MASTER IN COMPUTER SCIENCE

Evaluation of ARIS and BPMN using the UEML approach

Dossogne, Aurélie; Jeanmart, Cédric

Award date:
2007

Evaluation of ARIS and BPMN using the UEML approach

Aurélie Dossogne
Cédric Jeanmart

Abstract

Business process modelling appears as a central technique for dealing with entreprise strategy, processes development and usually predicating a need to change processes or identify issues to be corrected. Unfortunately, current process-oriented languages are not interoperable and not easily comparable with one another, nor with other modelling languages used to represent different information of an entreprise. This is a problem because the entreprise activity brings them to cooperate with each one, to exchange or to compare information. The design of the entreprise's business processes depends on the coordinated use of several modelling languages to represent different perspectives.

Our work applies a structured UEML 2.0 approach to describe two process-oriented languages: ARIS and BPMN. The main activity is to map the constructs of the languages onto an extensible ontology. We have defined the language semantics following the UEML method. The results have been validated through the UEML Validator and through a case study which result in improvement and consistency of ARIS and BPMN semantics definition.

Our thesis contributes to broadening UEML 2.0 by incorporating two process-oriented languages: ARIS and BPMN. As more process-oriented languages and other modelling languages are described and integrated into UEML, it should allow UEML to support entreprise model integration, translation, transformation through the mappings onto the common ontology. It permits languages comparison and checking the required global consistency between distinct entreprise models.

Keywords: Modelling language, ARIS, BPMN, Unified Entreprise Modelling Language (UEML), ontological analysis, common ontology.

Résumé

La modélisation des processus business apparaît comme une technique centrale concernant le développement de la stratégie et des processus de l'entreprise. Cette modélisation détecte habituellement le besoin de changer de processus ou identifie les problèmes à corriger. Malheureusement, les langages orientés processus existants ne sont pas interopérables et facilement comparables entre eux, ni avec d'autres langages de modélisation employés pour représenter certaines informations d'une entreprise. Ce fait pose problème étant donné que l'activité de l'entreprise les amène à coopérer entre eux, à échanger ou comparer de l'information. La conception des processus business de l'entreprise dépend de l'utilisation coordonnée de plusieurs langages de modélisation afin de représenter différentes perspectives.

Notre travail consiste à appliquer l'approche structurée UEML 2.0 pour décrire deux langages orientés processus : ARIS et BPMN. L'activité principale est d'identifier les correspondances entre les constructions du langage et une ontologie extensible. Nous avons défini la sémantique des langages suivant la méthode UEML. Les résultats ont été validés par l'UEML Validator et par une étude de cas. Ce qui a permis d'améliorer les résultats et de prouver la cohérence de ceux-ci.

Notre mémoire contribue à l'élargissement d'UEML 2.0 en incorporant deux langages orientés processus : ARIS et BPMN. D'autres langages orientés processus et langages de modélisation devraient être décrits et intégrés dans UEML. Cela devrait permettre à UEML de soutenir l'intégration, la traduction, la transformation de modèles d'entreprise grâce aux correspondances identifiées entre les constructions du langage et une ontologie commune. La comparaison entre les langages et la vérification de l'uniformité globale exigée entre les modèles distincts d'entreprises deviennent réalisables.

Mots-clefs : Langage de modélisation, ARIS, BPMN, Unified Entreprise Modelling Language (UEML), analyse ontologique, ontologie commune.

Preface

This report is mainly the result of a project carried out in the autumn of 2006 in the University of Bergen (Norway) and continued in the University of Namur in spring 2007. In Norway, we worked under the supervision of Prof. Andreas L. Opdahl who is a main actor in the UEML 2.0 version being part of INTEROP Network of Excellence.

We would like to thank Andreas L. Opdahl warmly for his guidance, his availability and his assistance brought during all our internship.

We would also like to thank our supervisor, Michaël Petit, for his advice given during the writing and for the opportunity he offered us to work on an interesting subject.

A special thanks goes to Raimundas Matulevičius who patiently read the different versions of the thesis and helped us improve it.

We would like to thank Patrick Heymans for his collaboration.

We would also like to thank *InterMedia*, the department which welcame us during our internship, for their warm welcome and for the place they gave us to work.

We would finally like to thank all the persons who helped and encouraged us during the work.

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

ARIS	Architecture of Integrated Information Systems
BPD	Business Process Diagram
BPM	Business Process Management
BPMI	Business Process Management Initiative
BPMN	Business Process Modeling Notation
BWW	Bunge-Wand-Weber representation model of information systems
EEML	Extended Entreprise Modelling Language
EM	Entreprise Modelling
EML	Entreprise Modelling Language
EPC	Event-Driven Process Chain
ER	Entity Relationship
GRAI	Graphs with Results and Actions Inter-related
IEM	Integrated Modelling Language
OWL	Web Ontology Language
PC	Program Committee
PCC	Program Committee Chair
UEML	Unified Entreprise Modelling Language
UML	Unified Modeling Language

Chapter 1

Introduction

The present-day enterprise world is such as it obliges enterprises to be flexible, and adaptable to frequent changes. An enterprise has usually to change its organization, its process, its products to satisfy the customers and continue to evolve. Then, they need to understand and to know the way they work. Enterprise Modelling (EM) is one of the solutions that allows enterprises to represent the way they work and all their important information in models. The purpose of EM is, on the basis of the models it creates, to allow enterprises to reconsider their process or more generally their strategies.

The purpose of EM gave rise to Enterprise Modelling Languages (EML). Those languages are the means that allow modelling of the enterprise information. They are the basis of all enterprise models. The problem is that the number of EML is quite large.

One group of EML is the process-oriented language. Process-oriented language is used by entreprises to model their business processes. It appears as a central technique for dealing with entreprise strategy, process development. It usually predicts a need to change processes or identify issues to be corrected. These modelling languages allow achieving analyses of the strategic business processes. The strategic business process analysis identifies key goals, business areas, preliminary new business process (which need to be designed) and even weakspots. The analysis also helps to decide which new information technology should be deployed. Unfortunately, current process-oriented languages and other modelling languages are not interoperable and easily comparable with one another. This is a problem because the enterprise activity leads them to cooperate with each one, to exchange or to compare information. The multiple languages do not facilitate the work. Enterprises lose a lot of time trying to understand a model created using another language. The syntax and the semantics are not the same, some languages provide information that others do not, some equivalent constructs can be slighty different because of their attributes. Thus, the design of the entreprise's business processes depends on the coordinated use of several modelling languages to represent different perspectives.

Unified Entreprise Modelling Language (UEML) was set up in an attempt to contribute to the need for EMLs interoperation. UEML is an intermediate language between existing EML. Its main objective concerns the support of enterprise model integration, translation, transformation and to support the required global consistency between distinct enterprise models. UEML would serve as an interlingua between EM tools, i.e. a common language spoken between different EM tools. Two versions (UEML 1.0 [UEMa] and UEML 2.0 [UEMb]) are developed. These two versions are related but different in their approach. The first version is based on three different languages while the second version aims at integrating new languages.

The main goal of our work is to evaluate two process-oriented languages, ARIS and BPMN, to be able to incorporate them in UEML. In order to integrate these languages, we analyse several modelling constructs by using the UEML approach. The main activity is to map the constructs of the languages onto an extensible ontology. Through their mappings onto the common ontology, ARIS and BPMN become available for comparison. Indeed, our work also compares the constructs of ARIS and BPMN. We contribute to the UEML 2.0 version. It developed a method to analyse EMLs and to define the semantics these languages.

The results have been validated through the UEML Validator and a case study which allows the

improvement of the mappings and prove the consistency of the results. The definition of the semantics allows the languages comparison and interoperability.

The document is divided into three parts. First, the background explained what the two UEML versions are. The BWW model and the UEML template will be explained. We present the two analysed languages, ARIS and BPMN.

Chapter 2 provides an overview and a comparison of the two versions of UEML. It also explains the UEML 2.0 meta-meta model.

Chapter 3 describes the main concepts of the BWW model. The concepts of the BWW model are maintained in a common ontology and provide a way of defining modelling constructs.

Chapter 4 explains how the UEML construct template works. The template is the technique used by UEML 2.0 to analyse languages.

Chapter 5 introduces the modelling language, ARIS. It explains the meta models useful for the analysis, an example and the main ARIS elements.

Chapter 6 deals with the modelling language, BPMN. This chapter provides the meta models used, an example and a explanation of the BPMN elements.

In the contribution part, we describe our research method for an ontological analysis. The analysis of ARIS and BPMN will be presented in two chapters. This part also shows the extension of the common ontology.

Chapter 7 shows our research method for ontological analysis, provides an evaluation of it and a proposal of a new method.

Chapter 8 focus on the analysis of ARIS. It describes in detail the analysis of one modelling construct and explains in brief all the other analyses.

Chapter 9 focus on the analysis of BPMN. This chapter also describes the analysis of one construct and explains the others in brief.

Chapter 10 shows the extension of the common ontology by providing the hierarchy of this common ontology and a explanation of two additions.

The third part will provide a validation, an evalution of the analyses. To validate this, two means are used, the UEML Validator and a case study. A comparison between ARIS and BPMN will be explained. To finish this part, we will evaluate the UEML approach.

Chapter 11 explains the use of the UEML Validator and the results obtained from its application.

Chapter 12 deals with a modelling of the process of conference's organization in ARIS and in BPMN. It provides a verification of the mappings of the analyses.

Chapter 13 focus on the comparison between ARIS and BPMN. It explains the possible results, the comparison between the constructs of each languages.

Chapter 14 gives an evaluation of the UEML approach, in particular the general ideas of the UEML approach, the UEML template, the common ontology, the similarity identification and the tools used.

Finally, we will conclude and propose future developments.

Chapter 15 provides a conclusion of the work, the limitions of this one and proposes future developments.

The work is followed by eleven appendixes:

Appendix A presents a description of the main concepts of BWW model.

Appendix B explains the concepts of the common ontology that we used in our analyses.

Appendix C provides the UEML 2.0 template tutorial.

Appendix D describes the ARIS meta models used for the ARIS analysis.

Appendix E explains the BPMN meta models used for the analysis of BPMN.

Appendix F shows the UEML 2.0 graphical representation standard used as legend for the figures representing the scene of modelling constructs.

Appendix G illustrates the legend of BPMN constructs.

Appendix H provides all the filled in templates of ARIS analysis.

Appendix I groups the filled in templates of each analysed modelling construct of BPMN.

Appendix J gives the complete results of the UEML Validator's application.

Appendix K gathers the comparisons between ARIS constructs and BPMN constructs by means of tables.

Part I

Background

Chapter 2

Overview of UEML

UEML stands for **U**nified **E**nterprise **M**odelling **L**anguage. The idea of a UEML has been introduced since 1992 and 1997 in the ICEIMT (International Conference of Enterprise Integration and Modelling Techniques) Conference series aiming at solving problems in Enterprise Modelling in order to reach a better enterprise integration and to provide an underlying formal theory for Enterprise Modelling languages [Ber03].
UEML can also be defined in a broad sense by explaning each acronym capital letter:

"**Unified** means
unified and shared linguistic context for
Enterprise Modelling means
supporting all the needed tasks for representing and utilizing enterprise knowledge through a
Language means
with well-defined syntax and, possibly, semantics." [Ber03]

In other words, unified means that UEML should have an unique definition, shared between all of its stakeholders. Then, UEML should allow supporting typical tasks found in methodologies for enterprise engineering and integration, in that sense it should be related to entreprise modelling. Finally, UEML should be a language. It should possess a formally defined syntax and a well-defined semantics.

UEML is an intermediate language between existing Enterprise Modelling languages (EML). Its main objective concerns the support of enterprise model integration, translation, transformation and to support the required global consistency between distinct enterprise models. These models are supposed to be represented in distinct Enterprise Modelling languages. UEML aims at providing correspondences between those languages to facilitate the model exchange and the required global consistency. In the longer term, the UEML can thus potentially support for comparison, consistency checking, update reflection, view synchronisation and, eventually, model-to-model translation across modelling language boundaries [OB06c].

In [Ber05], G. Berio explains that "dealing with Enterprise Modelling raises two key issues that make the objectives of UEML very challenging:

1. which languages are Enterprise Modelling languages;

2. the informal nature of the underlying meaning (also called semantics) of many Enterprise Modelling languages.

According to point (1) several languages can be included, each of them concerning some aspects of enterprises. Some of these languages often come from other disciplines; as a consequence, languages for Enterprise Modelling are often very different in their nature, therefore difficult to be related by the advocated basic correspondences.
According to point (2), the meaning of a construct of an Enterprise Modelling language is often provided by a text in English, French, Italian etc. Therefore, the phenomena, that those languages are able to represent, are often unclear."

A first UEML version was established in the UEML Thematic Network (TN) (2002-2003), whereas a a second version is currently being developed in the INTEROP Network of Excellence (NoE) (2003-2007).

In Section 2.1 we will explain UEML 1.0, in Section 2.2 we will discuss UEML 2.0. We will provide the UEML 2.0 meta-meta model in Section 2.3 and compare both approaches in Section 2.4.

2.1 UEML 1.0

UEML is a Themantic Network Project (IST-2001-34229) financed by European Union (EU). The project started on March 1st 2002 and ended on May 30th 2003. The aim of UEML is to create working groups to develop UEML core. [UEMa]

As reported in [PD02], the idea of the UEML project was to contribute to solve the problems of multiple Enterprise Modelling Languages. The long term objective of UEML is the definition of a Unified Enterprise Modelling Language, which would serve as an interlingua between Enterprise Modelling (EM) tools. This language will:

- Provide a common visual, template based language
- Provide standardised mechanisms for sharing knowledge models and exchanging enterprise models among projects, overcoming tool dependencies
- Support the implementation of enterprise model repositories to leverage enterprise knowledge

2.1.1 Strategy for UEML 1.0

UEML 1.0 is defined by integrating three specific entreprise modelling languages *GRAI*, *IEM*, *EEML*. During the project, a methodology named "Strategy for UEML" has been applied to define UEML 1.0 with these three languages [Ber03]. This process allows understanding and analysing choices, hypotheses and how tasks have been performed during the UEML definition. Figure 2.1 gives a precise overview of planned tasks in this process.

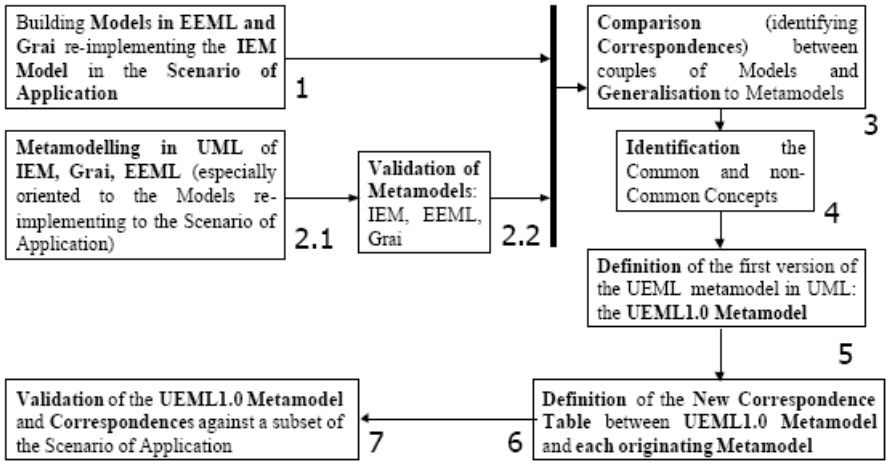


Figure 2.1: The Strategy for UEML 1.0 (from [Ber03])

The steps of this strategy are as follows :

1. A scenario (a concrete reference situation to be modelled) is defined. This scenario is modelled in each of the three considered languages. This task must be achieved by experts in each considered languages. Despite of that, subjectivity remains possible during the modelling.

2. **Meta models**

   2.1 The meta models of each language are defined in an UML classes model. These meta models represent the abstract syntax of the languages by a set of meta models' artefacts and the relationships between these.

   2.2 UML experts validate these meta models.

3. The correspondences between the models (built during the first step) must be identified and the meta models are generalized.

4. The common concepts and non-common concepts are identified.

5. On the basis of the common concepts identified at the step 4, a version of UEML can be defined and represented by an UML classes model (UEML 1.0 meta model).

6. A final version of the semantic correspondences between each considered languages and UEML 1.0 meta model is defined.

7. A final validation of UEML 1.0 meta model and the correspondences is done.

The UEML meta model which results of the application of the strategy is shown by the Figure 2.2.

Table 2.1 (result of the step 6) sums up the semantic correspondences between the UEML 1.0 meta model and the meta models of the three considered languages. For instance, for the first lane, the common concept is the *Activity* which is a class of the UEML 1.0 meta model (See Figure 2.2). This class is characterized by the *extended activity* in *GRAI*, the *action state* in *IEM* and the *task* in *EEML*.

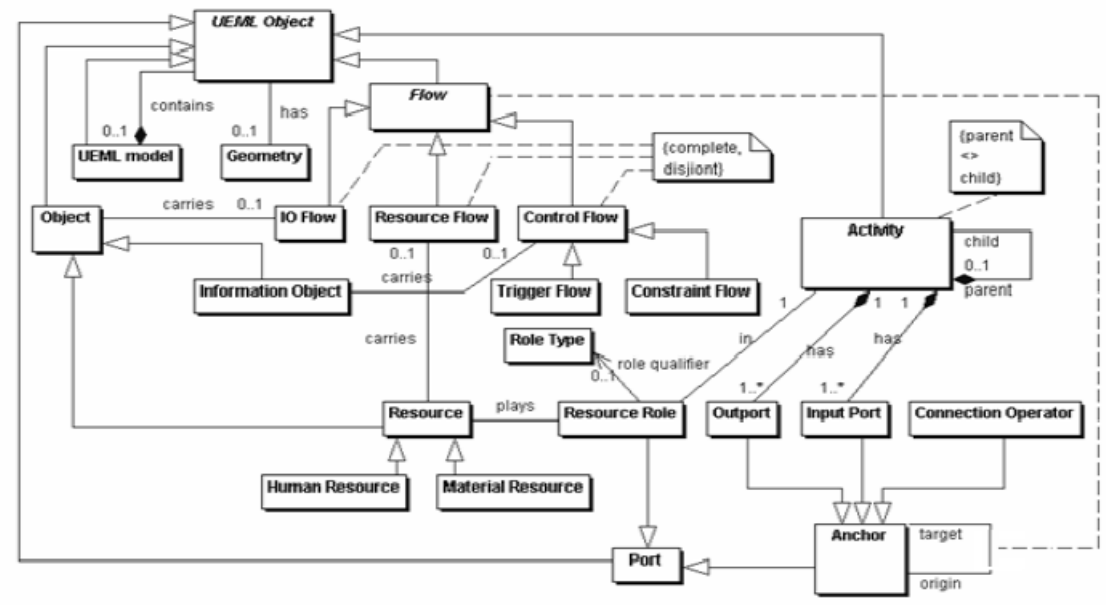


Figure 2.2: UEML 1.0 meta model (from [BPP04])

Table 2.1: Semantic correspondences between IEM, EEML, GRAI and UEML 1.0 (from [BPP04])

CONCEPTS COMMUNS	GRAI	IEM	EEML
ACTIVITY	Extended activity	Action state	Task
ROLE	Not explicit	IEM Object state	Role
RESOURCE	Resource	Resource class	Resource
INPUT/OUTPUT FLOW	Input / Flow Ouput / Flow	Successor / ProcessElement	Flow (with is control flow = false)
CONSTRAINT FLOW	Control / Flow (with trigger=false)	No direct equivalence	Flow (with is control flow = false and Role)
CONTROL FLOW	Control / Flow (with trigger=true)	ControlSuccesor / ProcessElement	Flow (with is control flow = true)
RESOURCE FLOW	Resource / Flow (with trigger = false)	ResourceSuccessor / Resource State	Role (with Task)
CONNECTION OPERATOR	Logical operator	Connection Element State	DecisionPoint (not (Inport or Outport))
PORT	Connector	Port	Decisionpoint (Inport or Outport)

2.1.2 Benefits and problems

The benefits and problems of UEML 1.0 given in [Ber05] can be summarized as follows:

Benefits

- *Practical:* The method proposes at least one suitable way to map one language onto another (through a set of correspondences and UEML constructs).
- *Conceptual:* The model artefacts represented in a language and further represented by UEML constructs, are understandable in terms of what they are intended to represent.
- *Potential:* Apart from the simple exchanges of models that can be realised by using only the identified correspondences, more complex exchanges may also be achieved if a mapping language is available for representing mappings over metamodel artefacts.
- *Architectural:* The approach makes possible to implement an architecture in which it is possible to provide a uniform interface for accessing models represented in several languages.
- *Methodological:* New relationships between UEML constructs, available between language constructs (because distinct languages are not related) can be identified (and new methods and methodologies can be developed).

Other advantages of the use of the strategy for UEML can be pointed out as quoted in [BPP04]:

- to provide a well-defined context to make choices in connection with the manner of building UEML by re-using existing EMLs
- to base on existing and proved approaches coming from theories suggested before for the integration of diagrams and data's in the field of data bases
- to suggest the useful mechanisms to carry out translations of models between several EMLs, thanks to UEML, definite independently of particular models

Problems

- The proposed approach seems difficult to be generalised, to be suitable for managing situations of potential inconsistencies of correspondences (e.g. a construct in a language can represent several constructs in another language), and to be independent from the models used to find out the correspondences and modellers building these models.
- The approach does not guarantee that the exported model artefacts according to the correspondences are "formally semantically equivalent" i.e. meaning preserving. Indeed, between two fully formalised languages (even between a language and itself) there are several possible correspondences: the problem is still how to identify correspondences that might be basic correspondences.
- The advocated specific mappings that realise complex model exchanges can be represented if a specific mapping language is available: in this case, we can say that models are exchanged by using this mapping language. However, once more time, the UEML 1.0 approach does not help to formally proof (correctness) properties of these exchanges.

2.2 UEML 2.0

UEML 2.0 is currently being developed in the INTEROP Network of Excellence (NoE) (2003-2007).

The Unified Enterprise Modelling Language (UEML) is an ongoing effort to develop an intermediate language for modelling enterprises and related domains, such as information systems [OB06c]. Being an intermediate language, the aim of UEML is not to propose new modelling constructs or new visual model presentations. Instead, the aim is to integrate existing modelling languages in a structured and cohesive way [Opd06b]. In the longer term, the UEML can thus potentially support for comparison, consistency checking, update reflection, view synchronisation and, eventually, model-to-model translation across modelling language boundaries [OB06c]. The UEML work in INTEROP has three main activities [Opd06b]:

1. determining requirements for UEML
2. selecting languages to incorporate into UEML
3. describing the modelling constructs that are chosen as part of UEML

The focus of our work is related to the third activity. On the basis of some constructs of two languages, we tried with the UEML template to reveal classes, properties, states and events instanciating the meta meta model.

The UEML 2.0 general approach is shown by Figure 2.3.

2.2.1 The approach

In this subsection, we explain the mechanisms of the UEML 2.0 approach. We present the three activities of the UEML 2.0 general approach and the UEML construct template. This subsection is inspired by [BOAD05].

The UEML requirements determination approach

The first step is intended to determine the requirements for UEML. Those requirements are derived from the users' needs. A method for the elicitation and the collection of those requirements have been defined. This method is based on a requirement template. This template is simple and fonded on the fact that the users can explain the needs in more details. In that sense, the template is enacted to reformulate the needs using different statements. The first set of requirements was constructed from different Enterprise Modelling application domains and from various users' requirements.

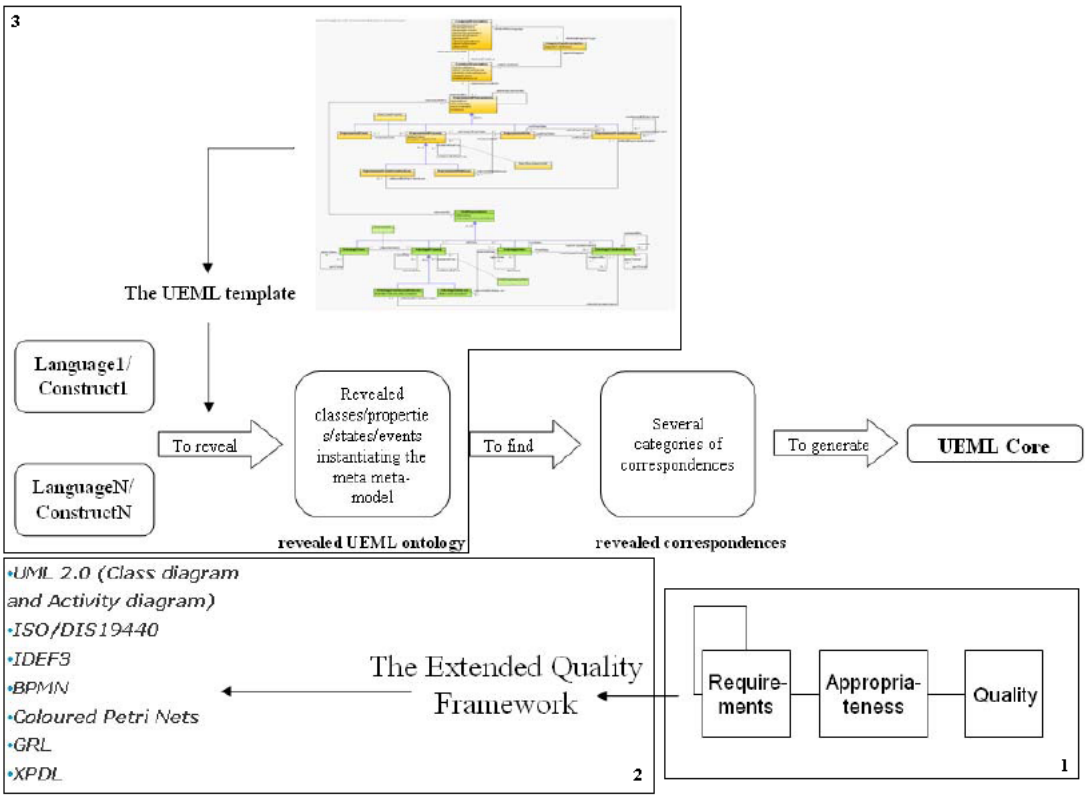


Figure 2.3: UEML 2.0 general approach (adapted from [Ber06])

The languages selection approach

The second activity is the selection of the languages to incorporate into UEML. The approach is based on the previous step and uses a set of quality criteria linked to the requirements of the first step. These selected languages are also evaluated with a language template. Then, they are incorporated construct-by-construct on the basis of the information collected by the INTEROP partners with the construct template.

More explicitely, as shown in Figure 2.4, the language selection begins with the definition and the update of the language list. This is done on the basis of the states of the art and of the partners' experience. According to that, the evaluation of the qualities of languages requires the definition of well-defined and/or measurable quality criteria. A definition and an update of quality criteria must be defined. To facilitate this step, the quality framework [Kro95] essentially provides the neutral definition of the several quality types of a language: neutral means that the quality framework can be applied to languages that are used for modelling without any regards to the specific domains. The quality framework helps in assessing the coverage degree of the quality criteria, according to the various quality types. Quality criteria can be defined starting from elicited requirements or just independently according two ways. From those quality criteria, the language template must be updated. This language template has been defined for collecting information about languages (for instance, "number of constructs") to be used by the methods for evaluating quality criteria. Another way to collect information about languages is the criteria-driven questionnaires. The objective is to perfom a statistical evaluation of the quality criteria. Then, the language template must be filled in. On this basis, languages are evaluated according to the quality criteria. All this process allows carrying out the languages selection. This selection can probably not be done at once and goals for using the approach have to be included also in the selection process.

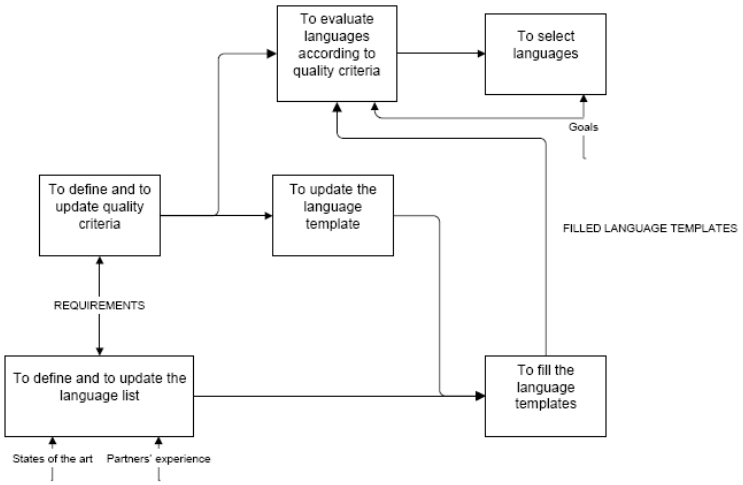


Figure 2.4: Tasks and their dependencies to perform the selection of languages for UEML (from [BOAD05])

The UEML construct description approach

The UEML construct description approach works as follows [Opd06b]:

1. A construct description is created in a structured and cohesive manner for each modelling construct that is to be incorporated into UEML (Figure 2.5).

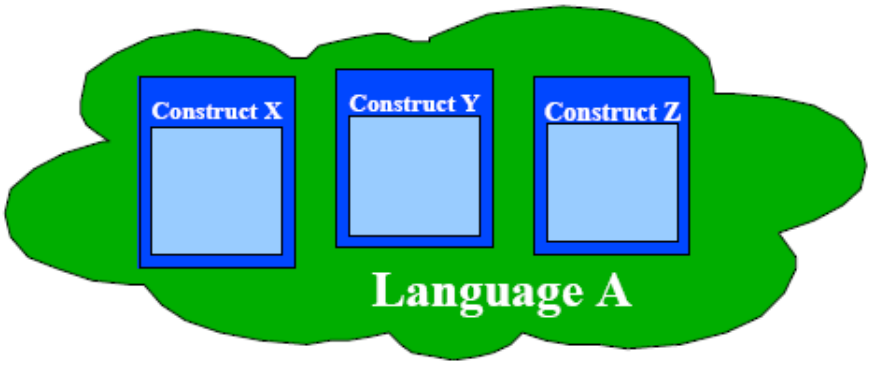


Figure 2.5: Incorporating a language into UEML (from [Opd06b])

2. The construct description has both a presentation part and a representation part (See Chapter 4). The presentation part describes the visual presentation of the modelling construct and the representation part describes the phenomena it represents, i.e. the semantics.

3. The representation part uses a referential decomposition to split each modelling construct into its ontologically atomic parts, defined as a part that maps one-to one with an ontology concept (See Chapter 3).

4. The ontology concepts are maintained in a common ontology, which grows incrementally as more modelling constructs are incorporated into the UEML (See Chapter 3).

5. The common ontology is hierarchically organized, shown in Figure 2.6.

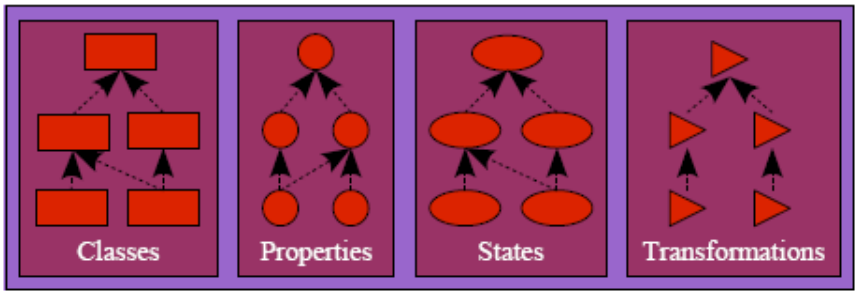


Figure 2.6: The common ontology (from [Opd06b])

6. Three types of interrelations can exist between constructs at the most detailed possible level via the common ontology: If two modelling constructs are identical, they will map onto the exact same ontology concepts. If two modelling constructs do not overlap at all, they will map onto completely distinct ontology concepts, i.e., onto concepts which are not even hierarchically related. The third case is likely to be most common, where two modelling constructs map onto some identical ontology concepts, some ontology concepts that are hierarchically related and some ontology concepts that are completely distinct. But in all cases, the hierarchically organised common ontology enables to determine the exact representational ("semantic") relationship between any pair or group of modelling constructs. In Figure 2.7, constructs A-X and B-W designates the same ontology class and properties, but B-W designates more specific states and transformation.

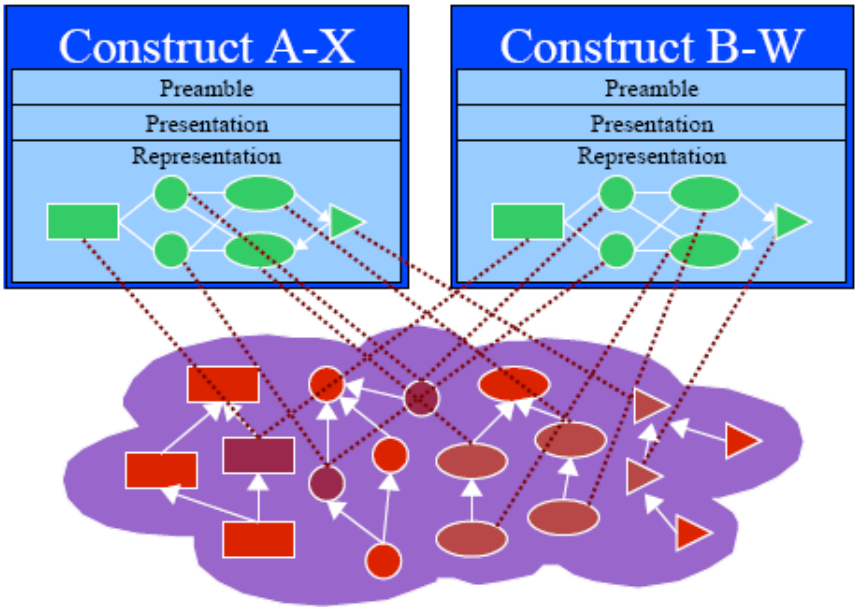


Figure 2.7: Representational relationship bewteen modelling constructs (from [Opd06b])

7. A meta-meta model is provided to account for the representation part of the UEML approach (it does not yet account for the presentation part) (See figures 2.8 and 2.9).

UEML contruct template

The UEML construct template provides a standard for describing modelling constructs. The template was used for compiling the initial version of UEML 2.0. It provides a common structured format for describing modelling constructs, guiding information collection, supporting distributed work and ensuring that the resulting descriptions are consistent. It clearly and systematically distinguishes how a construct is presented, e.g., lexical and syntactical issues, from what a construct represents, i.e., reference, an important aspect of semantics, and from how a construct is used, e.g., pragmatics, although pragmatics has been less investigated in INTEROP so far [OB06a].
This template is thoroughly explained in Chapter 4.

2.2.2 Tools

UEMLBase

Based on the meta-meta model shown by Figures 2.8 and 2.9, a prototype tool for managing construct descriptions, UEMLBase, has been developed using the OWL plug-in for the Protégé tool. The intention is that the tool should replace the paper-based construct template used so far. Construct descriptions created using the UEML template are currently being negotiated between the partners and entered into UEMLBase, revealing a first version of the common ontology.

UEML Validator

The UEML Validator allows to check the UEMLBase consistency and in particular the analyses. It is the main objective of the Validator, besides this goal, the validation enables to emphaze usual mistakes in order to create a list of mistakes that can be easily avoided. This checking is based on a prolog file composed of rules written to verify the consistency of the UEMLBase [Mah06].

2.2.3 Ontological analysis

The UEML 2.0 approach is an ontological approach. Indeed, an ontological analysis can be defined as the evaluation of a selected modelling grammar from the viewpoint of a pre-defined and well-established ontology [GR05b]. Particularly, the UEML approach uses the common ontology to evaluate some selected modelling grammars which are indeed the selected languages. Thus, the UEML approach is a particular case of ontological analysis.

Several shortcomings in the current practice of ontological analysis can be identified. The identification of such shortcomings will provide a basis upon which the practice of ontological analysis can be improved. Those reported in [GR05b] and [GRI04] are summarized as follows:

- Shortcomings related to the quality of the input data:
  - Lack of Understandability: An ontology may not be clear and intuitive. It can lead to misinterpretations.
  - Lack of Comparability: Unless the ontology and the grammar are specified in the same language or a precise mapping of the two languages exists, it will be up to the coder to "mentally convert" the two specifications into each other, which adds a subjective element to the analysis.
- Shortcomings related to the process of the ontological analysis:
  - Lack of Completeness: The difficulty in clearly specifying the boundaries of the analysis, as well as the limited consideration of relationships between the ontological constructs, lead sto a potential lack of completeness.
  - Lack of Guidance: There are hardly any recommendations on where the analysis should start, i.e. in what sequence the ontological constructs and relationships will be analysed.
  - Lack of Objectivity: The individual interpretations of the involved researcher in the analysis adds significant subjectivity to the results.

- Shortcomings related to the outcomes of the analysis:
  - Lack of Adequate Result Representation: The result's tables can become quite lengthy and are typically not sorted out in any particular order.
  - Lack of Result Classification: In general, the ontological analysis does not make any statements regarding the relative importance of identification of ontological deficiencies based on a comparison of the constructs in the ontology and the modelling technique.
  - Lack of Relevance: An ontological analysis should be perceived as relevant by the related stakeholders.

The shortcomings identified above motivate the development of an enhanced methodology for ontological analyses. The main purpose of this methodology is to increase the rigour, the overall objectivity and the level of detail of the analysis. The proposed methodology for ontological analyses, in [GR05b], is structured in three phases. These are: input, process and output.

Input

The first step is to convert the ontology, as well as the selected modelling grammar, to meta models using the same language (e.g., ER models or UML class diagram). If the meta models for the ontology and the modelling grammar are specified in the same language, the ontological analyses evolves into a comparison of two conceptual models.

Process

The second step is to clearly specify the scope of an analysis using those meta models. Then, the analysis can start with the representation mapping. That is, selecting the meta model of the ontology and subsequently identifying the corresponding elements in the modelling grammar. The first construct to be analysed should be the most central entity type - that is, in the case of the BWW models the entity type thing. This analysis should be followed by a cluster-by-cluster approach. Starting with the core constructs in a cluster, this approach allows a more structured and focused analysis of the completeness of a modelling grammar. The analysis of the entity types is followed by the relationship types and the cardinalities. The representation mapping is followed by an analysis of the clarity. That is, the interpretation mapping. In this case the meta model of the grammar under analysis is the starting point. The general procedure is similar.

In order to improve the validity of the analysis, a research methodology can be adopted that undertakes individual analyses of a particular grammar by at least two members of a research team, followed by consensus as to the final analysis by the entire team of researchers. The methodology consists of three steps:

1. Using the specification of the grammar in question, at least two researchers separately read the specification and interpret, select and map the ontological constructs to candidate grammatical constructs to create individual first drafts of the analysis.

2. The researchers involved in step 1 of the methodology meet to discuss and defend their interpretations of the modelling technique analysis. Then a concurrence score is determined from their initial analyses. This meeting leads to an agreed second draft version of the analysis that incorporates elements of each of the researchers' first draft analyses. The overlap in the selection of the constructs and in the actual ontological analysis can be quantified by concurrence/agreement scores that are used in content analysis and other more qualitative research.

3. The second draft version of the analysis of the modelling technique is used as a basis for defence and discussion in a meeting involving the entire research team. The outcome of this meeting forms the final analysis of the grammar in question.

Output

The meta models that have been used as input for the ontological analyses are also an appropriate medium to visualise the outcomes of the entire analysis process. At the present time, the process of an ontological analysis results in the identification of ontological incompleteness and ontological clarity through the identification of missing, overloaded or redundant grammatical constructs. While the end result identifies such problems, it fails to account for their relative importance. There is a need for the

development of a scoring model that enables the calculation of the "goodness" of a grammar with respect to the ontology.

The UEML approach has some similarities with the enhanced methodology for ontological analyses described above. For example, as explained previously, both methods use an ontology to evaluate modelling grammars. Also, the UEML approach and the reference methodology make an interpretation mapping. Those particular things show that the UEML approach is really a particular case of the reference methodology.

2.2.4 Benefits and problems

All the benefits concerning UEML 1.0 and mentioned in Section 2.1.2, are mostly valid for the UEML 2.0. Additionally, the UEML 2.0 approach improves the UEML 1.0 approach as follows:

- It guides, according to a meta-meta model, the representation of abstract syntax, semantics and semantic domain for any Enterprise Modelling language.
- It allows to infer basic correspondences between distinct languages, once their semantics and the semantic domain have been represented.
- While not suggested, it allows to represent some UEML constructs and their semantics whenever the semantic domain contains enough information (i.e. whenever a significant number of relevant languages have been represented).

As for UEML 1.0, the undertaken approach does not allow:

- To formally proof properties of basic correspondences and more complex exchanges; while the meta-meta model (Figures 2.8 and 2.9) is represented with an UML class diagram, it can be represented in some formal language enabling reasoning.

Other advantages of the UEML approach can be pointed out as quoted in [Opd06b] :

- It offers more detailed advice on how to proceed when analysing individual language constructs.
- It thereby encourages construct description at a high level of details, which tends to integrate languages at a fine-grained, precise level.
- It is also systematic, producing construct descriptions that are complete and easily comparable.
- It supports ontological analysis in terms of particular classes, properties, states and events, and not just in terms of the concepts in general.
- It acknowledges that a language construct often represents a scene played by several ontological concepts together.
- It suggests a path towards tool-supported, integrated use of models expressed in different languages, through the structured format in combination with the common ontology.
- It has positive externality, in the sense that each construct becomes easier to incorporate as more constructs are already added to the UEML and each language becomes easier to incorporate as more languages are already added.

As given in [Ber05], several problems of the UEML 2.0 approach can be underlined. Accordingly, the first approached problem is how to organise the semantic domain. Generally speaking, the semantic domain represented according to the meta-meta model is organized in two distinct parts: a static part and a behavioural part. In the static part, there are classes of things, the construct is intended to represent, and the relevant properties of these things. In the behavioural part, there are events and states concerning these things. The second approached problem is what classes, properties, events and states should be used for representing a language. In the first attempts performed in INTEROP, the idea has been to use an existing ontological theory, specifically the very general BWW, that allows to distinguish between several real world phenomena (while we are not constrained by any ontological theory).

2.3 UEML 2.0 meta-meta model

A UEML 2.0 meta-meta model (Figures 2.8 and 2.9) is provided to account for the representation part of the UEML approach. It is called a meta-meta model because it is a model of how to model languages and because models of languages are called meta models [Opd06b]. The UEML template is based on this meta-meta model.

The meta-meta model is divided into three parts: the "preamble" part, the "represented" part and the "ontology" part. The "preamble" part is represented by the top layer of the meta-meta model and deals with modelling languages, their diagram types and their modelling constructs. This part is composed of the classes *LanguageDescription*, *DiagramTypeDescription* and *ConstructDescription*. Indeed, a construct description belongs to a language and is used in a diagram type. The "represented" and "ontology" parts are represented by the middle and bottom layers and deal with individual construct descriptions and with the common ontology. The "represented" part is composed of the class *RepresentedPhenomenon* which is divided in four classes: *RepresentedClass*, *RepresentedProperty*, *RepresentedState*, *RepresentedTransformation*. This represented phenomenon described the construct by defining the class/property/state and transformation that the construct represents. The "ontology" part is composed of the class *OntPhenomenon* which is divided into four classes: *OntologyClass*, *OntologyProperty*, *OntologyState*, *OntologyTransformation*. In the meta-meta model (Figures 2.8 and 2.9), we can see a link (represents) between *OntPhenomenon* and *RepresentedPhenomenon*. Thus, an ontology phenomenon represents a represented phenomenon.

2.4 Comparison

As explained in [Ber05], the UEML 2.0 approach can be compared to the UEML 1.0 approach by Figure 2.10. In Figure 2.10, the most important similarities become evident: both require to represent abstract syntaxes but in UEML 2.0 there is a standardised way to do the work, according to the meta-meta model. Additionally, the UEML 2.0 requires to represent the semantic domain and the semantics explicitly. Specifically, the semantic domain corresponds to objects of classes OntologyTransformation, OntologyState, OntologyProperty and OntologyClass depicted in Figure 2.9. The semantics corresponds to the links between RepresentedClass, RepresentedProperty, RepresentedState and RepresentedTransformation (which objects nearly correspond to abstract syntaxes), depicted in Figure 2.8, and the associated classes of the semantic domain.

These two approaches share similar ideas but the envisioned mechanisms to implement these ideas are different. While UEML 1.0 is pragmatics, i.e. guided by pratical experience rather than theory, UEML 2.0 explicitly requires the definition of a common semantic domain for languages, grounding the future work on correspondences on this domain. However, an important research work is nevertheless required to fully characterize the semantic domain and how basic correspondences can be characterized in term of properties on this domain.

Therefore, the UEML 1.0 approach is different than the UEML 2.0 approach. UEML 1.0 was defined on the basis of three languages set up by project partners (*GRAI*, *IEM* and *EEML*). The idea developed in the UEML 1.0 was to state some basic correspondences between those three languages by using examples. Whereas UEML 2.0 is intended to incorporate construct-by-construct from different existing Entreprise Modelling Languages. UEML 2.0 isn't reduced to only three languages. UEML 2.0 is more developed than UEML 1.0. The aim is to integrate existing modelling languages in a structured and cohesive way. Besides the opening to all languages, UEML 2.0 described all the modelling constructs of a language in a standard form using a template. The approach of UEML 2.0 also uses an ontology, all the described constructs are defined with ontology concepts. Those concepts are maintained in a common ontology which grows incrementally as more modelling construcs are incorporated into the UEML. This common ontology is based on Bunge's ontological model ([Bun77], [Bun79]) and the Bunge-Wand-Weber representation model (the BWW model, [WW88], [WW93], [WW95]) of information systems.

The following Table 2.2 summarized all the comparison between UEML 1.0 and UEML 2.0 on the basis of four criteria. Both first ones are the general approach and the number of languages to incorporate in UEML. The third one is the representation of the abstract syntax, i.e. a syntax that focuses on the

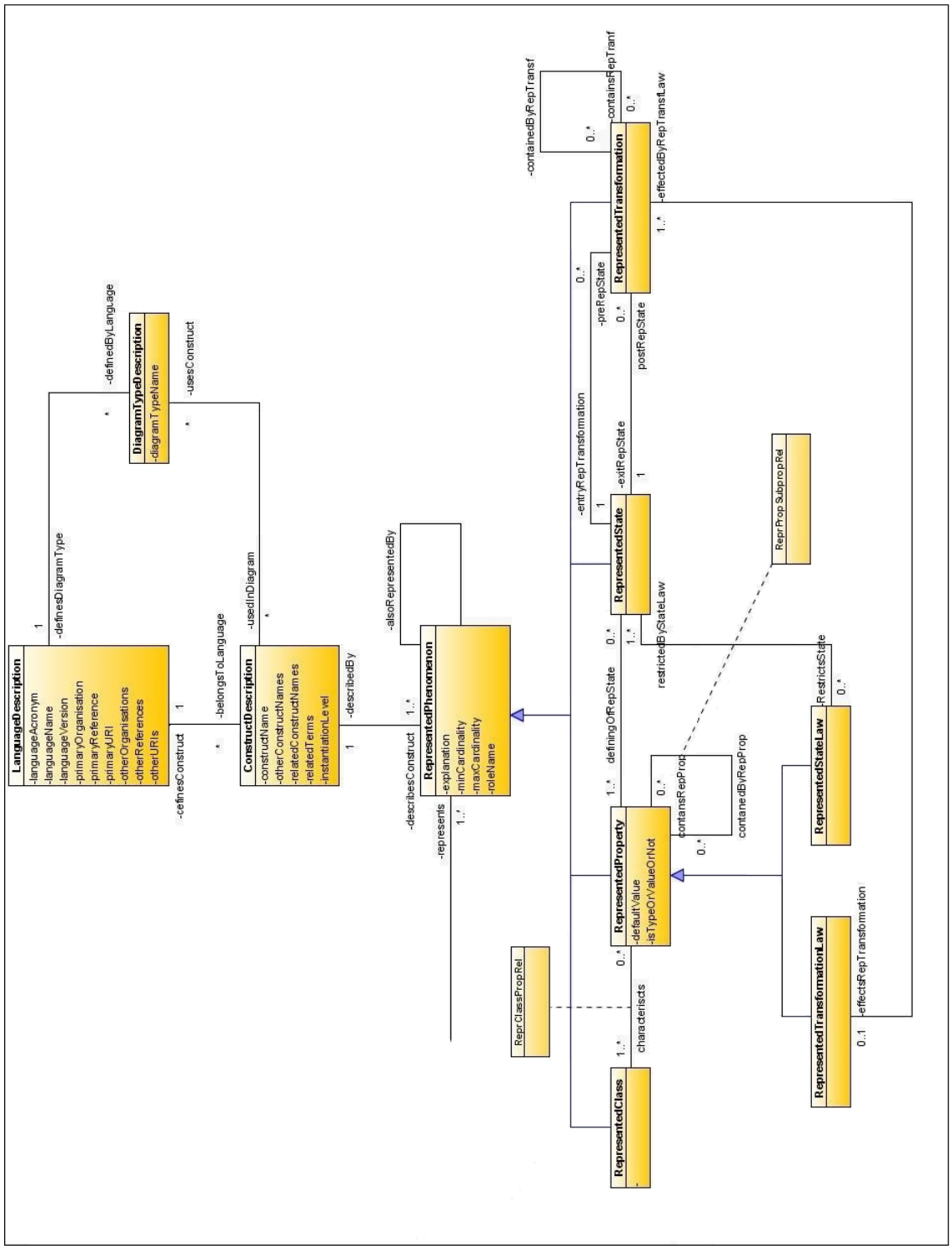


Figure 2.8: UEML 2.0 meta-meta model - Upper part

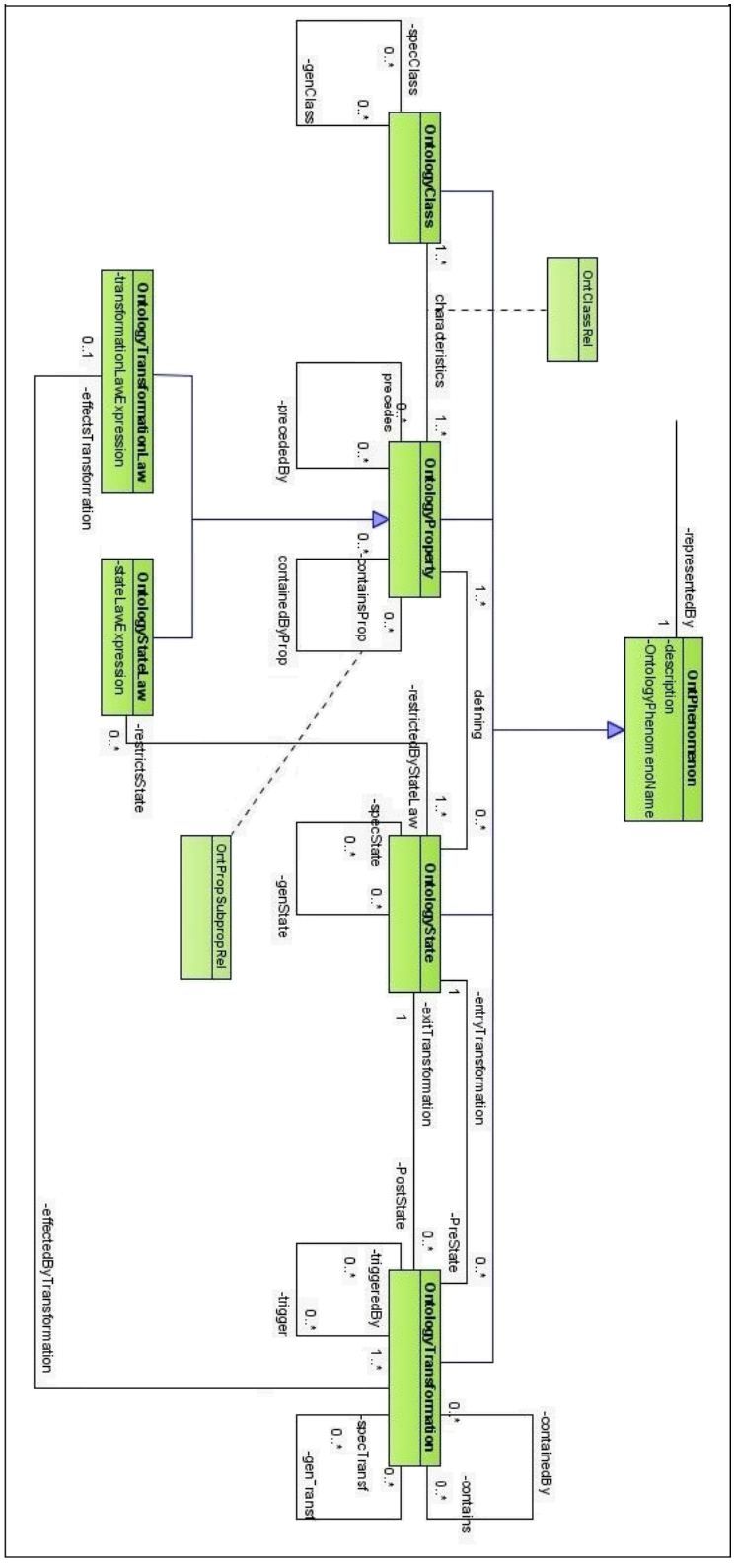


Figure 2.9: UEML 2.0 meta-meta model - Lower part

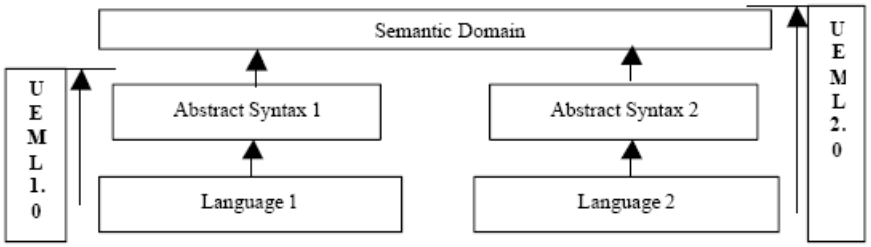


Figure 2.10: Comparing UEML 2.0 approach and UEML 1.0 approach

constructs of a language and their structure [Ber05]. The last one is the representation of the semantic domain and semantics.

Table 2.2: Comparison between UEML 1.0 and UEML 2.0

Criteria	**UEML 1.0**	**UEML 2.0**
Approach	State some basic correspondences between three languages	Incorporate construct by construct by using the UEML template and an ontology
Number of EMLs to incorporate	Three languages	Open to all languages
Abstract syntax representation	Yes	Yes, but in a standardised way
Semantic domain and semantics representation	No	Yes

2.5 Summary

Enterprise model integration, transformation, translation are today an essential issue for building complex systems that show high autonomy of constituents, and robustness to changes and evolution. UEML was created to address this issue, and to make the enterprise integration easier.

As described in [BPP04] and [Ber05], UEML alone is not the full solution to the model exchange and to consistency. Indeed, the need to support model integration, transformation and translation also appears whenever models are expressed in only one language. The language is always the same, while the things the language represents (in the models) are not necessarily the same ones. Starting from this point of view, UEML would tend to describe possible basic correspondences between distinct modelling languages that are, in principle, context independent. In that sense, UEML gives a way to tackle the problem at the language level. To understand correctly models elements which are put in common with UEML, additional information is needed. This need is linked to the distinction between the two instantiation levels (language/model). A clear understanding of the semantic links existing between models is necessary to the exchange or interoperability of models thanks to a common language.

In this chapter, we explained the purpose of UEML and its two versions: UEML 1.0 and UEML 2.0. Then, we have compared both versions.

As reported in [Ber03], UEML 1.0 represents the successful application of the "Strategy for UEML". It provides the correspondences between the three modelling languages *IEM*, *EEML*, *GRAI* integrated and the UEML 1.0 meta model. The "Strategy for UEML" can be made generic and can be reused for improving the UEML 1.0 by applying it with other enterprise modelling languages.

The three main activities of UEML 2.0 are (1) the determination of requirements, (2) the languages selection and (3) the description of modelling constructs with the UEML template. As reported in [OB06c], starting from high level requirements established in the UEML TN (UEML Thematic Network), more focused requirements were collected by using a new method based on a requirement elicitation template. Next, languages were selected using a set of quality criteria, linked to the collected requirements, evaluated on information collected from INTEROP partners using a language template. Currently, the selected languages are being incorporated construct-by-construct into UEML using a construct template.

Chapter 3

The BWW model

As explained in [OHS04], the BWW model was used, for instance, for general analyses of information system design theory and object-oriented modelling constructs. The BWW was therefore a good start for defining enterprise modelling constructs. The main idea was to provide a way of defining modelling constructs in terms of the BWW model. The concepts are then maintained in a common ontology [1] (based on the BWW model) which grows incrementally as more modelling constructs are incorporated into the UEML.

This common ontology is based on Bunge's ontological model and the Bunge-Wand-Weber representation model of information systems. The initial classes, properties, states and events are drawn from Bunge's ontology and the BWW-model, but the common ontology also grows dynamically as more specific classes, properties, states and events are included. Ontology classes are organised using generalisation/specialisation and using aggregation/decomposition. Properties are organised using property precedence (being human precedes having a responsibility). States are similarly organised using super-/substate relationships and events using super-/subevent relationships.

3.1 Description of the BWW model

In [Opd], A.L. Opdahl describes all the different categories of the BWW model and their relations with the template. We'll now present them.

Thing, property

Thing and *property* are the two most fundamental ontological concepts. The world consists of things with properties. Things possess properties and properties belong to things.

Properties cannot themselves have properties. But a property can be a set of "sub-properties".

Model thing, attribute

We cannot know things in the world or their properties directly. We only know our mental (conceptual, cognitive) representations of those things and their properties. We call our mental representations of real things *model things*. We ascribe attributes to model things to represent the real properties we think the corresponding real things possess.

Collection of things, "group" of properties

When we want to refer to more than one thing, we can talk about a *collection of things*. When we want to refer to more than one property, we can talk about a *group of properties*.

Complex property, "sub-property", "intersection"

A property is *complex* when it has "sub-properties". Distinct complex properties may have some (but not all) of the same "sub-properties". We say that complex properties with common sub-properties are "intersecting".

[1] The concepts of the common ontology are explained in Appendix B

Property precedence/"preceded by"

A property p precedes another property q if and only if all real things that possess q also possess p. A "sub-property" precedes its complex property. A complex property p precedes another complex property q if all sub-properties of p are also sub-properties of q. More strongly, p precedes q if each sub-property of p precedes some sub-property of q.

We say that q entails p if and only if p precedes q. Hence, "preceded by" is the inverse of precedence. A complex property entails its "sub-property". A complex property q entails another complex property p if each sub-property of q entails some sub-property of p.

Property precedence/"preceded by" is central in the template approach because it is the main mechanism for structuring properties in the common ontology.

Property in particular and in general

We can talk about *particular properties* of individual things, as in "My bike is red." Here, the redness of my particular bike is a particular property. We can also talk about *general properties* possessed by many things, as in "Red bikes are nice." Here, the redness of any red bike is a general property.

An individual thing possesses a particular property. A collection of things may possess the same general property.

Class, characteristic property

A *class* is a collection of things that all possess the same general property. We say that the class is characterised by (or defined by) the property. All the things in the class must possess the class' *characteristic property*, and no things not in the class can possess it. The characteristic property of a class can be a complex property with sub-properties.

Subclass, "generalisation", "specialisation"

A class Q is a subclass of another class P if all the characteristic properties of P are also characteristic of Q. More strongly, Q is a subclass of P if and only if each characteristic property of P precedes a characteristic property of Q.
When Q is a subclass of P, we say that P generalises Q and that Q specialises P.

Class generalisation/specialisation is central in the template approach, because it is the main structuring mechanism for classes in the common ontology. Generalisation/specialisation relationships between classes parallel precedence/"preceded by" relationships between properties.

Property function, time, datatype, value

The *time* is represented as an ordered set of points in time. A datatype was defined as a set of values. Then, a property of a thing can be described as a property function with a subset of time as domain and a datatype as co-domain.

The *property function* is defined for each point in time at which the property is real (belongs to a thing). For each such point in time, the property function assigns a value to the property. This value must of course be a member of the appropriate datatype.

The lifetime of a thing is the set of points in time at which there is at least one property function defined for the thing. Hence, the lifetime of the thing represents the time during which the thing is real (possesses at least one property). Like time, a lifetime is an ordered set of points in time.

State, event, history

For an individual thing at a particular point in time, the property functions that are defined at that point in time can be arranged to form a vector. For each point of time in the lifetime of the thing, this vector in turn defines a vector of property values for the thing. This vector of property values is called the *state* of the thing.

A change of state in a thing is called an *event*. An event is defined by its pre- and post-state. An event in a thing occurs at a particular point in time.

The sequence of states a thing goes through in its lifetime is the *thing's history*. Alternatively, a thing's history can be defined as the sequence of events it undergoes in its lifetime.

Classes do not have states, events or histories, but it is possible to talk about (the set of) states that things in a class can have and (the set of) events that things in a class can undergo. In this sense, we can talk about the "states", "events" and even "history of a class".

Consecutive events, process

Two events in a thing are *consecutive* if and only if the post-state of one is the pre-state of the other. Consecutive events form chains and trees of events. A tree of events in a thing is called a *process*. Because an event is defined as a pair of states, a process can also be defined as a tree of states.

Classes do not undergo processes, but it is possible to talk about the set of processes that a class can undergo. In this sense, we can talk about the "processes of a class".

Law property, state law, transformation law, "non-law property"

Certain properties restrict the values that other properties can have. Such properties are called *laws*. A law property can restrict the values of one or more other properties, but they must all be *non-law properties*, i.e., a law cannot restrict the value of another law. Furthermore, the properties restricted by a law must all belong to the same thing as the law, i.e., a law cannot restrict the value of properties that does not belong to the same thing as the law.

A *state law* is a law that constrains the values that other properties can have for individual states the thing can be in, i.e., state laws are structural/static.

A *transformation law* is a law that constrains the values that other properties can have across multiple states, i.e., transformation laws are behavioural/dynamic. When a thing is in a particular state, a transformation law may effect an event.

Natural kinds, sub-kinds

A *natural kind* is a class with at least one characteristic property that is a law. Generalisation/specialisation relationships are defined for natural kinds as for all classes. A *sub-kind* is a natural kind that specialises another.

Intrinsic/mutual property

Some properties only belong to one thing. They are *intrinsic* to the thing. Other properties belong to two or more things. They are *mutual* between those things.

Mutual properties play a central role in the ontology because they are one of the two ways in which things/classes can be related.

Acting on, interaction

One thing *acts on* another thing if and only if the history of the second thing would have been different had the first thing not existed. We say that one thing *is acted on* by another thing if and only if the second thing acts on the first. Two things *interact* if they both act on one another. In either case - whether one thing acts on the other or they interact - the two things are said to be coupled.

Binding mutual property, coupling

One thing acts on another thing by changing a property that is mutual between the two things. Such a mutual property, which is used by one thing to act on another thing, is called a *binding mutual property*.

When there is a binding mutual property between two or more things, we say that there is a *coupling* between them or that the things are coupled. This is another way to say that the things act on or interact with one another.

Binding mutual properties play a central role in the ontology because they are one of the two ways in which things/classes act on one another. For the same reasons, binding mutual properties play an important role in the property entry of the template too.

Internal/external event, coupled events

When a thing changes a binding mutual property, this is an event both in the acting thing and in

the other things that possess the property. We say that the event is *internal* in the acting thing and *external* in the acted-on things. The internal event and the external events are *coupled* because they always co-occur.

The internal event in the acting thing is always caused by a transformation law of the acting thing. The external events in the acted-on things are also caused by the same transformation law of the acting thing.

Composite/component thing

A thing can have one or more other things as *components*. A thing that has components is called a *composite* thing. Accordingly, we can speak of "composite" and "component" classes.

Part-whole relation

A composite thing possesses a *part-whole relation* property in common with each of its component things.

Part-whole relations play a central role in the ontology because they are the second of the two ways in which things/classes can be related. Accordingly, they are the second of the two ways in which things/classes act on one another. (The first way was through mutual properties again.)

System

A composite thing is a *system* if and only if its collection of component things cannot be partitioned in such a way that no thing in either partition is coupled with any thing in the other. A class is a "system" if all its things are systems.

Resultant/emergent property

A property of a composite thing (including a system) is *resultant* if and only if it is derived from one or more properties of its components. For example, the "serial number" of a "car" is a resultant property because it is the same as the "serial number" of the car "frame" component. And the "weight" of the "car" is resultant because it is the sum of "weights" of all the car components. A resultant property of a composite thing is not qualitatively different from the properties of its components.

A property of a composite thing (or system) is *emergent* if and only if it is not derived from one or more properties of its components. For example, the "driving comfort" of a "car" is not clearly derived from any properties A resultant property of a composite thing is qualitatively different from the properties of its components. A "person" can be an "idiot" although none of the persons "body parts" can.

Regular property

A lot of subtypes of properties have been introduced. It is sometimes useful to have a specific term for properties that are not whole-part relations, neither mutual and nor laws. Such properties are called "regular".

3.2 Summary

In this chapter, we briefly presented the common ontology. As explained in [OHS04], Wand & Weber have provided a tabular description of the main concepts in their model, from which Table A.1 was derived (See Appendix A). This table should be an asset to help the understanding of the different concepts.

Chapter 4

The UEML construct template

A need for UEML 2.0 is to define in a standard way modelling constructs to be integrated in UEML. These constructs come from different entreprise modelling languages. In that sense, a UEML construct template was proposed. This template provides, thus, a standard form describing modelling constructs by filling in standard set of "entries", in a way that facilitates language integration. The main idea is that the template is based on the Bunge-Wand-Weber (BWW) representation model of information systems, in order to make the definitions cohesive and, thus, learnable, understandable and as directly comparable to one another as possible [Opd06b].

In this chapter, we will detail the different parts of the template and how to use it. The UEML template is composed of three sections: the preamble, the presentation and the representation sections. For each modelling construct we analysed, we filled in each entries of the three parts. Here is a brief explanation of the UEML template. More information on each entry is provided in Appendix C.

4.1 The preamble section

The preamble section corresponds to the "preamble" part of the UEML meta-meta model (Figure 2.8). This part provides general information about the modelling construct, such as the construct name, alternative construct names, relations to other constructs or terms, the diagram types and language it belongs to, as well as acronyms and external resources.

To fill in the preamble section, we have read documentation about the language itself and try to find general information about the modelling construct.

4.2 The presentation section

The presentation section of the template is not accounted for by the UEML meta-meta model. This part describes the visual presentation of the modelling construct and has been kept informal at this stage. It includes lexical information, syntax and some pragmatics.

Lexical information deals with things as icons or line styles that are used to represent graphically the modelling construt.

The syntax describes how this construct is connected to other constructs. It can be both relations to other modelling constructs within the same diagram type or relations to constructs in other diagram types in the same language.

The pragmatics mainly deal with layout conventions. There are two kinds of layout conventions. The first ones are the diagram layout conventions which represent informal, tacit, social conventions that affect how this modelling construct is used when drawing diagrams. The second ones are other usage conventions which represent any other social conventions that affect how this modelling construct is used.

To fill in the presentation section, we have read the specification (meta model) about the language to find information about how the modelling construct is represented in the diagram, how it is connected to other constructs.

4.3 The representation section

The representation section corresponds to the "represented" and "ontology" parts of the UEML meta-meta model (Figures 2.8 and 2.9). It describes the instantiation level, classes, properties and kinds of dynamic behaviour that a modelling construct can be used to represent.

The instantiation level of a construct describes whether the modelling construct is intended to represent either classes, properties, states or transformations at the type level or at the instance level or both.

In the third entry of this section, we can indicate which classes of things that the modelling construct is intended to represent. Along with the property and behaviour entries, described next, this entry is the most central in the representation section. Along with the property and behaviour entries, it is also the most complex. Each modelling construct somehow represents one or more classes of things in the domain. Indeed, a property belongs to one or more classes of things; a state is defined in terms of properties that characterize a class; a transformation possesses pre- and post- states.

The next entry describes which properties and relationships the modelling construct is intended to represent, if any. As for classes, the modelling construct somehow represents one or more types of properties of classes in the domain. Even if the primary purpose of a construct is to represent classes, states, events or processes, it represents a class, state, event or process that involves one or more property types, each of them playing a particular role in the context of the construct.

The kind of behaviour, that the construct is intended to represent, is mentioned in the representation section. Some constructs are apparently not intended to "represent behaviour" at all. They just represent the existence of certain classes, properties. Other constructs represent particular states or events or processes.

The representation part is the most difficult one to complete. It needs a good knowledge of the language, the modelling construct and also of the common ontology. Moreover, most existing modelling language definitions describe semantics using text only, so the entries in this template section usually cannot be filled in without interpreting the language definition, looking "between the lines" to some extent, and also looking at examples of how the language is used in practice [Opd06b].

4.4 Summary

In this chapter, we presented the UEML template that has been used to describe modelling constructs to incorporate into version 2 of UEML. Central ideas behind the template are separation of presentation and representation and describing representation using referential decomposition of modelling constructs onto a common ontology. Another important idea is to provide a way of defining modelling constructs not only generally, in terms of whether they represent "classes", "properties" or other ontological categories, but also in terms of which classes and/or properties they represent, in order to make the definitions more clearly and precisely related to the enterprise.

As explained in [Opd06b], the experiences from the UEML work in INTEROP indicate that the template-based approach for describing modelling constructs is sufficiently powerful to support integrated use of a broad variety of languages and models. It can form the core of a new theory of language and model integration. However, the UEML must be broadened and developed, formalised and documented further, and proof-of-concept prototype tools must be developed. The resulting theory and tools also need to be empirically validated and evaluated in real case studies, while incorporating an increasingly wider selection of existing modelling languages.

Chapter 5

ARIS

ARIS stands for *Architecture of Integrated Information Systems*. The concept is intended to provide a framework that spans the gap between business theory and information and communication technology. In that sense, ARIS is a unique and internationally renowed method for optimizing business processes and implementing application systems [Sch98] [Sch99].

Professor Scheer [Sch98] defines the ARIS fonctionalities as follows:

- offers a framework (architecture) to describe standard software solutions
- methods for modeling information systems, for describing business processes
- provides reference models as tools
- leveraging standard software solutions

In modeling enterprise models, ARIS uses a modeling language known as Event-driven Process Chains (EPC), which is an important aspect of the ARIS-model. EPC is the center of the ARIS House and connects all other views, as well as describing the dynamics of the business process.

In this chapter, we will first describe the ARIS house. Then, we will give a brief explanation about the EPC. Thirdly, we will describe the ARIS elements chosen to be analysed. Next, we will show and explain an example. To finish, we will provide a meta model used for the analysis of ARIS.

5.1 The ARIS House

As reported in [Sch98], ARIS builds different small models, related them to others. Each models may contain many items (objects) and many connections (relationships). In order to provide structure, these models are organised into five views. These views are the basis of the ARIS architecture and shown by Figure 5.1. The grouping of the ARIS concepts and their relationships into views serve the purpose of structuring and simplifying business process models. We will define the content of each view.

- **Function view:** The function view comprises the process (function) which transforms input into output. Functions support goals, i.e. are controlled by them. It includes the static models of process tasks.
- **Organization view:** The class of organization view creates hierarchical organization structure, to group responsible entities or devices executing the same work object. It includes the static models of the structure of the organization.
- **Data view:** Data view comprises the data processing environment as well as the messages triggering functions or being triggered by functions. Information services objects are captured in data views. It includes the static models of business information.
- **Output view:** Output view contains all physical and non-physical input and output, including funds flows.

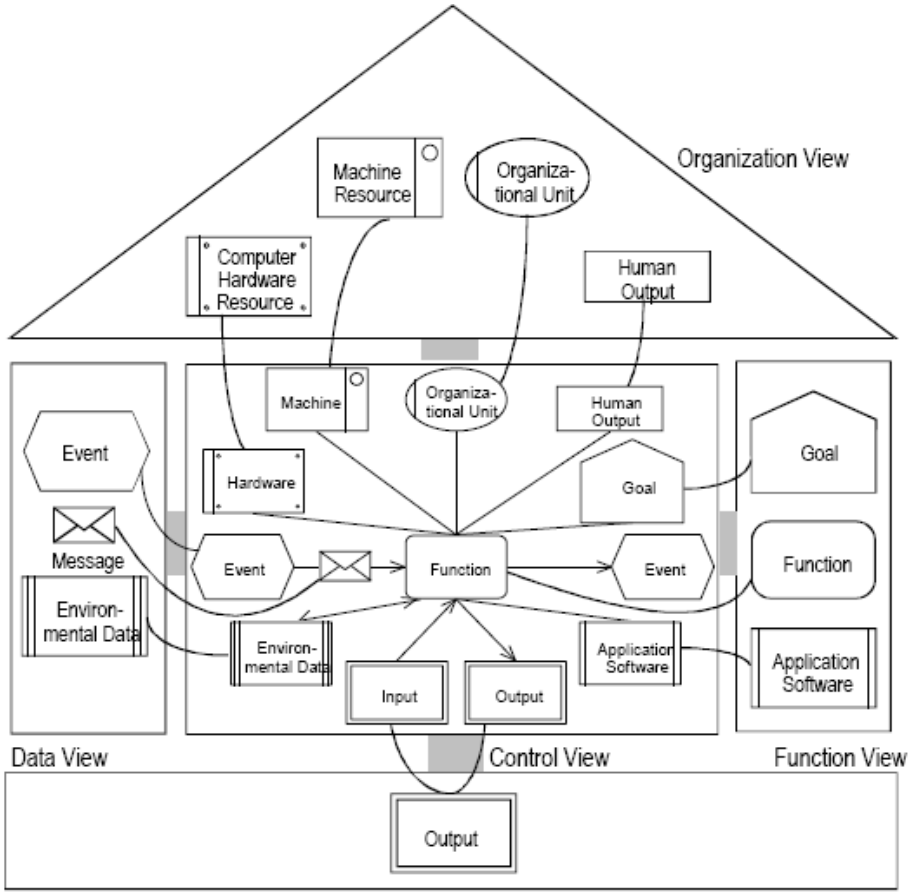


Figure 5.1: Views of the ARIS house (from [Sch98])

- **Control view/Process view:** The control view is where the respective classes with their view-internal relationships are modelled. Relationships among the views as well as the entire business process are documented in the control or process views. It includes the dynamic models that show the behaviour of processes and how they relate resources, data and functions of the business environment.

In system theory, we can draw a distinction between the structure and the behaviour of a system. Structures encompass the static view of a system, whereas behaviour describes its dynamics. In business process models, dynamics are expressed by event control and message flow. The function, organization, data and output views describe the system structure. Control view shows all the structural coherences of the views and the dynamic behavioral aspect of the business process flow.

5.2 The Event-Driven Process Chain (EPC)

The Event-Driven Process Chain (EPC) is the central model for all business modelling in ARIS [Dav01]. It is a dynamic model that brings together the static resources of the business (systems, organization and data) and organizes them to deliver a sequence of tasks or activities that adds business value. There are usually four types of objects used in the EPC: Events, Functions, Rules and Resources.

Events represents the changing state of the world as a process proceeds. Events represents the pre-conditions and post-conditions for each step in the process. **Functions** represent the activities or tasks that are carried out as part of a business process. In ARIS an Event activates a Function and a Function will always create one or more Events. Thus, Events trigger Functions and Functions produce Events. The Events in turn trigger further Functions, and so on, to produce a chain of Functions and Events –

the *Event-Driven Process Chain*. Of course this is a simplist way of representing processes. It is for that reason that the concept of decisions and multiple process paths was introduced. Decisions that change the process flow are always taken by Functions, but to represent possible outcomes and multiple triggers we need to use rules. There are three basic **Rules** in ARIS and they have different usage depending on whether they follow or precede Functions. These Rules are summarized in Table 5.1.

Table 5.1: Three basic Rules in ARIS

Operator	Following a Function	Preceding a Function
OR	OR decision: One or many possible paths will be followed as a result of the decision.	OR trigger: Any one Event, or combination of Events, will trigger the Function.
XOR	Exclusive OR decision: One, but only one, of the possible paths will be followed.	Exclusive OR trigger: One, but only one, of the possible Events will be the trigger.
AND	AND branch: Process flow splits into two or more parallel paths.	AND trigger: All Events must occur in order to trigger the following Function.

ARIS provides some additional Rules that provide combinations of AND and OR operators, but it can be confusing and ambigous in their interpretation and therefore we won't speak about them.

The real power of ARIS is achieved when we start to model the relationships between the process and the business environment in which it operates [Dav01]. Thanks to a fourth category of objects called **Resources**, we can model the relationships between the process and the organization, the systems, the data, the knowledge and the products.

As reported in [Dav01], the important rules to bear in mind when using the EPC are:

- Every model must have at least one start Event and one end Event
- Functions and Events always alternate
- Functions and Events only have a single incoming and outgoing connection
- Process paths always separate and combine using Rules
- Multiple Events triggering a Function combine using a Rule
- Decisions are taken by Functions (The name of the Function should describe the taken decision, the Function connects to a Rule which determines the logic of the possible outcomes (e.g. *OR* or *XOR*))
- Functions that take decisions are always followed by Rules
- Rules show the valid combination of paths that follow a decision
- Events following Rules indicate the actual outcomes of decisions
- Rules cannot have multiple input and multiple output

5.3 ARIS elements

In this section, we will describe each modelling construct we have analysed through the UEML template, on the basis of [Sch98], [Sch99] and [Dav01].

Function

The function defines an activity or a task to be executed in a process. Accordingly, they are defined as operations applied to objects for the purpose of supporting one or more goals. For instance, in Figure 5.3, a function is "Write up purchase order".

Goal

A goal is met as a result of executing the process. It represents the finality of a function. In that sense, a function supports goals. We can also say that the function is controlled by goals. A goal can be linked to one another with a subordinate goal supporting overriding goals. For instance, in Figure 5.3, the function "Check order" supports one goal, "Customer satisfaction - Short lead time". It implies that the function should maximize the satisfaction of the customer and also not to take long to check the order.

Application software

An application software executes a function. In an application software, computer-aided processing rules of a function are defined. In other words, an application software is a software running on a computer that is used to support the execution of a function. For instance, in Figure 5.3, the function "Check order" is executed by the application software "Sales Information System".

Organizational unit

An organizational unit is an entity involved in the business process. This entity is responsible for multiple functions. It can be a group of companies, a company, a department, a service. External business partners as well, such as customers, suppliers or the public sector, are instances of the class "Organizational Unit". For instance, in Figure 5.3, the organizational units are the "Customer", "Sales", "Purchasing" and "Manufacturing". The "Customer" is reponsible for the function "Write up purchase order" because it is the customer who wants to order something.

Position

A position is a role performed by individual people. An organizational unit may be composed of positions, e.g. "Customer Service Advisor" or a position may have a specific responsibility for the department, e.g. a "Department Manager". The position is defined by the function amount that an individual employee can handle. This concept is not used in the example (Figure 5.3).

Human output

A human output processes a function. Is is a responsible entity and it represents the people involve in the output's realization. This concept is not used in the example (Figure 5.3).

Machine resource

The machine resources are the different material equipment having a role to play in a process. They are responsible entities. They are used by a function. For instance, if we continue the order's process, at one moment, the entreprise should deliver the marchandises to the customer. To be realized, this function needs the use of a "Truck". This "Truck" is a machine resource.

Computer hardware

Computer hardware represents all the devices capable of accepting and storing computer data, executing a systematic sequence of operations on computer data, or producing control outputs. It can be a laptop, a desktop, a printer, a switch, an operating system or a server. A computer hardware is a responsible entity. For instance, in Figure 5.3, all the functions use a computer hardware which is a "PC", i.e. a personal computer.

Event

Events characterize activities containing facts (what) that occur at certain point in time (when). Events are (data) status modifications, created by functions or by actors outside of the model. Events trigger functions by means of messages and are the result of functions. The ARIS definition of an event is: "An event is an instance of a state that influences or controls the further flow of the processes". As a main component of EPC, event controls dynamic processes in business processes. For instance, in Figure 5.3, the function "Write up purchase order" is triggered by the event "Demand reported" and creates an event "Order accepted".

Message

Messages determine how functions react to events. Messages can contain additional attributes besides information regarding the beginning of the event. Control flows are executed by events and messages that they trigger, after which information regarding the beginning of the event is transferred to the next entity. Messages indicate that the events have been detected, pass them on to successive functions and then activate them. Accordingly, events trigger messages, transmitting the fact that an event has occurred.

Environmental data

The environmental data represents the data used and added to databases during the instantiation of a function. Functions process environmental data by transforming input data into output data. For instance, in Figure 5.3, the function "Check order" processes the environmental data "Customers suppliers".

Logical operator "AND"

The logical operator "AND" expresses compounded events (operator beween events). We can find two kinds of "AND". The first one is the *AND trigger*: all Events must occur in order to trigger the following Function. The second one is the *AND branch*: process flows splits into two or more parallel paths. Splitting the process into parallel branches implies that they can be done at the same time because they have no dependence on each other. The branches normally recombine in an AND later on in the process. Where they re-combine, the process cannot continue until all the branches that join in the AND have been completed. For instance, in Figure 5.3, the *AND branch* is used because the process flow splits into two parallel paths, one with the function "Write up purchase order" and the other with the function "Plan manufacturing".

Output

The output is the result of processes, it is heterogeneous and can be used at various levels of details. It can be physical (material output) or non-physical (services). The general term output can be distinguished between material output and services, the latter in turn being divised into information services and other services. The classification of the types of output/input is illustrated by Figure 5.2. Material output can be defined, for example, by the delivery of material (cars, machines), manufactured parts or even the finished products. The term services is more difficult to define because it comprises heterogeneous services. These are some of the many kinds of services:

- Theatrical performances (concerts, theaters), where the output is the act of performing; consumption of this output is simultaneous with its production
- Banks providing services in the form of loans or credits; in this case, the service consists of providing the necessary funds and is in itself the result of other banking services, such as credit checks, depositary services, ...
- Insurance services
- Services in the public sector (issuing drivers licenses, IDs, ...)

An information service can be, for instance, a certificate, a instruction manual. Another service can be a warranty or an inspection.
One characteristic of output is that it can be required by a party other than the party providing them. In that sense, an output can be an input of another function. For instance, in Figure 5.3, the function

"Write up purchase order" creates an output "Customer order". This output is used by the next function "Check order" as an input.

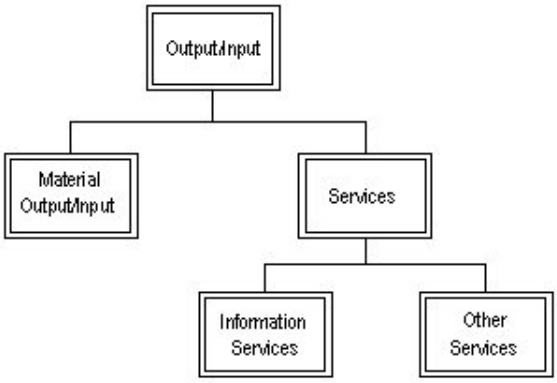


Figure 5.2: Classifying types of output/input (from [Sch98])

5.4 ARIS example

In this section, we will present and explain briefly an example of an ARIS business process model. This model is shown by Figure 5.3. This example is a part of the entire example presented in [Sch98]. It is the control view which is modelled but the other views can also be identified in this business process model.

This example represents the beginning of an order's process. A customer orders something and the entreprise should satisfy this purchase order. The first function of the process is *Write up Purchase Order* and is under the responsibility of *Customer*. Indeed, it is the customer who writes up a purchase order for the entreprise. This function is triggered by an event, *Demand reported*, and is executed by the software, *Ordering Program*. It also uses a computer hardware, *PC*, and the environmental data, *Suppliers*. This function creates an event, *Order accepted*, and an output *Customer Order*.

This event will trigger the second function of the process, *Check order*, which uses the output *Customer Order* as an input. The *Sales* department of the entreprise is reponsible for the function *Check order*. This function is executed by the software, *Sales Information System*, and uses a computer hardware, *PC*, and the environmental data, *Customers Suppliers*. It is controlled by a goal, *Customer statisfaction - Short Lead time*. It implies that the function should maximize the satisfaction of the customer and also not to take long to check the order. The function creates an event, *Order accepted*, and an output *Checked Order*.

After this event, the control flow is split into two parallel paths, one with the function *Write up purchase order* under the responsibility of *Purchasing* department and the other with the function *Plan manufacturing* under the responsibility of *Manufacturing* department. These two functions use a computer hardware, *PC*. The one under the responsibility of the *Purchasing* consists in writing up a purchase order for the supplier (Supplier not presented in the example). It is executed by the same software application used by the first function of the process, *Ordering Program*. This function uses an environmental data which is *Suppliers Material* and supports a goal *Short Processing Time*. It implies that the function should take the minimum time for the processing. The last function shows in the example is *Plan manufacturing*. This function uses an environmental data, *Work Schedule*. It is executed by a software *Manufacturing Plan* and is controlled by the goal *Short Lead Time*. As we can see, it creates an event *Manufacturing Plan Completed* and an output *Manufacturing Plan*.

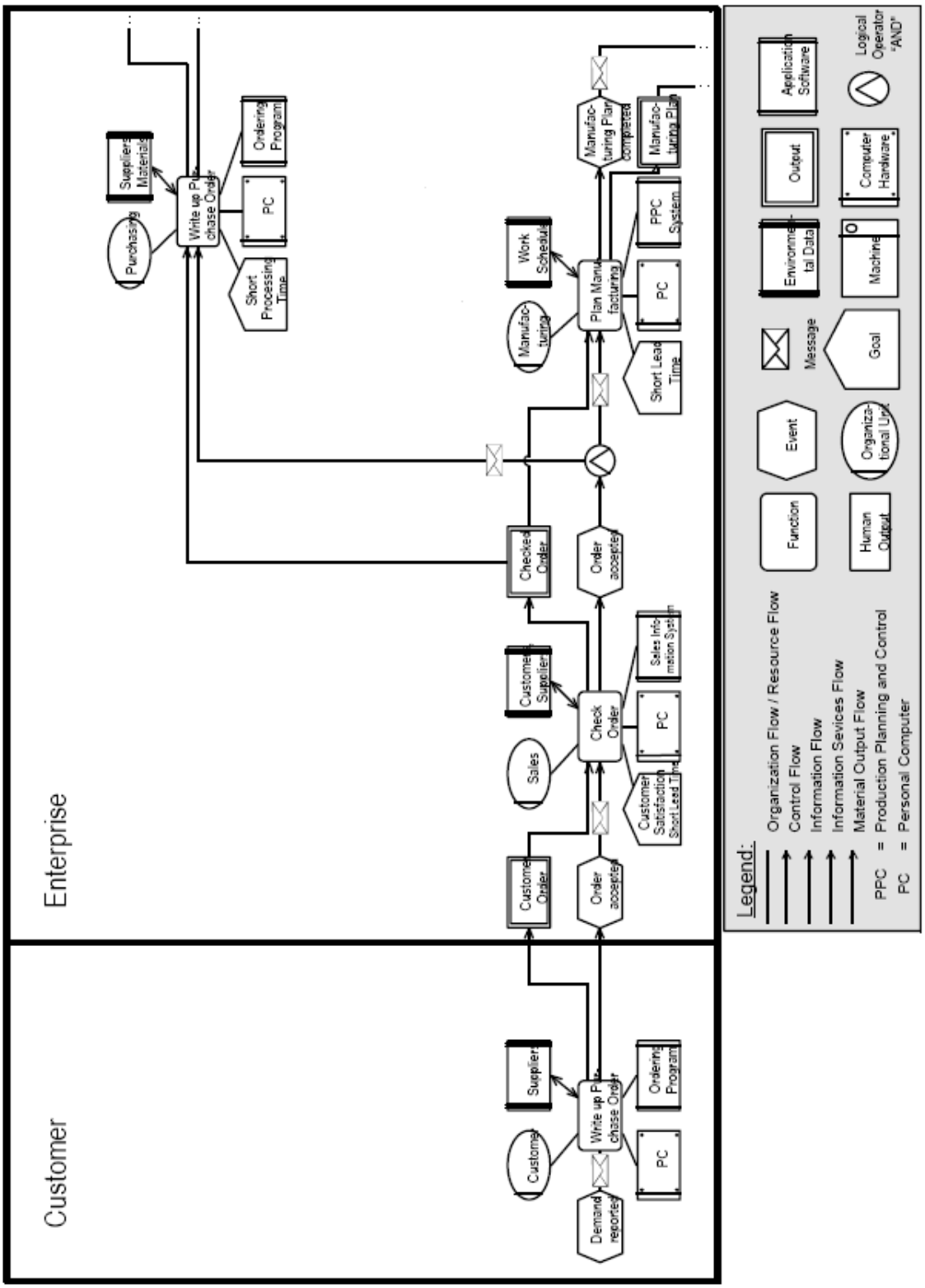


Figure 5.3: ARIS business process model - Example (adapted from [Sch98])

5.5 Meta models

The ARIS house establishes a framework for classifying the descriptive components of a business process [Sch99]. We will now take a look at the individual building blocks of a business process, along with their relationships. We will show and describe the general ARIS meta model where the elements of a general business process are captured. The more details meta models are provided in Appendix D. The general ARIS meta model is based on the ARIS business process model, depicted in Figure 5.4. This figure shows the ARIS concepts and their relationships. Each relationship has a name. For instance, the function is linked to the organizational unit by an organization flow.

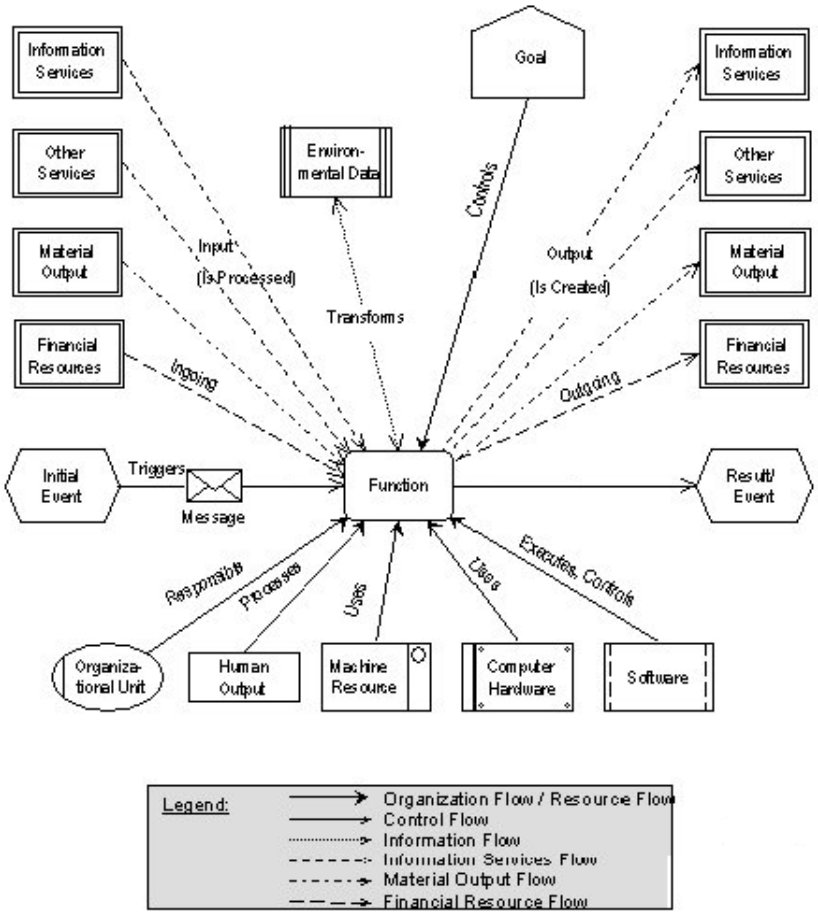


Figure 5.4: The general ARIS business process model (from [Sch98])

The Unified Modeling Language (UML) will be used to describe those information. The UML description language enables to individually model object classes and association classes, respectively, in the various views. This description is known as the **ARIS meta model** or **ARIS information model**, depicted in Figure 5.5. This figure shows the various classes (which represent an ARIS concept) being in each view and the associations between the classes. For instance, the association between the classes *Organizational Unit* and *function* is a *:* (many to many) association. Indeed, an organizational unit can be responsible for multiple functions and a function can be under the responsibility of multiple organizational units. Thus, this ARIS meta model permits us to see the most general associations between the classes and their cardinalities.

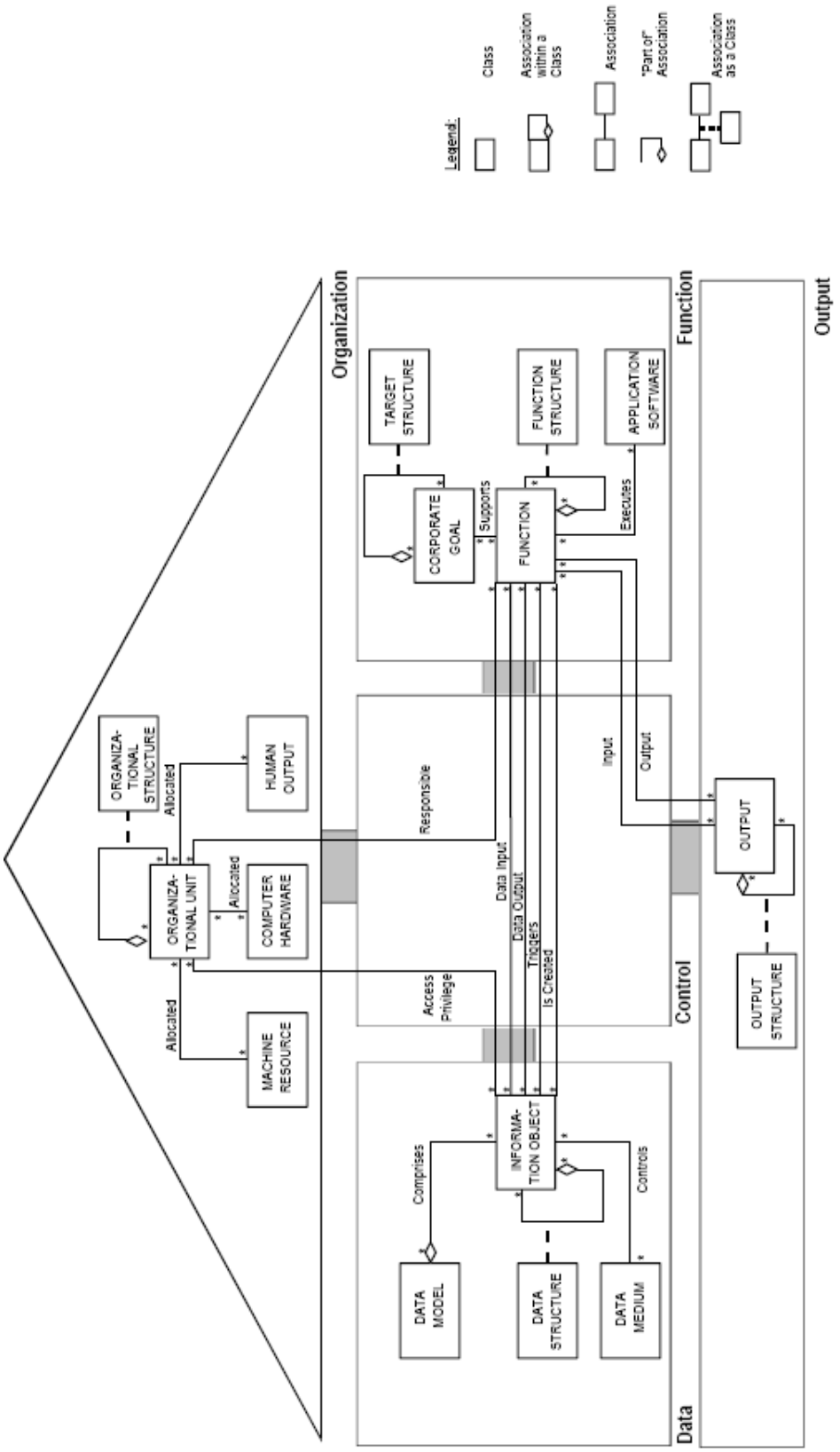


Figure 5.5: Preliminary ARIS information model (from [Sch98])

The explanation of Figure 5.5 is largely inspired by [Sch98].

Function view

The starting points of the function model, in Figure 5.5, are the *corporate goals* controlling the functions. It implies that certain functions must be executed to reach a particular goal. Corporate goals are generally classified hierarchically. *Functions* can be derived in subfunctions. The linking of functions, as well as the linking of functions to support goals, implies that function and corporate goals are associated by means of *:* association.

Organization view

The central designation in the organization view is *organizational unit*. This class has instances such as position, department or entreprise. The responsible entities or devices, respectively, *machine resource*, *computer hardware* and *human output* are allocated to organizational units.

Data view

On the left side of the ARIS house, the data view depicts the data structure model. The class *information object* characterizes objects to be described by database attributes. There are associations between their instances, such as item data and customer data. Information objects of an area with correlated contents can be grouped in a class diagram or a data model.

Output view

In the output view, the class *output* represents every kind of output (material output, service and information output). The instances in question would be application-related output classes such as item, materials, spare parts, assembly hours, warranty services or certificates. Here as well, various kinds of output can be linked with another by "part of" associations.

Control view

The associations between the four components, i.e. organization, function, information, and output, are depicted in the control view. The link between *organizational unit* and *function* is expressed by the *responsible* association. Indeed, an organizational unit is responsible for functions. Certain privileges relating to *information objects*, expressed by the *access privilege* association, can be allocated to organizational units. Functions transform input data into output data. Events trigger functions and are also the result of functions. These activities are illustrated as associations between the *information objects* and *functions*. An *output* can be an input or an output of function, expressed by the *input* association and the *output* association.

5.6 Summary

In this chapter, we explained the main concepts of ARIS. It includes the ARIS house representing the five views: organization, function, data, output and control views; the preliminary ARIS information model which represents the relationships between these views. The analysed ARIS elements were explained and an example of an ARIS business process model was discussed.

On the basis of this explanation, we will relate the analysis of ARIS in Chapter 8.

Chapter 6

BPMN

Initiatives in the field of Business Process Management (BPM) wanted to create a BPM standard notation. The Business Process Management Initiative (BPMI) brought then the Business Process Modeling Notation (BPMN). The BPMN 1.0 specification was released to the public in May, 2004.

As explained in [OMG06], the primary goal of the BPMN effort was to provide a notation that is readily understandable by all the people involved in a business process, i.e. the business analysts that create the initial drafts of the processes, the technical developers responsible for implementing the technology that will perform those processes, and finally, the business people who will manage and monitor those processes.

The BPMN 1.0 specification defines the notation and semantics of a Business Process Diagram (BPD), which is based on a flowcharting technique tailored for creating graphical models of business process operations. A Business Process Model, then, is a network of graphical objects, which are activities (i.e., work) and the flow controls that define their order of performance.

In this chapter, we will first describe the BPM. In Section 6.2, we will give an explanation of the BPMN elements. In Section 6.3, we will give an example of BPMN model to illustrate them. Finally, we will provide the general meta model of BPMN and one centered on the activities.

6.1 Business Process Management

Business Process Management (BPM) is about processing the business and achieving the success of this business. It concerns the capacity to underline the different ways to achieve the success of the business. The goal is to have an adaptive, innovative and effective organisation [BPMa].

There are plenty of definitions of BPM, but the [BPMa] tried to fix the best one:

> "Business Process Management (BPM) is a natural and holistic management approach to operating business that produces a highly efficient, agile, innovative, and adaptive organization that far exceeds that achievable through traditional management approaches."

Another definition of BPM is:

> "Business process management (BPM) is a systematic approach to improving an organization's business processes. BPM activities seek to make business processes more effective, more efficient, and more capable of adapting to an ever-changing environment. BPM is a subset of infrastructure management, the administrative area of concern dealing with maintenance and optimization of an organization's equipment and core operations. A business process is a set of coordinated tasks and activities, conducted by both people and equipment, that will lead to accomplishing a specific organizational goal. [BPMc]

Thus, BPM includes methods, techniques and tools to support the design, enactment, management and analysis of operational business processes [vdAtHW03].

BPMN allows creating Business Process Diagrams (BPDs), which is a diagram designed for use by the people who design and manage business processes. The BPD is in fact the needed tool to allow the BPM.

The BPD is the modelling of the enterprise business process from an expert view. This management is intended to have a better global view of the different process of an enterprise in a goal of optimisation. That is exactly in this sense that BPMN was created. [OMG06]

6.2 BPMN elements

This section provides a summary of the BPMN graphical objects and their relationships, as reported in [OMG06]. The four basic categories of elements of BPMN are:

- Flow objects
- Connecting objects
- Swimlanes
- Artifacts

6.2.1 Flow objects

A BPD has a small set of core elements, which are the flow objects, so that modelers do not have to learn and be familiar with a large number of different shapes. The three flow objects are:

Event

An event is something that "happens" during the course of a business process. These events affect the flow of the process and usually have a cause (trigger) or an impact (result). Events are represented by circles with open centers to allow internal markers to distinguish different triggers or results.

Figure 6.1: Event (from [OMG06])

There are three types of events, based on when they affect the flow:

- *Start*: The start event indicates where a particular process will start. In Figure 6.12, the process in the pool supplier starts by a start event. There are six types of start events in BPMN: None, Message, Timer, Rule, Link, Multiple. A start event is optional.
- *Intermediate*: Intermediate events occur between a start event and an end event. This is an event that occurs after a process has been started. It will affect the flow of the process, but will not start or (directly) terminate the process.
- *End*: The end event indicates where a process will end.

Activity

An activity is a generic term for work that company performs, work that is performed within a business process. An activity can be atomic or non-atomic (compound).

Figure 6.2: Activity (from [OMG06])

The types of activities that are a part of a BPD are:

- *Process*: Process is an activity performed within a company or organization. In BPMN a process is depicted as a graph of flow objects, which is a set of other activities and the controls that sequence them. A Business Process, as shown in a BPD, may contain more than one separate process. Each process may have its own sub-processes and would be contained within a pool. The individual processes would be independent in terms of sequence flow, but could have message flow connecting them.
- *Sub-Process*: A sub-process is a compound activity in that it has detail that is defined as a flow of other activities. A sub-process is a graphical object within a process flow, but it also can be "opened up" to show another process. In Figure 6.12, a sub-process is represented by the rectangle called "Check Credit" containing two gateways and four tasks.
- *Task*: A task is an atomic activity that is included within a process. A task is used when the work in the process is not broken down to a finer level of Process Model detail. Generally, an end-user and/or an application are used to perform the task when it is executed. In Figure 6.12, a task is for example "Receive request", the first task located in the pool supplier.

Gateway

Gateways are modelling elements that are used to control how sequence flow interact as they converge and diverge within a process. The term "gateway" implies that there is a gating mechanism that either allows or disallows passage through the gateway - that is, as Tokens arrive at a gateway, they can be merged together on input and/or split apart on output as the gateway mechanisms are invoked. To be more descriptive, a gateway is actually a collection of "gates". There are different types of gateways and the behaviour of each type of gateway will determine how many of the gates will be available for the continuation of flow. A gateway is represented by the familiar diamond shape and is used to control the divergence and convergence of sequence flow. Thus, it will determine traditional decisions, as well as the forking, merging, and joining of paths. Internal markers will indicate the type of behaviour control.

Figure 6.3: Gateway (from [OMG06])

Gateways can define all the types of business process sequence flow behaviour:

- *Exclusive Decision/Merge (XOR)*: Exclusive gateways (decisions) are locations within a business process where the sequence flow can take two or more alternative paths. The exclusive decision defines a set of alternative paths for the Token to take as it crosses the flow. There are two types of exclusive decisions:
  - *Data-Based Exclusive Decision*: The set of gates for data-based exclusive decisions is based on the boolean expression contained in the `ConditionExpression` attribute of the outgoing sequence flow of the gateway. These expressions use the values of process data to determine which path should be taken (hence the name data-based).
  - *Event-Based Exclusive Decision*: On the input side, their behaviour is the same as a data-based exclusive gateway. On the output side, the basic idea is that this decision represents a branching point in the process where the alternatives are based on events that occurs at that point in the process.

  Exclusive gateways can also be used as a merge for alternative sequence flow.
- *Inclusive Decision/Merge (OR)*: This decision represents a branching point where alternatives are based on conditional expressions contained within outgoing sequence flow. All sequence flow with

a *True* evaluation will be crosses by a Token. When the inclusive gateway is used as a merge, it will wait for (synchronize) all Tokens that have been produced upstream. It does not require that all incoming sequence flow produce a Token (as the parallel gateway does).

- *Complex Decision/Merge*: BPMN includes a complex gateway to handle situations that are not easily handled through the other types of gateways. Complex gateways can also be used to combine a set of linked simple gateways into a single, more compact situation.

- *Parallel Fork/Join (AND)*: Parallel gateways provide a mechanism to synchronize parallel flow (Joining) and to create parallel flow (Forking).

In Figure 6.12, two Gateways can be observed. The first one is called "Approve?" and the second one "Type of customer?".

6.2.2 Connecting objects

The flow objects are connected together in a diagram to create the basic skeletal structure of a business process. There are three connecting objects that provide this function. These connectors are:

Sequence Flow

A sequence flow is used to show the order (the sequence) activities will be performed in a process. Each flow has only one source and only one target. The source and target must be from the set of the following flow objects: events, activities, and gateways. During performance (or simulation) of the process, a Token will leave the source flow object, traverse down the sequence flow, and enter the target flow object. A sequence flow is represented by a solid line with a solid arrowhead. Note that the term "control flow" is generally not used in BPMN. In Figure 6.12, a sequence flow links the start event to the "Receive request", for instance.

Figure 6.4: Sequence Flow (from [OMG06])

Message Flow

A message flow is used to show the flow of messages between two entities (Process participants) that are prepared to send and receive them. It must connect two Pools, either to the pools themselves or to flow objects within the pools. They cannot connect two objects within the same pool. A message flow is represented by a dashed line with an open arrowhead. In Figure 6.12, message flows can be identified as the "credit request", the "credit report" and finally the "credit response" fows.

Figure 6.5: Message Flow (from [OMG06])

Association

An association is used to associate information and artifacts with flow objects. An association is also used to show the activities used to compensate for an activity. An association is represented by a dotted line with a line arrowhead. Associations are used to show the inputs and outputs of activities.

Figure 6.6: Association (from [OMG06])

6.2.3 Swimlanes

Many process modelling methodologies utilize the concept of swimlanes as a mechanism to organize activities into separate visual categories in order to illustrate different functional capabilities or responsibilities. BPMN supports swimlanes with two main constructs. The two types of BPD swimlane objects are:

Pool

A pool represents a participant in the process. A participant can be a specific business entity (e.g, a company) or can be a more general business role (e.g., a buyer, seller, or manufacturer). Graphically, a pool is a container for partitioning a process from other pools, when modeling business-to-business situations, although a pool need not have any internal details (i.e., it can be a "black box"). In Figure 6.12, there are three identifiable pools: the customer, the supplier and the credit agency.

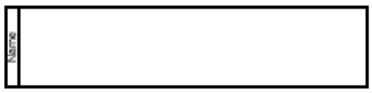


Figure 6.7: Pool (from [OMG06])

Lane

A lane is a sub-partition within a pool and will extend the entire length of the pool, either vertically or horizontally. Text associated with the lane (e.g., its name and/or any attribute) can be placed inside the shape, in any direction or location, depending on the preference of the modeller or modelling tool vendor. Lanes are used to organize and categorize activities within a pool.

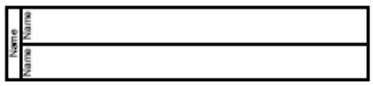


Figure 6.8: Lane (from [OMG06])

6.2.4 Artifacts

BPMN provides modellers with the capability of showing additional information about a process that is not directly related to the sequence flow or message flow of the process. BPMN defines only three types of BPD artifacts, which are:

Data Object

Data objects provide information about what the process does. That is, how documents, data, and other objects are used and updated during the process. They are a mechanism to show how data is required or produced by activities. They are connected to activities through associations. A data object is a portrait-oriented rectangle that has its upper-right corner folded over that must be drawn with a solid single black line.

Figure 6.9: Data object (from [OMG06])

Group

The group object is an artifact that provides a visual mechanism to group elements of a process informally. A group is represented by a rounded corner rectangle drawn with a dashed line. The grouping can be used for documentation or analysis purposes, but does not affect the sequence flow.

Figure 6.10: Group (from [OMG06])

Annotation

Annotations are a mechanism for a modeller to provide additional text information for the reader of a BPMN Diagram. The text annotation object can be connected to a specific object on the diagram with an association, but do not affect the flow of the process. Text associated with the annotation can be placed within the bounds of the open rectangle.

Figure 6.11: Annotation (from [OMG06])

6.3 BPMN example

In this section, we will present a short example of a business process diagram. This diagram is depicted in Figure 6.12 [1]. This example represents a simplified credit request system. This model shows only the point of vue of the Supplier. Anyway the model is composed of three pools: the *Supplier*, the *Customer* and the *CreditAgency*.

The process of the Supplier begins with a start event, but the reception of the credit request implies the realization of the task *Receive request*. The link between the start event and this task is a sequence flow. We can see a message flow which links the pool *Customer* and the task *Receive request*. This message flow represents the transmission of the credit request from the customer to this supplier's task. After that the supplier receives the credit request, the sub-process called *Check Credit* is executed. The starting of this sub-process instanciates a task consisting of the reception of a credit report from the credit agency, *Receive credit report*. This report permits to decide if the request is approved or not.

[1] The legend of Figure 6.12 is shown in Appendix G.

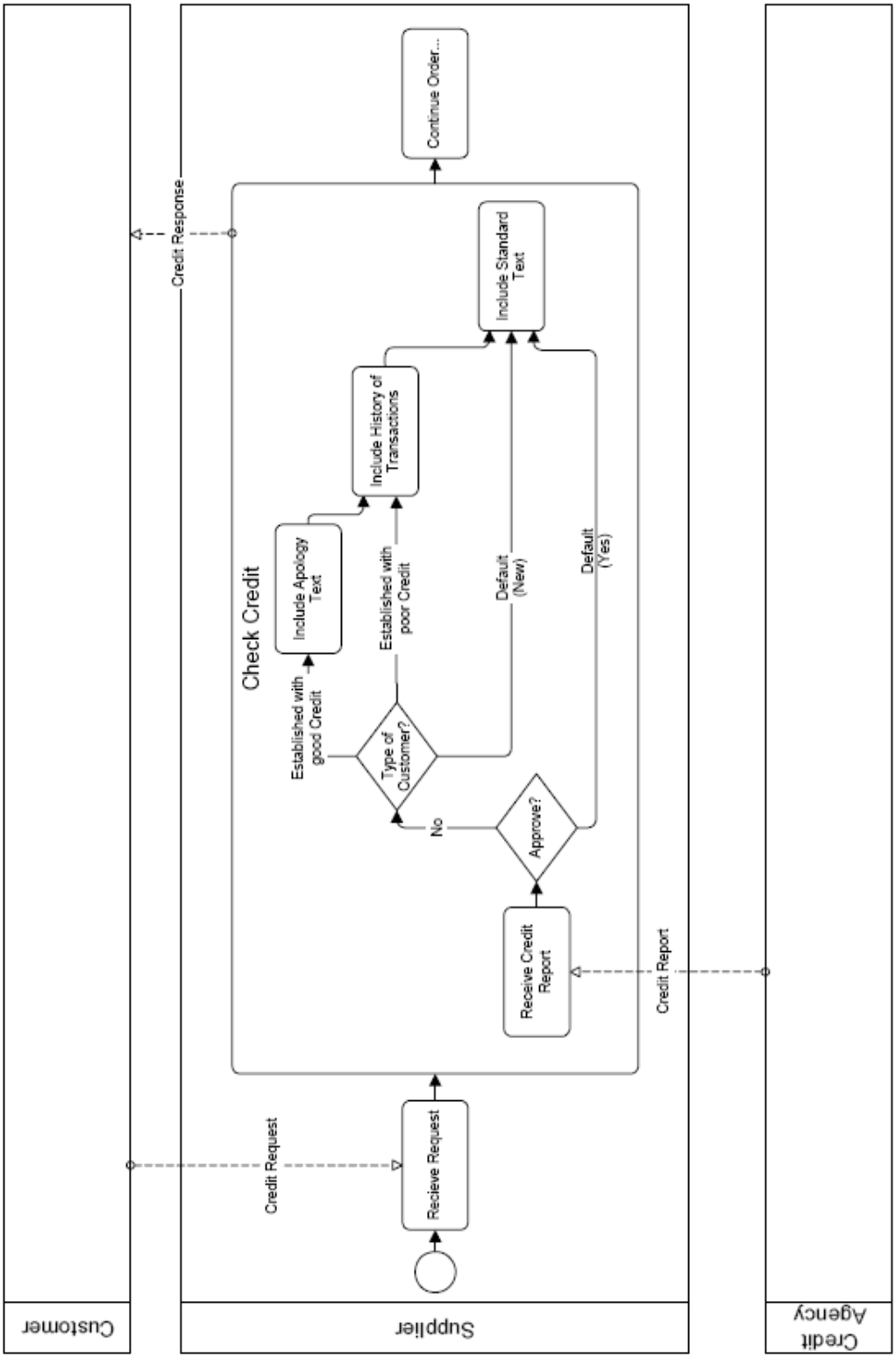


Figure 6.12: BPMN example (from [OMG06])

This decision is illustrated by a gateway. The default gate, materialized by the fact that the request is approuved, starts a task consisting of including the standard text in the response for the customer. If the report is negative it's necessary to check the type of the customer. The gateway representing this choice has three outgoing sequence flow. The default one, representing the new customer arrives to the task *Include Standard Text* in the response. The second gate, if the customer is established with poor credits, the supplier includes the history of the transactions and then the standard text in the response. Two tasks are used to represent this fact: *Include history of transactions* and *Include Standard Text*. The last gate, if the customer is established with good credit, the supplier includes apology text, then the history transactions and finally the standard text. In all those cases, the sub-process ends with the task "Include standard text". At the and, the sub-process sends a message to the pool *Customer* which represents the credit response. When the sub-process is terminated, the supplier continues with the task "Continue Order".

It's necessary to underline that this process is not complete, the process of the supplier is not ended by an end event. Usually all the processes are ended by those end events. There are also any associations and any artfacts represented in this example. And finally we can indicate that there isn't any lane in the three pools.

6.4 Meta models

We will now present a BPMN meta model that described its constructs (elements which can be used to create BPDs) and the relationships that exist among them. This meta model has been formalised in a UML class diagram. There is actually no BPMN meta model defined yet. But it is under definition, though still not released officially. That is why we produced our own first draft of the meta model. By concerns of clarity, we have decided to make a high level general meta model and to devide it in some parts. The general meta model of BPMN will be explained. This meta model is divided into four parts: one is centered on the activities, another on the events, a third one on the gateways and a last one on the artifacts. The meta model centered on the activities will be presented and the others are shown in Appendix E. This meta model is presented because it is needed to understand the environment of the activity to be allowed achieving the analysis of the activity in Chapter 9. The achievement of meta models was needed to facilitate our analysis, it was easier to understand the relationships between the constructs in a meta model than in the textual specification. By the way, the completion of the template was easier and quicker to do.

6.4.1 General BPMN meta model

In the general BPMN meta model (Figure 6.13), the BPD represents one or more processes. The BPD is composed by four categories of constructs: the *Flow Object*, the *Connecting Object*, the *Swimlane* and the *Artifact*. All of those classes can be specialised in some constructs. For all those generalisation/specialisation relations, the set of the subclasses is disjoint and complete, i.e. each instance of the superclass is an instance of a subclass and can not belong to simultaneously to multiple subclasses. The *Artifact* can be specialised in *Data Object*, *Text Annotation* or *Group*. By the same way, the *Swimlane* can be specialised in *Pool* and *Lane*. Those two classes have a particular relationship because the lanes compose the pool. Regarding the *Connecting Object*, it can be specialised in *Sequence Flow*, *Message Flow* and finaly in *Association*. To end the scope of the general view of BPMN, the *Flow Object* can be specialised in *Event*, *Gateway* or *Activity*. With this information, we have the whole global structure of BPMN.

For the process, each process, represented in a BPD, is contained within a *Pool*. A *Process* is composed as a set of *Flow Object* and *Artifact* that add information to the set of flow objects. A *Sequence flow* links two *Flow Objects* and one of those objects can be the extremity of multiple sequence flows. Regarding the *Message Flow*, they connect two pools, either to the pools themselves or to *Flow Objects* within the pools. They cannot connect two objects within the same pool.

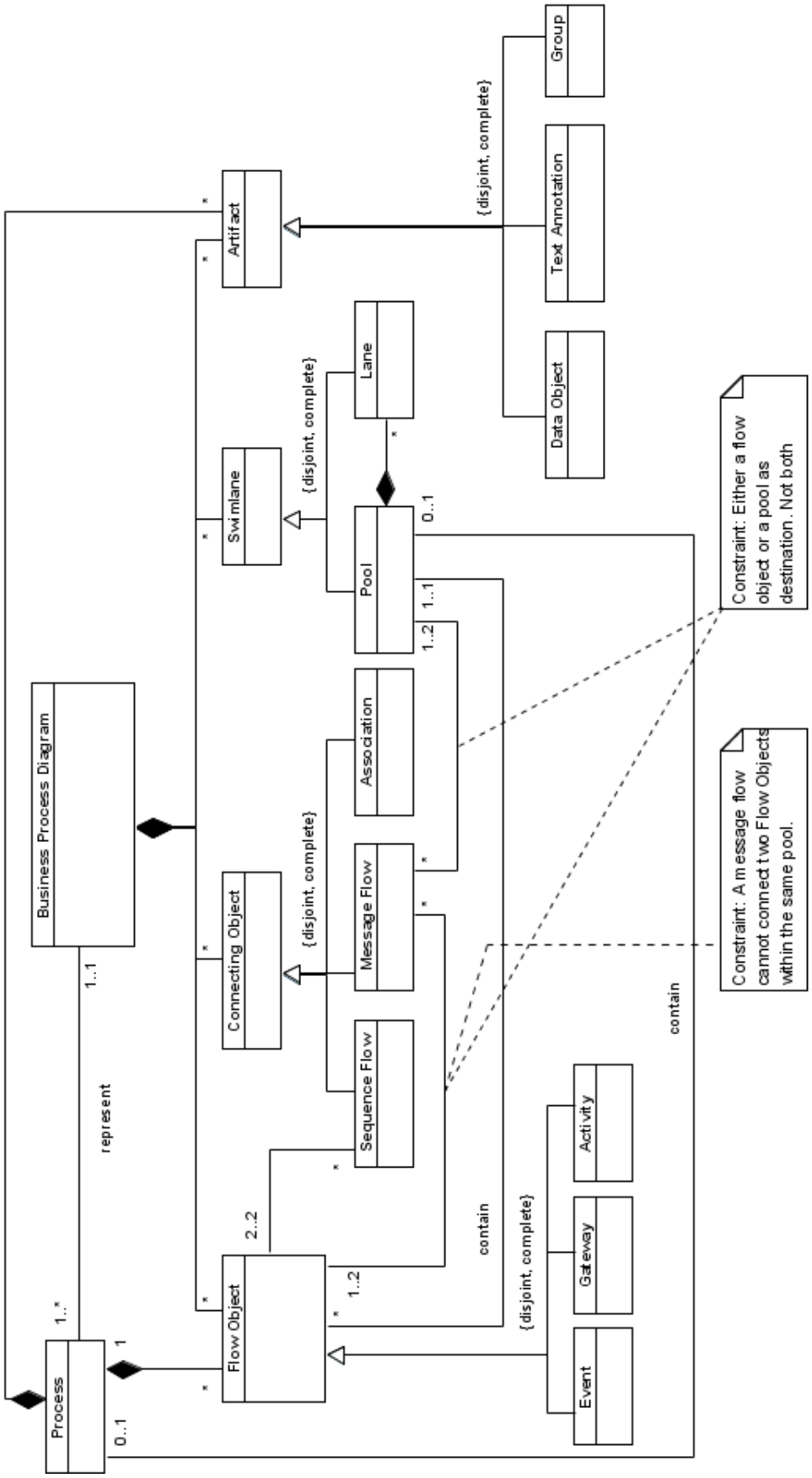


Figure 6.13: General meta model of BPMN

6.4.2 Meta model centered on the activities

Figure 6.14 shows the meta model centered on the activities. The *Activity* can be specialised in two categories, the *Sub-Process* or the *Task*. This generalisation/specialisation relation is disjoint and incomplete. Disjoint because an instance of *Activity* cannot be simultaneously an instance of *Sub-Process* and *Task*. Incomplete because the types of activities that are a part of a BPD are: process, sub-process, and task. However, a process is not a specific graphical object. Instead, it is a set of graphical objects [OMG06]. It is why the *Process* is represented as class containing the *Flow Object* (See Figure 6.13) and not like a specialisation of *Activity*. Those constructs are directly linked to the different flows. The task and the sub-process can be the target or the source of many sequence flows. Conversly, one *Sequence Flow* can have one *Sub-Process* or *Task* as source or target. In the same way, a *Sub-Process* can be the source or the target of multiple message flows and a *Message Flow* can have a particular *Sub-Process* as source or as target. For the *Task*, it can be the source of multiple message flows but it can only be the target of one *Message Flow*. And finally a *Message flow* can have a *Task* as source or as target.

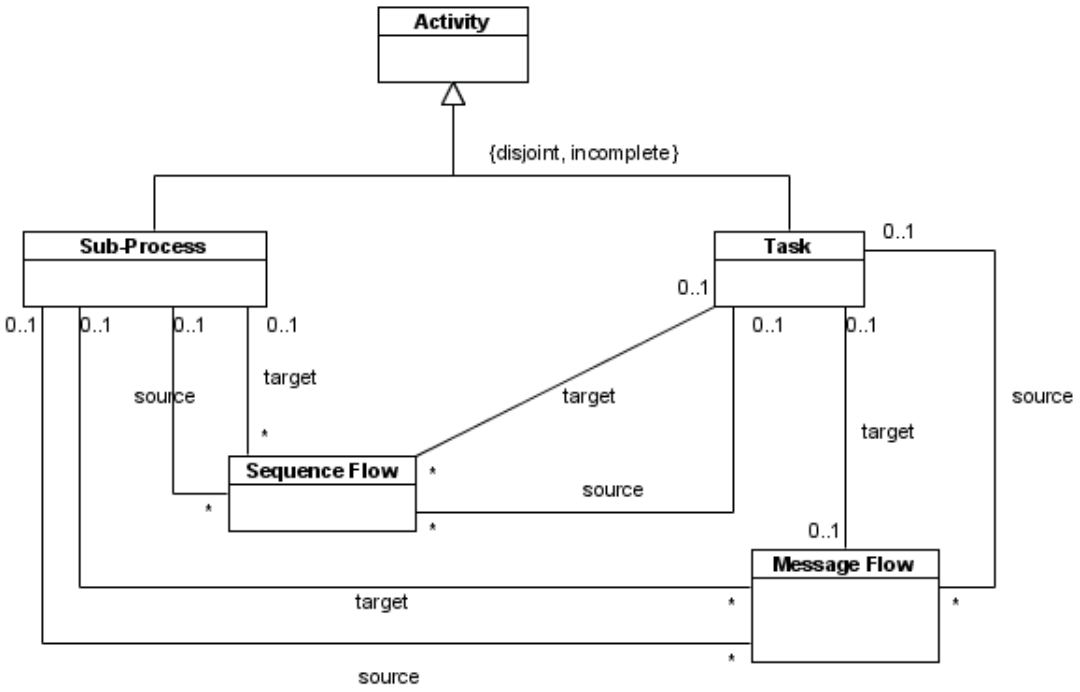


Figure 6.14: Meta model centered on the activities

6.5 Summary

In this chapter, we briefly described what is BPM. Then, we presented the four main concepts of BPMN: the flow objects, the connecting objects, the swimlanes and the artifacts. In Section 6.3, an example of an BPMN business process model was discussed. To finish, we explained the BPMN general meta model and one centered on the activity.

On the basis of this description, we will explain the analysis of BPMN in the Chapter 9.

Part II

Contribution

Chapter 7

Our research method for ontological analysis

The research method that we applied during all the work is inspired by the UEML 2.0 approach and the reference methodology explained in [GR05b] (See Chapter 2). However, we haven't used the whole UEML approach, we just applied only some sub-steps of the approach. The particular step which was interesting for us was the third one: describing the modelling constructs that are chosen as part of UEML using the UEML template. It is really this step that we have achieved. Concerning the reference methodology, we can find several similarities between this one and our research method.

First, we will explain our research method, i.e. a report of what we have done (the way of working). Then, we will assess our method and list the similarities with the UEML approach. To finish, we will propose a new method.

7.1 Our methodology

We will present our research method by explaining each process steps in general way. These steps are shown by Figure 7.1.

1. **Gain knowledge of UEML [1]:** The knowledge is gained after reading the UEML documentation. This knowledge consists in understanding the UEML approach.

2. **Gain knowledge of language:** The knowledge is gained after reading the language documentation. This knowledge consists in understanding the language and the relationships between the constructs by using meta models and examples.

3. **Analyse language:** We have applied the UEML paper template. First, the analysis of static aspects is achieved, i.e. mappings of classes and properties. Then, the analysis of dynamic aspects is made, i.e. mappings of states and transformations. The language analysis report which is the filled in UEML paper template is produced.

4. **Refinement:** The analysis is revised to have a final filled in UEML paper template for each construct. For that, the language analysis report is discussed with the teammate(s) and with a meta modelling expert.

5. **Enter data in the Protégé tool:** The UEMLBase is completed with the language analysis report. This step produces a new extended version of the UEMLBase.

6. **Validate the analysis:** The validation package (UEML Validator) is used in Protégé to verify the new UEMLBase. The possible errors are corrected in this UEMLBase and on each paper template. This step brings us in a state where stable results are available.

[1] This step 1 is realized once even if more than one language is analysed.

7. **Carry out a case study:** A case study is carried out to validate the mappings. This step brings us in a state where we learn some lessons. The case study will be reported in Section 12.

8. **Refinement:** From the lessons learnt, the results are revised and corrected.

7.2 Evaluation of our reseach method

Similarities/Differences with the UEML approach and the reference methodology [GR05b]

There are still some similarities and differences between our research method, the UEML approach and the reference methodology. These similarities and differences are summarized in Table 7.1. The symbol "+" means that the activity defined in the criterion is present in the corresponding approach. The symbol "-" means that the activity defined in the criterion is not present in the corresponding approach. The sentence "Not specified" indicates that the activity defined in the criterion is not specified in the corresponding approach.

Table 7.1: Similarities/Differences with the UEML approach and the reference methodology

Criteria	**Our research method**	**UEML approach**	**Reference methodology [GR05b]**
Describe construct-by-construct	+	+	+
Use the UEML template	+	+	-
Analyse the static and the dynamic separatly	+	-	Not specified
Determine requirements for UEML (Activity 1 of UEML)	-	+	-
Select languages to incorporate into UEML (Activity 2 of UEML)	-	+	-
Convert ontology to meta models	-	Not specified	+
Convert modelling grammar to meta models	+	Not specified	+
Define the scope of the analysis using meta models	+	Not specified	+
Carry out the representation mapping	-	Not specified	+
Carry out the interpretation mapping	+	Not specified	+
Apply the three steps for a team's work	+	Not specified	+
Determine the excess, overload, redundancy in the language	-	Not specified	+

Some criteria need to be explained a bit:

- About the criterion *Convert ontology to meta models*, we did not convert the common ontology to a meta model but we used the UEMLBase Draft [Com06] which gathers the explanations of all the concepts of the common ontology on a textual shape. This document shows the common ontology with the different classes, properties, states, events and tranformations and the relationships between them.

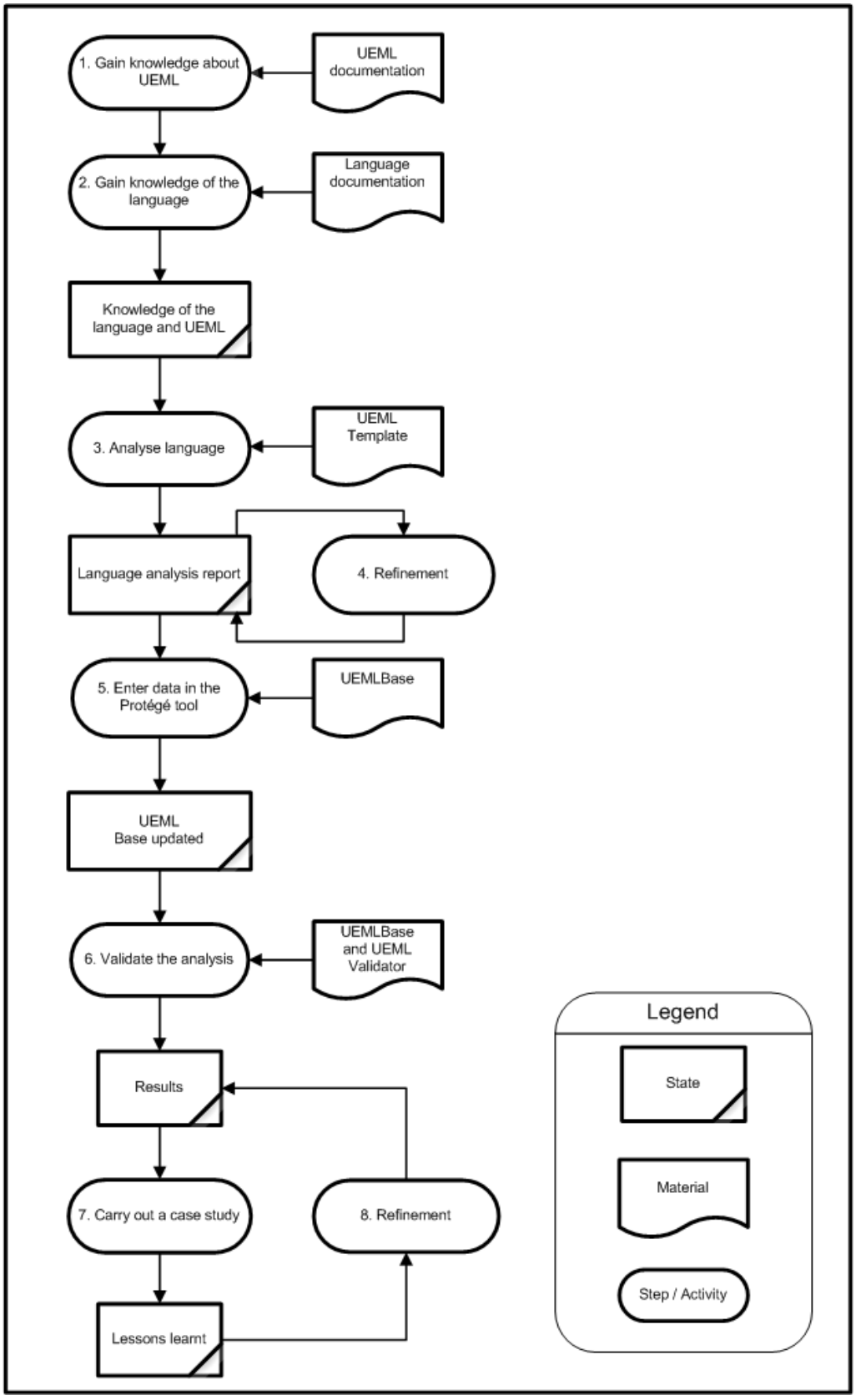


Figure 7.1: Outline of our research method

- For the criteria *Carry out the representation mapping* and *Carry out the interpretation mapping*, in the reference methodology, they recommend starting with the representation mapping. *"That is, selecting the meta-model of the ontology and subsequently identifying the corresponding elements in the modelling grammar"*. We haven't done the representation mapping. We have carried out the interpretation mapping. We have worked by selecting one modelling construct and tried to find the corresponding elements of the ontology. As reported, we have used a cluster-based analysis.

Evaluation of our method

Our method has not more specific defects than another. Some weaknesses are mainly due to the tools used. The use of the UEML template and of the common ontology brings to a lack of comprehension at the beginning.

Dealing with several shortcomings of current ontological analyses identified in Section 2.2.2, we will try to underline some of these shortcomings in our method:

- **Lack of understandability:** An ontology is a combination of textual descriptions and formal structure, where the text is a necessary part. In that sense, some concepts in the common ontology can be ambiguous.
- **Lack of guidance:** We do not have any guidelines to choose the constructs and their order for the analysis. We have selected and analysed the modelling constructs from the most- to the less important.
- **Lack of result classification:** This lack is not directly applicable to our method because we haven't tried to identify some ontological deficiencies, then we didn't have to classify them.

Some strenghts can be underlined:

- **The iterative process:** The iterative process is not a waste of time because we had to to ask some questions and receive a feedback from a meta modelling expert to be able to go ahead correctly and effectively.
- **A pooling of our two opinions and of the one of Andreas L. Opdahl:** It allows to discuss on different sensible points of the analysis and to make it more objective.

Our research method remains the same if we analyse one or more languages. It does not bring anything more if several languages are analysed. But it is important to underline that the analysis of the second language was based on the analysis of the first one.

- **Extension of the common ontology:** The common ontology was extended with the analysis of ARIS, for instance we added the property *MutualLaw* for the logical operator "AND". This property was also used in the analysis of BPMN.
- **Influence of the ARIS templates:** The representation part of the UEML template of several BPMN constructs was influenced by the template of some modelling constructs of the first analysis. For example, the template of the Parallel Gateway in BPMN and the template of the logical operator "AND" in ARIS.

7.3 Proposal of a new method

Base on the experience of applying the method describe in Section 7.1, we have defined a new better method. This method represent the way we would like to work if we have to do this work again. This method was not tested in practice. We cannot prove that this will work, but it is the results of what we have learnt with our two analyses. We will present a detailed outline of a sequence of major process steps and give some advice to facilitate the realization of each steps. These steps are shown on Figure 7.2.

1. **Gain knowledge of UEML [2]:** Read UEML documentation to have a whole idea of the UEML approach.
2. **Gain knowledge of language:** Read language documentation to have a whole idea of the language and of the relationships between the constructs by using the meta models and examples. Understanding the definition, the semantic and the utilization of each construcs is needed.
3. **Choose a certain number of modelling constructs to be analysed on the basis of some guidelines:** Choose first the most central to finish with the most local. Most central means most important, i.e. the construct has a lot of relationships with others and properties and conversely for the most local.
4. **Fill in the preamble and presentation section of the UEML template:** Achieve a paper version for each modelling construct. For a team work, each member has to do it individually.
5. **Discuss the results of step 3 with the teammate(s):** To have a common written paper for each construct.
6. **Discuss the results of step 4 with a meta modelling expert:** To have some precisions concerning the different entries of the filled in template.
7. **Carry out the representation and interpretation mappings explained in [GR05b]:** To have more certitude concerning the mapping and to familiarize with the concepts of the BWW model. It also provides an easy way to emphasize the sensible points of the analysis. Use a graphical representation of the mappings to visualize the relations between the represented classes of things, properties, states, transformations and the common ontology.
8. **Fill in the representation section of the UEML template:** Complete the paper version of the step 3 for each modelling construct. For a team work, each member has to do it individually.
9. **Discuss the results of step 7 with the teammate(s):** To have a complete common written paper for each construct.
10. **Discuss the results of step 8 with a meta modelling expert:** To have some precisions concerning the different entries of the respresentation section. Discuss and argue the choice of the classes of things, properties and behaviour (states and transformations).
11. **Revise the analysis:** To have a final filled in UEML paper template for each construct.
12. **Enter data in the Protégé tool:** Complete the UEMLBase with the language analysis.
13. **Validate the analysis:** Use the validation package (UEML Validator) in Protégé to verify the new UEMLBase. Correct the possible errors in Protégé and on each paper template. This step brings us in a final state where the results are available. They are a summary of the representation and interpretation mappings in two tables.

[2] This step 1 is realized once even if more than one language is analysed.

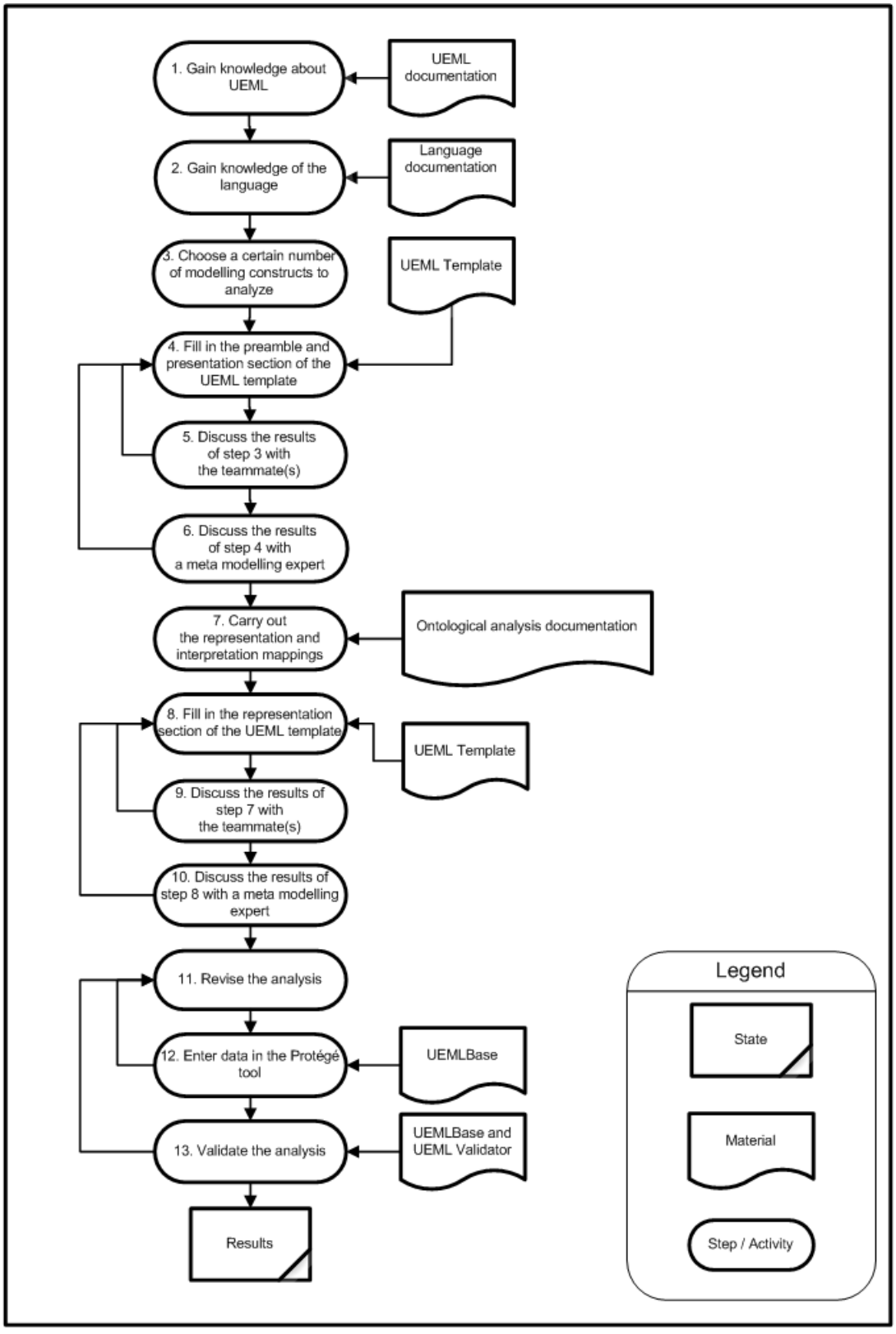


Figure 7.2: Outline of the proposal of a new method

7.4 Summary

In this chapter, we explained our initial research method to evaluate ARIS and BPMN. Then, we compared it to the reference methodology and the UEML approach and made a short evaluation of the way we did the work. Finally, we tried to propose a new method. This new method was not applied practically, but we believe it will bring better results than the previously used methodology. Checking of this hypothesis would require further validation of the methodology through its application on some language analysis.

Chapter 8

Analysis of ARIS

In previous sections, we have surveyed both the UEML approach and the two languages to which we have applied this approach. In this chapter, we start our analysis with ARIS.

We decided to analyse 17 modelling constructs which are the most central in ARIS. We had as guidelines for the choice of these to take 15 to 20 modelling constructs which are the central ones. We considered the others are less important and can be left for another analysis. For instance, the different kinds of data objects. Some are a specialisation of another construct. That implies that the general mapping doesn't change or implies the addition of a subclass of one existing in the common ontology. For instance, the logical operator "OR" which has the same general mapping than the logical operator "AND".

First, we will present the analysis of the *function* which is the most central construct in ARIS. Next, we will give overview of other construct mappings. Appendix H contains more information about the mappings. During the process, we will add new classes to the common ontology. These additions will be justified here and each of the added classes will be described in Chapter 10.

8.1 Function construct

The *function* is the central modelling construct in ARIS. It defines an activity or a task to be executed in a process. Almost all concepts in ARIS are connected to the function. Thus, the functions are the key points of the process.

Following the UEML approach, we have filled in the three parts of the UEML template. We will explain the most relevant entries of those parts. The filled template of the function and the other templates of all ARIS modelling constructs can be found in Appendix H.

Preamble part

Four entries of the preamble part are not filled in: builds on, built on by, related construct names and related terms. Indeed, the function's description is not built on one another and conversely and it does not have any closely related constructs or terms. The others are filled in. The *function* has some alternative construct names like *activity*, *transformation*, *task* and *process*. The language to which this construct belongs to is "Architecture of Integrated Information systems" (ARIS). The last entry of the preamble section is the diagram types where the construct is modelled. In our case, it's the ARIS business process model.

Presentation part

The first relevant entry of the presentation part concerns the icon, the linestyle and the text. In ARIS, the *function* is represented by a soft rectangle (Figure 8.1).

The construct can have some user-definable attributes, this is the following entry. The *function* has one user-definable attribute which is the function's name. Another relevant entry indicates the relations to other constructs. The *function* belongs to one ARIS model and has many relationships to other constructs.

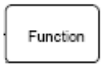


Figure 8.1: Graphical representation of the *function*

- It is under the responsability of 1 to N organizational unit.
- It can be executed by 0 to N software.
- It can use 0 to N machine resource and 0 to N computer hardware.
- It can create 0 to N outputs and process 0 to N inputs. For instance, in the example presented Chapter 5 (Figure 5.3), the function "Accept merchandise" processes two inputs: "Shippment of item" and "Shipping order".
- It can support 0 to N goals i.e. can be controlled by 0 to N goals.
- It can be processed by 0 to N human output.
- It can transform 0 to N environmental data.
- It can be triggered by 1 to N event (in particular, by 1 to N message of the events) and produce 1 to N event.

Representation part

The most important part is the third one, the representation part. The first entry concerns the instantiation level. The *function* can be of type and instance level. Indeed, the construct is of type level, i.e. it represents a whole class, and of instance level, i.e. it represents a single individual thing. The *function* is connected to modelling constructs which are of type and instance level.

This construct is the central one in ARIS, then it has many relationships to other constructs. That makes the modelling construct mapping quite huge. We have decided to divide the model in three parts. These are described by a simplified description based on a graphical representation. The graphical representation (Figure 8.5 [1]) comprises *represented classes*, *represented properties*, *represented states* and *represented transformations* which are all mapped onto the common ontology. First, we have mapped onto the common ontology and, then, we have validated the mapping by using the tool "UEML Validator" and the case study. These are explained in more details in Part III.

Classes of things and properties

For the analysis of the *function*, we have first analysed the static structure. For that, we identified the classes and properties that the modelling construct is intended to represent. One or more classes of things and properties can be identified, each of them playing a particular role in the context of the construct [Opd06b]. For the function, we identified eight classes of things and twelve properties. It concerns the *Classes of things* and *Properties and relationships* entries of the UEML template's representation section. This is shown by Figure 8.2 [2].

Properties belong to a class of things

Organizational unit is responsible of the function

The **function** is represented by a property, *FunctionLaw*. It describes the law that makes the function happen. This property was added to the common ontology because the function doesn't map exactly onto *TransformationLaw*: "A law that restricts the combinations of properties that an (active) thing can possess before and after an event" [Com06]. It is more precise. Thus, the *FunctionLaw* is preceded by the *TransformationLaw* and has the following definition: "A transformation law that manipulates the resources in a repository and produces appropriate outputs in response to the inputs it receives". The function is under the responsibility of an **organizational unit**. So, this property belongs to the class

[1] The legend of Figure 8.5 is shown in Appendix F.
[2] The legend of Figure 8.2 is shown in Appendix F.

organizational unit which is mapped onto *OrganizationalUnit* [3]. We added this class to the common ontology because the organizational unit doesn't map exactly onto the *ActiveThing* class: "A changing thing that acts on at least one other thing through an acting-on relation, a particular type of coupling" [Com06]. It is more precise i.e. an organizational unit is responsible for another thing. So, we decided to create another subclass of *ActiveThing* which is the *OrganizationalUnit* class. (Figure 8.2)

Human participates in the function

The function is a complex property, it means that it has some subproperties. Some are in relation with a class of things. First, there is the **participation** that describes the participation of a human in the creation of an output of the function. This property is a subproperty of the function because this function law must know when and what for the human intervenes in the function. It is the participation law that plays this role. This property is mapped onto *ParticipationLaw* which is a property added to the common ontology. We added it because the definition of the *TransformationLaw* is too large. Thus, the *ParticipationLaw*: "A transformation law that marks the participation of a thing in the creation of another thing" (Chapter 10), is preceded by a *TransformationLaw*. This property belongs to the **human output** class which is mapped onto *HumanOutput* in the common ontology. We added this class as a subclass of *OrganizationalUnit* because the human output is a specialization of the organizational unit. It is a responsible entity and it processes a function [Sch98]. (Figure 8.2)

Software executes a part of the function

The function possesses a subproperty, the **application law**, which is a law. It belongs to the **software**. This law describes the fact that the software executes a part of the function (as the participation law with the function law). It is why we added the *ApplicationLaw* to the common ontology. The software is mapped onto *ExecutingThing* because of the definition of the application software: "An application software is a software running on a computer that is used to support the execution of a function." [Sch98]. (Figure 8.2)

Machine and computer hardware are used by the function

Two subproperties of the function are the **useLawMachine** and **useLawComputerHardware** which both mapped onto *UseLaw*. We added this law because a part of the function uses a machine resource or a computer hardware. Thus, the definition of *UseLaw* is "A transformation law that marks the part of a thing used by another (active) thing". The useLawMachine and useLawComputerHardware properties belong respectively to the **machine** and **computerHardware**. The class machine is mapped onto *MachineResource* and the class computerHardware onto *ComputerHardware*. We added those two classes of things to the common ontology because no class was suitable. These two are subclasses of a new class *Equipment*. This is described as "An active thing that equips another thing". (Figure 8.2)

Environmental data is processed the function

Another subproperty of the function is the **information flow** which describes the information flow between the function and the environmental data. This property is mapped onto *InteractionRelation* and belongs to the **environmental data**. This class is mapped onto two ontology classes: *InputOutputThing* and *ReactiveThing*. They were chosen because the environmental data represents the data used and added to databases during the instantiation of a function. And, functions process environmental data by transforming input data into output data [Sch98]. Thus, the definition of *InputOutputThing*: "The class of things that are both targets and sources of flows" [Com06] applies well to the environmental data class. But there is something more than just being the target and the source of a flow. It is an *InteractingThing* ("A changing thing that both acts on and is acted on by another thing, i.e., it interacts with the other thing" [Com06]) that possesses any TransformationLaw. It is why we added the *ReactiveThing*. An environmental data actes on the function and is acted on by the function. (Figure 8.2)

[3] In the final version of the template, presented in Appendix H, the construct organizational unit is mapped onto *Participant*. However, we describe it here like this because this mapping has been modified according to the case study (Refer to Chapter 12).

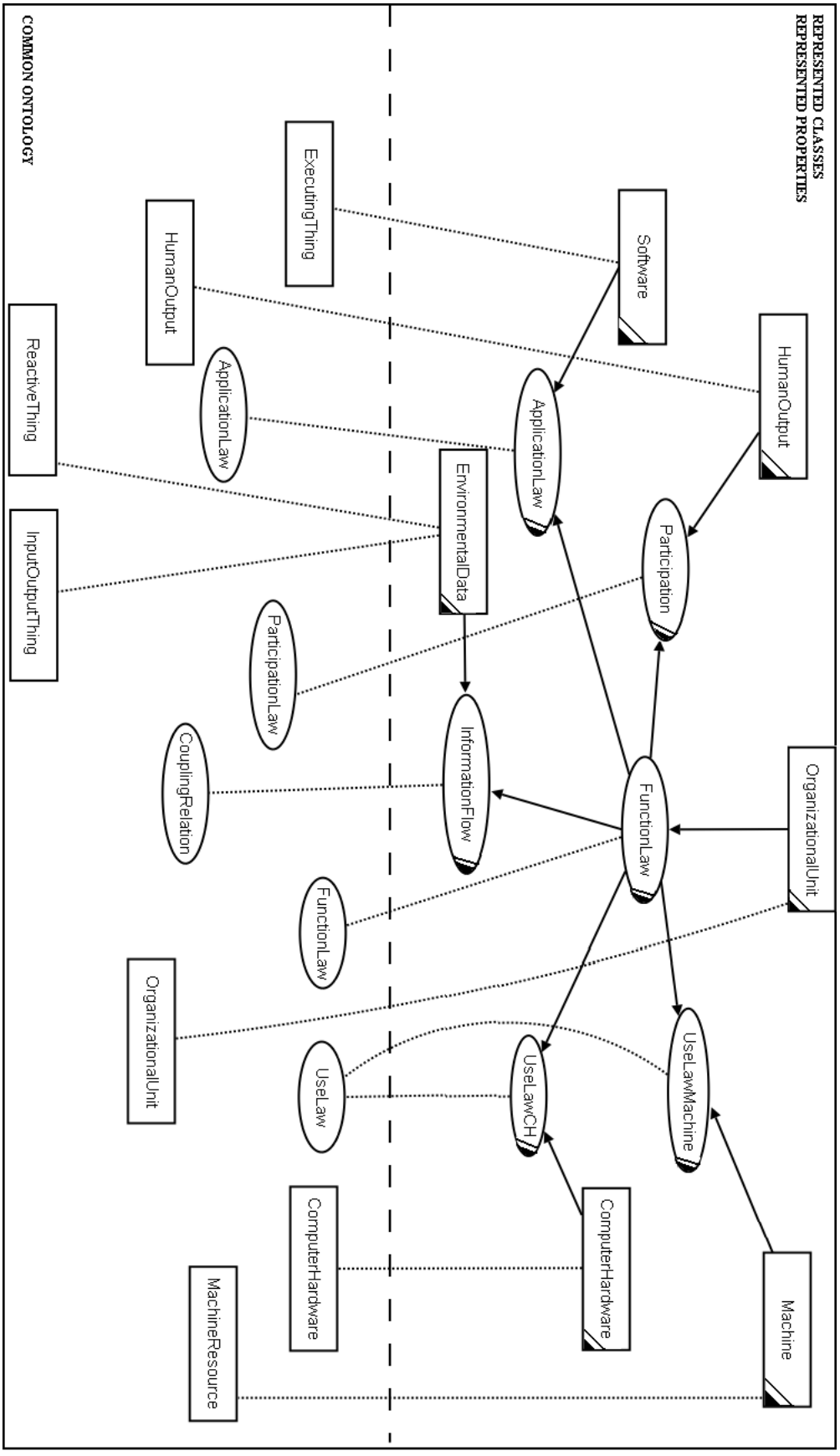


Figure 8.2: Simplified description of the function - Part 1

Complex properties

The function is a complex property. Thus, it possesses some subproperties. This is shown by Figure 8.3 [4].

The function creates ouput and processes input

Two other subproperties of the function are the **incoming flow** and **outgoing flow** of the function. These two properties are mapped onto *Flow*. This ontology property does represent the flow between two things. These are complex property. The incoming flow possesses the **OutputFlowInput** property that describes the content of the incoming flow. The outgoing flow possesses the **OutputFlow** property that describes the content of the outgoing flow. Both are mapped onto *FlowContent*. We added this property to the common ontology because there was any property to represent the content of a flow. Thus, the *FlowContent* is a regular or a mutual property that marks the content of a flow. The OutputFlowInput property belongs to **SourceOutput** which is intented to represent the input of the function. It is mapped onto two classes of things: *OutputThing* and *Repository*. The first one was chosen because the definition of this class: "The class of things that are sources of flows" [Com06], applies to the input. But an input is more than just a source of a flow. It is something that contains some information. So, it is why we created the *Repository* with the following definition: "A container that contains informational or material ressources" (Chapter 10). The OutputFlow property belongs to **TargetOutput** which is intented to represent the output of the function. It is mapped onto two classes of things: *InputThing* and *Repository* for the same reasons than the source output. (Figure 8.3)

Goal controls the function

Another subproperty of the function is the **goal**. The goal is mapped onto *Law*. We decided to map onto *Law* and not onto *TransformationLaw* or *StateLaw* because ARIS doesn't make the distinction between the goals which represent *StateLaw* and those which represent *TransformationLaw*. The goal has a subproperty which is the outgoing flow. These two properties are linked because a goal is met as a result of executing the function [Sch99]. (Figure 8.3)

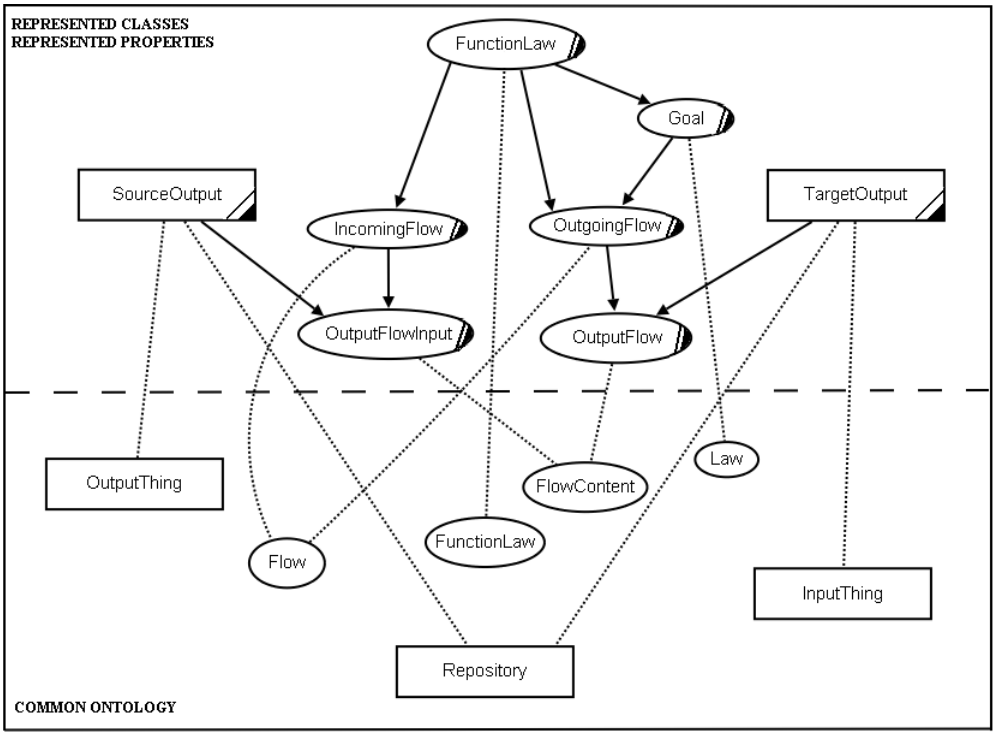


Figure 8.3: Simplified description of the function - Part 2

[4]The legend of Figure 8.3 is shown in Appendix F.

States and transformations

Then, we analysed the kind of dynamic behaviour that the function can be used to represent. The function is a dynamic construct. It represents a process which is a chain of alternating states or events. We identified two states in which the function can be and four transformations which allows to pass from one state to another. It concerns the *behaviour* entry. This is shown by Figure 8.4 [5].

Property used for the dynamic behaviour

As we can see in Figure 8.4 , the last subproperty of the function is **IsActive**. This property which is mapped onto *IsActive*. It describes "A manipulated property that marks whether an (executing) thing is active or non-active" [Com06]. It is a particular property which is used in the description of the dynamic part.

States

The function can be in two states. These states must be described as an invariant over the property roles defined in the *properties* entry [Opd06a]. The property role is the property "IsActive". The two states are **FunctioningState** and **NonFunctioningState**. The function is in the FunctioningState when it is active ("IsActive == true" which is the state constraint). The FunctioningState is played by *ActiveState*. When it is not active, the function is in the NonFunctioningState. This state is played by *InactiveState*.

Transformations

To pass from the NonFunctioningState to the FunctioningState, the **TriggeringFunction** transformation is achieved. The trigger of this transition is that an event occurs (an input is created) and there is a condition: all the inputs are available. When the transition occurs and the condition is true, the function is activated. This transformation is played by *Triggering*. To pass from the FunctioningState to the NonFunctioningState, the **TerminationFunction** is realized. The trigger is that an event occurs (an output is created) and there is a condition: all the outputs are available. When the transition occurs and the condition is true, the function is terminated. There remain two transitions when the conditions are not satisfied: **NotAllInputAvailable** and **NotAllOutputAvailable**. Both are mapped onto *AnyTransformation*. When the NotAllInputAvailable transformation occurs, the function waits to have all inputs available. When the NotAllOutputAvailable transformation occurs, the function continues to be executed to produce all the outputs.

[5] The legend of Figure 8.4 is shown in Appendix F.

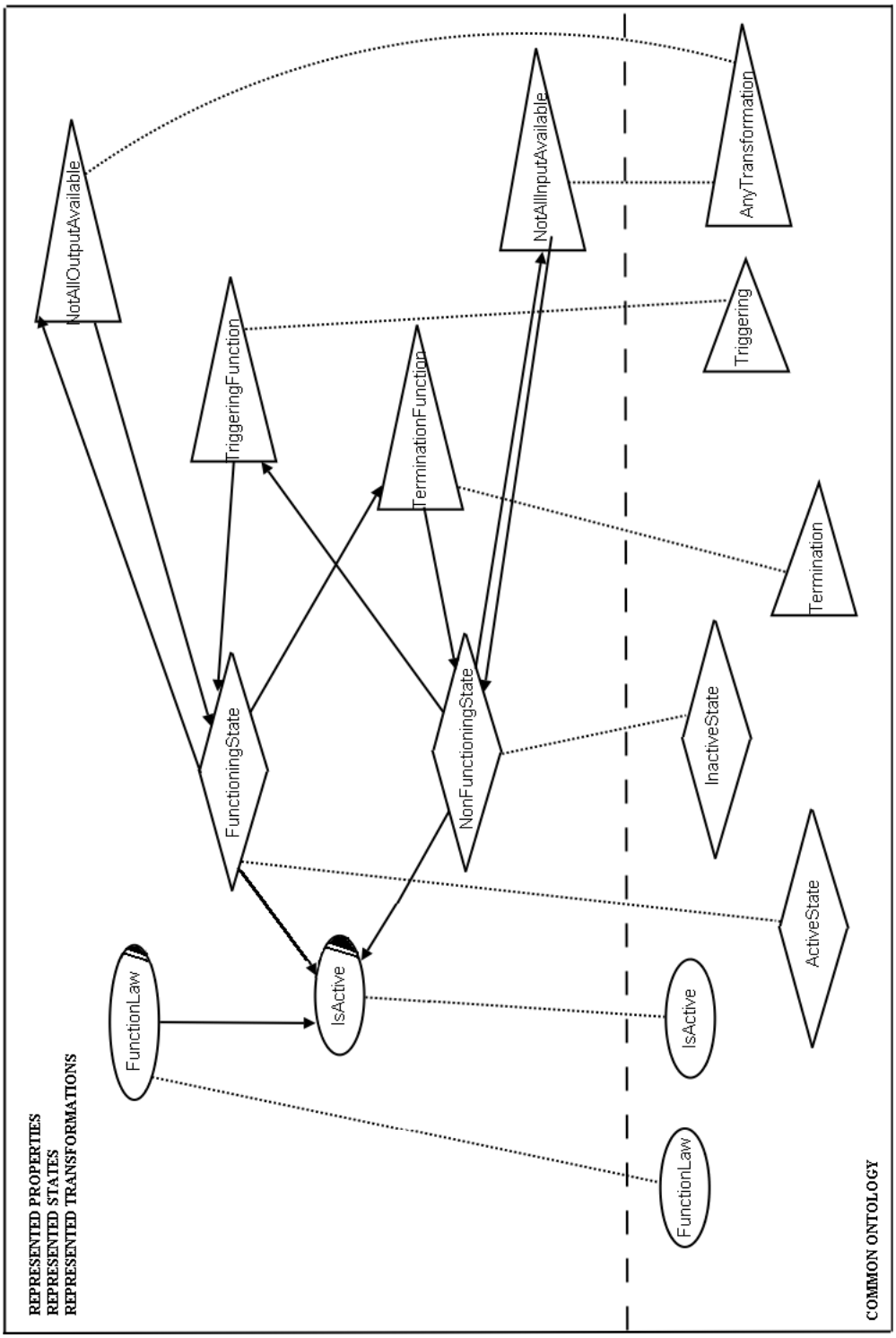


Figure 8.4: Simplified description of the function - Part 3

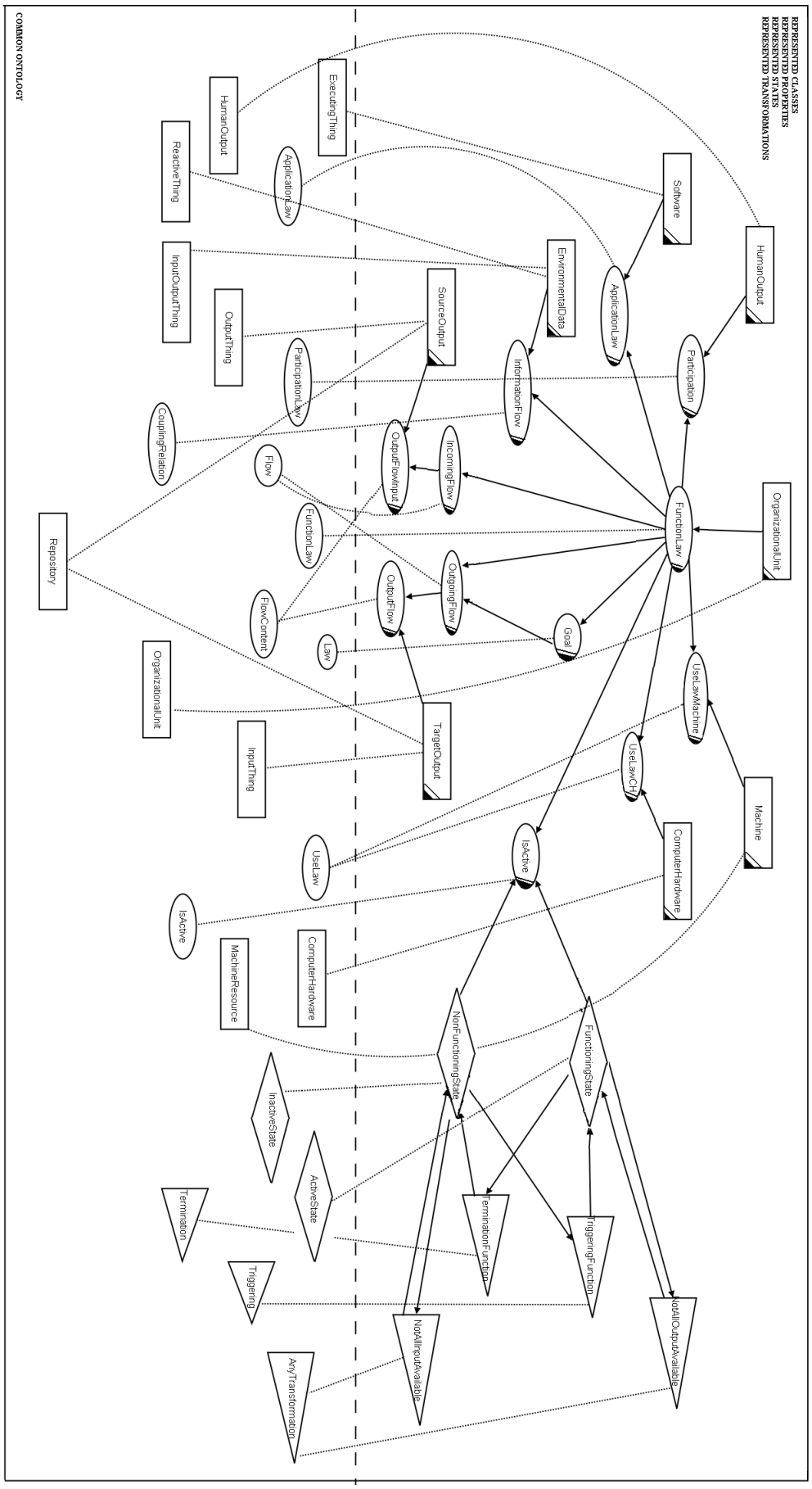


Figure 8.5: Complete description of the function

8.2 Mapping table

The following Table 8.1 lists the primary mappings of ARIS modelling constructs to the common ontology. The first column indicates the name of the modelling construct. The second one represents the mapping to the common ontology.

Table 8.1: Primary mappings of ARIS modelling constructs

Modelling construct	**Mapping to the common ontology**
Function	*FunctionLaw*
Organizational unit	*OrganizationalUnit* [6]
Position	*RoleHolder*
And	*MutualLaw*
Ouput	*Repository* and (*InputThing* or *OutputThing*)
Material output	*MaterialRepository* and (*InputThing* or *OutputThing*)
Services	*Service* and (*InputThing* or *OutputThing*)
Information services	*InformationService* and (*InputThing* or *OutputThing*)
Other services	*MaterialService* and (*InputThing* or *OutputThing*)
Environmental data	*ReactiveThing* and*InputOutputThing*
Event	*Flow* and *FlowContent*
Message	*StateLaw*
Application software	*ExecutingThing*
Human output	*HumanOutput*
Goal	*Law*
Machine resource	*MachineResource*
Computer hardware	*ComputerHardware*

We will discuss each modelling construct and their main mappings.

Function

The function is explained in detail in Section 8.1.

Organizational unit

Three represented classes and eigth represented properties have been identified. The major class is the organizational unit which is mapped onto *OrganizationalUnit* [6]. We added this class to the common ontology because the organizational unit doesn't map exactly onto the *ActiveThing* class. It is more precise, i.e. an organizational unit is responsible for another thing. So, we decided to create another subclass of *ActiveThing* which is *OrganizationalUnit*. The organizational unit is linked to two other classes by a part-whole relation: the human output and the position. The organizational unit is responsible for a function which is represented by a property. This property is mapped onto *FunctionLaw* (See above for explanation).

Position

The scene of the position can be divided into three parts. The main one is the relation between the organizational unit and the position. The organizational unit is mapped onto *OrganizationalUnit* as explained previously. The position is mapped onto *RoleHolder* in the common ontology. This class was added because of the definition of the position: "A position is a role performed by individual people. An organizational unit may be composed of Positions". The class *RoleHolder* is a subclass of *OrganizationalUnit*. The link between those classes is represented by a *PartWholeRelation* because an organizational unit may be composed of positions.

[6] In the final version of the template, presented in Appendix H, the construct organizational unit is mapped onto *Participant*. However, we describe it here like this because this mapping has been modified according to the case study (Refer to Chapter 12).

And

Two classes of things and three properties have been identified. The two classes represent the input and the output of the logical operator "AND". They are mapped onto *CoupledThing* in the common ontology because the "AND" associates its input and output. This operator is a mutual property between those classes and is mapped onto *MutualLaw*. We added this property because we needed a transformation law which is mutual and the definition of *TransformationLaw* says that "a transformation law is not mutual, not a part-whole relation and not a class-subclass relationship". *MutualLaw* is preceded by *CouplingRelation* in the common ontology. The two other properties (InputEnding and OutputEnding) are subproperties of the "AND" and properties of the classes. They are mapped onto *Flow* in the common ontology to show the fact that if there is input then there is output.

Output

Two classes of output have been identified: TargetOutput and SourceOutput, because the output can be an input or an output of the function. The cardinalities of those classes are 0-1. Both are mapped onto *Repository*. We added this class as a subclass of *LocationOfResource* because the definition of *LocationOfResource*: "A container whose contents are resources, a particular kind of content that is acted on by a process. A resource location may hold resources either transiently or durably" [Com06], was not enough precise. The *Repository* is a container that contains informational or material ressources which represent the outputs in that case. The TargetOutput is also mapped onto *InputThing* and SourceOutput onto *OutputThing* because of the definitions of these classes (*InputThing*: "The class of things that are targets of flows" and *OutputThing*: "The class of things that are sources of flows" [Com06]).

Material output

Like for the output, there are two classes of material output: one for the material input (TargetMaterialOutput) and one for the material output (SourceMaterialOutput) of the function. Both are mapped onto *MaterialRepository*. We added this class as a subclass of *Repository* because the output can be physical (material output) or non-physical (services). The TargetMaterialOutput is also mapped onto *InputThing* and SourceMaterialOutput onto *OutputThing*. Both classes possess a property which represents the function that uses/creates material output.

Services

It is the same than the output and material output for the main part. The two classes (TargetService and SourceService) are both mapped onto *Service* which was added in the common ontology. The reason of this adding was the classification of the types of output (See Figure 5.2).

Information services

The information services are a specification of the services (See Figure 5.2). Thus, the two classes of information services (TargetInformationService and SourceInformationService) are mapped onto *InformationService*. This class is a subclass of *Service*.

Other services

The other services are a specification of the services (See Figure 5.2). Thus, the two classes of other services (TargetOtherService and SourceOtherService) are mapped onto *MaterialService*. This class is a subclass of *Service*.

Environmental data

One class and four properties have been identified. The class is the environmental data which is mapped onto *InputOutputThing* and *ReactiveThing*. For the first mapping, the reason was that the function transforms input data into output data; thus the environmental data is the target and the source of the link between itself and the function. Environmental data is also mapped onto *ReactiveThing* because the environmental data interacts with the function. This class was added to the common ontology as an *InteractingThing* that possesses any TransformationLaw. The two main properties of this class are the function and the information flow. The function is mapped onto *FunctionLaw*. The information flow represents the link between the function and the environmental data and it is also a subproperty of the function. It is mapped onto *InteractionRelation* because of the fact that function interacts with the environmental data.

Event

The scene is divided into two parts. The main one is composed of the organizational unit which possesses the FunctionLaw property. This property has a subproperty which is the flow of the function (flow between functions). This subproperty is mapped onto *Flow* and is a complex property. Its subproperty is called Event and is mapped onto *FlowContent*. We added this property to the common ontology because we had to represent the fact that the flow can have a content. Thus, the definition of *FlowContent* is "A regular or a mutual property that marks the content of a flow". The event modelling construct is represented by the *Flow* and also the *FlowContent* because of the definition of the event: "An event is an instance of a state that influences or controls the further flow of the processes".

Message

The message is a subproperty of the FunctionLaw property and the Event property (mapped onto *FlowContent*). It is mapped onto *StateLaw*. The definition of the *StateLaw* is "A law that restricts the combinations of properties that a thing can possess in a state". It is the case with the message because the message determines how functions react to events. It is also why the property Message is a subproperty of FunctionLaw. It is also a subproperty of Event because "Messages indicate that the events have been detected, pass them on to successive functions and then activate them. Accordingly, events trigger messages, transmitting the fact that an event has occurred" (Section 5.3).

Application sofware

The main part of the scene is composed of one class and two properties. The represented class is the software which is mapped onto *ExecutingThing*. We have chosen to map onto this class of thing because a software executes a part of the function. Thus, the definition of the *ExecutingThing*: "An active thing that is not active all the time, but has an active state and an inactive state and corresponding triggering and terminating events. In other words, it is an active thing that has execution behaviour." [Com06], is appropriated. The class software possesses a property which is the FunctionLaw and an ApplicationLaw which represents the fact that the application software executes a part of the function. This property is mapped onto *ApplicationLaw*. We added this property in the common ontology because the definition of *TransformationLaw* was not enough precise. Thus, the *ApplicationLaw* is preceded by *TransformationLaw* and has the following definition: "A tranformation law that marks the part of a thing executed by a (executing) thing".

Human output

The scene of the human output can be divided into three parts. The main one is the relation between the organizational unit and the human output. The class HumanOutput is mapped onto *HumanOutput*. This class was added to the common ontology because of the definition of the human output: "A human output processes a function. Is is a responsible entity and it represents the people involve in the output's realization". The class *HumanOutput* is a subclass of *OrganizationalUnit*. The link between those classes is represented by a *PartWholeRelation* because an organizational unit may be composed of human output as shown in Figure D.2. The class HumanOutput possesses a property which represents the participation of the human in the function (mapped onto *ParticipationLaw*).

Goal

The scene of the goal is composed of one class and three properties. The class represents the organizational unit and possesses a property (FunctionLaw). This property has a subproperty: OutgoingFlow which is mapped onto *Flow* because it represents the outgoing flow of the function. The last property of the scene is the goal. The goal is mapped onto *Law*. We choose to say that the general Law represents the goal. Because ARIS doesn't make the distinction between the goals which represent a StateLaw or a TransformationLaw. The property OutgoingFlow is a subproperty of the goal because of the definition of the goal: "A goal is met as a result of executing the process. It represents the finality of a function", i.e. the goal is linked to the function by means of its outgoing flow.

Machine resource

The scene is divided into four parts. The first main one is the relation between the machine resource and the organizational unit. To remember, the definition of the machine resource is "The machine resources are the different material equipments having a role to play in a process. They are responsible entities. They are used by a function". Thus, we decided to add a new class to the common ontology: *MachineResource*. The machine resource modelling construct is mapped onto *MachineResource*. For

that reason, we had to create a new class which is *Equipment*. In that sense, *MachineResource* is a subclass of *Equipment* because of the definition of the machine resource. We put as definition of the *MachineResource* class the following one: "An equipment thing that is a device designed for doing a work". The machine resource is allocated to an organizational unit. This relation is represented by a property (Allocation) which is mapped onto *MutualProperty* because this property is mutual between the classes OrganizationalUnit and Machine. The second main part is the relation between the function and the machine resource. The OrganizationalUnit has a FunctionLaw property and the Machine has a UseLaw property mapped onto *UseLaw*. This property represents the fact that a part of the function uses a machine resource. Thus, we had to add a new law which represents this fact. We called it *UseLaw*. And, this UseLaw is a subproperty of the FunctionLaw.

Computer hardware

The scene of the computer hardware is nearly the same as the machine resource. The definition of computer hardware is "Computer hardware represents all the devices capable of accepting and storing computer data, executing a systematic sequence of operations on computer data, or producing control outputs". Thus, we had to create a new class to represent that fact: *ComputerHardware* as a subclass of *Equipment*. The class ComputerHardware is mapped onto *ComputerHardware*. We have also identified the properties FunctionLaw and UseLaw like in the scene of the machine resource.

8.3 Summary

In this chapter, we explained in detail the analysis (template) of one modelling construct, the *function*, and we discussed briefly of all the ARIS modelling constructs we analysed. The status of this chapter is the status before validation. We will validate the analysis of ARIS by using the UEML Validator and through a case study, presented in the Validation/Evaluation part. After this analysis, we have entered the filled in paper template into the tool Protégé and we especially focused on the dynamic part of ARIS, i.e. the modelling constructs which are intended to "represent behaviour" (particular states or events or processes).

After the achievement of ARIS analysis, we can give a general opinion of this language. The main advantage of ARIS is that it allows representing information as individual views of business process. This permits to have a global view of the process with the control view and then more details views on some parts of the process. For instance, the relations between the different organizational units. The language is intuitive and it does not possess too many constructs, i.e. it allows creating models simply. A difficulty lies in the range of the responsible entities. Indeed, a lot of responsible entities are described: the organizational unit, the human output, the machine resource and the computer hardware. Sometimes it is difficult to see in what they are implied in the realization of a function. A problem in the definition of ARIS is that there is a lack of explanation on the construct's attributes.

Chapter 9

Analysis of BPMN

On the basis of the UEML and BPMN chapters, we are going to explain our analysis of BPMN.

Our analysis was made on the basis of 18 modelling constructs which are the most important in BPMN. Like for the analysis of ARIS, we have chosen the most central constructs. Those are the ones which have more relationships with other constructs, in that sense they are the most important ones. Some can be considered as secondary, we decide to let them for a future possible analysis. To illustrate what we name "secondary", let us take the example of the artifacts of BPMN. This construct provides modellers with the capability of showing additional information about a process that is not directly related to the sequence flow or message flow of the process. It is not really important to understand a model, it is just some additionnal information. That is why we have called it secondary. Another example should relate the possible specification of a construct. The task can be a *Send task*, in this case, the mapping for this construct (Send Task) doesn't change or implies the adding of a subclass of an existing one in the common ontology in comparison with the mapping of the *task*.

We will present the analysis of the *activity* which is the most central construct in BPMN and a brief explanation of the other constructs.

9.1 Activity construct

The *activity* defines the work that is performed within a business process. It can be atomic or non-atomic (compound). The types of *activities* that are a part of a Business Process Diagram are: Process, Sub-Process, and Task. We will only explain the activity analysis. All the analyses (filled in templates) can be found in Appendix I.

Preamble part

The preamble is divided into some entries, as explained in the Chapter 4. We will briefly explain all the entries of the template. Firstly, we have to complete the entry "build on" a construct. This means that the construct is based onto the mentioned name, it's a variant, a refinement. The "built on by" entry is the inverse of the "buid on". For example, the *activity* is the base of two other modelling constructs, the *task* and the *sub-process*. They are a specialization of the *activity*, they are then "built on by" this construct. The following entries are the "construct name", the "alternative construct name", the "related construct names" and the "related terms". We just need a few explanations for the related terms. In fact the related terms correspond to additionnal terms that are used usually but different from the construct name. For the *activity*, there are some related construct names, as *task*, *sub-process* and *process*. This means that those constructs are directly related to the *activity*, the first two ones are built on *activity* and the last one is a set of graphical objects which are a set of other activities and the controls that sequence them [OMG06]. The next entry, "language" entry identifies the version of BPMN that we analysed. We used the version 1.0 of BPMN to make this analysis. The last entry of the preamble section is the "diagram types" where the construct is modelled. In our case, it is the Business Process Diagram (BPD).

Presentation part

The first relevant entry of the presentation part concerns the icon, the linestyle and the text. In BPMN, the *activity* is represented by a rounded corner rectangle (Figure 9.1).

Figure 9.1: Graphical representation of the *activity*

The construct has some user-definable attributes, this is the following entry. The *activity* has some attributes as a name, some assignements, a pool, some lanes, the activity type, the status, some properties, some inputs sets, some outputs sets, some IO rules, a start quantity and a loop type (See Appendix I to learn more about those properties). The last relevant entry of this part is the relationships to other constructs. The *activity* belongs to one Business Process Diagram. It can be the source of 0 to N sequence flow. It can be the target of 0 to N sequence flow. It can be the source of 0 to N message flow. To finish, it can be the target of 0 to N message flow.

Representation part

In this representation section, the first entry concerns the instantiation level. The *activity* can be of type and instance level. Indeed, the construct is of type level, i.e. it represents a whole class, and of instance level, i.e. it represents a single individual thing. The *activity* is connected to modelling constructs which are of type and instance level.

This construct is the central one in BPMN, then it has a lot of relationships with other constructs. According to that, the *activity* has a lot of attributes. That makes the modelling construct mapping quite huge. We then decided to divide the model in three parts. The two first parts concern the static part: classes of things and properties. The third part is the dynamic part. Those three parts are the explanation of the "Classes of things", "Properties and relationships" and "Behaviour" entries. These are described by a simplified description based on a graphical representation. The graphical representation (Figure 9.5 [1]) comprises *represented classes*, *represented properties*, *represented states* and *represented transformations* which are all mapped onto the common ontology. First, we have mapped onto the common ontology and, then, we validated the mapping by using the tool "UEML Validator" and the case study. These are explained in more details in Part III.

Classes of things and properties

For the analysis of the *activity*, we proceeded like the analysis of the *ARIS.function*. First, we analysed the static structure. We identified one class of things and twenty-two properties. It concerns the *Classes of things* and *Properties and relationships* entries of the UEML template's representation section. This is shown by Figures 9.2 and 9.3 [2].

Properties belong to a class of things

The **activity** is represented by a property, *ActivityLaw*, that describes the law that makes the activity happen [Com06]. This property was added to the common ontology because the activity did not map exactly onto *TransformationLaw*. This *TransformationLaw* is defined in the common ontology as "a law that restricts the combinations of properties that an (active) thing can possess before and after an event" [Com06]. More than restricting the combinations of properties that an active thing can possess, the activity in BPMN produces outputs according to the imputs. In [OMG06], an activity is a generic term for work that company performs. That is why we decided to add the *ActivityLaw*, as a transformation law that produces appropriate outputs in response to the inputs it receives. This activity is performed by a **participant**. It is mapped onto *ActiveThing* [3] in the common ontology because this participant performs an activity. This is shown by Figure 9.2.

[1] The legend of Figure 9.5 is shown in Appendix F.

[2] The legend of Figures 9.2 and 9.3 is shown in Appendix F.

[3] In the final version of the template, presented in Appendix I, the participant is mapped onto *Participant*. However, we describe it here like this because this mapping has been modified according to the case study (Refer to Chapter 12).

Complex properties

The activity is a complex property. Thus, it possesses some subproperties. This is shown by Figures 9.2 and 9.3.

Attributes

This activity is a complex property, it means that it has some subproperties. Those subproperties represent the parameters of the activity. Firstly, there are the attribute **properties**, which represent the properties defined by the modeller. Beside, there is the **assignement** property. To end this first group of properties, there are the **pool** and the **lane**. The pool identifies the location of the activity as the lane represents the composition of the pool. All this four properties are mapped onto the *RegularProperty*. They are simple properties, without special signifactions and without any special type definitions, that is why we choose the most general kind of property for all of them. Afterwords, there is the **token** property, it represents the number of token of a single sequence flow that are already arrived to the activity. This is thus a property for which its value change, this is typically mapped onto the *MutableProperty* in the sense of the common ontology. Then there is a new group of subproperties. They are all mapped onto the *RegulareStringProperty* because they are all defined as a String. They are the **ActivityType**, the **LoopType**, the **name** of the activity and finally the **status** of this one. The last property is the **StartQuantity**. It represents the number of tokens that must arrive from a single sequence flow before the activity can begin and it was created to represent the dynamic part explained later in this section. This property is an integer and doesn't change with the evolution. That is why we have decided to map it onto *RegularNaturalProperty*. (Figure 9.2)

Sequence flow of the activity

As explained previously, the activity is a complex property representing a law (*ActivityLaw*). This activity is in direct relation with some flows, the **sequence flow** and **message flow**. We decided to represent the sequence flow according two different parts, the incoming and the outgoing flows. Those flows are mapped onto *Flow* in the common ontology. The contents of those flows are represented by the incoming and outgoing flow contents (IncomingSequenceFlowContent and OutgoingSequenceFlowContent). Those properties are mapped onto *FlowContent* in the common ontology. (Figure 9.3)

Message flow and rules of the activity

The message flow is a bit more complex. For this part, the activity has a IORules property, which defines "the relationship between one input set and one output set" [OMG06]. That is why the **IORules** is mapped onto *Law*. As explained in the definition, the IORules is in direct relation with the inputSets and the outputSets. As their relationship is defined by the IORules, those two sets are subproperties of this IORules property. Those **inputSets** and **outputSets** define respectively "the data requirements for input to the activity" and "the data requirements for output from the activity" [OMG06]. As they are requirements, they are also mapped onto *Law*. InputSets has a subproperty, **IncomingMsgFlow** and outputSets has a subproperty, **OutgoingMsgFlow** because it that case, the message of the incoming/outgoing message flow is linked to the data requirements for input/output to the activity. Those incoming and outgoing message flows are mapped onto *Flow* and are subproperties of the activity. As for the sequence flow, they have each one a flow content (IncomingMessageFlowContent and OutgoingMessageFlowContent) which are mapped onto *FlowContent* in the common ontology. (Figure 9.3)

States and transformations

Then, we have analysed the kind of dynamic behaviour that the activity can be used to represent. The activity is a dynamic construct. It represents a process which is a chain of alternating states or events. We identified two states in which the activity can be and three transformations which allow passing from one state to another. It concerns the *behaviour* entry. This is shown by Figure 9.4 [4].

Property used for the dynamic behaviour

As we can see in Figure 9.4, the last subproperty of the activity is **IsActive**. It is a particular property, it's intended to represent the dynamic part of the activity, i.e. the activity or non activity. That is why we mapped this property onto *IsActive* in the common ontology. This property is defined as "a manipulated property that marks whether an (executing) thing (e.g. the activity) is active or

[4] The legend of Figure 9.4 is shown in Appendix F.

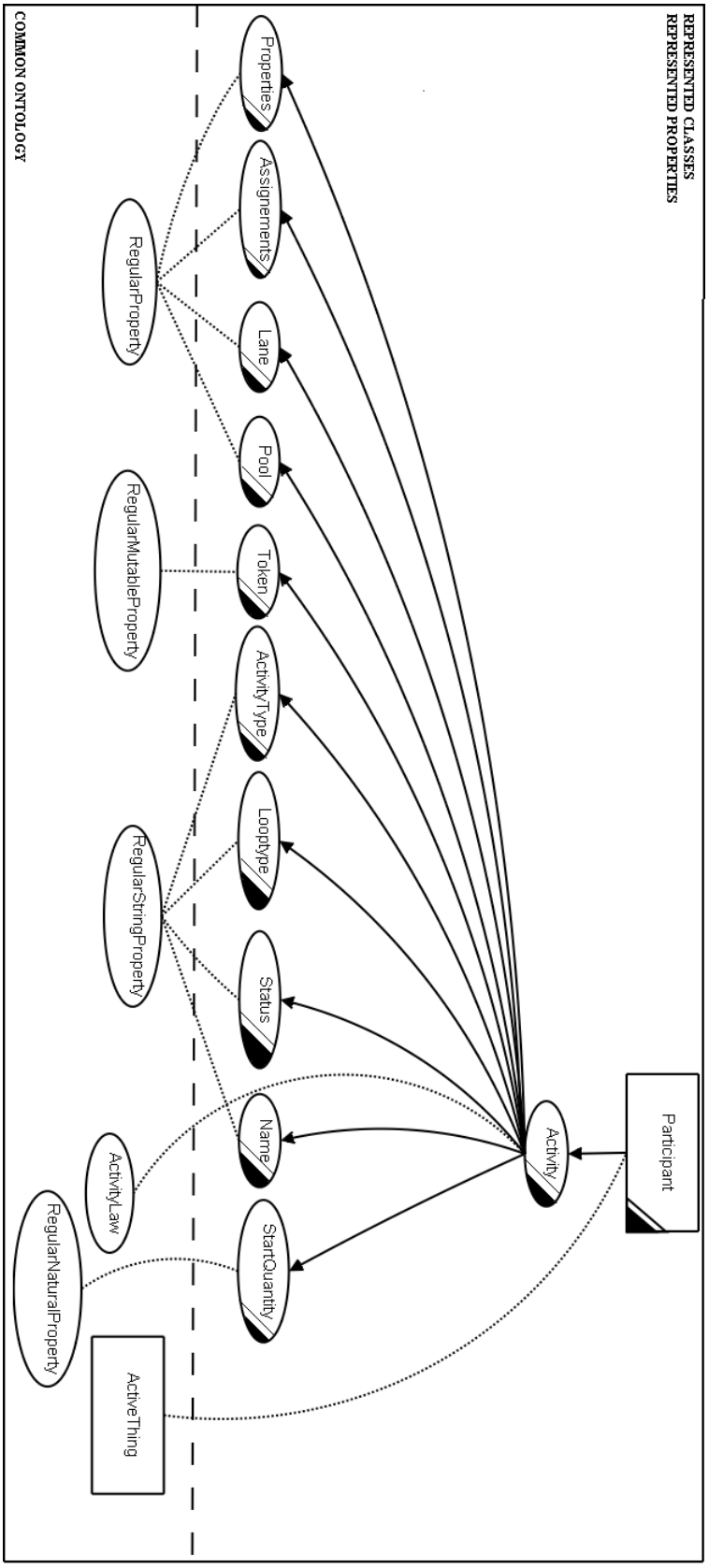


Figure 9.2: Simplified description of the activity - Part 1

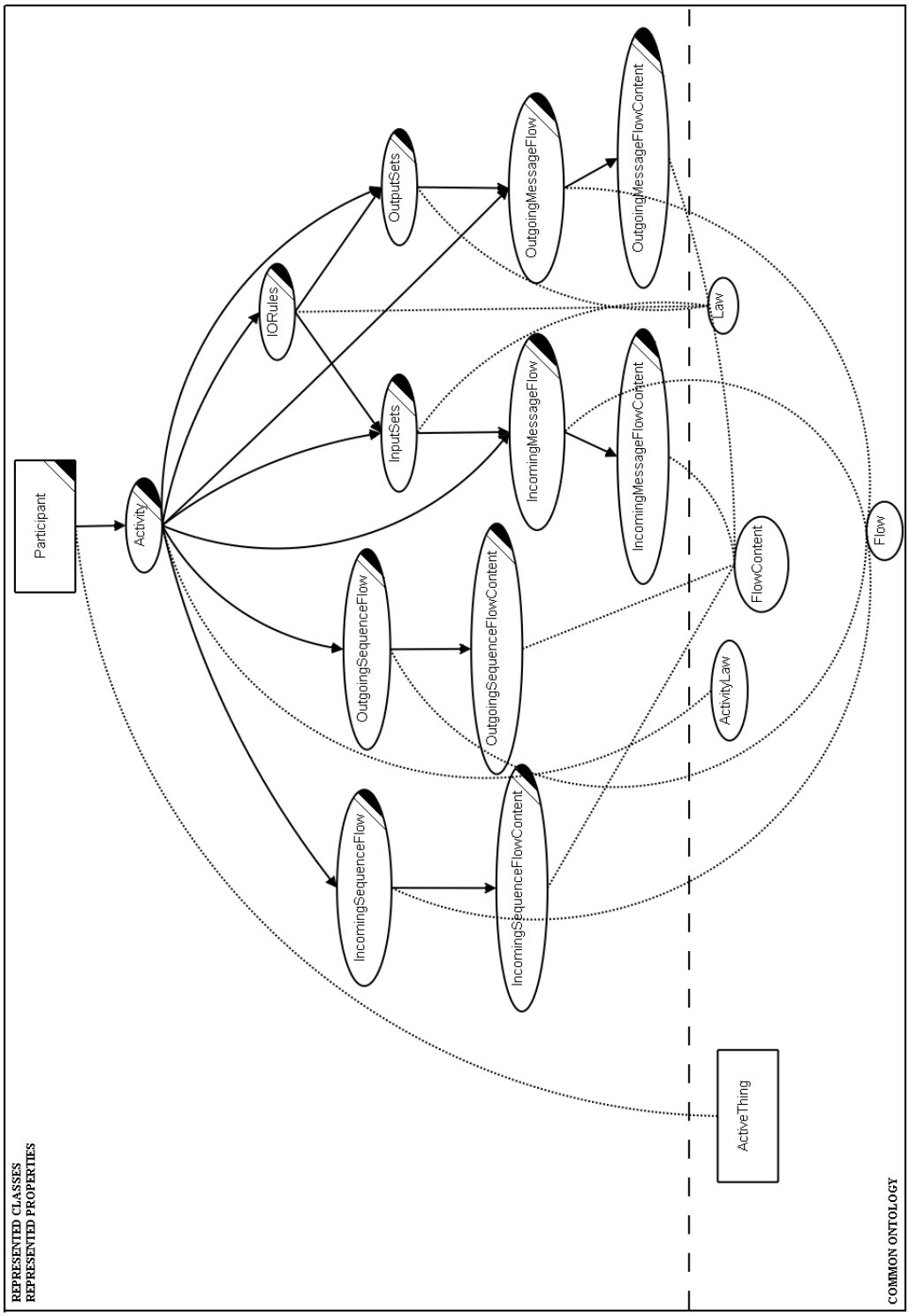


Figure 9.3: Simplified description of the activity - Part 2

non-active" [Com06], thus exactly what we were looking for.

States

The activity can be in two different states: **ActiveState** and **InactiveState**. The activity is obviously in the ActiveState when it is active and it is in the InactiveState when it is inactive. This activity is active when the number of token is reached. It is inactive when all the tokens are not generated for all outgoing sequence flows, i.e. while the number of tokens is not yet reached. The active state is mapped onto *ActiveState* as the inactive state is mapped onto *InactiveState* in the common ontology.

Transformations

The **TriggeringActivity** transformation allows to pass from the InactiveState to the ActiveState. The trigger of this transition is the arrival of a token from an incoming sequence flow but the realization of this transformation needs to wait that the condition - which indicates that the start quantity is equal to the number of tokens needed - is true. This transformation is mapped onto *Triggering* in the common ontology because this transformation lets the activity pass from an enabled state to an activity state. The **NotAllTokenAvailable** transformation is quite special, it intends to represent that the activity is always in the ActiveState while not all the tokens are available for each outgoing sequence flow. The trigger of this transformation is the generation of a token for an outgoing sequence flow but the realization of this transformation needs to assure that the condition - which indicates that the not all the tokens are generated for the outgoing sequence flow - is true. This transformation is mapped onto *AnyTransformation* in the common ontology because any of the Firing, Triggering or Termination transformations correspond to this one. The last transformation, **TerminationActivity**, represents the end of the activity and then makes the activity passe from the ActiveState to the InactiveState. The trigger is the generation of a token for an outgoing sequence flow, but must wait that the condition - which indicates that all the tokens must be generated for each outgoing sequence flow- is true. Finally, this transformation is mapped onto *Termination* in the common ontology because it represents a change from an activate state to another state.

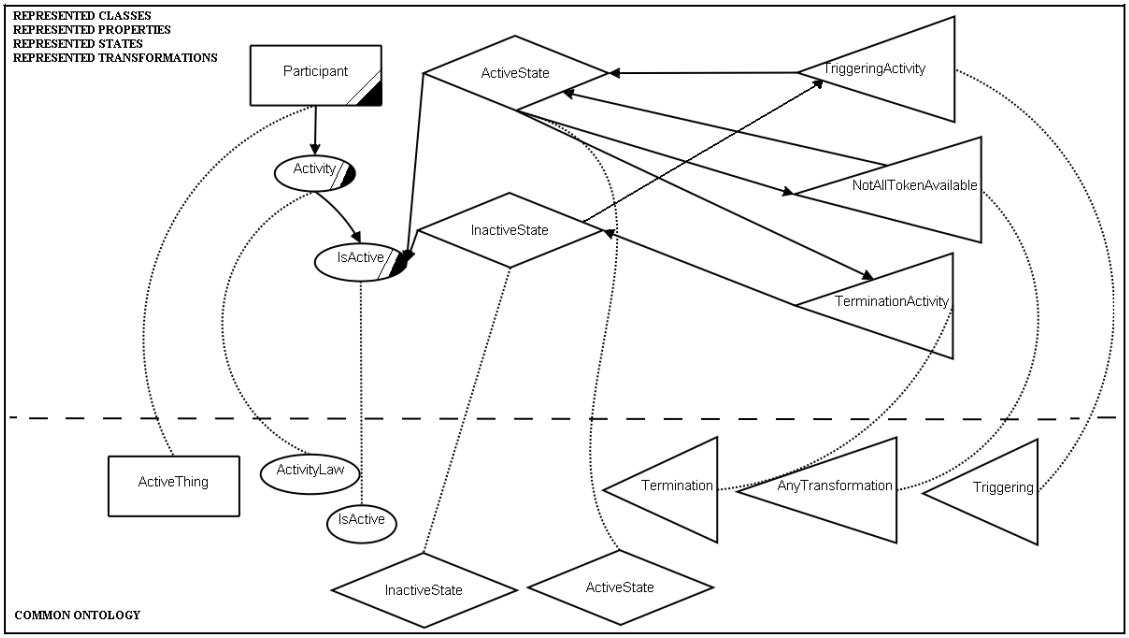


Figure 9.4: Simplified description of the activity - Part 3

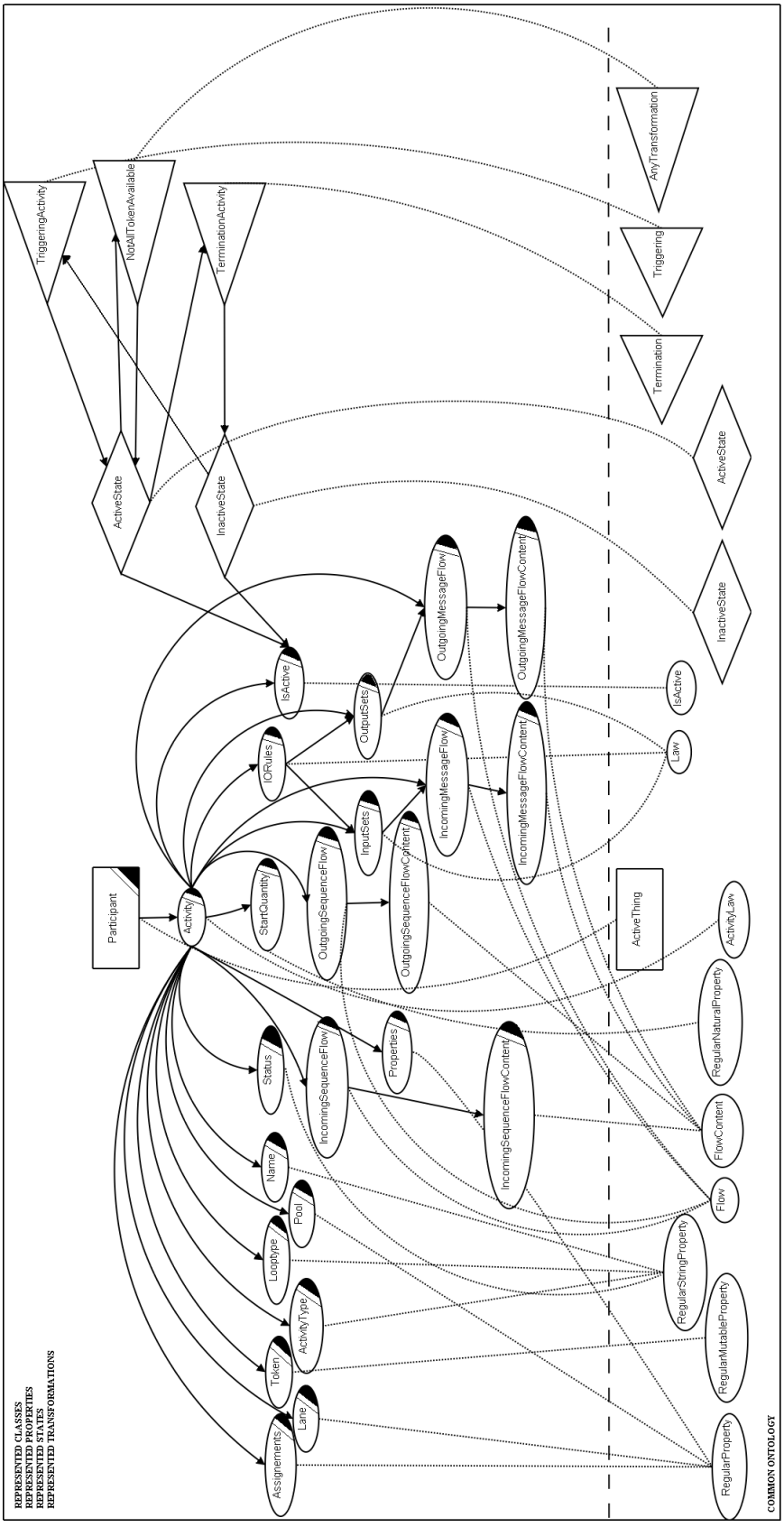


Figure 9.5: Complete description of the activity

9.2 Mapping table

The following Table 9.1 lists the primary mappings of BPMN modelling constructs to the common ontology. The first column indicates the name of the modelling construct. The second one represents the mapping to the common ontology.

Table 9.1: Primary mappings of BPMN modelling constructs

Modelling construct	**Mapping to the common ontology**
Event	*Flow* and *FlowContent*
StartEvent	*Flow* and *FlowContent*
IntermediateEvent	*Flow* and *FlowContent*
EndEvent	*Flow* and *FlowContent*
Activity	*ActivityLaw*
Task	*ActivityLaw*
Process	*TransformationLaw*
Sub-Process	*ActivityLaw*
Gateway	*MutualLaw*
Data-Based Exclusive Gateway	*MutualLaw*
Event-Based Exclusive Gateway	*MutualLaw*
Inclusive Gateway	*MutualLaw*
Complex Gateway	*MutualLaw*
Parallel Gateway	*MutualLaw*
Pool	*ActiveThing* [5]
Lane	*ActiveThing*
SequenceFlow	*Flow*
MessageFlow	*Flow*

We will dicuss each modelling construct and their mappings.

Event

The scene of the event can be seen as three main parts. Firstly, the part with the sequence flow, secondly the part with the message flow and finally the part representing the event's parameters. The event is directly linked to the sequence and message flow because the event brings changes in the flows. An event is then represented by the flow but also the content of the flow (EventParameters). For the sequence flow, there are two classes of things: the target of the flow (TargetSF) and the source of this one (SourceSF). They are respectively mapped onto *InputThing* and *OutputThing*. It's a natural mapping because *InputThing* represents the targets of flows and *OutputThing* represents the sources of flows. Those classes are linked together by the a mutual property, SequenceFlow. This property is mapped onto *Flow* because it is exactly what the sequence flow is intended to represent. The flow in the common ontology is defined as "a binding mutual property through which things flow from an output thing to an input thing" [Com06]. The message flow is totally similar to the sequence flow. We have identified two classes, the source of the message flow (SourceMF) and the target of the message flow (TargetMF). Those classes are linked by a mutual property, MessageFlow. Those classes and this property are mapped onto the same common ontology concepts as the sequence flow and for the same reasons. Finally, the event is a subproperty, called EventParameters, of those two flows because they are directly conditioning the event. This property is mapped onto *FlowContent*. Thus, those parameters represents the characteristics of the event and then the characteristics of what is travelling into the flows.

Start event

The start event is a specialization of the event. The only differences are that a start event cannot be the source of an outgoing message flow and cannot be the target of an incoming sequence flow. Then, we have been obliged to specify the incoming and outgoing part of the flow. The outgoing sequence flow and the incoming message flow are here just concerned. In the scene of the start event, we only have a target of an outgoing sequence flow and the source of an incoming message flow as classes of things. As

for the event, they are respectively mapped onto *InputThing* and *OutputThing*. The rest is exactly the same than the event with the specification of the flows.

Intermediate event

The case of the intermediate event has the same basis. The flows are divided into incoming and outgoing parts as previously. Here no message flows go out of the intermediate event, that is why there is not any class target of the outgoing message flow. For the rest of the scene, the functionning is exactly the same as for the previous events.

End event

For the end event, we identify an outgoing message flow and an incoming sequence flow. Thus, we have identified two classes of things: the target of the outgoing message flow (TargetOMF) and the source of the incoming sequence flow (SourceISF). The functionning is the same as for the previous events.

Activity

The activity is explained in detail in Section 9.1.

Task

The scene of the task is divided into two parts. Firstly, we identify the relation between the process and the task and secondly, we have all the properties of the task. We will just explain the first part. In this part, we have identified two classes of things: the process and the participant. The process is mapped onto *System* because the definition of the process in BPMN: "A process is depicted as a graph of Flow Objects, which is a set of other activities and the controls that sequence them" [OMG06], corresponds to a composite thing whose components are coupled and interacting. This exactly fits *System* ("A composite thing whose components [or parts] are coupled [or interact] [Com06]) in the common ontology. Besides the process, we have another class which is the participant. It is mapped onto *ActiveThing* in the common ontology. The participant is reponsible for the task, it is then acting on it. The best matching was then to map it onto *ActiveThing* because it is defined as "a changing thing that acts at least on one another" ([Com06]). The participant has a property which is the task, "an atomic activity that is included within a Process" [OMG06]. This property is mapped onto *ActivityLaw* in the common ontology. Finally, we identify another property, ProcessLaw, representing the law that makes the process happen. This law is mapped onto *TransformationLaw* because it modifies the properties that the process possesses during the execution. It corresponds exactly to the definition of the *TransformationLaw*. ProcessLaw has as subproperty the task.

Process

The scene of the process is divided into three parts. The main one is composed of the process, the flow objects which compose the process and the relation between these and the process. The process is identified as a class of things and mapped onto *System*. The flow objects are represented by a class of things and simply mapped onto *Component* because they compose the process. The process and the flow objects are linked together by a property representing the composition of the flow objects into the process. This property is mapped onto *PartWholeRelation* in the common ontology because the definition of the *PartWholeRelation*, "a property that relates a composite to one of its components" [Com06], applies to the relation between the process and its components.

Sub-process

The principal part of the scene is exactly the same as the task, replacing the task by the sub-process. The difference between these two constructs is that the sub-process is a set of tasks. The difference is made by some particular attributes of each construct.

Gateway

The scene of the gateway is divided into two parts. The main one is composed of the input and the output of the gateway, the gateway itself and the coupling relation between the input and output through the gateway. The input and the output are represented by two classes of things. They are mapped onto *CoupledThing* because they are both coupled with the gateway. We will now describe the input side, and we don't explain the output side because it has exactly the same functionning than the input side. The input class has a property which represents the gateway. This property is mapped onto *MutualLaw* for the same reasons as for the mapping of the ARIS.And. This class has another property, Ending, which represents the coupling between the input and the output through the gateway. This property is mapped onto *CouplingRelation* to represent this coupling relation between the two classes (Output

and Input). The gateway has a subproperty: IncomingSequenceFlow which is mapped onto *Flow*. The IncomingSequenceFlow has as subproperty the Ending property because the inputs are linked in a sense to the incoming sequence flow of the gateway.

Data-Based Exclusive Gateway

The scene of the Data-Based Exclusive gateway is nearly the same as the most general gateway described here-above. We mean that the construct has the same primary mapping with some little particularities. The difference is that this construct has some other supplementary properties in comparison to the general gateway. This gateway has the XORType, MarkerVisible, Gates, OutgoingSequenceFlow, Assignements, DefaultGate, OutgoingSequenceFlow (of the default gate) and Assignements (of the default gate) attributes.

Event-Based Exclusive Gateway

The scene of the Event-Based Exclusive gateway is nearly the same as the most general gateway described here-above. As for the previous gateway, this construct has a particular attribute: InstantiateFalse.

Inclusive Gateway

The scene of the inclusive gateway is nearly the same as the most general gateway described here-above. In this case, the construct has two other properties: Gates and DefaultGate.

Complex Gateway

The scene of the complex gateway is nearly the same as the most general gateway described here-above. This construct has has two particular attributes: IncomingCondition and OutgoingCondition.

Parallel Gateway

The scene of the parallel gateway is nearly the same as the most general gateway described here-above. It is the construct with least supplementary attributes, it has the Gates, the OutgoingSequenceFlow and Assignements of those gates.

Pool

The scene of the pool is quite simple. The pool is a participant in the process, it can be specific or general. We decided to map it onto *ActiveThing* [6] because as the definition explains "a changing thing that acts on at least one other thing" [Com06], the participant acts on the process. The pool is composed of lane(s). As explained in [OMG06], a lane is a sub-partition within a Pool. We then identified a class of things which represents the lanes which is mapped onto *ActiveThing*. We decided to keep this general mapping because of the use of the lane is not clear. As explained in [OMG06], the meaning of the lanes is up to the modeller. BPMN does not specify the usages of lanes. Lanes are often used for such things as internal roles (e.g. Manager, Associate), systems (e.g. an entreprise application), an internal department (e.g. shipping, finance). As we can see, a lane can often be used for three different things but also for others things which depend of the modeler. To explain the specialization relation between the two, we added a subproperty to those two things. This property is called Lanes and it is mapped onto *PartwholeRelation* (explained above) because it is the closest property to represent the generalisation/specialisation relation. The pool possesses six properties that are not explained here.

Lane

The scene of the lane is composed of two classes and three properties. The two classes are the lane and the pool. Both are mapped onto *ActiveThing* as explained above. These two classes are linked by a mutual property, ParentPool, which identified the parent of the lane. This property is mapped onto *PartWholeRelation* because a lane is a component of a pool. The lane class possesses two other properties but we will not describe these here.

Sequence flow

The main objects of the scene of the sequence flow are the source, the target and the flow itself. The source of the flow is represented by a class of things, the *OutputThing* because *OutputThing* is the class of things that are the sources of flows. Identically the target is mapped onto *InputThing* because it is the class of things representing the targets of flows. Those classes are linked together by a mutual property

[6] In the final version of the template, presented in Appendix I, the pool is mapped onto *Participant*. However, we describe it here like this because this mapping has been modified according to the case study (Refer to Chapter 12).

representing the sequence flow. This property is mapped onto *Flow* in the common ontology because of the definition of *Flow* applies well to the sequence flow.

Message flow

The scene is exactly the same as the sequence flow replacing the SequenceFlow property by the message flow. There are two classes of things, the source and the target of the message flow, and the property representing the message flow.

9.3 Summary

In BPMN, a lot of elements and elements' attributes are described and used. This variety poses some difficulties for us to realize the mappings because sometines we are lost in all this information. The definition of some elements can be ambiguous. For instance, the concept of lane is ambiguous because it can be used for representing several things and there is no precise definition, thus it is difficult to understand what it can represent to be able to map it correctly. The language is good to describe precisely and in detail a process and the several scenarios through the specialization of the events, gateways and tasks. In that sense, a process can be described at a high level and then, subprocesses can be used and detailed at a low level.

This chapter is composed of two main parts. The first one we explained in detail the analysis of the *activity* construct. In this explanation we also described the whole template and how we filled it in. The second part is composed of the mapping table of BPMN. This table represents the primary mapping of every constructs. In this part we have also given a brief discussion about the mappings of every constructs.

This analysis was validated with the UEML Validator, it will be the object of the Chapter 11. The mappings of the analysis will be verified in the Chapter 12 with a case study. And finally a comparison between ARIS and BPMN will be made in Chapter 13.

Chapter 10

Adding to the common ontology

During our two analyses, we needed to add some new classes of things and properties to the common ontology. These additions are 27 on the whole. They all concern the classes and properties, we didn't add any states or transformations during our work.

First, we will list all these additions with a brief description. Then, we will explain in detail one class of things and one property.

10.1 Extension of the common ontology

Thirteen classes of things and fourteen properties were added to the common ontology. It allows the common ontology to grow and to suggest more and more concepts. In this way the common ontology becomes more specific and allows representing the reality with more precision.

Classes of things

- **Participant**[1]**:** An active thing that is responsible for another thing.
- **RoleHolder:** An active thing that represents the role performed by individual people.
- **HumanOutput:** An active thing that is a human-being which is implied in the creation of a thing.
- **Equipment:** An active thing that equips another thing.
- **MachineResource:** An equipment thing that is a device for doing a work.
- **ComputerHardware:** An equipment thing that is a computer hardware resource.
- **ReactiveThing:** An interacting thing that doesn't possess any TransformationLaw.
- **ProActiveThing:** An interacting thing that possesses TransformationLaw [2].
- **Service:** An associated thing that is a service.
- **InformationService:** An associated thing that is an information service.
- **MaterialService:** An associated thing that is another service (not an information service).
- **Repository:** A container that contains informational or material ressources.
- **MaterialRepository:** A container that contains material resources.

Figure 10.1 shows the hierarchy of the classes of things of the common ontology, the existed and added classes of things. The classes in grey are the classes we have added.

[1] *Participant* was created after the achievement of the case study. Refer to Chapter 12 for more information.
[2] The class *ProActiveThing* was not used in our analyses but was added as the opposite of *ReactiveThing*.

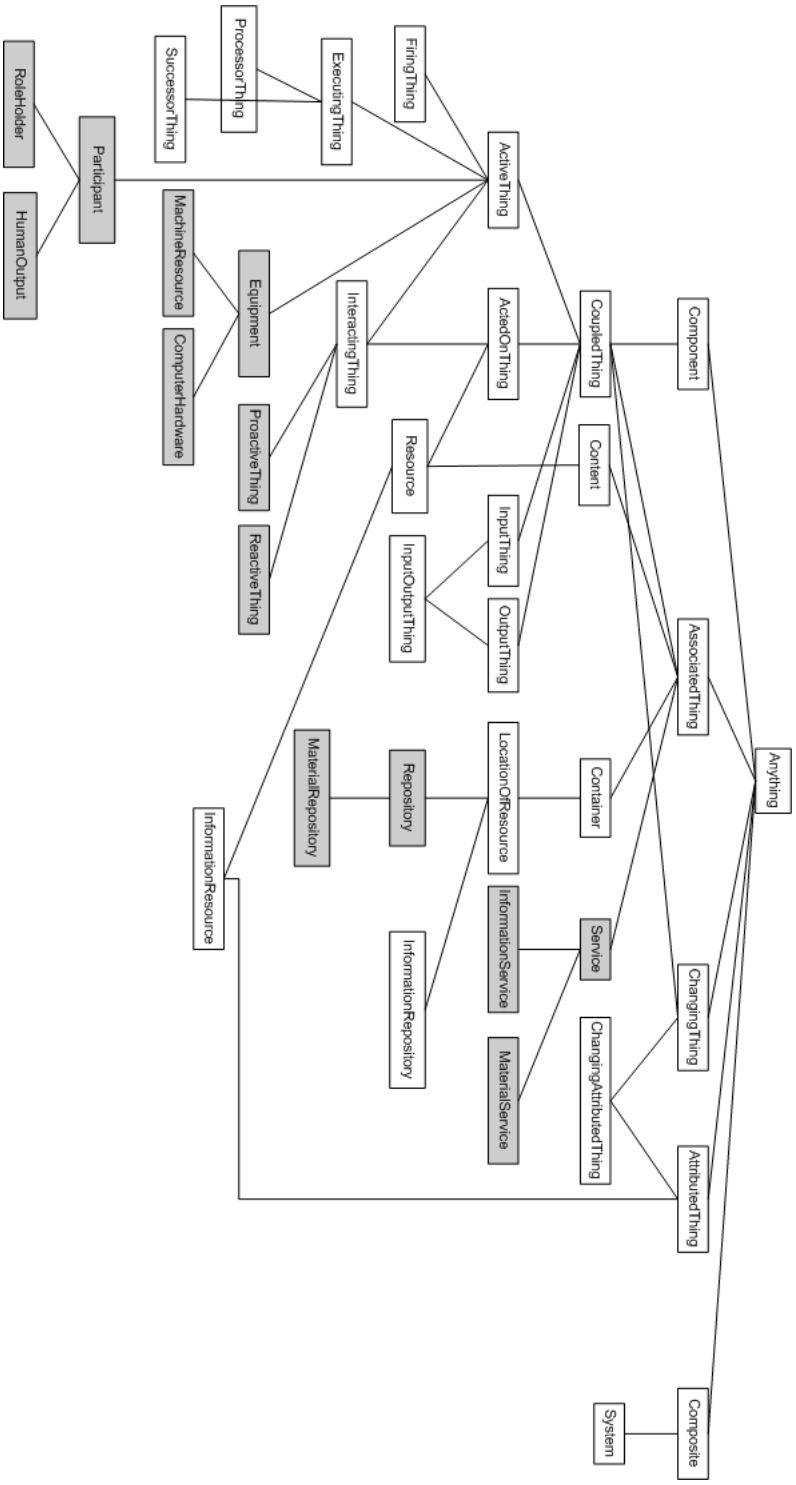


Figure 10.1: Hierarchy of the classes of things of the common ontology

Properties

- **FunctionLaw:** A transformation law that manipulates the resources in a repository and produces appropriate outputs in response to the inputs it receives.
- **ActivityLaw:** A transformation law that produces appropriate outputs in response to the inputs it receives.
- **ApplicationLaw:** A tranformation law that marks the part of a thing executed by a (executing) thing.
- **ParticipationLaw:** A transformation law that marks the participation of a thing in the creation of another thing.
- **UseLaw:** A transformation law that marks the part of a thing used by another (active) thing.
- **MutualLaw:** A transformation law that is mutual.
- **Responsability:** A law that shows the responsibility of a thing.
- **Realization:** A law that shows the realization of the work by a thing.
- **Name:** A regular property that is the name of the thing.
- **Location:** A regular property that is the location of the thing.
- **RegularMutualProperty:** A property that is not a law, not a part-whole relation and not a class-subclass relationship.
- **FlowContent:** A regular or a mutual property that marks the content of a flow.
- **MutualFlowContent:** A mutual property that marks the content of a flow.
- **RegularFlowContent:** A regular property that marks the content of a flow.

Figures 10.2 and 10.3 show the hierarchy of the properties of the common ontology, the existed and added properties. The properties in grey are the ones we have added.

10.2 Explanation of *RoleHolder* and *FunctionLaw*

We will explain the class of things *RoleHolder* and the property *FunctionLaw*.

RoleHolder

The definition of *RoleHolder* is the following: "An active thing that represents the role performed by individual people". In Figure 10.1, this class is one of the subclasses of *Participant*.

We have decided to add this class to the common ontology because no present class was enough precise to represent the ARIS.Position. Indeed, the construct ARIS.OrganizationalUnit is mapped onto *Participant*[3]. Thus, it is an active thing that is responsible for another thing. An organizational unit may be composed of positions (e.g. composed of some roles like "Custommer service advisor" or "Department manager"). It implies that a position is also an active thing because a position can be responsible for a function. But it is more precise than *Participant* because it represents a role. It is why we have decided to create a new class *RoleHolder* which underlines the role performed by an indivudual thing.

[3]See Chapter 12 for more details

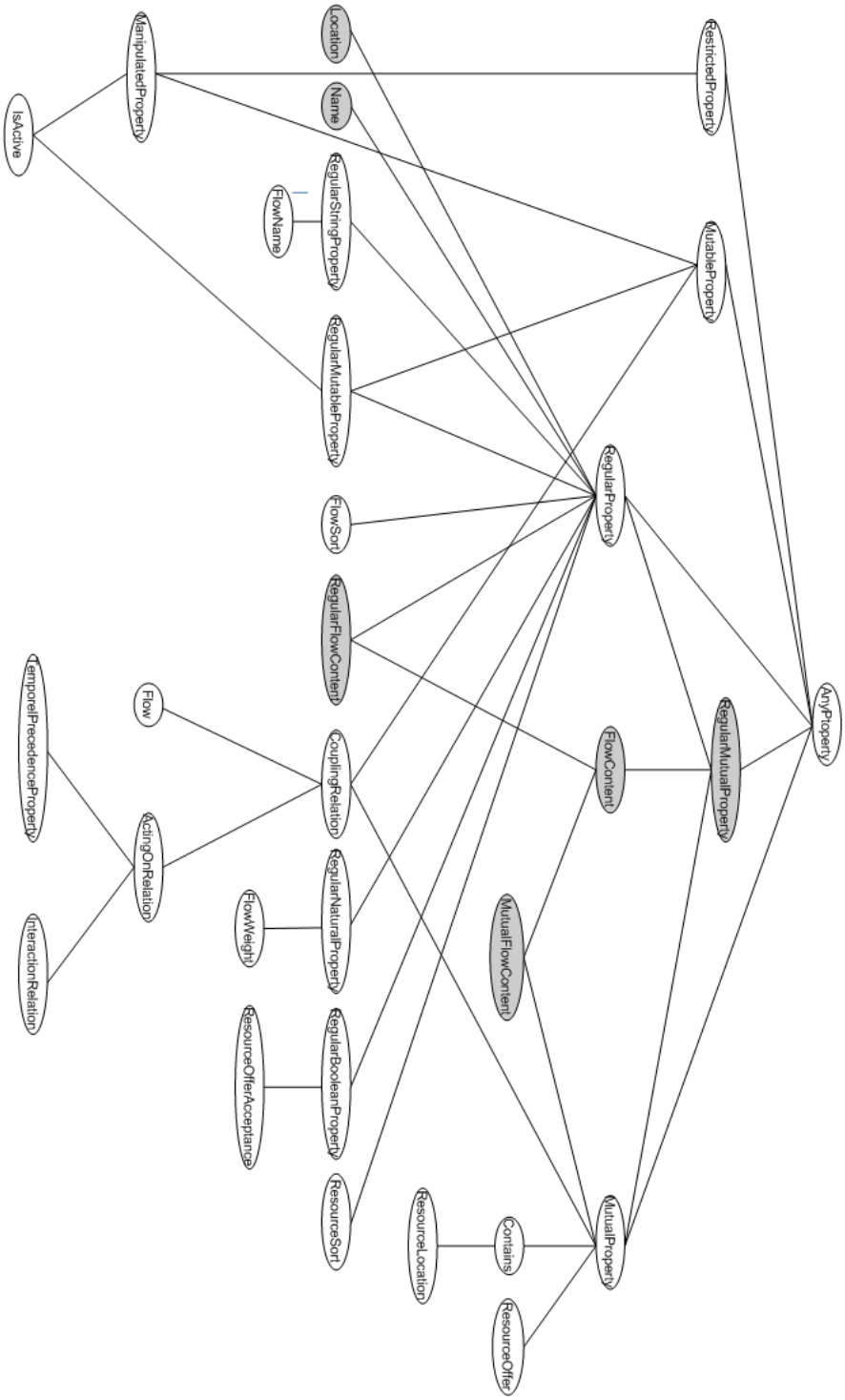


Figure 10.2: Hierarchy of the properties of the common ontology - Part 1

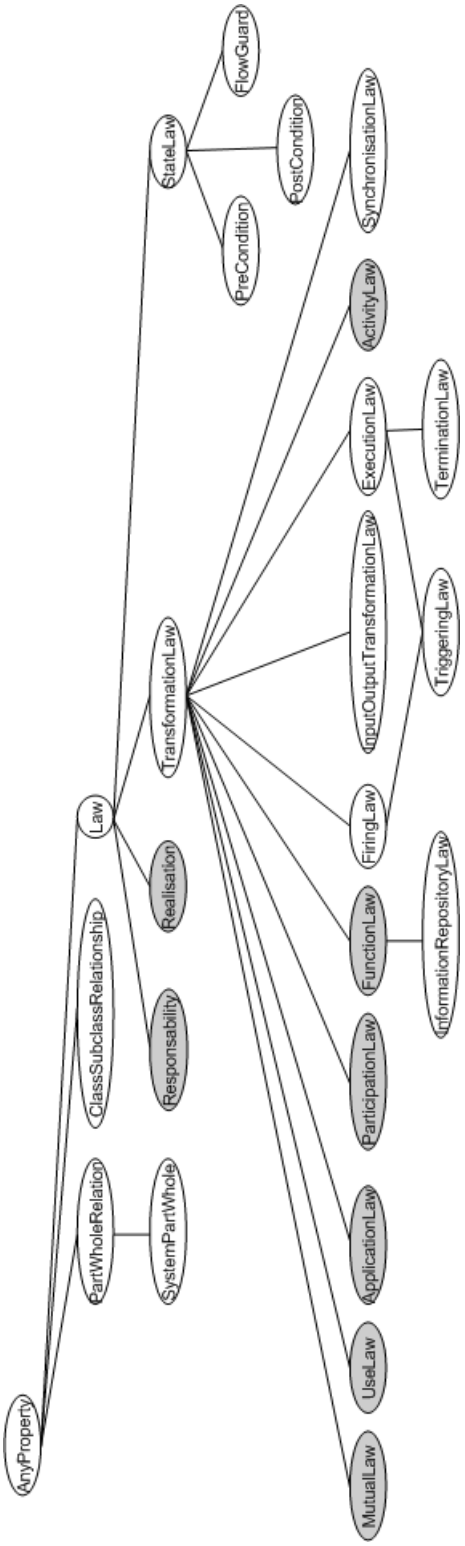


Figure 10.3: Hierarchy of the properties of the common ontology - Part 2

FunctionLaw

The *FunctionLaw* is a transformation law that manipulates the resources in a repository and produces appropriate outputs in response to the inputs it receives. In Figure 10.3, the *FunctionLaw* is preceded by the *TransformationLaw* and precedes the *InformationRepositoryLaw*.

We have decided to create the *FunctionLaw* property because the ARIS.Function doesn't map exactly onto *TransformationLaw*. The *TransformationLaw* is defined as "a law that restricts the combinations of properties that an (active) thing can possess before and after an event. (An event is an occurrence of a transformation.) Like any laws (that are currenly accounted for in the UEML ontology), a transformation law is not mutual, not a part-whole relation and not a class-subclass relationship" [Com06]. Furthermore, functions are applied to objects, e.g. inputs and outputs, then they are a little bit more than something that restricts the combinations of properties of an active thing. In ARIS, the function defines an activity or a task to be executed in a process. Accordingly, they are defined as operations applied to objects for the purpose of supporting one or more goals. That is why we added the *FunctionLaw* in the common ontology, to underline the fact that this *TransformationLaw* in the sense of the ontology manipulates the resources to produce the outputs corresponding to the inputs.

10.3 Summary

In this chapter, we presented the classes of things and properties we had added to the common ontology and the hierarchy of the common ontology with these extensions. To finish, we explained one class of things, *RoleHolder*, and one property, *FunctionLaw*.

Part III

Validation / Evaluation

Chapter 11

UEML Validator

In this chapter, we will briefly explain the use of the UEML Validator. We will show the results of the application of the UEML Validator on the UEMLBase and describe with examples how we managed to fix some errors.

11.1 Use of the UEML Validator

After having entered our two anlyses in Protégé, we used the UEML Validator to check the consistency of the new UEMLBase. In fact, we have used the UEML Validator to validate our analyses. This application provides a list of the mistakes which can be found in the UEMLBase. In all these errors, we identified those linked to our analyses and put them into another file. This file can be found in Appendix J. In the results, we underlined three kinds of mistakes: the correctable mistakes, the bad rule mistakes and the generated mistakes. The **correctable errors** consist of errors we can correct. For instance, it can be one or many non filled-in entries, a problem of superfluous relations recognizable by transitivity, a problem of possible duplication of classes or a problem of precedence of the properties and their relation to the class. The **bad rule errors** consist of errors that come from the rule. It means that the rule is not correct. The **generated mistakes** concern the *RepresentedPhenomenon* we can find in the UEMLBase. It consists of errors that come from automatic generated things. In the two last cases, we submitted to Andreas L. Opdahl the possibility of the existence of a mistake in this rule or in the generation of things.

11.2 Consequences

We will now categorize the mistakes the UEML Validator has identified, show an example for each of them, and how we fixed it. As we said, we had three different kinds of mistakes: the correctable mistakes, the bad rule mistakes and the generated mistakes. In the correctable mistakes, we still have different kinds of errors. There are different examples of all the previous categories of errors:

- **Bad rule mistakes:**
  Error: *No generalisation relationship from ontology subclass Anything to superclass ProActiveThing.*
  Explanation: This sentence means that there is a generalisation/specialization relationship between *Anything* and *ProActiveThing* missing. The class *Anything* should be a subclass of *ProActiveThing.*
  Solution: It is impossible that the highest class of the common ontology *Anything* is considered as a subclass of another one, in this case *ProActiveThing.* Thus, this is a mistake in one of the rules. In this case, we have noticed Andreas L. Opdahl that the rule which gives this error can be incorrect.

- **Generated mistakes:**
  Error: *Represented phenomenon _ARIS_ApplicationSoftwareClassRole_Software_ARIS_ ApplicationSoftwarePropertyRole_Rule_RepresentedClassPropertyRelation does not describe any constructs.*
  Explanation: This sentence means that the entry *describedConstruct* of the represented phenomenon is empty.

Solution: The represented phenomenon is an automatic generated thing. Thus, this is a mistake in the generation of these represented phenomenon. In this case, we noticed Andreas L. Opdahl of this mistake.

In the correctable mistakes, we identify different kinds of errors:

- **Non-filled entry:**
  Error: *Language ARIS has no name and/or version.*
  Explanation: It means that the name/version entry is not filled for the ARIS language. This is the simplest error we had.
  Solution: The solution is to fill in the corresponding entry. But in the case of this example, it is impossible to mention the version simply because it is not indicated in the source books ([Sch98] and [Sch99]). Thus, we have let this entry empty.

- **Duplication of classes:**
  Error: *Ontology classes Component and OrganizationalUnit possess exactly the same ontology properties.* This error is shown by Figure 11.1 [1].

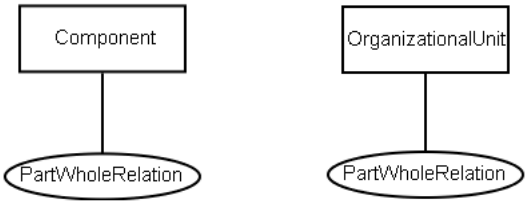


Figure 11.1: Error of duplication of classes

Explanation: This error indicates that the classes *Component* and *OrganizationalUnit* represent exactly the same thing because they possess the same properties.
Solution: Actually it is not the case, it is not what we wanted to represent. Thus, we have added the property *FunctionLaw* for the *OrganizationalUnit*. The solution of the error is to add some particular properties to the classes and is shown by Figure 11.2 [2].

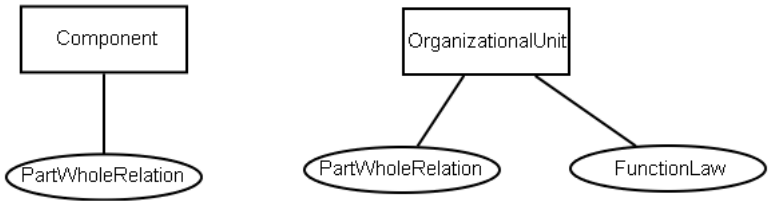


Figure 11.2: Solution of the error - Duplication of classes

- **Duplication of represented phenomenon:**
  Error:*Represented phenomenon DUPLICATE_1_OF__ARIS_ApplicationSoftwareClassRole_Software_ARIS_ApplicationSoftwarePropertyRole_Rule_RepresentedClassPropertyRelation does not describe any constructs.*
  Explanation: This indicates that there is a duplication of a *RepresentedClassPropertyRelation*
  Solution: We have deleted all these duplications.

[1] The legend of Figure 11.1 is shown in Appendix F.
[2] The legend of Figure 11.2 is shown in Appendix F.

- **Proprety belongs to any class:**
  Error: *Construct ARIS_ Event describes represented state ARIS-EventStateRole_ PostState but not any represented class whose properties define ARIS-EventStateRole_ PostState.*
  Explanation: This indicates that there is no class of things in the environment of the construct *Event*.
  Solution: Every property should belong to a class or be a subproperty of another one. Thus, we have added a class of things in the scene of the *Event* which was in this case *OrganizationalUnit*.

- **Recognizable superfluous relation (by transitivity):**
  Error: *Ontology property InformationRepositoryLaw precedes FunctionLaw, which in turn precedes TransformationLaw, so there is no need for a precedence relationship between the first and last.* The error is shown by Figure 11.3.

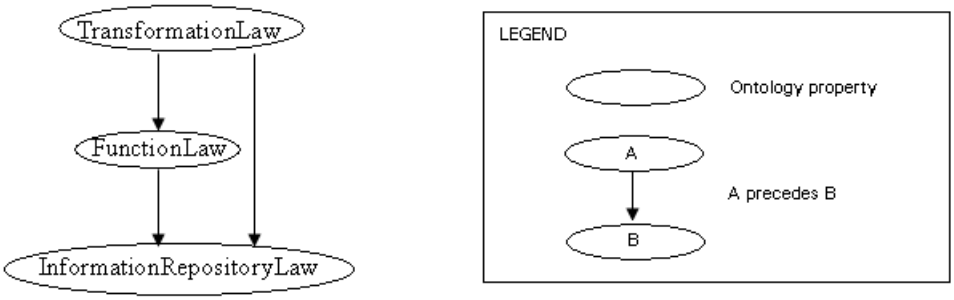


Figure 11.3: Error of superfluous relation

Explanation: It means that there is a transitive relationship between two properties: *InformationRepositoryLaw* and *TransformationLaw*. This is typically a case of possible recognition by transitivity.
Solution: The relation between *InformationRepositoryLaw* and *TransformationLaw* is not needed. Thus, we have deleted this relationship. The solution of the error is shown by Figure 11.4.

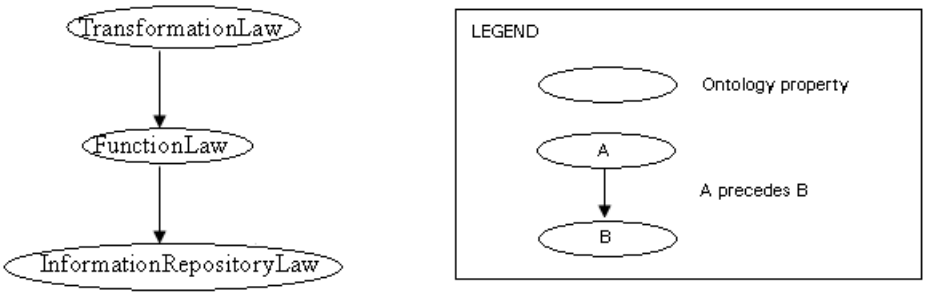


Figure 11.4: Solution of the error - Superfluous relation

- **Precedence of the properties and their relation to the class:**
  Error: *Ontology class CoupledThing possesses property MutualLaw and also its precedent TransformationLaw.*
  Explanation: This indicates that the class *CoupledThing* possesses two properties whose one (*TransformationLaw*) precedes another (*MutualLaw*). Then, the preceded property must be possessed by a subclass of the class that possesses the most general property.
  Solution: We changed the *MutualLaw* in the common ontology. It is now preceded by the property

CouplingRelation. Then, the *MutualLaw* and the *TransformationLaw* can be possessed by the class *CoupledThing*.

11.3 Summary

In this chapter, we explained how we used the UEML Validator and which were the kinds of identified mistakes. We also explained the corrections we made in relation with the different kinds of mistakes. The application of the UEML Validator allows the validation and the improvement of the analyses by underlining and correcting erros in the UEMLBase.

Chapter 12

Case study

During the analysis of two languages, we have realized that these analyses imply a part of subjectivity. To decrease this subjectivity and to validate our analyses, we have carried out a case study. For this, we have modelled the process of conference organization in ARIS and BPMN. The main objective of this modelling was to verify that the mapping between ARIS and the common ontology and between BPMN and the common ontology were coherent.

In this chapter, we will, first, describe the process of conference organization. Then, we will present and explain our ARIS and BPMN models. On the basis of these, we will verify if the mappings are coherent.

12.1 Description of the process of conference organization

On the basis of [IEE], [con] and [DMBF05], we will explain briefly the process of conference organization. This process can be seen as a outline of major steps/tasks. These steps are identified as follows:

1. **Selection of the contributions:** One of the most important task in the organization of a conference is the selection of the contributions. This selection is done under the responsibility of the Program Committee Chair (PCC). The PCC is helped in his task by the members of the Program Committee (PC) also named reviewers.

2. **Review process:**

   - *Submission phase:* The review process on the contributions submitted by authors to a conference starts after the deadline for the paper submission phase.
   - *Selection of reviewers:* When the submission phase ends, suitable members of the PC are selected, which will act as reviewers, in order to evaluate the submitted papers.
   - *Collected submissions and review forms sent to the reviewers:* The PCC sends the collected submissions with review forms to individual reviewers. The review form consists of a set of questions to assess the quality of the paper, that the reviewers must fill in and return it to the PCC. Each submission is typically examined and evaluated by two or three reviewers.

3. **Program Committee meeting:** The review process ends with the Program Committee meeting, where the papers are discussed on the basis of collected review forms, in order to their acceptance or rejection for presentation at the conference. The PC meeting starts at the envisaged date even if all the review forms are not collected.

4. **Reviewer's comments sent to the authors:** After this meeting, anonymous extracts of the review forms (reviewer's comments) are typically sent back to all the authors, so that they can improve their paper, regardless they were accepted or not.

5. **Submission of a new version of the author's paper:** The authors of accepted papers may submit a new version of their paper, in the so-called camera-ready format, to the PCC.

6. **Proceedings printed:** PCC will send the new version of the author's paper, together with the preface and the table of contents of the book, to the publisher in order to have the proceedings printed.

We have added some elements to be closer to the reality. For instance, when the author sends his paper after the deadline, he is notified that his paper is rejected. There is also the case when the reviewer forgets to fill in the review forms and to send them back to the PCC, some recalls are sent to the reviewer. It is the same case when the author forgets to submit his final version, but we have decided to omit it into the model because we already represented these constructions.

We have identified the main concepts and parts of the process of conference's organization. The main concepts are:

- **Participants:**
  - *PCC:* Program Committee Chair.
  - *PC:* Program Committee.
  - *Reviewers:* They are members of the PC.
  - *Authors:* They submit their papers.
- **Documents:**
  - *Paper:* The paper submitted by the author is also called contribution.
  - *Review form:* The review form consists of a set of questions to assess the quality of the paper.
  - *Reviewer's comments:* The reviewer's comments are anonymous extracts of the review forms.
  - *Proceedings:* The PCC writes the preface and the table of contents to create the proceedings.

The main parts are:

- **Review process:** It consists of the submission phase, the selection of the reviewers and the sending of the collected submissions and review forms to the reviewers.
- **Recall process:** It consists in recalling the reviewers who have not filled the review form or sent it.
- **The PC meeting:** In the PC meeting, the papers are discussed on the basis of collected review forms, in the order of their acceptance or rejection for presentation at the conference.
- **Improvement process:** On the basis of the reviewer's comments, the authors can improve his paper and send a new version.
- **Printing of the proceeding process:** It consits in writing the preface and the table of contents to create the proceedings and in sending them to the publisher in order to print the proceedings.

12.2 Description of the research question

The goal of this case study is to validate the analyses of ARIS and BPMN. In that sense, the objective of this modelling was to verify that the mapping between ARIS and the common ontology and between BPMN and the common ontology are coherent. The term coherent implies that for two constructs of different langages representing the same thing in the world, are mapped onto the same concepts of the common ontology.

This case study allows more objective judgement of the mappings and the choices made. It permits to revise our analyses and to underline some important and delicate points.

12.3 ARIS model

The modelling of the process of conference organization in ARIS is shown by Figure 12.6. This model is divided into five parts shown by Figures 12.1, 12.2, 12.3, 12.4 and 12.5 [1].

Review process

The first part consists of the review process shown by Figure 12.1.

This process begins by the submission phase during which the authors can submit their paper. This phase begins by a *call for papers*. This event will trigger the function *Submit paper* which represents the submission of the papers by the authors. The result of this first function is the information service defined by the *paper*. The goal *Deadlines kept* controls this function. All the papers have to be submitted before a given date. If the paper is submitted after this date, the author is notified that his paper is rejected. It is represented by the function *Notify the author that the paper is rejected* which is under the responsibility of the PCC. If the paper is submitted before this date, the function *Select reviewers* is realized. These two paths are represented by the logical operator "Or".

The function *Select reviewers* is controlled by the goal *High matching*, i.e. to try to obtain the high matching between the papers and the reviewer's skills. This function uses a computer hardware *PC* and an application software *Reviewers selection system* to achieve this goal. The environmental data *Reviewers assignation list* represents the link between the reviewer and several papers. This list is modified by the function. This function creates the event *Reviewers selected*.

After the selection of the reviewers, the PCC is responsible for the sending of the papers and the review forms to the reviewers. It is represented by the function *Send papers and review forms to the reviewers*. This function has two inputs: *Review forms* and *Papers*. The environmental data *Reviewers assignation list* is used to know which papers are assigned to the reviewers. This function creates the event *Papers and review forms arrived* which represents the fact that the reviewer receives the papers he has to evaluate and the review forms he has to fill in.

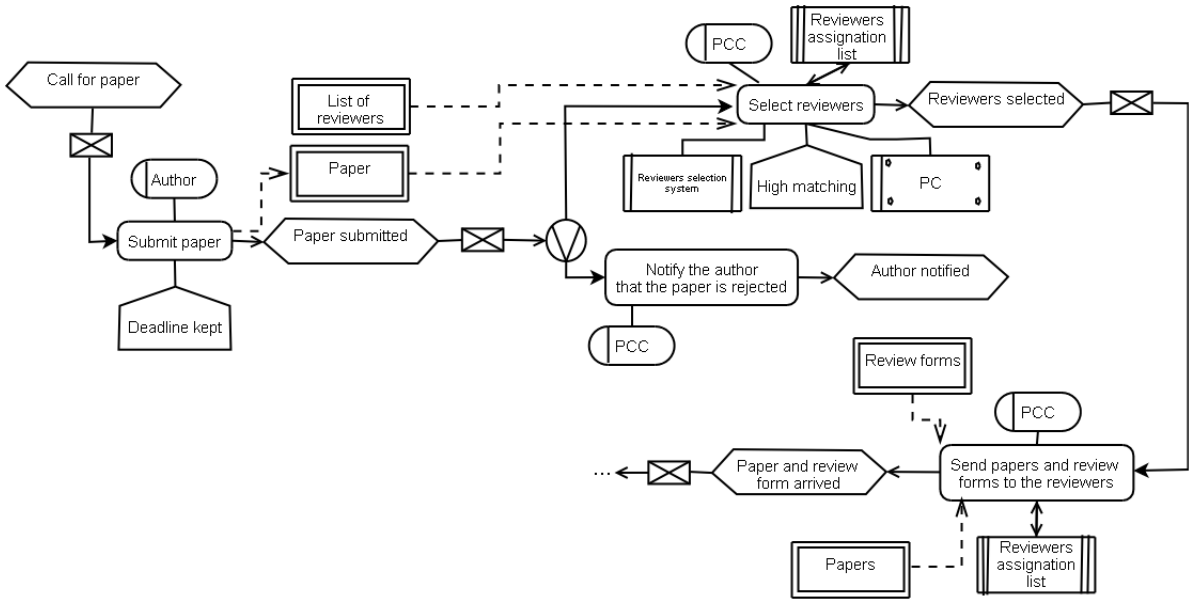


Figure 12.1: ARIS model - Part 1

[1] The legend of Figures 12.6, 12.1, 12.2, 12.3, 12.4 and 12.5 is depicted in Section 5.4 by Figure 5.3.

Recall process

The second part consists of the recall process shown by Figure 12.2.

After the reviewers receive the papers and review forms, they have to fill in the review forms, *Fill in the review form* and send them back to the PCC, *Send review form to PCC*. If they don't send them before the deadline or don't fill in them, the PCC sends a recall to the reviewer with the function *Send recall to the reviewer*. This function is triggered by the event *Dealine missed*. After this function, two cases can occur. It is shown by the logical operator "Or". Either, the reviewer reads the recall, represented by the event *Recall read*, and decides to fill in the review form or to send them directly. This choice is represented by another logical operator "Or". Or whether, the recall is not readed and the review forms not sent to the PCC. It is represented by the event *Recall not read* and it is the end of the process.

The function *Fill in the review form* has two inputs: *Review forms* and *Papers*, and creates an output which is *Review forms filled*.

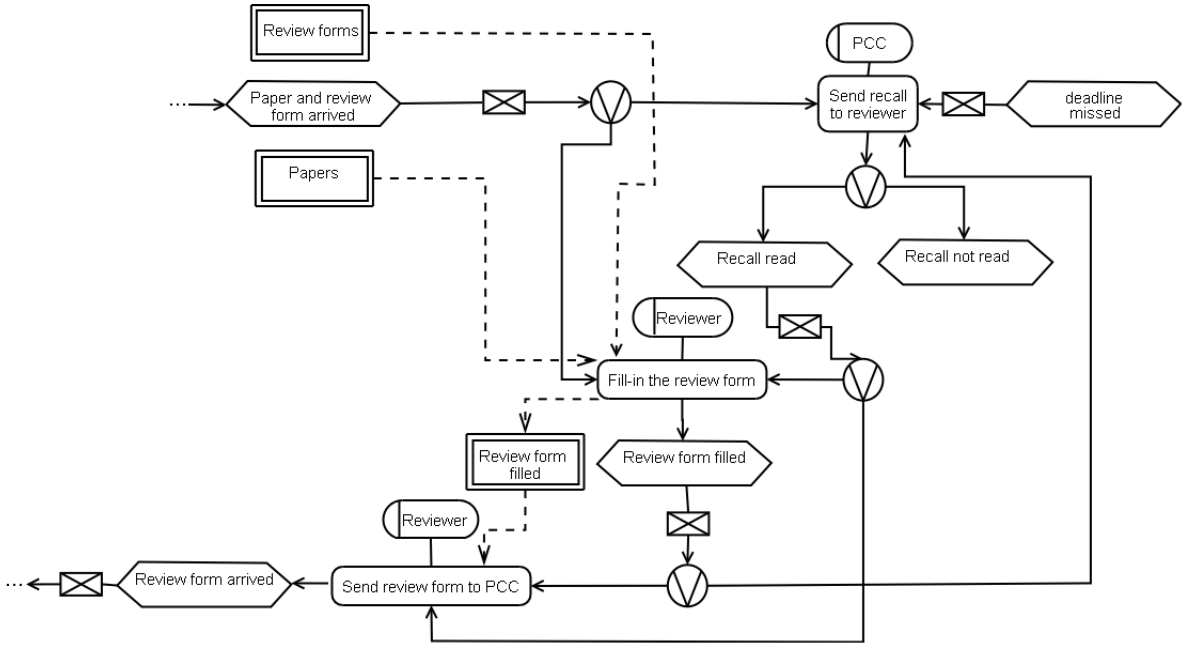


Figure 12.2: ARIS model - Part 2

The PC meeting

The third part consists of the PC meeting shown by Figure 12.3.

The PCC has to collect the review forms by the means of the function *Collect review forms*. This function creates the event *Review forms collected*. The PC has to attend the PC meeting which is trigged by the events *Review forms collected* and *Meeting's date arrived*. These two events are linked by the logical operator "And". It means that if the date of the meeting arrived, the PC meeting starts even if all the review forms are not collected. The function *Attend the PC meeting* creates an output which is the *Reviewer's comments*.

When the PC meeting is ended, the decision of acceptance or rejection of each paper has to be taken. This phase is represented by two events, *Paper accepted* and *Paper rejected*. For each paper, it is accepted or rejected. This is shown by the logical operator "Or". If the paper is rejected, it is the end of the process. The author is notified of the decision by the means of the function *Notify the author of acceptance or rejection of the paper* which is under the responsibility of the PCC. In parallel, the PCC sends the reviewer's comments to the author, *Send reviewer's comments to the author*.

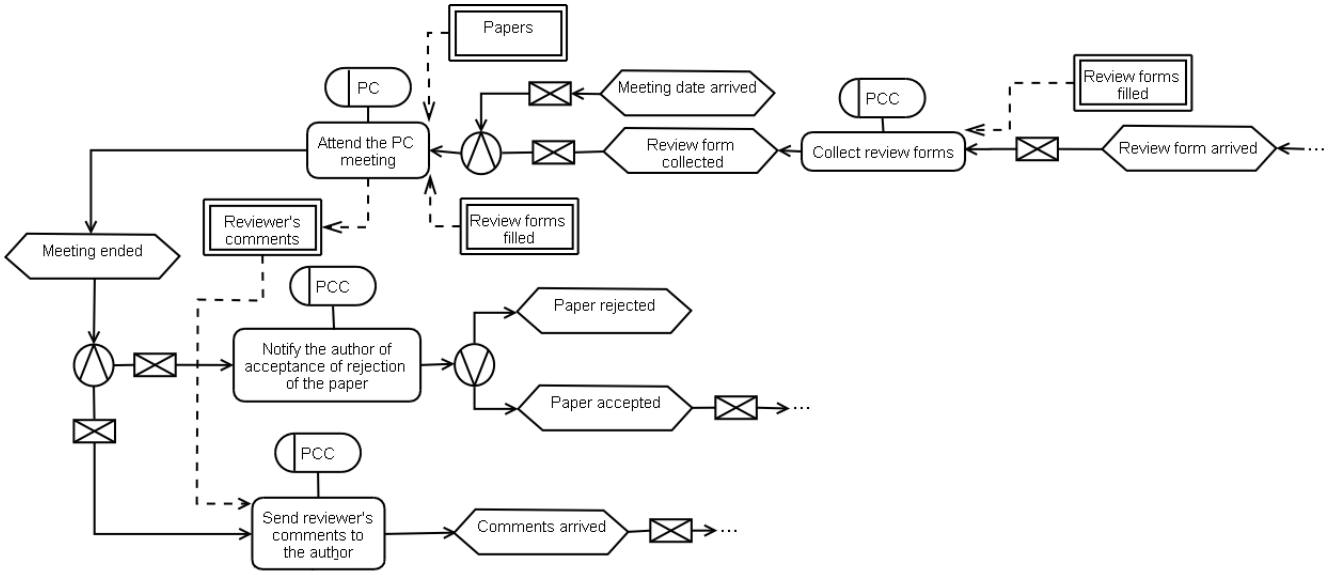


Figure 12.3: ARIS model - Part 3

Improvement process

The fourth part consists of the improvement process shown by Figure 12.4.

The function *Improve paper*, under the responsibility of the author, is achieved when the two events, *Paper accepted* and *Comments arrived*, occur. This fact is represented by a logical operator "And". This function needs two inputs: *Paper* which represents the paper the author has submitted and *Reviewer's comments*. It also creates one output which is *Final version* which represents the last version of the author's paper. After the paper has been improved, represented by the event *Paper improved*, the author submits the final version by the means of the function *Submit final version*. We prefer to keep two different functions to be clear and to follow the explanations given in the three articles used.

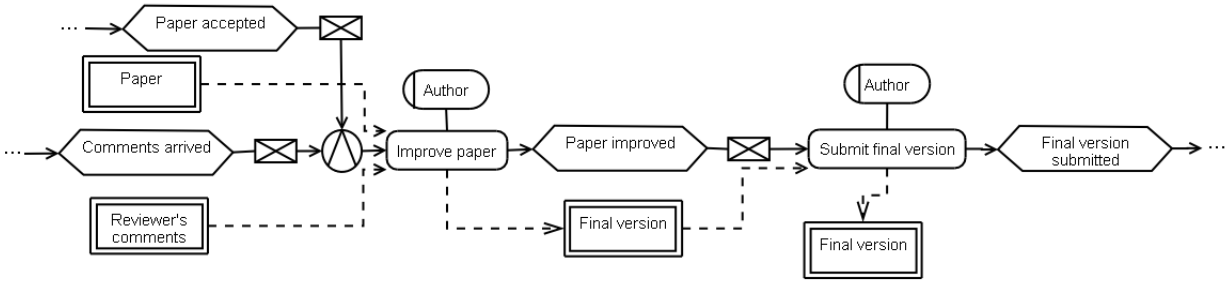


Figure 12.4: ARIS model - Part 4

Printing of the proceedings process

The last part consists of the printing of the proceedings process shown by Figure 12.5.

To achieve this process, the PCC has to collect the final versions of the paper, *Collect final versions*. This function has one input: *Final version*. Then, when the event *Final version collected* occurs, the PCC writes the preface and the table of contents to create the proceedings which will be printed, by the means of the function *Write the preface and the table of contents*. This function creates an output which represents the proceedings. Next, the PCC sends the proceedings to the publisher, *Send proceedings to publisher*. When the publisher receives the proceedings, *Proceedings received*, he will verify the proceedings, *Verify the proceedings*. He will transmit the whole of the proceedings to the printer by means of the function *Send proceedings to printer*. When the printer receives the proceedings, he will print the proceedings, *Print proceedings*. The output, *Proceedings*, is used as input of these tasks.

The all process of conference organization is finished when the proceedings are printed, with the event *Proceedings printed*.

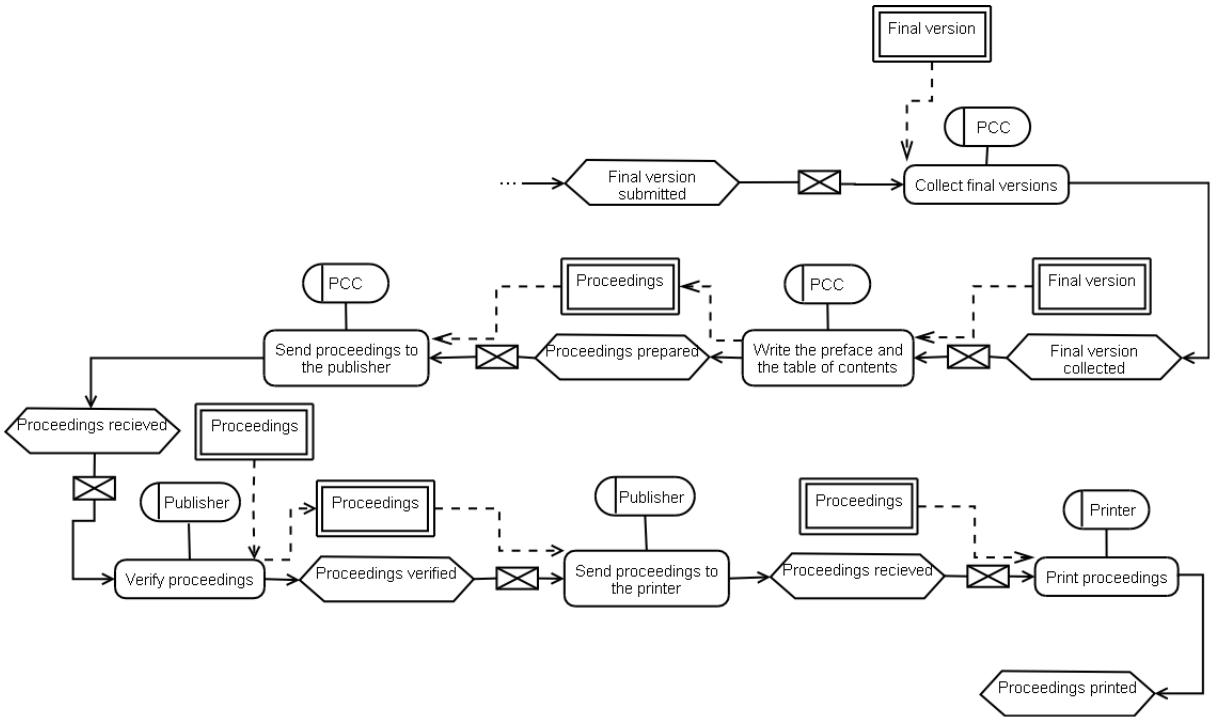


Figure 12.5: ARIS model - Part 5

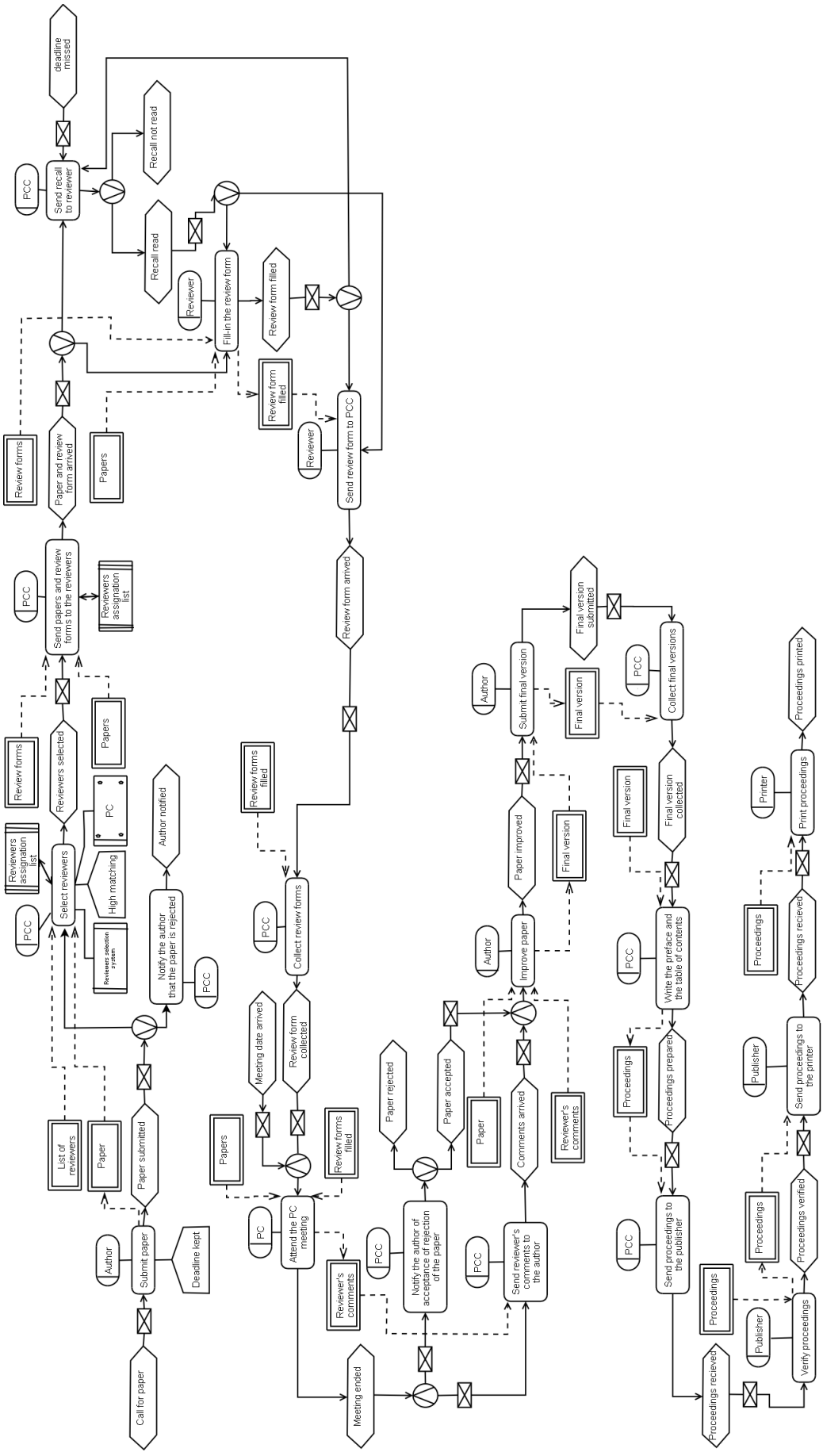


Figure 12.6: ARIS model

12.4 BPMN model

The modelling of the process of conference organization in BPMN is shown by Figure 12.12. This model is divided into five parts shown by Figures 12.7, 12.8, 12.9, 12.10 and 12.11 [2].

To explain the BPMN model, we will use the concept of Token that will traverse the sequence flow and pass through the flow objects in the process. The behaviour of the process can be described by tracking the path(s) of the Token through the process [OMG06].

Review process

The first part consists of the review process shown by Figure 12.7.

The process begins with the start event *Call for paper* which will generate a Token. This Token will arrive to the task *Submit paper*. This task is in the pool *Author* which is a participant in the process. A message flow links this task to an intermediate event message located in the lane *PCC* of the pool *PC*. This kind of intermediate event has the following definition: "A message arrives from a participant and triggers the Event. This causes the Process to continue if it was waiting for the message." [OMG06]. A Token does not traverse the message flow since it is a message that is passed down those flows (as the name implies). Thus, after this intermediate event occurs, a Token is generated and passes through the OR gateway *Deadlines keept?*. If the dealine is over, the PCC notifies the author of the rejection of his paper by the task *Notify the author that his paper is rejected* and the Token is passed to an end event. This end event consumes the Token and indicates the end of the process. If the answer is true, the tasks *Select reviewers* and then *Send papers and review forms to the reviewers* are achieved by the *PCC*. The Token will pass through these two tasks.

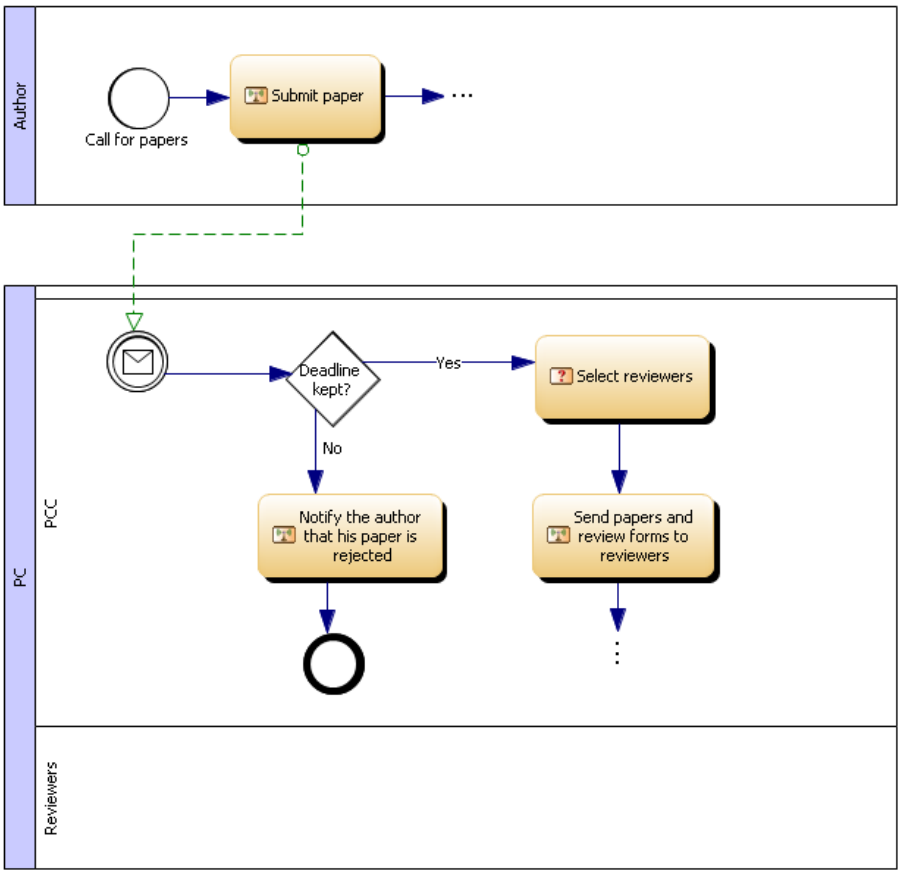


Figure 12.7: BPMN model - Part 1

[2] The legend of Figures 12.12, 12.7, 12.8, 12.9, 12.10 and 12.11 is depicted in Appendix G.

Recall process

The second part consists of the recall process shown by Figure 12.8.

After the reviewers receive the papers and review forms, they have to fill in them by the way of the task *Fill in review forms*. But they can forget to do it. This fact is shown by a an inclusive decision using an OR gateway. Either, they fill in the review forms, or an intermediate event *Deadline missed* occurs. This intermediate event will trigger the task *Send a recall to the reviewer* achieved by the *PCC*. After the task *Fill in review forms*, there is another OR gateway which represents the fact that the reviewer can send the filled in review forms to the PCC or not. If not, the Token is passed to the intermediate event *Deadline missed*. When the reviewer receives the recall (if he had to receive one), there is a third inclusive decision using an OR gateway. There are three alternatives: the reviewer fills in the review forms, the reviewer sends the filled in review forms or the reviewer does nothing shown by an end event. This end event consumes the Token and indicates the end of the all process.

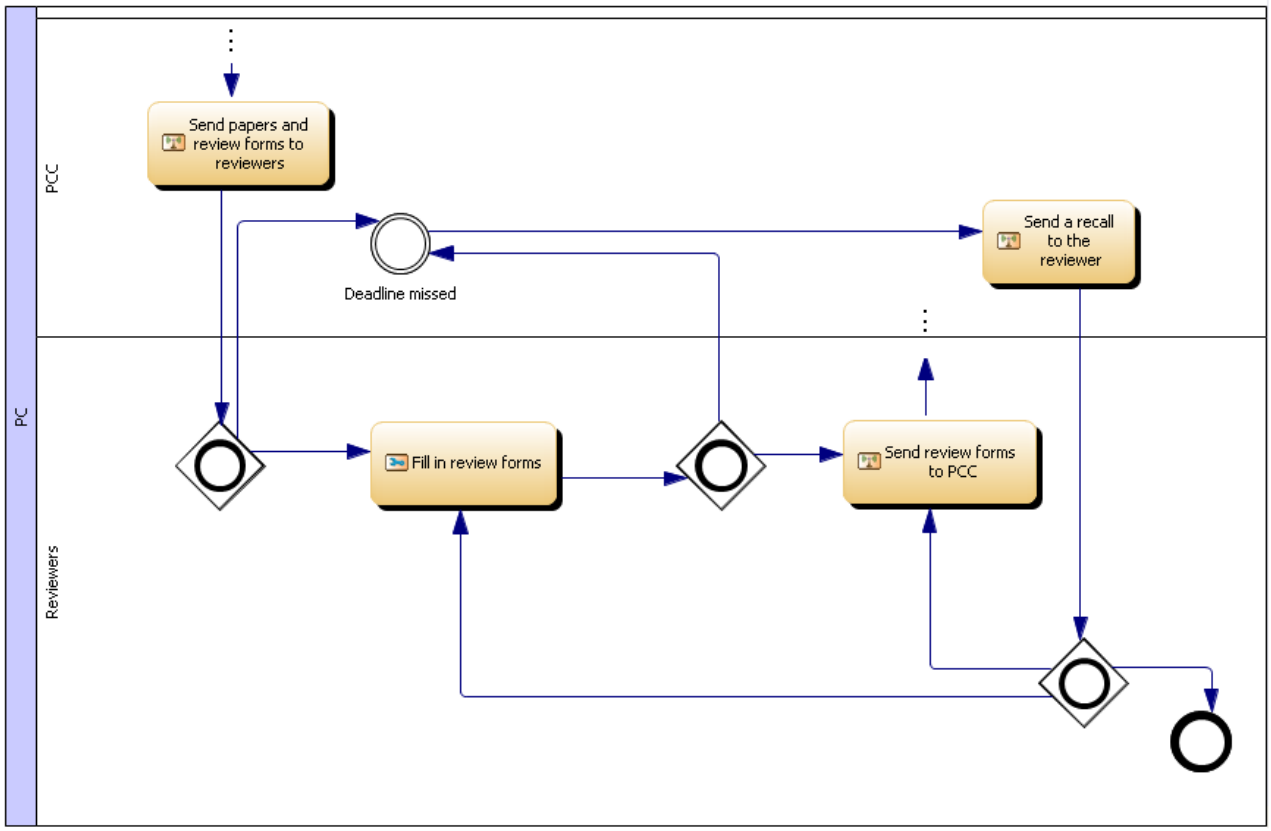


Figure 12.8: BPMN model - Part 2

The PC meeting

The third part consists of the PC meeting shown by Figure 12.9.

After the achievement of the task *Send review forms to the PCC*, the Token is passed to the task *Collect review forms* which represents the fact that the *PCC* collects all the review forms filled in by the reviewers. Through a sequence flow, the Token is passed to a parallel gateway. This parallel gateway will join two parallel paths. The condition is "Process flow shall continue when a signal (a Token) has arrived from all of a set of Sequence Flow (i.e., the process will wait for all the signals arriving before it can continue)" [OMG06]. A Token is well receive from the first path (coming from the task *Collect review forms*). The other Token will be generated by the intermediate event *Date of the meeting arrived*. When the two Token arrive to the parallel gateway, the Token will pass through the outgoing sequence flow of the gateway.

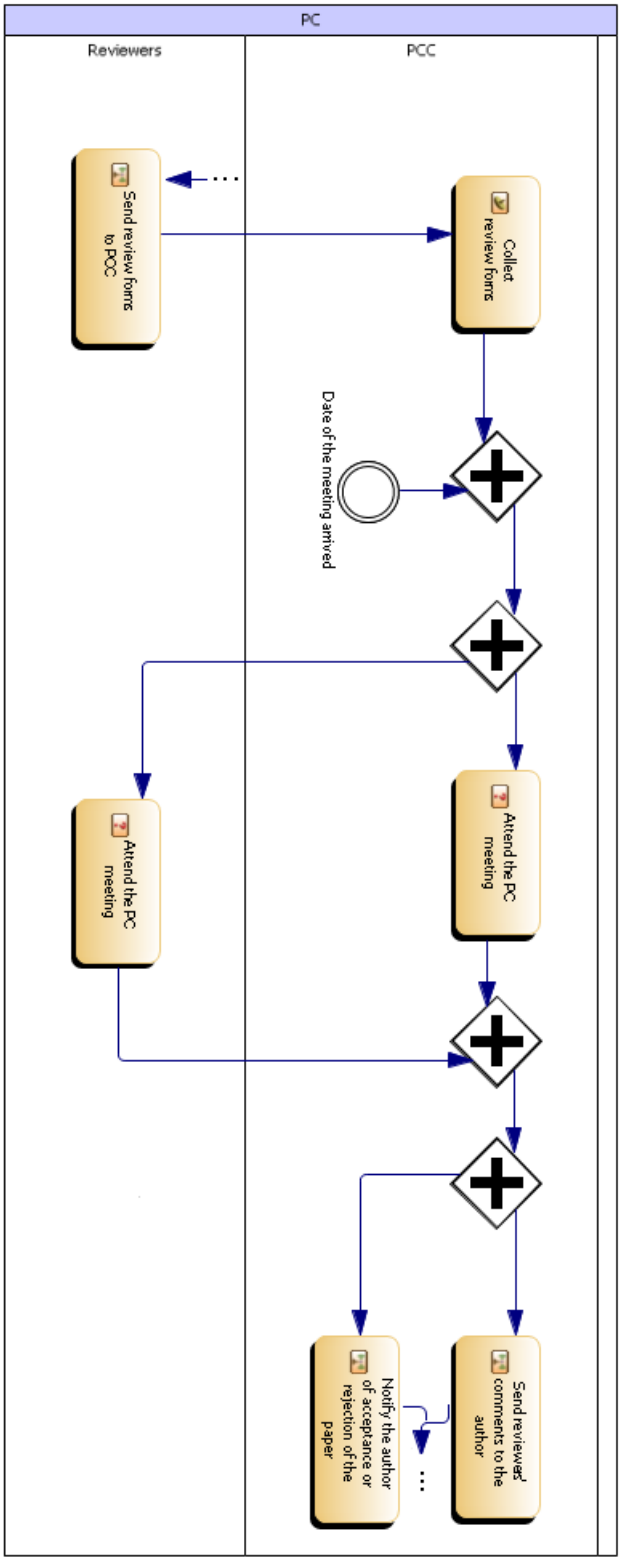


Figure 12.9: BPMN model - Part 3

Then, there is another parallel gateway which will create two parallel flows. Thus, one Token will go in each flow. The first flow brings to the task *Attend the PC meeting* which represents the fact that the PCC has to attend the PC meeting. The second flow brings to the task *Attend the PC meeting* which is located in the lane *Reviewers* of the pool *PC* because the reviewers also have to attend the PC meeting. This task is in the lane *PCC* and also *reviewers* because all, the PCC and the reviewers, attend the PC meeting. Putting this task in the general pool *PC*, which represents all the members of this meeting, should be another representation.

When the PC meeting is ended, two parallel gateways are needed to synchronize the flows and create two other ones in parallel. We cannot use one gateway with two incoming flows and two outgoing flows [OMG06]. The first path brings to the task *Notify the author of acceptance or rejection of the paper* which implies that the author is notified of the acceptance or rejection of his paper by the PCC. The second path brings to the task *Send reviewer's comments to the author.*

Improvement process

The fourth part consists of the improvement process shown by Figure 12.10.

The author receives the reviewer's comments and is aware of the acceptance or rejection of his paper. These facts are represented by two receive tasks: *Receive reviewer's comments* and *Receive and acknowledge.* After this, the gateway *"Paper accepted ?"* is used to check if the paper is accepted or not. If it is rejected, it is the end of the process. If it is accepted, the process continues with the improvement of the paper and the submission of the final version by the author, by the way of two tasks: *Improve paper* and *Submit final version.* The Token is passed to an intermediate event which will terminate the process located in the pool *Author.*

Printing of the proceedings process

The last part consists in printing the proceedings process shown by Figure 12.11.

A message flow goes out of the task *Submit final version* to arrive to an intermediate event message. This event will trigger the task *Collect final versions*, i.e. the PCC will collect all the final versions of the papers. Then, he writes the preface and the table of contents to create the proceedings which will be printed. The Token is passed to the task *Send the proceedings to the publisher* achieved by the PCC. Next, the Token is consumed by an intermediate event which marks the end of the process achieved in the pool *PC.*

A message flow goes out of this task to arrive to an intermediate event message. This event triggers the task *Verify the proceedings* which is located in the pool *Publisher.* Indeed, the publisher has to verify the proceedings before sending them to the printer. This is the second task of the publisher, *Send proceedings.* This task has an outgoing sequence flow which brings to an intermediate event and an outgoing message flow which brings to an intermediate event message. The intermediate event located in the pool *Publisher* indicates the end of the process of this pool. The intermediate event message located in the pool *Printer* triggers the last task of this all process, *Print proceedings* which is achieved by the printer. The all process is finished when the proceedings are printed, with an end event. The Token is well consumed.

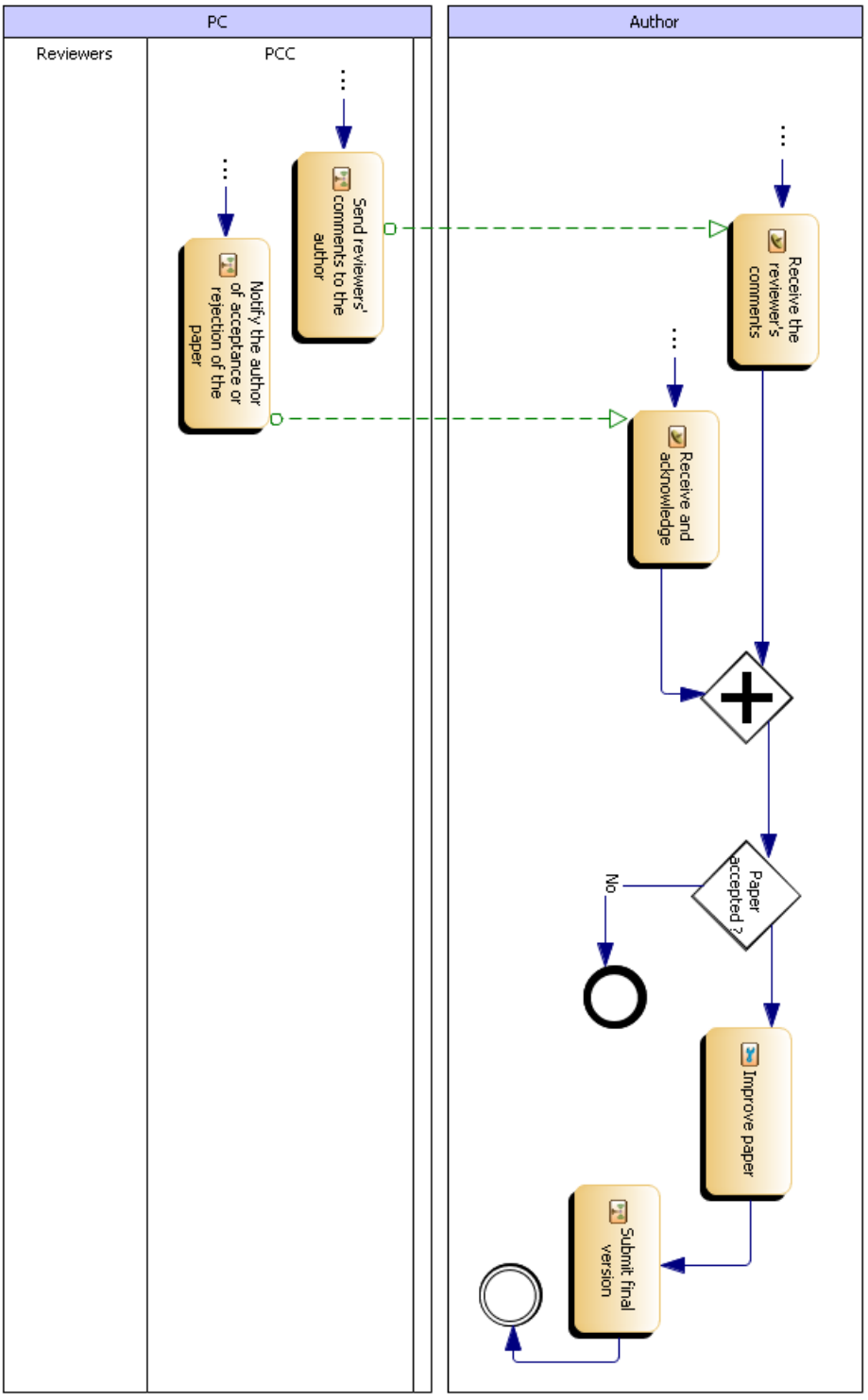


Figure 12.10: BPMN model - Part 4

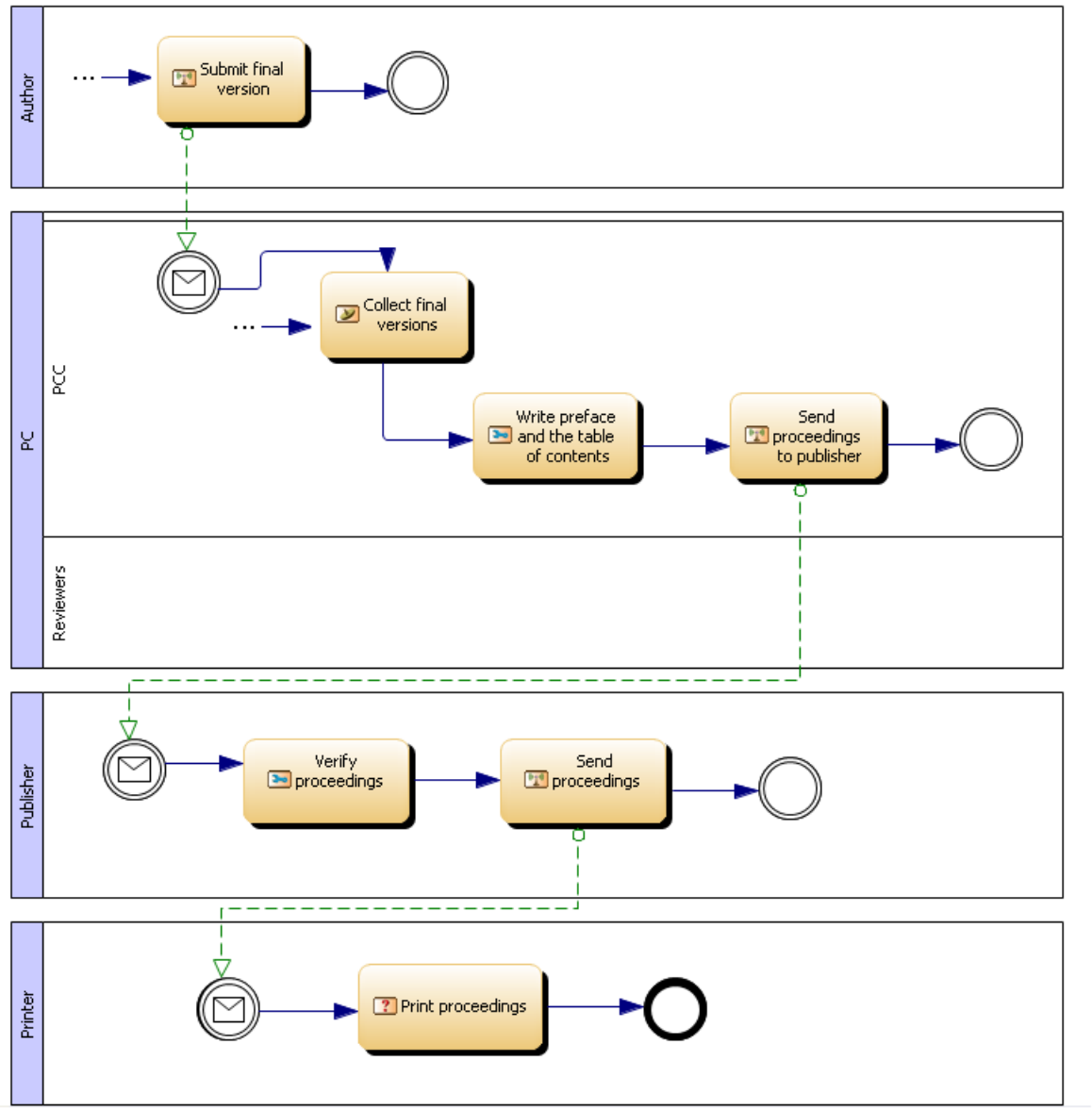


Figure 12.11: BPMN model - Part 5

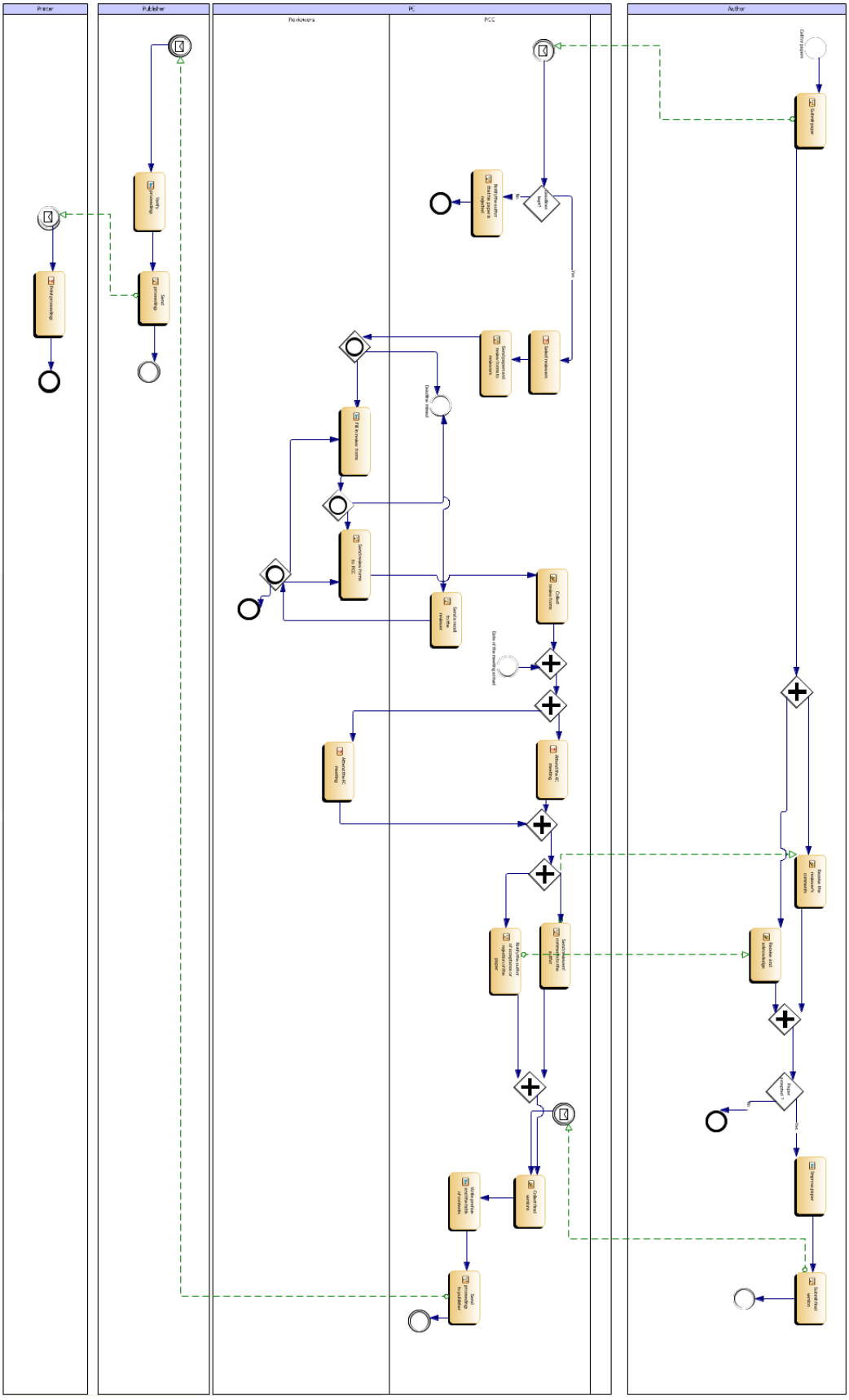


Figure 12.12: BPMN model

12.5 Verification of the mappings

With the case study, we were able to validate the analyses of ARIS and BPMN. Before explaining how we will proceeded, we have to underline the difference between two levels: the language level and the model level. At the language level, some mappings were defined from "ARIS or BPMN concepts → common ontology concepts". At the model level, some correspondences can be identified between "Elements of ARIS or BPMN model of the conference organization" and "Elements of common ontology model of the conference organization". The level we want to verify is the language level. The intention is to use the model level to validate the mappings at the language level. We can increase our confidence that a mapping is correct if its consistency applied in the case study (i.e. if the correspondences between the same kind of elements found in models) can be justified by the application of a mapping defined at the language level. This distinction is shown by Figure 12.13. When we have a C1 BPMN construct which has been used to create a E1 element of BPMN model and a C2 ARIS construct which has been used to create a E2 element of ARIS model. These two constructs, C1 and C2, mapped onto C3 in the common ontology. The two elements, E1 and E2, have to correspond in the common ontology to an unique and same concept (E3). We illustrate the distinction of the two levels by using the line 1 of Table 12.1:

- C2 = Event
- C1 = Start event
- C3 = *FlowContent*
- E1 = Call for paper
- E2 = Call for paper
- E3 = FlowContent"CallPaper"

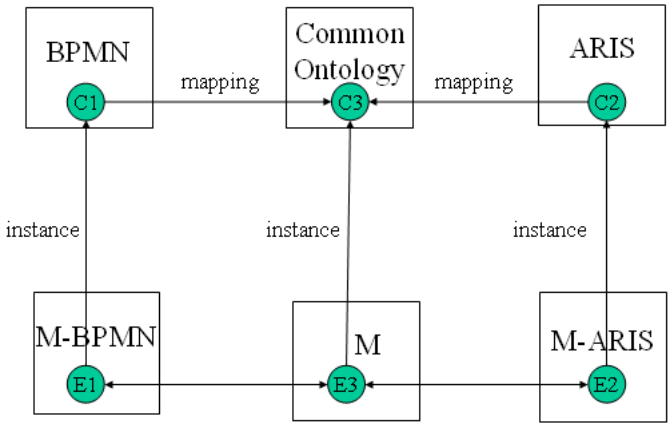


Figure 12.13: Distinction between language and model levels

To verify the mappings, we follow a certain method. For each concepts of the conference system, we identified the ARIS construct and BPMN construct which represent this concept. Then, for each ARIS and BPMN constructs, we identified the mapping to the common ontology. We compared both mappings of the constructs intended to represent the same concept of the conference system. To be coherent, the mappings should be the same or close. This has to be true for each instance of a construct used in the model. If the mappings are identical, it will not say that the mappings are correct. If a concept of the conference system is only represented by a construct of one of the two languages because one language does not provide a construct to represent this aspect, the mappings cannot be verified. If the mappings of constructs representing the same concept are different, they could be not coherent or the initial models could be not good.

The following Table 12.1 shows the concepts of the conference system, the constructs which represent this concept and the mappings between ARIS construct and the common ontology and between BPMN and the common ontology.

Table 12.1: Verification of the mappings

	Concepts of the conference system	**Language**	**Constructs**	**Mapping to the common ontology**
1	Call for paper	ARIS	Event	*FlowContent*
		BPMN	StartEvent	*FlowContent*
2	*Letter symbol*	ARIS	Message	*StateLaw*
		BPMN		
3	Submit paper	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
4	Author	ARIS	Organizational Unit	*OrganizationalUnit*
		BPMN	Pool	*ActiveThing*
5	Deadline kept	ARIS	Goal	*Law*
		BPMN		
6	Paper submitted	ARIS	Event	*FlowContent*
		BPMN		
7	*Or*	ARIS	Logical operator "Or"	*MutualLaw*
		BPMN	Inclusive Gateway	*MutualLaw*
8	Notify the author that the paper is rejected	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
9	Author notified	ARIS	Event	*FlowContent*
		BPMN		
10	Paper	ARIS	Information Service	*InformationService* & *InputThing* *InformationService* & *OutputThing*
		BPMN		
11	List of reviewers	ARIS	Information Service	*InformationService* & *OutputThing*
		BPMN		
12	Reviewers assignation List	ARIS	Environmental Data	*ReactiveThing* & *InputOutputThing*
		BPMN		
13	PC	ARIS	Computer Hardware	*ComputerHardware*
		BPMN		
14	Reviewers selection system	ARIS	Application Software	*ExecutingThing*
		BPMN		
15	PCC	ARIS	Organizational Unit	*OrganizationalUnit*
		BPMN	Lane	*ActiveThing*
16	Select reviewers	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
17	High matching	ARIS	Goal	*Law*
		BPMN		
18	Reviewers selected	ARIS	Event	*FlowContent*
		BPMN		
19	Papers	ARIS	Information Service	*InformationService* & *OutputThing*
		BPMN		

Continued on next page

	Concepts of the conference system	**Language**	**Constructs**	**Mapping to the common ontology**
20	Review forms	ARIS	Information Service	*InformationService* & *OutputThing*
		BPMN		
21	Send papers and review forms to the reviewers	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
22	Paper and review forms arrived	ARIS	Event	*FlowContent*
		BPMN		
23	Send recall to reviewer	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
24	Deadline missed	ARIS	Event	*FlowContent*
		BPMN	Intermediate event	*FlowContent*
25	Recall readed	ARIS	Event	*FlowContent*
		BPMN		
26	Recall not readed	ARIS	Event	*FlowContent*
		BPMN		
27	Reviewer	ARIS	Organizational Unit	*OrganizationalUnit*
		BPMN	Lane	*ActiveThing*
28	Fill in review forms	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
29	Review forms filled	ARIS	Event	*FlowContent*
		BPMN		
30	Review forms filled	ARIS	Information Service	*InformationService* & *InputThing* *InformationService* & *OutputThing*
		BPMN		
31	Send review form to PCC	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
32	Review forms arrived	ARIS	Event	*FlowContent*
		BPMN		
33	Collect review forms	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
34	Review forms collected	ARIS	Event	*FlowContent*
		BPMN		
35	Meeting date arrived	ARIS	Event	*FlowContent*
		BPMN	Intermediate event	*FlowContent*
36	Attend the PC meeting	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
37	PC	ARIS	OrganizationalUnit	*OrganizationalUnit*
		BPMN	Pool	*ActiveThing*
38	Meeting ended	ARIS	Event	*FlowContent*
		BPMN		
39	*And*	ARIS	Logical Operator "And"	*MutualLaw*
		BPMN	Parallel Gateway	*MutualLaw*
40	Reviewer's comments	ARIS	Information Service	*InformationService* & *InputThing* *InformationService* & *OutputThing*

Continued on next page

	Concepts of the conference system	**Language**	**Constructs**	**Mapping to the common ontology**
		BPMN		
41	Notify the author of acceptance or reject of the paper	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
42	Send reviewer's comments to the author	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
43	Paper rejected	ARIS	Event	*FlowContent*
		BPMN	End event	*FlowContent*
44	Paper accepted	ARIS	Event	*FlowContent*
		BPMN		
45	Comments arrived	ARIS	Event	*FlowContent*
		BPMN		
46	Receive and acknowledge	ARIS BPMN	Task	*ActivityLaw*
47	Improve paper	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
48	Paper improved	ARIS	Event	*FlowContent*
		BPMN		
49	Final version	ARIS	Information Service	*InformationService* & *InputThing* *InformationService* & *OutputThing*
		BPMN		
50	Submit final version	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
51	Final version submitted	ARIS	Event	*FlowContent*
		BPMN	Intermediate event	*FlowContent*
52	Collect final version	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
53	Final version collected	ARIS	Event	*FlowContent*
		BPMN		
54	Write the preface and the table of contents	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
55	Proceedings	ARIS	Information Service	*InformationService* & *InputThing* *InformationService* & *OutputThing*
		BPMN		
56	Proceedings prepared	ARIS	Event	*FlowContent*
		BPMN		
57	Send proceedings to the publisher	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
58	Proceedings received	ARIS	Event	*FlowContent*
		BPMN	Intermediate event	*FlowContent*
59	Publisher	ARIS	OrganizationalUnit	*OrganizationalUnit*
		BPMN	Pool	*ActiveThing*
60	Verify the proceedings	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*

Continued on next page

	Concepts of the conference system	**Language**	**Constructs**	**Mapping to the common ontology**
61	Proceedings verified	ARIS	Event	*FlowContent*
		BPMN		
62	Send the proceedings to the printer	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
63	Printer	ARIS	OrganizationalUnit	*OrganizationalUnit*
		BPMN	Pool	*ActiveThing*
64	Print proceedings	ARIS	Function	*FunctionLaw*
		BPMN	Task	*ActivityLaw*
65	Proceedings printed	ARIS	Event	*FlowContent*
		BPMN	End event	*FlowContent*
66	*Message Flow*	ARIS		
		BPMN	Message Flow	*Flow*
67	*Sequence Flow*	ARIS		
		BPMN	Sequence Flow	*Flow*

Four cases can be underlined in Table 12.1:

1. For a given concept, one ARIS construct and one BPMN construct are mapped onto the same common ontology concept.
2. For a given concept, one ARIS construct and one BPMN construct are mapped onto close common ontology concepts.
3. A given concept is represented by one ARIS construct, but no BPMN construct represents the concept.
4. A given concept is represented by one BPMN construct, but no ARIS construct represents the concept.

A last case that might have existed could be: "for a given concept, one ARIS construct and one BPMN construct are mapped onto distinct common ontology concepts". This case is not present in our case study.

Now we will illustrate how it is done in our work. We will explain the different cases by using some concepts represented in Table 12.1:

- "Call for paper" (line 1)
- "*And*" (line 39)
- "Submit paper" (line 3)
- "Author" (line 4)
- "List of reviewers" (line 11)
- "*Message flow*" (line 66)
- "*Sequence flow*" (line 67)

The first case, "for a given concept, one ARIS construct and one BPMN construct are mapped onto the same common ontology concept", can be illustrated by two concepts:

"Call for paper"

This concept is represented by ARIS.Event construct and BPMN.StartEvent construct. Those two constructs are mapped onto *FlowContent* in the common ontology and they have the same definition:

"An Event is something that "happens" during the course of a business process. These Events affect the flow of the process". Those constructs represent the same phenomenon. We can thus conclude that the mappings are coherent. Looking through the table, we can see that at each time we identify an ARIS.Event/BPMN.Event, it is mapped onto *FlowContent*.

"And"

The concept "*And*" can be represented by the logical operator "And" in ARIS and by the Parallel Gateway in BPMN. These two constructs are mapped onto *MutualLaw* in the common ontology. The mappings are coherent because the constructs are mapped onto the same property, *MutualLaw*. They represent the same concept and have the same definition (See Sections 5.3 and 6.2).

The second case, "for a given concept, one ARIS construct and one BPMN construct are mapped onto close common ontology concepts", can be illustrated by two concepts:

"Submit paper"

The "submit paper" concept is represented by the ARIS.Function and the BPMN.Task.
The ARIS.Function is mapped onto *FunctionLaw* and the BPMN.Task is mapped onto *ActivityLaw*. Each one (*FunctionLaw* and *ActivityLaw*) is preceded by *TransformationLaw* in the common ontology. These two constructs overlap but are not identical. It is why we decided to keep the two different laws. We can conclude that the mappings are coherent because those construcs overlap and the two contructs map each one onto a specialization of *TransformationLaw*, they can be qualified as close common ontology concepts. Looking through the table, we can see that all ARIS.Function are mapped onto *FunctionLaw* and all BPMN.Activity are mapped onto *ActivityLaw*.

"Author"

The ARIS.OrganizationalUnit construct and the BPMN.Pool construct represent the concept "Author". ARIS.OrganizationalUnit is mapped onto *Organizational Unit* and BPMN.Pool are mapped onto *ActiveThing*. *Organizational Unit* is a specialization of *ActiveThing* but the two constructs, ARIS.OrganizationalUnit and BPMN.Pool, are intended to represent the same thing: it is a responsible entity, a participant in a work. Thus, we decided to create a class of things, *Participant* which represents the concept of a participant in a process. This class will replace the class *Organizational Unit* in the common ontology, i.e. *Organizational Unit* is renaming in *Participant*. ARIS.OrganizationalUnit and BPMN.Pool would be mapped onto *Participant*. The mappings are coherent because the constructs, ARIS.OrganizationalUnit and BPMN.Pool, are mapped onto the same class, *Participant*, in the common ontology and they represent the same conference system concepts of Table 12.1.

The third case, "a given concept is represented by one ARIS construct, but no BPMN construct represents the concept", can be illustrated by the "List of reviewers" concept:

"List of reviewers"

The concept "List of reviewers" can be represented by ARIS.InformationService construct. The construct is mapped onto *OutpuThing* and *InformationService*. This concept represents a particular kind of input of a function in the conference system. In BPMN, an or more outputs/inputs must be defined for the Activity's OutputSets/InputSets. An output/input is an Artefact, usually a Data Object: "A Data Object will be shown as being an input, then an output of a Process. Directionality added to the Association will show whether the Data Object is an input or an output. Also, the state attribute of the Data Object can change to show the impact of the Process on the Data Object." [OMG06]. The Data Object provides information about what the process does. That is, how documents, data, and other objects are used and updated during the process. While the name "Data Object" may imply an electronic document, they can be used to represent many different types of objects, both electronic and physical [OMG06]. Thus, in some cases the ARIS.Output can be considered as a BPMN.DataObject but not all the time. In that sense, we can say that these two constructs are not identical. Thus, for all the concepts of the conference system identifying an input or output of a function, there is no similar construct in BPMN to represent these concepts. The mapping of the ARIS.Output hasn't being revised.

For the fourth and last case, "a given concept is represented by one BPMN construct, but no ARIS construct represents the concept", can be illustrated by two concepts:

"Message Flow"

The concept of a "message flow" is represented by the BPMN.MessageFlow construct. This construct is mapped onto *Flow*. There are some flows in ARIS: the Control Flow, the Organization Flow, the Information Flow, the Information Services Flow, the Financial Resource Flow and the Material Output Flow. But none of these flows corresponds to the BPMN.MessageFlow.

"Sequence Flow"

The concept of a "sequence flow" is represented by the BPMN.SequenceFlow construct. This construct is mapped onto *Flow*. We can say that the ARIS.ControlFlow is nearly similar to the BPMN.SequenceFlow. All the other flows in ARIS have no similar construct in BPMN. The ARIS.ControlFlow is mapped onto *Flow*. The two constructs map onto the same property in the common ontology but the mappings are not coherent. There is a subtle difference: in a BPMN.SequenceFlow, sequencing is achieved by some passing of token, but a ARIS.ControlFlow does not assume a token: events in things may be sequenced by inherent laws in each thing, without any coordination going on between them.

12.6 Lessons learnt

On the basis of the modelling of a common scenario in two different languages, the case study allows revising the mappings, identifying and discussing sensible points. With the two models achieved in ARIS and BPMN, we are induced to review the choices made, in particular the mappings between the constructs and the common ontology. In that sense, the main point gained by the case study is the improvement of two mappings: the mappings of ARIS.OrganizationalUnit and BPMN.Pool. Now, both map onto *Participant* in the common ontology.

The case study enables to describe an existing system by means of models. The application of a concrete case was useful to better understand the constructs. The use of those in a real case allows checking again their definition and having a more general view on the constructs of each languages to revise their mappings.

Threats to validity:

- The case study is not so easy to achieve. We had to create two models of the same scene in two different languages. We would ideally have to totaly forget how we realized the first model while we were doing the second one. But it was not so easy to do. At some moments, we were influenced by the first model. It is a difficult point to manage when case studies are achieved.
- The complexity of the conference system was difficult to represent simply. The case was only modelled with a general view.
- The subjectivity of the modeller is a limitation of the case study. Every person implied in the analysis would be influenced by his proper subjectivity. For example, we decided to do not represent the event, "Paper and review forms arrived" (line 22 of Table 12.1), in BPMN. It is an arbritary choice of modelling. The reception of those paper and review forms should not be necessarily represented. But another person should have a different opinion and decide to represent it.

There are often several different ways to present the same set of issues. The results are influenced by the subjectivity of the modeller, by the way of modelling and by the different languages used. If the case study is carried out by different or more people, the models could be different and it could perhaps influence the results. Different results could influence the validation of the mappings. The subjectivity has some consequences on this validation. According to modelling choices, some problems cannot be identified. The subjectivity is a limitation of the case study.

If the case is different, for instance a financial system, we think that these comments would also apply. The results would probably not be influenced by the case but well by the people carrying out the case study.

12.7 Summary

In this chapter, we explained the main research question and the process we pursued by modelling a case study in ARIS and in BPMN. Then, we described the two models. To finish, we validated the mappings. Indeed, the case study allowed us to discuss about the mapping of some constructs like the *Organizational Unit* and *Pool*. It enables to think about certain modifications of the mapping and the addition of a new class (for instance, *Participant*) in the common ontology.

Chapter 13

Languages comparison

In this chapter, we will make a comparison between the **primary mappings** of ARIS and BPMN to the common ontology. The primary mappings correspond to the most important classes and properties of the modelling construct. The results of the comparison will allow us to verify the conclusions of the case study.

Firstly, we will explain the possible results that we can obtain (+, -, +/-). Then, we will present the comparison between ARIS and BPMN by means of Table 13.1. We will focus on two comparisons, one between ARIS.OrgnanizationalUnit and BPMN.Pool, the other between ARIS.Function and BPMN.Activity. The comparisons between the rest of the constructs can be found in Appendix K. Finally, in Section 13.5, we will evaluate this comparison.

13.1 Explanation of the possible results

This section is inspired by [MOH06] and [MOH07]. We use the same correspondences (+, -, +/-) to explain the comparison between the two languages. The correspondences are based on three criteria:

- The primary mappings: It corresponds to the most important classes and properties of the modelling construct.
- The definition of the construct: It is the definition given by the authors of the language.
- The other mappings: It corresponds to the classes, properties, states and transformations which are not a primary mapping.

Same primary mapping (+)

A "+" correspondence indicates that two modelling constructs of different languages represent the same scene or a part of the scenes is common to both constructs. The scene is the environment of the construct, i.e. the set of the represented classes, properties, states and transformations. In other words, using the criteria, the constructs have the same primary mappings and the same definition. However, we accept in this category that the others mappings of the constructs are different, i.e. a construct can overlap the other.

As we can see in Table 13.1, the "+" correspondence concerns:

- ARIS.OrganizationalUnit compared to BPMN.Pool
- ARIS.And compared to BPMN.ParallelGateway
- ARIS.Or compared to BPMN.InclusiveGateway
- ARIS.Xor compared to BPMN.ExclusiveGateway
- ARIS.Event compared to BPMN.Event

Nearly same primary mapping (+/-)

A "+/-" correspondence indicates some semantic similarities and some differences between two constructs. We have identified two cases:

- The primary mappings are different but a part of the other mappings is identical. And the definitions of both constructs are not necessary exactly similar.
- The primary mappings are identical and the constructs represent the same concept but the definitions are not identical.

Table 13.1 shows wich comparison can be qualified as nearly same primary mapping. It concerns the main part of the constructs. For example, it is the comparison between ARIS.Function and BPMN.Activity.

Different primary mapping (-)

A "-" correspondence means that a construct in one language has no corresponding construct in the other.

For instance, the construct Output in ARIS has any correspondence in BPMN. To be complete, we can conclude the same for the Material output, Services, Information services, Other services, Human output, Message, Environmental data, Application software, Computer hardware, Machine resource and Goal in ARIS. It is the same with the Process, Message Flow and Complex Gateway in BPMN.

13.2 Comparison ARIS/BPMN

Table 13.1 lists the correspondences between ARIS and BPMN constructs, showing the primary mapping to the common ontology. The mapping indicates the primary represented class/property.

In the following sections, we will illustrate the "+" and "+/-" correspondences by explaining two comparisons: one between the ARIS.OrganizationalUnit and the BPMN.Pool and the other between ARIS.Function and BPMN.Activity.

Table 13.1: Comparison between ARIS and BPMN

ARIS		**BPMN**		**Comparison**
Constructs	**Mapping**	**Constructs**	**Mapping**	
Function	*FunctionLaw*	Activity	*ActivityLaw*	+/-
		Task	*ActivityLaw*	+/-
		Sub-Process	*ActivityLaw*	+/-
		Process	*TransformationLaw*	-
Organizational unit	*Participant*	Pool	*Participant*	+
Position	*RoleHolder*	Lane	*ActiveThing*	+/-
And	*MutualLaw*	Parallel Gateway	*MutualLaw*	+
		Gateway	*MutualLaw*	+/-
		Complex Gateway	*MutualLaw*	-
Or	*MutualLaw*	Inclusive Gateway	*MutualLaw*	+
		Gateway	*MutualLaw*	+/-
XOR	*MutualLaw*	Data-Based Exclusive Gateway	*MutualLaw*	+/-
		Event-Based Exclusive Gateway	*MutualLaw*	+/-
		Exclusive Gateway	*MutualLaw*	+
		Gateway	*MutualLaw*	+/-
Ouput	*Repository & InputThing*			-

Continued on next page

ARIS		**BPMN**		**Comparison**
Constructs	*Repository* & *OutputThing* **Mapping**	**Constructs**	**Mapping**	-
		Data Object	*DataObject*	-
Material output	*MaterialRepository* & *InputThing* *MaterialRepository* & *OutputThing*			- -
Services	*Service* & *InputThing* *Service* & *OutputThing*			- -
Information services	*InformationService* & *InputThing* *InformationService* & *OutputThing*			- -
Other services	*MaterialService* & *InputThing* *MaterialService* & *OutputThing*			- -
Human output	*HumanOutput*			-
Event	*Flow* & *FlowContent*	Event	*Flow* & *FlowContent*	+
		Start event	*Flow* & *FlowContent*	+/-
		Intermediate event	*Flow* & *FlowContent*	+/-
		End event	*Flow* & *FlowContent*	+/-
Message	*StateLaw*			-
Environmental data	*ReactiveThing* & *InputOutputThing*			-
Application software	*ExecutingThing*			-
Goal	*Law*			-
Machine resource	*MachineResource*			-
Computer hardware	*ComputerHardware*			-
Control Flow	*Flow*	Sequence Flow	*Flow*	+/-
		Message Flow	*Flow*	-

13.3 Comparison of ARIS.OrganizationalUnit and BPMN.Pool

The Table 13.2 illustrates the comparison between the organizational unit and the pool. Organizational unit belongs to ARIS and pool belongs to BPMN. This table summarizes the mappings of ARIS and BPMN onto the common ontology with the similarities. It is divided in three groups corresponding to the division of the scene of the organizational unit and of the pool. The scene of the organizational unit is shown by Figure 13.1. The one of the pool is shown by Figure 13.2 [1].

[1] The legend of Figures 13.1 and 13.2 is provided in Appendix F.

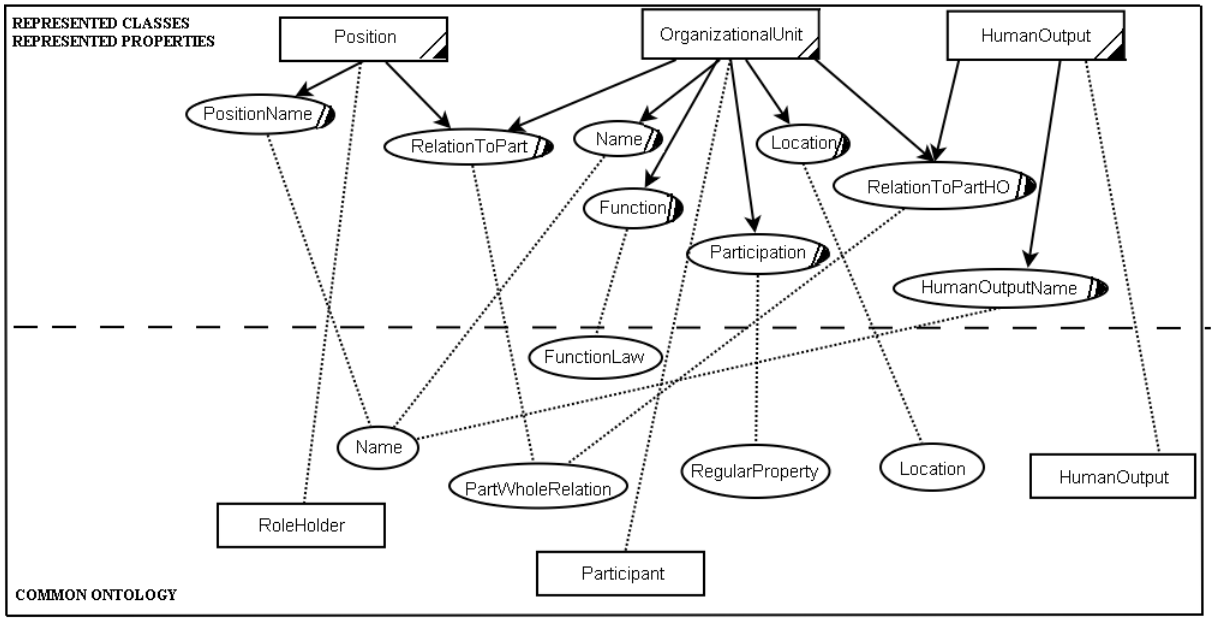


Figure 13.1: Scene of the organizational unit

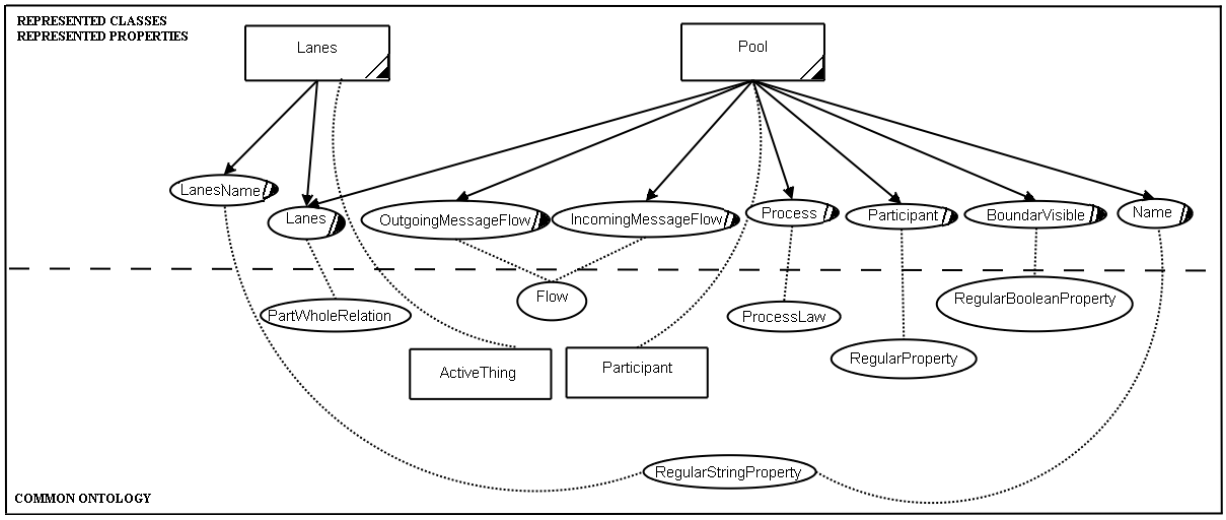


Figure 13.2: Scene of the pool

The first split of Table 13.2 shows that the `organizational unit` and the `pool` are both mapped onto *Participant* in the common ontology because these two constructs identified the participant/entity involved in a function or in a process. The organizational unit can be composed of positions. The `RelationToPart` property illustrates the possible composition of positions into an organizational unit. This fact is represented by a class `Position` which is mapped onto *RoleHolder* and a property `RelationToPart` which is mapped onto *PartWholeRelation*. In BPMN, the pool can be composed of several lanes. It is represented by the class `Lanes` which is mapped onto *ActiveThing* and a property `Lanes` which is mapped onto *PartWholeRelation*. The position has a name and the lane too. Those names are represented as two properties (`LanesName` and `PositionName`). The `name` of the position is mapped onto *Name* and the `name` of the lane onto *RegularStringProperty* because it represents the text description of the lane. This part of the scene of the constructs is nearly the same for ARIS and BPMN. The names of position and lane are mapped onto different concepts but *Name* and *RegularStringProperty* have the

same superclass, *RegularProperty*. Sometimes, the lane can be the same than a position according to its definition "Lanes are often used for such things as internal roles, systems (e.g., an enterprise application), an internal department" [OMG06].

In the second split of Table 13.2, the organizational unit and the pool have several properties. The organizational unit has four other properties. We identify the `Name`, `Function`, `Location` and `Participation`. Regarding the pool, we identify the `Name`, `BoundaryVisible`, `Participant`, `Process`, `IncomingMessageFlow`, `OutgoingMessageFlow`. Two properties are mapped onto the same property of the common ontology: the `Participation` and the `Participant`. The `Participation` represents the type of participation of the organizational unit in the function under his responsibility and is mapped onto *RegularProperty*. The `Participant` can be either a role or an entity. This defines the role that a particular entity or role the pool will play in a diagram that includes collaboration [OMG06]. This property is mapped onto *RegularProperty*. In ARIS and BPMN, we can see that the organizational unit has a name and the pool too but they are not mapped onto the same concept. The `name` of organizational unit is mapped onto *Name* in the common ontology because it represents excatly the name of the organizational unit. The `name` of the pool is mapped onto *RegularStringProperty* because it is a text description of the pool. Thus, one property in ARIS and one BPMN are mapped onto the same property of the common ontology. Three properties are found in the scene of the organizational unit and not in the one of the pool. Five properties are found in the scene of the pool and not in the one of the organizational unit.

The last split of Table 13.2 underlines a third part in the scene of the organizational unit. It includes the `HumanOutput`, `RelationToPartHumanOutput` and `HumanOutputName`. This part does not exist in the scene of the pool.

We can conclude that these two constructs overlap but are not identical. They have the same primary mapping and there are some properties not defined in ARIS or in BPMN. We decide to put a "+" because they are similar in the sense of the definition but it will not mean that we can transform a construct in another. We have to take in account all the information (different properties). If we make a transformation from a language to another, we lose some information and we need to define/add information.

Table 13.2: Comparison between ARIS.OrganizationalUnit and BPMN.Pool

Splits	**ARIS**	**BPMN**	**Common ontology**
1	OrganizationalUnit	Pool	*Participant*
	Position	-	*RoleHolder*
	-	Lanes	*ActiveThing*
	RelationToPart	Lanes	*PartWholeRelation*
	PositionName	-	*Name*
	-	LanesName	*RegularStringProperty*
2	Participation	Participant	*RegularProperty*
	Function	-	*FunctionLaw*
	Name	-	*Name*
	Location	-	*Location*
	-	Process	*ProcessLaw*
	-	Name	*RegularStringProperty*
	-	BoundaryVisible	*RegularBooleanProperty*
	-	OutgoingMessageFlow	*Flow*
	-	IncomingMessageFlow	*Flow*
3	HumanOutput	-	*HumanOutput*
	HumanOutputName	-	*Name*
	RelationToPartHumanOutput	-	*PartWholeRelation*

13.4 Comparison of ARIS.Function and BPMN.Activity

The Table 13.3 illustrates the comparison between the function and the activity. Function belongs to ARIS and activity belongs to BPMN.

As shown in Figure 8.5, the `Function` is mapped onto *FunctionLaw* and, as shown in Figure 9.5, the `Task` is mapped onto *ActivityLaw*. Those results can be seen in the first split of Table 13.3. Each property, *FunctionLaw* and *ActivityLaw*, is preceded by *TransformationLaw*. They represent the same concept with few particular aspects in each one. Those two constructs are under responsibility of an `organizational unit` in ARIS and a `participant` in BPMN. They are both mapped onto *Participant*.

The activity and the function are complex properties. They have a lot of subproperties representing the activity or function characteristics. Those properties are all listed in the second split of Table 13.3. We identify the `Assignements`, `Lane`, `Token`, `ActivityType`, `LoopType`, `Pool`, `Name`, `Status`, `StartQuantity`, `IORules`, `InputSet`, `OutputSet`, `Properties`, `IsActive`, `IncomingSequenceFlow`, `IncomingMessageFlow`, `OutgoingMessageFlow`, `OutgoingSequenceFlow`, `OutgoingSequenceFlowContent`, `OutgoingMessageFlowContent`, `IncomingSequenceFlowContent` and `IncomingMessageFlowContent` for the activity. Regarding the function, we identify the `Participation`, `ApplicationLaw`, `InformationFlow`, `IncomingFlow`, `OutgoingFlow`, `Goal`, `UseLaw`, `OutputFlow`, `OutputFlowInput` and `IsActive`. As we can see, some are mapped onto the same concepts of the common ontology:

- `IncomingFlow`, `IncomingMessageFlow` and `IncomingSequenceFlow` are mapped onto *Flow*
- `OutgoingFlow`, `OutgoingMEssageFlow` and `OutgoingSequenceFlow` are mapped onto *Flow*
- `OutputFlow`, `OutgoingSequenceFlowContent` and `OutgoingMessageFlowContent` are mapped onto *FlowContent*
- `OutputFlowInput`, `IncomingSequenceFlowContent` and `IncomingMessageFlowContent` are mapped onto *FlowContent*
- `IsActive` is present in both models and mapped onto *IsActive*

The rest of the second split of Table 13.3 is particular to one language.

The shared property *IsActive* is the property which concerns the dynamic structure of the construct. As summarized in the third split of Table 13.3, the function has a `Functionning` state and a `NonFunctionning` state. Those are the "to and from states" of four transformations: `TriggeringFunction`, `TerminationFunction`, `NotAllInputsAvailable` and `NotAllOuputsAvailable`. The first one is mapped onto *Triggering*, the second one onto *Termination* and the two last ones onto *AnyTransformation*. Nearly similarly, the activity has also an `ActiveState` and `InactiveState`. Thus, the two constructs have the two same states which mapped onto the same properties of the common ontology (*ActiveState* and *InactiveState*). For the activity, we have identified three transformations: `TriggeringActivity`, `TerminationActivity` and `NotAllTokenAvailable`. The first one is mapped onto *Triggering*, the second one onto *Termination* and the last one onto *AnyTransformation*. The activity has a transformation less than the function but the other transformations are identical to these of the function.

Besides that, many of the subproperties of the function are linked to a particular class because the human and material participants are more described into ARIS than into BPMN. For instance, `ComputerHardware`, `Machine`, `EnvironmentalData`, `Software`, `HumanOutput`, `SourceOutput` and `TargetOutput` are identified in ARIS and not in BPMN. They can be seen in the fourth split of Table 13.3.

To conclude, we can say that the primary mappings of those two constructs are not exactly the same. They have many differences. On the other hand, the "dynamic" part, i.e. the states and transformations, is nearly the same. The function and the activity are not identical. They have some differences but have a similar part of the scene. Because this part doesn't concern the primary mappings, we decide to put a "+/-".

Table 13.3: Comparison between ARIS.Function and BPMN.Activity

Splits	ARIS	BPMN	Common ontology
1	Organizational Unit	Participant	*Participant*
	Function	-	*FunctionLaw*
	-	Activity	*ActivityLaw*
2	-	IORules	*Law*
	-	InputSet	*Law*
	-	OutputSet	*Law*
	-	StartQuantity	*RegularNaturalProperty*
	-	Properties	*RegularProperty*
	-	Status	*RegularStringProperty*
	-	Name	*RegularStringProperty*
	-	Pool	*RegularProperty*
	-	LoopType	*RegularStringProperty*
	-	ActivityType	*RegularStringProperty*
	-	Token	*RegularMutableProperty*
	-	Lane	*RegularProperty*
	-	Assignements	*RegularProperty*
	IncomingFlow [2]	IncomingMessageFlow	*Flow*
		IncomingSequenceFlow	*Flow*
	OutgoingFlow [3]	OutgoingMessageFlow	*Flow*
		OutgoingSequenceFlow	*Flow*
	OutputFlow [4]	OutgoingSequenceFlowContent	*FlowContent*
		OutgoingMessageFlowContent	*FlowContent*
	OutputFlowInput [5]	IncomingSequenceFlowContent	*FlowContent*
		IncomingMessageFlowContent	*FlowContent*
	Participation	-	*ParticipationLaw*
	ApplicationLaw	-	*ApplicationLaw*
	InformationFlow	-	*InteractionRelation*
	Goal	-	*Law*
	UseLaw	-	*UseLaw*
	IsActive	IsActive	*IsActive*
3	FunctionningState	ActiveState	*ActiveState*
	NonFunctionningState	InactiveState	*InactiveState*
	TriggeringFunction	TriggeringActivity	*Triggering*
	NotAllInputAvailable	NotAllTokenAvailable	*AnyTransformation*
	TerminationFunction	TerminationActivity	*Termination*
	NotAllOutputAvailable	-	*AnyTransformation*
4	ComputerHardware	-	*ComputerHardware*
	Machine	-	*MachineResource*
	EnvironmentalData	-	*ReactiveThing*
	Software	-	*ExecutingThing*
	HumanOutput	-	*HumanOutput*
	SourceOutput	-	*Repository & OutputThing*
	TargetOutput	-	*Repository & InputThing*

[2]IncomingFlow in ARIS represents all the flows that arrive to a function, it is a general concept. So, the IncomingMessageFlow and the IncomingSequenceFlow can be compared to the IncomingFlow.

[3]OutgoingFlow in ARIS represents all the flows that go out of a function, it is a general concept. So, the OutgoingMessageFlow and the OutgoingSequenceFlow can be compared to the OutgoingFlow.

[4]OutputFlow in ARIS represents the content of the outgoing flows of a function, it is a general concept. So, the OutgoingSequenceFlowContent and the OutgoingMessageFlowContent can be compared to the OutputFlow.

[5]OutputFlowInput in ARIS represents the content of the incoming flows of a function, it is a general concept. So, the IncomingSequenceFlowContent and the IncomingMessageFlowContent can be compared to the OutputFlowInput.

13.5 Comparison evaluation

This comparison of languages was not so easy to achieve. ARIS and BPMN have not the same level of details but anyone can be caracterized as more complete than one other. BPMN has a lot of constructs mainly with generalization/specialization relations and described much attributes. That makes BPMN widely defined on the basis of just some general constructs. On the other hand, ARIS has more different constructs, not specialized as in BPMN. ARIS describes much the different human and material participants in a process. Thus, the comparison implies to abstract some information (attributes) of the scene of a BPMN construct to be able to compare it with an ARIS construct and conversly.

The comparison between ARIS and BPMN is a good basis for the transformation from one language to another. The Table 13.1 shows which constructs of ARIS could be transformed in a construct of BPMN and conversly. It also underlines the constructs that have no similar construct in the other language. In fact, the comparison shows what the similarities between the two languages are. It allows identifying which information has to be deleted from a construct and which information has to be added to a construct when transforming a construct of a language in another of the second language. Thus, the comparison permits to underline which information is existing in one language and not the other. We only discussed two comparisons, but the tables for all the constructs comparisons, are provided in Appendix K.

The comparison between languages should be useful for future works around UEML. In particular the correspondences identified in Table 13.1 could be used to translate models created in one language to the others. The transformations might be done by tools which would suggest model similarities.

13.6 Summary

In this chapter, we compared ARIS and BPMN on the basis of the mappings achieved during the analyses. The possible results were described. Then, we listed the correspondences between ARIS and BPMN constructs, showing the primary mapping to the common ontology. To finish, we explained two comparisons: one between ARIS.OrganizationalUnit and BPMN.Pool which represents a "+" correspondence and another between ARIS.Function and BPMN.Activity which represents a "+/-" correspondence.

The comparison between ARIS and BPMN shows that some constructs of ARIS are similar, nearly similar to another of BPMN or have any corresponding constructs in the BPMN, and conversly. This comparison is useful for the transformation of a language in another. It allows the identification of the information existing in one construct of a language and not in another of another language.

Chapter 14

Evaluation of the UEML approach

Several advantages and disadvantages of the UEML approach given in [Opd06b] can be summarized as follows:

Advantages:

- It offers more detailed advice on how to proceed when analysing individual language constructs.
- It encourages construct descriptions at a high level of details, which tend to integrate languages at a fine-grained, precise level.
- It produces complete construct descriptions that are easily comparable.
- It supports ontological analysis in terms of particular classes, properties, states and events, and not just in terms of the concepts in general.
- It acknowledges that a language construct often represents a scene played by several ontological concepts together.
- It suggests a path towards tool-supported, integrated use of models expressed in different languages, through the structured format in combination with the common ontology.
- It has a positive externality, in the sense that each construct becomes easier to incorporate as more constructs are already added to the UEML and each language becomes easier to incorporate as more languages are already added.

Disadvantages:

- The approach is based on a particular way of thinking.
- The approach produces subjective results.

We will underline the advantages and disadvantages of the UEML approach, of the UEML template, of the common ontology and of the tools used. We will specify if we have identified the same advantages and disadvantages as [Opd06b]. In Section 14.5, we will also evaluate the similarities identification.

14.1 Evaluation of the general ideas of the UEML approach

Advantages

- **Several ontological concepts:** The UEML approach acknowledges that a language construct often represents a scene played by several ontological concepts together. We agree with [Opd06b] when saying that it is an advantage of the approach. The methodology never limits to the strict construct, but allows to study the whole environment of this one. The UEML template and the common ontology are done to allow the analyst to describe the relationships between the constructs and how they are really used in the language itself.

Disadvantages

- **Particular way of thinking:** The way of representing the scene was difficult to apply at the beginning of the analysis. We had some problems to see how we had to represent and to map some of the constructs. It corresponds to the disadvantage: "The approach is based on a particular way of thinking". It is sometimes difficult to adopt this special way of thinking. In a particular way, we mean that we had to think every construct as a representation of some phenomena of the common ontology. It was sometimes difficult to choose the classes, properties, states and transformations. And finally, it was not easy to know if you have to choose a concept of the common ontology or if you have to create a new one. All those things compose this particular way of thinking. It was difficult for us at the beginning to work by using this way of reasoning, but with the time and the experience, it becomes usual.
- **No guidelines:** There is no guidelines to indicate which constructs we have to choose to analyse and which concepts of the language are a modelling construct. It is in contradiction with the advantage given by [Opd06b]. Some steps, like the choice of the constructs, are not defined precisely. But we will explain later that some materials, like the template, allow more guidelines than previously.
- **Subjective results:** The UEML approach produces subjective results. We agree with [Opd06b] when saying that it is a disadvantage of the approach. It is impossible to do it otherwise, this method obliges the analyst to do some arbitrary choices. And those choices bring subjectivity. We encountered this disadvantage, even if we were working by two, our decisions depended on the way we saw the language, on our background and on the way of applying the method.
- **Any conversion of the grammar to meta models:** The UEML approach doesn't specify to convert the grammar of a language into meta models. But the analysis of a language leads to convert the constructs and the relationships between them into meta models. It should be better to have all those relations formalized into some meta models.
- **Specification of the mappings:** The UEML approach doesn't specify any precise mappings. The approach lets the analyst choose how he wants to accomplish the mappings. It is linked to the lack of guidelines and a solution should be inspired from the reference methodology [GR05b]. It should be useful to do the representation mapping and the interpretation mapping to have two different approaches to identify the mappings.

14.2 Evaluation of the UEML template

Advantages

- **Guideline:** The UEML template offers more detailed advice on how to proceed. It provides us guidelines and a structure for the description of the modelling constructs. This standard way allows us to understand and evaluate each construct according to the different entries. It allows a step-by-step way of work.
- **Clear:** The UEML template is clear and all-encompassing with the three sections which describe different information and the entries in each of them.
- **Comparison between constructs:** The UEML template produces complete construct descriptions that are easily comparable. We agree with [Opd06b] when saying that it is an advantage of the template. Its structure and the system of entries allow an easy comparison between different constructs because all those entries are composed of the same concepts.

Disadvantages

- **Particular way of thinking:** The UEML template implies to adopt a particular way of thinking. In fact, the method leads to consider a language, and particularly its constructs, as a representation of some phenomena of the common ontology. It needs that the users of the template have to be common with the concepts of the ontology and the different entries (concepts) of the template.

- **Misunderstanding of some entries:** At the beginning of the use of the template, we had some problems of misunderstanding. For instance, in the Preamble section, we did not see the difference between the *related construct names* and the *related terms*. In the Representation section, we had some difficulties to fill in the *Cardinality* and *Reverse cardinality* entries. This way of representing the cardinalities in three different ways was quite new for us. We also did not see clearly the differences between *existence*, *state*, *event*, *transformation* and *process*. There was no precise definition of these behaviours. The way of describing them was difficult to apply because it was abstract and there is any example in the UEML Template Tutorial [Opd06a]. With the experience, these entries were easier to understand and to complete.

To summarize, we can say that the UEML template is clearly representative and useful in the UEML approach, it has some advantages. But on top of that, we also can observe the disadvantages of this material.

14.3 Evaluation of the common ontology

Advantages

- **Analysis in terms of particular classes, properties, states and events:** The common ontology allows analysis in terms of particular classes, properties, states and events, and not just in terms of the concepts in general. Being composed of four different parts, the ontology classes, ontology properties, ontology states and ontology transformations, the common ontology permits to describe the construct in his whole scene. The whole scene consists of the construct and its relationships with other constructs, thus all its environment. The representation of the whole scene allows a better analysis and a more complete understanding of the construct.

- **Positive externality**: Each construct becomes easier to incorporate as more constructs are already added to the UEML and each language becomes easier to incorporate as more languages are already added. We agree with [Opd06b] when saying that it is an advantage. The analyses are easier to conduct because each previous analysis allows to enlarge the common ontology. In fact if you don't find the exact class in the ontology to represent a construct, you can add a new one that corresponds exactly what this construct represents. During our first analysis, we had to add some new classes and properties. Those addings to the common ontology were useful for the BPMN analysis. For instance, during the first analysis we added the property *MutualLaw* that we used for the second analysis.

- **Well-structured:** The hierarchy of the common ontology is a strong aspect of it. The hierarchy consists of the relations of specialization/generalization of the classes and precedes/preceded by of the properties. It is easier to find a concept in a hierarchy. It means that we begin with the most general concepts and we refine them until we find the correct one. If the most general one doesn't fit exactly with the modelling construct, we need to check which of its specializations fits the best. This aspect helps us in the research of the classes and properties.

- **Extension of the common ontology:** The common ontology grows dynamically as more specific classes, properties, states and events are included. It allows suggesting more and more concepts. In that sense, the common ontology becomes more specific and allows representing the reality with more precision.

Disadvantages

- **Abstract:** The use of the common ontology is not so easy at the beginning. Several names of classes and properties seemed abstract. It seemed that they did not really represent the real intended phenomenon. For instance, *StateLaw* and *TransformationLaw*. Some definitions were ambiguous and we needed a few explanations in addition. For instance, we did not see clearly the difference between the *CoupledThing* and *AssociatedThing*. Now, these definitions are clear but some need to be improved. The application of the common ontology requires a certain experience and time, to understand each concept.

- **Subjective results:** As explained, the UEML approach produces subjective results. This subjectivity comes, among others, from the choice made to determine which classes, properties, states and transformations can represent a modelling construct. This choice depends on the analyst's background and on his way to understand the concepts. For instance, we have the choice to use a general concept of the common ontology or to create a new one. We have decided when there is no exactly mapping to add a new one.

We can conclude by saying that the common ontology is well-structured but is composed of weak aspects. All those aspects naturally influence the advantages and disavantages of the UEML approach itself.

14.4 Evaluation of the tools used

Advantages

- **UEMLBase:** The UEMLBase is a useful tool which should replace the paper-based construct template used so far. This tool gathers the different analyses and the common ontology in a common base. It is then possible to see how previous analyses were carried out and to have an easy acces to the concepts of the common ontology.
- **UEML Validator:** The UEML Validator is a good tool to highlight some mistakes, for instance, not filling in entry or inconsistencies. It allows correcting the analyses and discussing some underlined problems. This tool allows to improve the analyses.

Disadvantages

- **UEMLBase:** The UEMLBase tool should be improved because it is not stable. Another disadvantage is that there is a lot of entries which are not in the paper-based template. It is sometimes difficult to understand these entries and to fill in them. Thus, the tool can be still closer to the UEML template.
- **UEML Validator:** The main disadvantage of the UEML Validator is that some of the rules are not correct. Those mistakes can be the object of a possible improvement of the tool. It can also be useful to add new rules to identify more mistakes.

14.5 Evaluation of the similarity identification

The indenfication of similarities concerns the comparison between the languages, i.e. the identification of the constructs which are similar to one or more constructs in another language.

Advantages

- **Good review method:** The indentification of similarities is useful. It allows detecting possible errors in the mappings and at least a good review of the analyses. It enables us to think about one or two more difficult points of the mappings and to discuss them.
- **Useful for languages transformation:** The similarity identification is the first step allowing the transformation from one language to another. With this comparison, it is easy to identify which information we have to add or to "delete", to tranform a construct of a language into a construct of another language.

Disadvantages

- **Difference between languages:** This identification is difficult to achieve. If two languages have two totally different degrees of representation, it is difficult to identify some similarities between the constructs. In our case, ARIS is more defined concerning the responsible entities acting in the process than BPMN. As BPMN is more defined concerning the gateways but also concerning the attributes of the constructs. Then, the similarities usually concern a part of the scene of the constructs.

14.6 Summary

The first analysis was difficult for us and took us a long time. We had to familarize ourselves with the special way of thinking, the common ontology and the entries of the UEML template. We had to select the modelling constructs we wanted to analyse. This selection seems easy in theory but in practice it is more complex. Some constructs in the language can be chosen by some experts and not by some others. It is linked to subjectivity. Another point is that it was hard to find the appropriate class, property for a modelling construct. It was also hard to choose between using an existing class, property and creating a new one. It is naturally linked to the particular way of thinking. Experience can fill in this gap. For the second analysis, it was easier to fill in the UEML template.

An important point to underline is the subjective results. Like in many analyses, the UEML approach implies a part of subjectivity. It is difficult to decrease this one even by a team work and the achievement of a case study. There will be always a part of subjectivity in this kind of work.

In this chapter, we evaluated the UEML approach. In particular, the general ideas of the UEML approach, the UEML template, the common ontology, the similarity idenfication and the tools used were evaluated.

Chapter 15

Conclusion

In this work we have analysed two process-oriented modelling languages using the UEML approach [BOAD05]. First, we haved overviewed the first version of UEML and in more details the second one. We have explored the BWW model and the UEML template which are the basis of UEML 2.0. The concepts of the BWW model are maintained in a common ontology for the UEML approach and provide a way of defining modelling constructs. The UEML template was proposed to define in a standard way modelling constructs of different languages. This template is divided into three parts: Preamble, Presentation and Representation. The third part, representation part, uses the common ontology to describe the constructs.

We have analysed two modelling languages: ARIS and BPMN, by using the UEML approach. We have carried out an analysis of the languages construct-by-construct. The analysed constructs have to be integrated in UEML. In that sense, we have filled in the entries of the UEML template for each analysed construct. For the third part of the template, we haved used the common ontology to identify the mappings, i.e. the correspondences between the construct and the concepts of the common ontology. According to our work, we have extended this ontology by adding classes and properties. We had to add phenomena when those of the common ontology weren't precise enough to represent the studied construct. During the languages evaluation, we have applied a research method for ontological analysis. The research method is inspired by the UEML approach and a reference methodology [GR05b]. We have applied some steps of one and the other method. With hindsight, we have proposed a new method that was not yet applied practically, but we believe that it could bring better results than the previously used methodology.

After the analysis, we have validated the results using the UEML Validator and a case study. First, we have validated them with the UEML Validator. The tool provides a list of the mistakes found in the analyses. That has allowed us to improve the analyses by correcting the errors. We have also identified some possible mistakes in the UEML Validator. Secondly, we have carried out a case study to verify the mappings of the analyses. We verified that two constructs representing the same thing in the world are mapped onto the same concepts of the common ontology.

Through their mappings onto the common ontology, ARIS and BPMN become interoperable and available for comparison. We have thus compared the two process-oriented languages. It has allowed reviewing the different mappings and to give the first step permitting the transformation from one model created in a language to others.

Finally, we have evaluated the UEML approach, more particularly the general ideas of the UEML approach, the UEML template, the common ontology, the tools used and the similarity identification. We have underlined the advantages and disavantadges of all those things.

The basic conclusions of our work are:

- Enterprise model integration, transformation, translation are today an issue for building complex systems that show high autonomy of constituents, and robustness to changes and evolution. UEML is created to address this issue, and to facilite the enterprise integration.
- The BWW model is the start for defining entreprise modelling constructs. The concepts are then

maintained in a common ontology which grows incrementally as more modelling constructs are incorporated into the UEML.

- The UEML template provides a standard form describing modelling constructs by filling in a set of entries. The template is powerful to support evaluation of a broad variety of languages.
- The combination of the UEML 2.0 approach and the reference methodology explained in [GR05b] allows us to apply a good research method but improvements still needed. The new proposed method was not yet applied practically, but we believe that it could bring better results than the previously used methodology.
- The analysis of ARIS and BPMN provides a description of the semantics of each construct as a representation of some phenomena of the common ontology.
- The extension of the common ontology allows the common ontology to grow and to suggest more and more concepts. It becomes more specific and it permits to represent the reality with more precision. New concepts were added when those of the common ontology weren't precise enough to represent the construct.
- The UEML Validator provides a list of the mistakes which can be found in the languages analyses. This tool allows the validation and the improvement of our analyses by underlining and correcting errors. The case study also permits to revise the mappings of some constructs and thus improves them by comparing the mappings of the constructs.
- The comparison between ARIS and BPMN shows that some constructs of one language are similar, nearly similar to another or have nor corresponding construct in the other language. The languages comparison is useful for the transformation of one language to another.

Our work, in particular the analyses of ARIS and BPMN, produces subjective results which consist in a description of each construct in term of mappings. The results are influenced by our subjectivity and the subjectivity of the mappings. Our subjectivity consists of our understanding of the languages, our background and the way of applying the method. The subjectivity of the mappings consists of the arbitrary choices the analyst had to make to identify the concepts of the common ontology which map with the construct. It also comes from the choice of creating a new concept or using an existing one of the common ontology.

Today, it remains a lot of modelling languages which are not analysed and not incorporated in UEML. Analysing more languages could be a future work. UEML should be broadened by incorporating more process-oriented languages and other kinds. It should be also important to add guidelines and frameworks to the UEML approach concerning the analysis of languages.

The case study and the languages comparison underline some corrections to do about the mappings. It would probably be possible to identify other modifications. Besides, some constructs of ARIS and of BPMN are not yet analysed. Revising, improving the analyses and completing them with the rest of the constructs could be future works. The definition of an official meta model of BPMN should be a good development to review the BPMN analysis. It could also be interesting to integrate a case study and a language comparison in the UEML approach. Those two activities could be useful to revise and validate an analysis. It allows underlining sensible points or mappings which should be discussed and possibly improved.

The application of the UEML Validator identifies some mistakes in rules. It could be interesting to improve this tool. Indeed, some mistakes in rules need to be corrected. The rules should be improved and it could be useful to add new rules. A future work could be to develop a tool to help the transformation from a language to another. This tool should try to identify the correspondences between two models of two languages in the UEMLBase. It should allow the translation of a construct to another by adding or deleting information.

Some concepts of the common ontology are still general. One of the next work could be to specify them by more precise concepts. It should facilitate the mappings and the comprehension of the ontology.

Finally, testing UEML in entreprises could be interesting. This UEML should be composed of five to ten languages. This version should be submitted to several entreprises which would use it in their usual work. It could be a good test and it should allow receiving a feedback to improve UEML. If the results are conclusive, UEML should be enlarged with some new languages.

Bibliography

[Ber03] Giuseppe Berio. Deliverable D3.1 - Requirements analysis: initial core constructs and architecture. May 2003.

[Ber05] Giuseppe Berio. UEML 1.0 and UEML 2.0: benefits, problems and comparison. *Business Process Management Workshops*, 2005.

[Ber06] Giuseppe Berio. DEM project presentation. May 2006.

[BOAD05] Giuseppe Berio, Andreas L. Opdahl, Victor Anaya, and Michele Dassisti. Delivrable DEM 1 - UEML 2.0. Novembre 2005.

[BPMa] BPMG. URL: http://www.bpmg.org/.

[BPMb] BPMN. URL: http://www.bpmn.org/.

[BPMc] http://searchcio.techtarget.com. Definition of BPMN.

[BPP04] Giuseppe Berio, Hervé Panetto, and Michaël Petit. UEML: Résultats et enjeux d'un langage unifié de modélisation d'entreprise. *Séminaires de l'Institut - ISI - Institut des systèmes d'information*, Septembre 2004.

[Bun77] Mario Bunge. Treatise on basic philosophy: Vol. 3: Ontology i: The furniture of the world. 1977.

[Bun79] Mario Bunge. Treatise on basic philosophy: Vol. 4: Ontology ii: A world of systems. 1979.

[Com06] UEMLBase Draft, June 21st 2006.

[con] http://www.sciences.univ-nantes.fr/info/lrsg/enseignement/uml_par_l_exemple/exo2.htm. Officiel website of University of Nantes, example of Electronic management of conferences.

[Dav01] Rob Davis. *Business Process Modelling with ARIS - A Practical Guide*. Springer, 2001.

[DEM] Deliverable 5.2: Overall Dissemination Final Report.

[DEM02] UEML Deliverable 2.1. 2002.

[DMBF05] Nicola Di Mauro, Teresa M.A. Basile, and Stefano Ferilli. GRAPE: An Expert Review Assignment Component for Scientific Conference Management Systems, 2005.

[FL03] Peter Fettke and Peter Loos. Ontological Evaluation of Reference Models using the Bunge-Wand-Weber Model. 2003.

[FL05] Peter Fettke and Peter Loos. Ontological Analysis of Reference Models. 2005.

[GR00] Peter F. Green and Michael Rosemann. Integrated process modelling: An ontological evaluation. *Information Systems*, May 2000.

[GR02] Peter F. Green and Michael Rosemann. Perceived ontological weaknesses of process modeling techniques: further evidence. 2002.

[GR05a] Peter F. Green and Michael Rosemann. *Business Systems Analysis with Ontologies*. Idea Group Publishing, 2005.

[GR05b] Peter F. Green and Michael Rosemann. Ontological Analysis of Business Systems Analysis Techniques: Experiences and Proposals for an Enhanced Methodology. 2005.

[GRI04] Peter F. Green, Michael Rosemann, and Marta Indulska. A reference methodology for Conducting Ontological Analyses. 2004.

[IEE] http://www.ieee.org/web/conferences/mom/all_manual.html. Officiel website of the IEEE, section Conferences Organization Manual.

[Kro95] Lindland O.I. & Sindre G. Krogstie, J. Defining quality aspects for conceptual models. In & A. Olive (Eds.) In E. D. Falkenberg, W. Hesse, editor, *Towards a consolidation of views*, pages 216–231, Marburg, Germany, March 28-30 1995. Proceedings of the IFIP8.1 working conference on Information Systems Concepts (ISCO3).

[Mah06] Jérémy Mahiat. A validation tool for the UEML approach. June 2006.

[MOH06] Raimundas Matulevičius, Andreas L. Opdahl, and Patrick Heymans. Comparison of Goal-oriented Languages using the UEML Approach. *E12N 2006 France*, 2006.

[MOH07] Raimundas Matulevičius, Andreas L. Opdahl, and Patrick Heymans. Comparing GRL and KAOS using the UEML Approach. *I-ESA 2007 Portugal*, 2007.

[OB06a] Andreas L. Opdahl and Giuseppe Berio. Interoperable language and model management using the UEML approach. *Information SystemsInternational Conference on Software Engineering - Proceedings of the 2006 international workshop on Global integrated model management*, 2006.

[OB06b] Andreas L. Opdahl and Giuseppe Berio. Interoperable Language and Model Management Using the UEML Approach. *International Conference on Software Engineering - Proceedings of the 2006 international workshop on Global integrated model management*, 2006.

[OB06c] Andreas L. Opdahl and Giuseppe Berio. Roadmap for UEML. *I-ESA conference programme*, 22 March 2006.

[OHS04] Andreas L. Opdahl and Brian Henderson-Sellers. A Template for Defining Enterprise Modelling Constructs. *Journal of Database Management, Vol. 15, No. 2*, 2004.

[OHS05] Andreas L. Opdahl and Brian Henderson-Sellers. A Unified Modeling Language Without Referential Redundancy, Data and Knowledge Engineering (DKE). 2005.

[OMG06] OMG. Business Process Modeling Notation (BPMN) Specification - Final Adopted Specification. February 2006.

[Opd] Andreas L. Opdahl. Introduction to the BWW-representation model and Bunge's ontology.

[Opd06a] Andreas L. Opdahl. UEML Template Tutorial. 2006.

[Opd06b] Andreas L. Opdahl. The ueml Approach to Modelling Construct Description. *I-ESA conference programme*, 22 March 2006.

[oVUI] Polytechnic University of Valencia (UPV) INTEROP. Tutorial - Business Process Modelling Language.

[PD02] Michaël Petit and Guy Doumeingts. Deliverable D1.1 - Report on the State of the Art in Enterprise Modelling. Septembre 2002.

[PH04] Michaël Petit and Patrick Heymans. Perspectives on the scope and definition process of the Unified Enterprise Modelling Language. 2004.

[Ros] http://aisel.isworld.org/article_by_author.asp?author_id=350. Michael Roseman's Articles website.

[RvBT04] Henk Jonkers (TI) René van Buuren (TI), Luuk Groenewegen (LIACS). Deliverable 2.2.3b: Mapping between archimate and standards. Technical Report https://doc.telin.nl/dscgi/ds.py/Get/File-38740, Leiden Institute of Advanced Computer Science (LIACS); Telematica Instituut (TI); Ordina Public Consulting, March 2004.

[Sch98] A.-W. Scheer. *"ARIS - Business Process Frameworks", Second, completely revised and enlarged edition.* Springer, 1998.

[Sch99] A.-W. Scheer. *"ARIS - Business Process Modeling", Third edition.* Springer, 1999.

[UEMa] UEML 1.0. URL: http://www.ueml.org.

[UEMb] UEML 2.0. URL: http://www.interop-noe.org.

[vdAtHW03] Wil van der Aalst, Arthur ter Hofstede, and Mathias Weske. Business process management - international conference bpm 2003, eindhoven the netherlands june 2003, proceedings. 2003.

[Whi04] Stephen A. White. Introduction to BPMN. *IBM*, May 2004.

[WK] Boris Wyssusek and Helmut Klaus. On the Foundation of the Ontological Foundation of Conceptual Modeling Grammars: The Construction of the Bunge–Wand–Weber Ontology.

[WW88] Yair Wand and Ron Weber. An ontological analysis of some fundamental information systems concepts, 1988.

[WW93] Yair Wand and Ron Weber. On the ontological expressiveness of information systems analysis and design grammars. 1993.

[WW95] Yair Wand and Ron Weber. On the deep structure of information systems. 1995.

Index

Part IV

Appendix

Appendix A

BWW Table

Here is presented a description of the main concepts of BWW model in Table A.1.

Table A.1: Description of the main concepts of BWW model

BWW concept	**Concept definition**
BWW-thing	"The elementary unit in our ontological model. The real world is made up of things." (Wand & Weber, 1995)
BWW-property of a thing	"Things possess properties" (Wand & Weber, 1995). "We know about things in the world via their properties" (Weber, 1997).
BWW-complex property	A complex BWW-property consists of other properties, which may themselves be complex. (Opdahl & Henderson-Sellers, 2005)
BWW-property function	"A property is modeled via a function that maps the thing into some value" (Wand & Weber, 1995). A BWW-property function represents how some BWW-property changes over time. BWW-property functions are also called state functions (Weber, 1997) or state variables (Parsons & Wand, 1997).
BWW-property co-domain	"A set of things that can be defined by their possessing a particular set of properties"(Weber & Zhang, 1996). 1) A BWW-class is defined by a "characteristic set" of properties. 2) All groups of BWW-properties that are possessed by at least one BWW-thing define a BWW-class.
BWW-subclass of things	"A set of things that can be defined via their possessing the set of properties in a class plus an additional set of properties" (Weber & Zhang, 1996). (Hence, a BWW-subclass is itself a BWW-class.)
BWW-intrinsic property of a thing	"A property that is inherently a property of an individual thing" (Wand & Weber, 1995).
BWW-mutual property of two or more things	"A property that is meaningful only in the context of two or more things" (Wand & Weber, 1995).
BWW-state of a thing	"The vector of values for all property functions of a thing" (Wand & Weber, 1995).
BWW-state law of a thing	A property that "[r]estricts the values of the property functions of a thing to a subset that is deemed lawful because of natural laws or human laws" (Wand & Weber, 1995).
BWW-event in a thing	"A change of state of a thing. It is effected via a transformation (see below)" (Wand & Weber, 1995).
BWW-process in a thing	"An intrinsically ordered sequence of events on, or states of, a thing" (Green, 1996). Processes may be either chains or trees of events (Bunge, 1977).

Continued on next page

BWW concept	**Concept definition**
BWW-transformation of a thing	"A mapping from a domain comprising states to a co-domain comprising states" (Wand & Weber, 1995).
BWW-transformation law of a thing	"Events are governed by transformation laws that define the allowed changes of state" (Parsons & Wand, 1997). (Wand & Weber, 1995) and other papers on the BWW model instead introduce BWW-lawful transformations, which define "which events in a thing that are lawful". The term "transformation law" instead of "lawful transformation" is chosen here to emphasise that a transformation law like a state law is a property of a particular thing.
BWW-law property of a thing	"Properties can be restricted by laws relating to one or several properties" (Parsons & Wand, 1997). 1) A law is either a state law or a transformation law of a particular thing. 2) A law is either a natural law or a human law (see below.)
BWW-natural law	"Natural laws are established by nature" (Weber, 1997). For example, a law of physics.
BWW-human law	"Some laws are human-made artifacts" (Weber, 1997), i.e., they are socially constructed and enforced by humans. Events and processes may sometimes violate human laws, but not natural ones.
BWW-natural kind of things	"A natural kind is defined by a set of properties and the laws connecting them" (Parsons & Wand, 1997). 1) Hence, a BWW-natural kind is itself a BWW-class, but all its characteristic properties must be BWW-laws. 2) In this paper, we refer to the "subclasses" of BWW-natural kinds as BWW-sub-kinds.
BWW-conceivable state space of a thing	"The set of all states that the thing may ever assume" (Wand & Weber, 1995).
BWW-possible state space of a thing	"[T]he space of states that are possible given our understanding of the laws of nature" (Weber, 1997).
BWW-lawful state space of a thing	"[T]he set of states of a thing that comply with the state laws of the thing" (Wand & Weber, 1995). Hence, lawful states satisfy both human and natural state laws, whereas possible states may violate human ones.
BWW-conceivable event space of a thing	"The set of all possible events that can occur in the thing" (Weber & Zhang, 1996).
BWW-lawful event space of a thing	"The set of all events in a thing that are lawful" (Wand & Weber, 1995). Weber (1997) adds " [...] because (a) nature permits them to occur, and (b) there are no human laws that denote them as unlawful".
BWW-composite thing	"A composite thing may be made up of other things (composite or primitive)" (Wand & Weber, 1995). "Things can be combined to form a composite thing" (Parsons & Wand, 1997).
BWW-component thing	Any BWW-thing that is in the composition of a composite thing.
BWW-whole-part relation	The property of being in the composition of another thing or, complementary, of having another thing as a component (according to Bunge, 1977)).
BWW-resultant property of a composite thing	"A property of a composite thing that belongs to a component thing" (Wand & Weber, 1995).
BWW-emergent property of a composite thing	A property of a composite thing that does not belong to a component thing (adapted from (Wand & Weber, 1995).)
BWW-acting on another thing, BWW-coupling of things	"A thing acts on another thing if its existence affects the history of the other thing. The two things are said to be coupled [. . .]" (Wand & Weber, 1995).

Continued on next page

BWW concept	Concept definition
BWW-direct acting on, BWW-binding mutual property	A thing acts directly on one or more other things when the former thing changes a BWW-binding mutual property they all possess. Changing the binding mutual property is an internal event in the former thing and an external event in each of the latter things.
BWW-system of things	"A set of things is a system if, for any bi-partitioning of the set, couplings exist among things in the two subsets" (Wand & Weber, 1995). 1) A BWW-system is itself a BWW-thing. 2) BWW-system things belong to BWW-system natural kinds.
BWW-system composition	"The things in the system" (Wand & Weber, 1995), i.e., its component things.
BWW-system environment	"Things that are not in the system but interact with things in the system" (Wand & Weber, 1995).
BWW-system structure	"The set of couplings that exist among things in the system and things in the environment of the system" (Wand & Weber, 1995).
BWW-subsystem	"A system whose composition and structure are subsets of the composition and structure of another system" (Wand & Weber, 1995).
BWW-system decomposition	"A set of subsystems such that every component in the system is either one of the subsystems in the decomposition or is included in the composition of one of the subsystems in the decomposition" (Wand & Weber, 1995).
BWW-level structure	"Defines a partial order over the subsystems in a decomposition to show which subsystems are components of other subsystems or the system itself" (Wand & Weber, 1995).
BWW-external event in a thing, subsystem or system	"An event that arises in a thing, subsystem or system by virtue of the action of some thing in the environment of the thing, subsystem or system. The before-state of an external event is always stable. The after-state may be stable or unstable" (see below) (Wand & Weber, 1995). 1) Stable and unstable states will be defined below. 2) We have not defined the subsystem-concept because we do not need it in this paper.
BWW-internal event in a thing, subsystem or system	"An event that arises in a thing, subsystem or system by virtue of lawful transformations in the thing, subsystem or system. The before-state of an internal event is always unstable. The after-state may be stable or unstable" (see below) (Wand & Weber, 1995).
BWW-unstable state of a thing	"A state that will be changed into another state by virtue of the action of transformation in the system" (Wand & Weber, 1995).
BWW-stable state of a thing	"A state in which a thing, subsystem or system will remain unless forced to change by virtue of the action of a thing in the environment (an external event)" (Wand & Weber, 1995).

Appendix B

Common ontology

Table B.1 presents a description of the concepts of the common ontology that we used in our analyses.

Table B.1: Description of the main concepts of the common ontology

Common ontology concept	**Concept definition**
InputThing	"The class of things that are targets of flows."
OutputThing	"The class of things that are sources of flows."
InputOuputThing	"The class of things that are both targets and sources of flows."
CoupledThing	"A thing that is coupled with one or more other things. According to Bunge and the BWW-model, two things are coupled iff the history of states and events undergone by (at least) one of the things is different from what it would have been had the other thing not existed."
ExecutingThing	"An active thing that is not active all the time, but has an active state and an inactive state and corresponding triggering and terminating events. In other words, it is an active thing that has execution behaviour."
ActiveThing	"A changing thing that acts on at least one other thing through an acting-on relation, a particular type of coupling. In addition, the acting-on thing possesses one or more transformation laws that restrict the combinations of properties that the active thing can possess before and after an event."
System	"A composite thing whose components (or parts) are coupled (or interact). According to Bunge and the BWW-model, - two things are coupled iff the history of states and events undergone by (at least) one of the things is different from what it would have been had the other thing not existed, and - a composite thing is a system iff its components cannot be bi-partitioned so that no component in the first partition is coupled to a component in the second."
Component	"A thing that is a component (or part) of other things. The characteristic property of a component things is PartWholeRelation, a particular type of property. A component thing must be part-whole related to one or more components, and it must play the role of "part" in these relations."

Continued on next page

Common ontology concept	**Concept definition**
RegularProperty	"A property that is not mutual, not a law, not a part-whole relation and not a class-subclass relationship. However, it can be emergent or resultant. A RegularMutableProperty may be valueless or it may hold a value of any type. It may also have subproperties that hold values. In other words, AnyRegularProperty corresponds well to what is often called an "attribute" or "property"."
PartWholeRelation	"A property that relates a composite to one of its components. Part-whole relations are considered properties in the Bunge-sense, although they are a particular type of property, because there is a specific set of axioms defined for them.Although a part-whole relation belongs to both a composite (whole) and a component (part) it is not considered mutual. (A system should never have mutual properties with its components for systems-theoretic reasons.)"
IsActive	"A manipulated property that marks whether an (executing) thing is active or non-active."
Law	"A property that restricts the value of other properties of the same thing (its restricted properties) and that is not mutual, a part-whole relation or a class-subclass relationship."
InteractionRelation	"A coupling between two things in which the history of the first thing depends on the history of the second thing and the second thing's history depends on the history of the first."
Flow	"A binding mutual property through which things flow from an output thing to an input thing. The active side can be either the output side or the intput side. Possibly, the two sides can even be working independently, but synchronised, so a flow is not preceded by one-way coupling. (When we say that things flow 'through' the binding mutual property, we mean that the flow can have a subproperty that is mutual with some resource, denoting that the resource location is moved along the flow. Resource offers and acceptances can move along flows in the same way as the resources themselves, although acceptances moves in the opposite direction.)"
TransformationLaw	"A law that restricts the combinations of properties that an (active) thing can possess before and after an event. (An event is an occurrence of a transformation.) Like all laws (that are currenly accounted for in the UEML ontology), a transformation law is not mutual, not a part-whole relation and not a class-subclass relationship."
RegularStringProperty	"A regular property that holds a string value."
RegularBooleanProperty	"A regular property that is either True of False (a flag)."
RegularNaturalProperty	"A regular property that holds a natural number (0, 1, 2...) as its value."
CouplingRelation	"A mutual property that reflects a coupling between a collection of things. According to Bunge and the BWW-model, - two things are coupled iff the history of states and events undergone by (at least) one of the things is different from what it would have been had the other thing not existed, and - a collection of things is coupled iff its components cannot be bi-partitioned so that no component in the first partition is coupled to a component in the second. (Hence, the collection of things is a system because of the coupling.)"

Continued on next page

Common ontology concept	Concept definition
RegularMutableProperty	"A changing attribute, i.e., a property that changes and that is not mutual, not a law, not a part-whole relation and not a class-subclass relation."
MutualProperty	"A mutual property that is not a law, not a part-whole relation and not a class-subclass relationship. (A mutual property is one that is possessed by more than one thing. The mutual property thereby associates those things.)"
StateLaw	"A law that restricts the combinations of properties that a thing can possess in a state. Like all laws (that are currenly accounted for in the UEML ontology), a state law is not mutual, not a part-whole relation and not a class-subclass relationship."
ActiveState	"The active state of an ActiveInactiveThing."
InactiveState	"The inactive state of an ActiveInactiveThing. (Although this state is not presently used, it will be needed later, e.g., when dealing with non-reentrantly executing things.)"
MutableState	"A state that can change (or is mutable). Mutable states therefore occur in ChangingThings, and they must be defined by at least one mutable property. Mutable states are transformed by AnyTransformations."
Triggering	"A transformation that occurs in an EnabledState and then (immediately) enters an ActiveState, with no intervening states. It is the first step of an Execution transformation. It is effected by a TriggeringLaw and occurs in an ExecutingThing."
Termination	"A transformation that occurs in an EnabledState and then (immediately) enters an ActiveState, with no intervening states. It is the first step of an Execution transformation. It is effected by a TriggeringLaw and occurs in an ExecutingThing."
AnyTransformation	"The most general transformation, defined by a TransformationLaw. It occurs in any ChangingThing and transforms a MutableState into AnyState (which may or may not be mutable)."

Appendix C

UEML 2.0 Template

Here is provided the UEML Template Tutorial.

UEML Template Tutorial

Document type:	Working document
Domain/Task/Topic:	DEM / UEML / Approaches
Version:	1
Date:	2006-01-26
Status:	1st iteration
Author:	Andreas L Opdahl
Additional contributors:	Giuseppe Berio and Petia Wohed have contributed to the explanations and examples.
Distribution list:	DEM
Document history:	

Explanation of the Template Entries

An entry-by-entry guide is given to the four template sections: The *Preamble*, *Presentation*, *Representation* and *Open Issues* parts. The *Representation* part will be described in most detail.

File name:

When starting to describe a new language, it may be most practical to have each construct description in a separate file, using the following naming convention:

UEML_construct_<LANGUAGE_ACRONYM>_<LANGUAGE_VERSION>_
<DIAGRAM_TYPE>_<CONSTRUCT_NAME>_<VERSION_NUMBER>

Again, <LANGUAGE_VERSION> and <DIAGRAM_TYPE> can be left out.

Example (*Composition* in the UML):
The file name would have been
UEML_construct_UML_2.0_ActivityDiagrams_Composition_1.doc.

As the number of files grows, is may become more convenient to place all the modelling constructs in the same language or diagram type in a single file, using the following naming conventions:

UEML_constructs_<LANGUAGE_ACRONYM>_<LANGUAGE_VERSION>_
<VERSION_NUMBER>

UEML_constructs_<LANGUAGE_ACRONYM>_<LANGUAGE_VERSION>_
<DIAGRAM TYPE>_<VERSION_NUMBER>

Again, <LANGUAGE_VERSION> can be left out.

Example (UML):
If all UML 2.0 class diagram definitions were in the same file, the file name would have been:
UEML_constructs_UML_2.0_ClassDiagram_1.doc.

If all UML 2.0 definitions were in the same file, the file name would have been:

UEML_constructs_UML_2.0_1.doc.

Document title:

The title of a modelling construct description should have the form:

<LANGUAGE_ACRONYM> <LANGUAGE_VERSION>
<DIAGRAM_TYPE>: <CONSTRUCT_NAME>

The <LANGUAGE_VERSION> and <DIAGRAM_TYPE> can be left out.

Example (*Composition* in the UML):
A description of the Composition construct in UML version 2.0 will have the following title:

UML 2.0 Class Diagrams: Composition

Initial comments: Start the template description with any overall comments you may have about the construct or your description of it. It is particularly nice to start by quoting a part of the construct description found in some "official" source, like the language definition.

The Preamble Section

The *Preamble* provides general information about the modelling construct, such as the construct name, relations to other constructs, the diagram types and language it belongs to, as well as acronyms and external resources.

Builds on:

Some modelling constructs are variants or refinements of other modelling constructs in the same language. If so, you should not have to duplicate the description of the other construct. Instead, you can just refer to the other construct by name here. Later entries in the template can then be filled in with

"As for <OTHER_CONSTRUCT> but with … replacing/modifying …"

If the entry for this construct is *identical* to the entry for the other construct, the entry can be left empty or filled with

"As for <OTHER_CONSTRUCT>."

The *builds on* field here in the *Preamble* indicates that *both* the *Presentation* and the *Representation* of this construct description builds on another construct. If only the presentation or only the Representation part of this description builds on another, there are additional, more specific, *builds on* entries in those two sections.

The *builds on* entries in this and later sections should only be used within the same language. To reuse construct definitions across languages, use good old cut-and-paste. (Maybe we will loosen this up in the future when tools become available to manage construct definitions, but for the moment we want to avoid the additional complexity of having to maintain cross-language references.)

Example (*Composition* in the UML):
Builds on the *Property* construct (in UML 2.0, both aggregation and composition are variants of properties, identified by an AggregationKind attribute of the Property metaclass.)

It is possible to describe modelling constructs that *build on* more than one other construct in the same language. *Builds on* is transitive, however, so it may not be necessary to list them all.

Built on by:

This entry is the inverse of the previous one, *Builds on*. Here you can list all the constructs that *build on* this construct.

Also transitive, not necessary to fill in all.

Construct name:

This entry is for the official name of the modelling construct.

Example (*Composition* in the UML):
Composition

Alternative construct names:

This entry is for other names that are sometimes used for the construct, whether formally or informally.

Example (*Composition* in the UML):
Composition relationship, composite AggregationKind, "black diamond", "black-diamond aggregation"

Related, but distinct construct names:

This entry can be used to mention related constructs with which this construct should not be confused.

Example (*Composition* in the UML):

- Aggregation: Use of aggregation in the UML is discouraged by some. The distinction between aggregation and composition is quite unclear at best. (See Barbier and Henderson-Sellers' work on this.)

Related terms:

The language (see below) may use additional terms that are not names of modelling constructs but that are nevertheless useful or necessary to fully understand and talk about the language.

Example (*Composition* in the UML):
None.

Language:

To which language does this construct belong? The language is described by a name, possibly a version and acronym and preferably an official URI or other reference.

Example (*Composition* in the UML):

- Unified Modelling Language, version 2.0 (UML 2.0), <www.uml.org>.

Diagram type:

Many modelling languages are organised as a family of *diagram types*, which can overlap to some extent. If so, this entry is used to list the diagram types in which this modelling construct can be

used. Each diagram type is specified by a name and possible an acronym. Here and elsewhere, entries in the template can be marked TBD and otherwise commented.

Example (*Composition* in the UML):

- Class diagrams. *(TBD: Check for other diagram types.)*

The Presentation Section

This section describes the *visual presentation* of the modelling construct. Presentation issues include lexical information (such as icons, line styles), syntax (how this and other modelling constructs connect in diagrams and repositories) and pragmatics (in particular layout conventions). The *Presentation* section of the template is kept informal at this stage, because we want to focus on the following *Representation* section, which we believe is more difficult.

Earlier versions of the template used the term “Syntax” to describe this section. We have renamed it because it is not only about syntax, but also about lexicals and pragmatics.

Builds on:

Whereas the *builds on* entry in the *Preamble* indicated that both the *Presentation* and R*epresentation* of one construct builds on another, this entry only applies to presentation. It can override *builds on* in the *Preamble* with a more specific construct (that builds on the construct listed in the *Preamble*), but it cannot otherwise contradict the *Preamble*.

Example (*Composition* in the UML):

The end points of a composition can have role and multiplicities like regular associations in UML. *(TBD: Also need to investigate relationship to links, and check the new 2.0 metamodel.)*

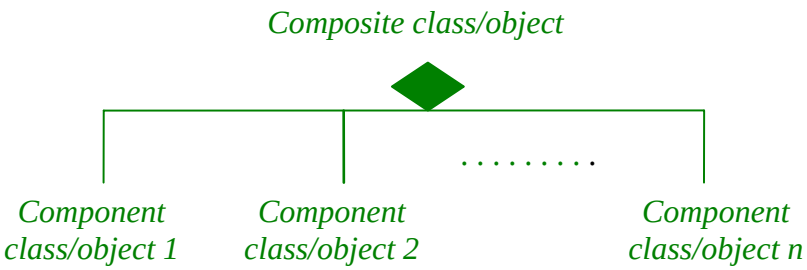
Built on by:

This entry is the inverse of the previous one, “Builds on”.

Icon, line style, text:

- *For a construct that is represented as a node:* This entry is used to show the corresponding icon and, if necessary, also to explain how it changes with the values of its attributes.
- *For a construct that is represented as an edge:* This entry is used to show the corresponding line style and, if necessary, also to explain how it changes with the values of its attributes.

Example (*Composition* in the UML):



A composition can have one or more component, and the graphic representation changes accordingly.

User-definable attributes:

This entry is for describing the attributes that can be specified for an instance of this construct. For example, *user-definable attributes* are found in the metamodels or textual specifications of a modelling language. Many of them also show up in modelling tools that implement the language, e.g., when a box or arrow that represents this construct in a diagram is ”opened”.

Typically, many of the attributes listed here will be listed again in the *property entry* of the *Representation* section. But the attributes listed here and the properties listed there will not match completely:

- Some user-definable attributes and relationships are only presentational. They are not intended to correspond to a property of the domain that is represented. An example is languages that offer some kind of *TBD* markings of models and model elements.
- Some represented properties are not presentational. They are implicit in the language or model and are never shown in diagrams or stored in a repository. An example is process languages where one element can interrupt or terminate the activity of other elements, but where the underlying communication relationship is never shown.

Example (*Composition* in the UML):
Each end of a composition can have a role name and a multiplicity, and the aggregation can have a name. *(TBD: There may be more here.)*

If an attribute is complex, its sub-attribute can be specified as other attributes, but indented.

Convention: Here are later, a reverse cardinality is written immediately following the forward cardinality, separate with the word **rev**: (0:1 **rev** 1:n).

All cardinalities express restrictions in the context of a single instance of the construct being described.

Relations to other constructs:

This entry is for describing the language-defined relations that can be specified for an instance of the construct. This can be both relations to other modelling constructs *within the same diagram type* or relations to constructs in *other diagram types* in the same language. For example, *user-definable relations* are (syntactic) diagram connexions that a box or arrow (representing this construct) may have with other constructs. But there can also be used-definable relationships that are not shown in diagrams, but maintained in a repository of some sort (for example, the relationship between a DFD process and its decomposion diagram). For each relation, cardinality constraints and preferrably also role names must be given.

As for user-definable attributes, many of the attributes listed here will be listed again in the *property entry* of the *Representation* section but, again, they will not match completely.

At the moment there is no fixed format for specifying constraints in the presentation part of the template.

Example (*Composition* in the UML):
- A composition must be related to one *composite* class or object.
- A composition must be related to at least one and can be related to more than one *component* class or object.
- The composite and components must either all be classes or all be objects.

- *TBD: If compositions are associations/links, can they be part of association classes and link objects?*

Convention: Here are later, a reverse cardinality is written immediately following the forward cardinality, separate with the word **rev**: (0:1 **rev** 1:n).

Diagram layout conventions:

This entry is for describing informal, tacit, social conventions that affect how this modelling construct is used when drawing diagrams. Typically, these conventions are not described in language definitions or made explicit in textbooks, but conveyed by example. Nevertheless, they must be captured to the extent possible so that the best possible tool support can eventually be provided for the UEML and for other languages.

The entry is not used to collect explicit or official *layout rules*, such as rules approved by a standards body. If it is a formal layout rule, it belongs under entries like “Icon/line style” or “Relationships to other constructs”.

Example (*Composition* in the UML):
The composite class/object should be positioned over its components when possible (and to the left of its components when not). The components should be aligned horizontally below (or vertically to the right of) the composite.

Other usage conventions:

This entry is for describing any other social conventions that affect how this modelling construct is used. Like the previous entry, this entry is work in progress.

The Representation Section

This section describes that instantiation level, classes, properties and kinds of dynamic behaviour that this modelling construct can be used to represent. Most existing modelling definitions describe representation using text only, so the entries in this template section usually cannot be filled in without interpreting the language definition, looking “between the lines” to some extent, and also looking at examples of how the language is used in practice.

This section will not fit well for constructs that define datatypes or particular values. If you encounter such a construct, just make a comment about it and fill in the rest the best you can.

Earlier versions of the template used the term “Semantics” to describe this section. We have renamed it because the “semantics” involves more than mere representation (or reference).

Builds on:

This entry is for describing whether the representation of this construct based on the representation of some other construct? In contrast to the *builds on* entries in the *Preamble* and *Presentation* sections, this entry only applies to representation. It can override *builds on* in the *Preamble* with a more specific construct (that builds on the *Preamble*-construct), but it cannot otherwise contradict the *Preamble* entry.

Example (*Composition* in the UML):
None

Built on by:

This entry is the inverse of the previous one, “Builds on”. Here you can list all the constructs that *Build on* the construct you are defining.

Instantiation level:

This entry describes whether the modelling construct is intended to represent either

- classes/properties/states/events/processes/etc. at the *type level*, or
- things/properties/states/events/processes/etc. at the *instance level*, or
- *both*.

Importantly, we distinguish between *instantiation* on the one hand and *execution*, or "temporal *unfolding*", on the other hand. So if a modelling construct is intended to represent some event or process, this entry describes whether the construct represents an event or process *in some individual thing* (at the instance level) or *generally in a class of things* (at the type level).

Example (UML Class/object diagrams):
Objects and Classes in the UML are almost identical constructs, but Objects are intended only to represent things at the instance level, Classes only at the type level. Similar with Properties and Attribtues.

Example (*Composition* in the UML):
Both type and instance level. (There is Composition in the UML both of Objects and of Classes.)

When we describe the other *Representation* entries, we will mostly talk about representing “classes of things” and “types of properties”. This applies most directly to modelling constructs *type level*. For modelling constructs at the *instance level*, when we talk about representing “classes” and “property types”, we really mean “*things* of those classes” and “*properties* of those types”.

Classes of things:

The class entry describes *which classes of things* that the modelling construct is intended to represent. Along with the *property* and *behaviour entries*, described next, this entry is the most central entry in the *Representation* section. Along with the property and behaviour entries, it is also the most complex.

A central idea behind the template is that most modelling constructs somehow represent one or more *classes of things* in the world. Even if the *primary* purpose of a construct is to represent certain properties, states, events or processes, the construct implicitly also represents a property of, state of, event in or process in *one or more classes of things*, each of them playing a particular *role* in the context of the construct. The class entry is there to describe these roles and the classes that play them.

There is one class entry for each such role played by a class of things in the context of a modelling construct. Each class entry has the following sub-entries:

- *A role name*. The same class can play several different roles in the context of a modelling construct. For example, uni-directional associations in the UML, which connect two or more classes of *AllThings*, and where some classes play the role of *navigable* end whereas others play the *non-navigable role*. Each role must be given a distinct name, which is local within the construct definition. A self-explanatory role name makes the construct definition easier to understand.

- *A cardinality constraint.* A role can be played one or more times in the context of a construct definition. For example, a UML composition can only have one (cardinality 1:1) class playing the *whole* role and only one class playing the *part* role, but a uni-directional UML association can have one or more (cardinality 1:n) classes playing both the *navigable* and *non-navigable end* roles.
- *An ontology class name*. This is the name of the ontological class that is playing the role. Bunge's ontology and the BWW-model suggest (sometimes indirectly) some top-level ontological classes that can be used, such as *AllThings*, *ChangingThings*, *ActiveThings*, *ActedOnThings*, *CompositeThings*, *ComponentThings*, *SystemThings*, *SystemComponentThings* (some of the major classes are described at the end of this tutorial). If none of the existing ones fit the class role you are trying to describe, you can propose your own classes or refine existing ones,
- *An explanation*. If the role name is not self-explanatory, the class role can be explained more elaborately here.
- *Also represented by*. The various constructs in a modelling language must be *connected* both syntactically and semantically. Syntactical connexions were described the Presentation part of the template, whereas semantic connexions between this modelling construct and others in the same language must be described here. The assumption is that modelling constructs are connected because they overlap semantically, most often because they represent the same types of properties, but sometimes also because they represent the same class(es) of things. If this modelling construct is connected to other constructs in the same language because they represent the same class(es), these other constructs are listed here.

Example (*Composition* in the UML):
Classes:
- *Role name:* "Whole"
- *Cardinality* 1:1.
- *Class:* The class of CompositeThings.
- *Also played by:* Class, object.

- *Role name:* "Part"
- *Cardinality* 1:1.
- *Class:* The class of ComponentThings.
- *Also played by:* Class, object.

An alternative short-hand notation is
- <MINCARD:MAXCARD> "<ROLENAME>" **played by** <ONTOLOGYCLASS>. (<EXPLANATION>)
  Also played by <OTHER_CONSTRUCTS>.

Example (*Composition* in the UML):
Classes:
- 1:1 "Whole" played by the class of CompositeThings.
  Also played by Class, Object.
- 1:1 "Part" played by the class of ComponentThings.
  Also played by Class, Object.

Convention: Here are later, role names are always "Quoted" and "WrittenWithCapitalisedWords".

Properties (and relationships):

The property entry describes *which properties* that the modelling construct is intended to represent, if any. The entry is also used to describe *which relationships* the construct is intended to represent, if any, because relationships are just properties that belong to more than one class (or thing).

As for classes, a central idea behind the template is that most modelling constructs somehow represent one or more *types of properties* of classes in the world. Even if the primary purpose of a construct is to represent classes, states, events or processes, it represents a class, state, event or process that involves *one or more property types*, each of them playing a particular *role* in the context of the construct. The property entry is there to describe these roles and the types of properties that play them. (Nevertheless, some constructs may clearly be intended to represent property-less classes. If so, leave this entry blank.)

Some properties are *complex*, because they have other properties as *sub-properties*. Some complex properties restrict the values of their sub-properties. Such properties are *law properties*.

There is one property entry for each role played by a type of property in the context of a modelling construct. Each property entry has the following sub-entries:

- *A role name*. A property type can play several different roles in the context of a modelling construct. Each role must be given a distinct name, which is local within the construct definition. A self-explanatory role name makes the construct definition easier to understand.
- *A cardinality*. How many times does the modelling construct represent the property in this role?
- *An ontology property name*. This is the name of the ontological property that is playing this role. Bunge's ontology and the BWW-model provide concepts that suggest some top-level ontological properties (some of the most important properties are described at the end of this tutorial). But you are also free to propose your own property types, or refine existing ones, if none of the ones already in the ontology fit your modelling construct.
- *An optional type* and *default value*. In particular, *complex properties* (i.e., properties with sub-properties, see below) do not have to have *type* and *default value* sub-entries. And *law properties* (i.e., properties that restrict the values of other properties, see below) have *law sub-entries* instead of *type/default value* entries.
- *Belongs to*. Each property playing this role must belong to one or more classes that also play roles in this construct definition. There is one *belongs to* sub-entry for each such class. Each *belongs to* sub-entry has the following sub-sub entries:
    - *Class role name*. Which class role does this property role belong to?
    - *Cardinality*. How many classes playing this role may this property role belong to?
    - *Reverse cardinality*. How many properties playing this role may the class role possess in the context of this construct definition?
- *An explanation*. Sometimes, a well-chosen role name suffices to explain a particular property role. If not, a more elaborate explanation can be given separately.
- *Also represented by*. The various constructs in a modelling language must be *connected* both syntactically and semantically. Syntactical connexions were described in the Presentation part of the template. Semantic connexions between this modelling construct and others in the same language are described here. The assumption is modelling constructs are connected because they overlap semantically, most often because they represent the same types of properties. If other modelling constructs in the same language as the current construct also represent this type of property, the constructs are listed in this sub-entry.

Example (*Composition* in the UML):

- The WholePartRelation property.
  - Belongs to each "Whole". Cardinality 1:1. Reverse cardinality 1:1.
  - Belongs to each "Part". Cardinality 1:1. Reverse cardinality 1:1.

Some properties have *sub-properties* of their own. They are specified almost like regular properties, but instead of one or more *belongs to* sub-entries, they have one or more *sub-property of* sub-entries. There is one *sub-property of* sub-entry for each sub-property. Each *sub-property of* sub-entry has the following sub-sub entries:

- *Property role name*. Of which other property role is this one a sub-property?
- *Cardinality*. How many classes playing this role may this property role belong to?
- *Reverse cardinality*. How many properties playing this role may the class role possess in the context of this construct definition?

It is possible for a property to *belong to* one class role and, at the same time, be a *sub-property of* a property of another class role.

In other words, *sub-property of* is almost identical to *belongs to*, except that the former lists a class role and the latter a property role. The sub-property relationship is transitive, so only *direct* sub-property relations need to be entered into the template. Indirect relations can then be inferred. Accordingly, if property *p* is a sub-property of *q* and *q* belongs to class *c*, then *p* also belongs to *c*. So we do not need to list which class roles a sub-property belongs to.

Finally, some properties represent *laws* that restrict the values of other properties. Such properties are called *law properties*. A law property must always be complex, i.e., it must have sub-properties, and the law can only restrict the values of those properties that are its sub-properties. A *state law* property restricts the combinations of values of its sub-properties *at the same time*, whereas a *transformation law* restricts the combinations of values *over time*.

Instead of a *type* and a *default value* sub-entry, a law property has a *law* sub-entry, which describes the law either informally or formally. An informal law description is just a textual explanation of what the law does. A formal law description expresses the law in some formal language, using the law property's sub-properties as parameters in the law expression. To express state laws, a regular first-order predicate logic-like language is sufficient. To express transformation laws, a language with some temporal notion is needed, at least with concepts to express pre versus post states. However, expressing transformation laws formally is less important, because they are detailed in the following *behaviour entry* anyway.

Behaviour:

What kind of behaviour is the construct intended to represent? Some construct are apparently not intended to "represent behaviour" at all. They just represent the *existence* of certain classes/properties/etc. Other constructs represent particular *states* or *events* or chains of alternating states or events, i.e., *processes*. Hence, this entry specifies if the modelling construct is intended to represent *existence* (previously called *lifetime*), a *state*, an *event* or a *process*.

- If the construct just represents existence, this concludes the specification.
- If the construct represents a state, that state must be described further, as an invariant over the *property roles* defined in the previous entry.
- If the construct represents an event, the *to* and *from states* must be specified according to the previous bullet point.
- If the construct represents a process, the *states* and *events* in the process must be specified according to the last two bullet points.

Example (*Composition* in the UML):

Existence.

In simple cases like this, you can express the state as a formal expression over Properties defined already in the “Properties (and relationships)” entry. But in most cases it is more informative (and easier at this early stage) to explain the state using simple text.

This way of defining events is probably overly simple. A more detailed way is to describe the transition that causes the event:

- SomeTransition
  - From: SomeState
  - To: SomeOtherState
  - Trigger: The event that triggers the transition
  - Condition: A condition described using simple text (later a formal expression)
  - Action: An action undertaken when the transition occurs

Finally, just like classes and properties, behaviour entries may connect modelling constructs semantically:

Also represented by. The various constructs in a modelling language must be *connected* both syntactically and semantically. Syntactical connexions were described in the Presentation part of the template. Semantic connexions between this modelling construct and others in the same language are described here. The assumption is that modelling constructs are connected because they overlap semantically, for example because they represent the same state, event or process. If other modelling constructs in the same language as the current construct also represent this type of behaviour, the constructs are listed in this sub-entry.

Modality (permission, recommendation etc):

We usually think of enterprise models as *assertions* of *facts* about a domain, e.g., assertions that something is the case or is not the case in the enterprise. But some model elements may instead state that *someone wants something to be the case*, or *that someone is not permitted to do something*, or that *someone knows something is the case*¸ or that *something will be the case some time in the future*. We call such statements *modal* (as opposed to *regular*) *assertions*, i.e., we use the term "modal" pretty much in the "modal logic" sense.

This entry is for describing the intended modality of a modelling construct. For most of the constructs we will encounter, the modality entry will just be "Regular assertion", i.e., the constructs *assert facts* about the domain. A fact states something to *be the case*, i.e., that the domain is in a particular state or not or that the domain is undergoing a particular event/process or not.

Other constructs may represent *permissions* or *mandates*. A permission means that something *may* or *may not* be the case. An obligation means that *must* or *must not* be the case. Both modalities can only be used about modelling constructs that represent at least one actor, i.e., at least one of whose represented classes is an *actor class* or a subclass thereof. Because premissions and obligations are socially constructed, the same or another *intentional actor class* or subclass may also be identified as the *issuer* of the permission or obligation.

Other constructs may represent *intentions* or *obligations*, i.e., whether someone *wants to* or *must make* something the case. Hence both modalities can only be used about modelling constructs that represent at least one intentional actor and, furthermore, one of those intentional actors must be identified as the *holder* of the belief or the knowledge.

Other constructs may represent *beliefs* or *knowledge*, i.e., whether someone *believes* or *knows* something to be the case. Hence both modalities can only be used about modelling constructs that represent at least one intentional actor and, furthermore, one of those intentional actors must be identified as the *holder* of the belief or the knowledge.

Because the modality entry is, to some extent, work still in progress, we do not formalise it further at this stage. Below is a summary of the modalities discussed above. As usual, you are free to propose others:

- Regular assertion
  - No particular restrictions on represented classes.
- Permission
  - At least one of the represented classes must be an *actor class*.
  - One represented intentional *actor class* may be identified as the *permission issuer*.
- Mandate
  - At least one of the represented classes must be an *actor class*.
  - One represented *intentional actor class* may be identified as the *mandate issuer*.
- Intention
  - At least one of the represented classes must be an *intentional actor class*, which is identified as the *intention holder*.
- Obligation
  - At least one of the represented classes must be an *intentional actor class*, which is identified as the *obligation holder*.
- Belief
  - At least one of the represented classes must be an *intentional actor class*, which is identified as the *belief holder*.
- Knowledge
  - At least one of the represented classes must be an *intentional actor class*, which is identified as the *knowledge holder*.

Note that we often create "modal enterprise models", e.g., enterprise models representing something we want to be the case some time in the future. Most modelling languages can be used like that, and it is different from what we want to capture in this entry.

Example (*Composition* in the UML):
Regular assertion.

Example (*Soft-goals* in i*):
Intention of the class of *IntentionalActors*.

The Open Issues Section

Everything else goes here, most importantly unresolved issues that do not fit well other places in the template.

This concludes the entry-by-entry guide is given to the four template sections.

The Common Ontology

An alternative would be to reuse one of the many upper ontologies that already exist. Indeed, it is a good idea to import parts of existing ontologies when they are clearly needed, to avoid re-doing existing work. However, reusing an existing ontology is also problematic:

- The imported parts would have to fit with the core constructs of Bunge's ontology and the BWW-model and with the template. For example, it would have to distinguish clearly between classes/things, properties, behaviour and modalities. At least at the most general levels, few existing ontologies satisfy these requirements. However, more specific parts of existing ontologies, i.e., parts that deal only with classes/things, with properties and with behaviour can still be imported and reused.
- Existing ontologies are typically very large, and they will contain lots of of the classes, properties, states, events and processes that are not used to describe any modelling construct. So parts of existing ontologies should only be imported when they are needed. Otherwise, the common ontology will become much too complex.

Examples of Ontology Classes (Unfinished)

This section provides examples of general ontology classes, to aid filling in the *class entry* of the template. Remember that these are only examples. If none of them fit, you can propose your own classes, refining one or more of the existing ones.

The following list is based on an earlier preliminary analysis UML version 1.4, presented in Opdahl & Henderson-Sellers, J Database Manag., 15(2), 39–73, April-June 2004. The entries taken from this paper is marked (*) below. The list is therefore slanted towards the UML, although a few other classes have been added to complete the picture:

AllThings*(*)*
The most general class of things, characterised by the *ability to associate*.

Pieces*, aka PieceThings*
Bunge introduces a fundamental distinction between *pieces* and *fields*. It may help to think of pieces as lumps of matter and of fields such as electromagnetic fields, although this analogy is too restrictive. The characteristic property of pieces is their *ability to compose*, a more specific property than the ability to associate. According to Bunge, piece composition is idempotent, associative and commutative.

Fields*, aka FieldThings*
The characteristic property of fields is their *ability to juxtapose*, another more specific property than the ability to associate. The template approach does not attempt to deal with fields because they are rarely covered by enterprise and IS modelling languages. The BWW-model does not mention them. Below, we therefore continue talking about various classes of "Things", although all the classes we describe could more precisely have been named "Pieces". (The reason we mention pieces and fields at all here is to give a more complete picture of the highest levels of Bunge's ontology.)

AssociatedThings*(*) (subclass of AllThings)*
An associated thing is *actually associated* with (not only *able to associate* with) one or more other things. The characteristic property of this class is *mutual property*.

ChangingThings*(*) (subclass of AllThings)*
According to Bunge, all things change and all change occurs in a thing. So the class of changing things may be the same as the class of all things. For our purpose, however, it is useful to distinguish between them because some modelling constructs are intended to represent things as changing and some as unchanging. The characteristic property of this class is *ability to change*.

(A more specific property would be to be *actually changing*, but this seems less useful for describing modelling constructs: There are many modelling constructs intended to represent that a

particular change *may occur* (given certain circumstances), but not many constructs that represent that a particular change *always occurs* (regardless of circumstance) in a domain. An additional more specific class can be introduced later to account for this if needed.)

CompositeThings*(*) (subclass of AllThings)*
The characteristic property of composite things is the *part-whole relation*, a particular type of property. A composite thing must be in a part-whole relation with one or more components, i.e., it must be on the whole-side of the relation. Almost all real things are composite, but not all modelling constructs are intended to represent them as such, so this is still a very useful class.

Components*, aka ComponentThings(*) (subclass of AllThings)*
The characteristic property of component things is also the *part-whole relation*. But to be a component, the thing must be on the part-side of the relation. Again, almost all real things are components, but this class is still very useful because not all modelling constructs are intended to represent those things as components.

Systems*, aka SystemThings(*) (subclass of CompositeThings and ChangingThings)*
Systems are composite things whose components are *coupled* (see below). Specifically, a composite is a system if its components cannot be bi-partitioned so that no component in the first partition is coupled to a component in the second. The characteristic properties of systems are most likely *emergent behaviour* and/or *interaction* among its components. (The details need to be sorted out here.)

Subsystems*, aka SubsystemThings(*) (subclass of Systems)*
A subsystem is a system all of whose components are also components of another, larger system. The characteristic property of a subsystem is the *subsystem of* property.

CoupledThings*(*), aka SystemComponentThings(*) (subclass of AssociatedThings and ChangingThings)*
Two things are coupled iff their histories depend on one another, i.e., iff the history of one of the things would have been different had the other thing not existed. The characteristic property of coupled things is the *coupling* property, a particular type of (and perhaps identical to) *binding mutual property*.

In (*), a distinction was made between *CoupledThings* and *SystemComponentThings*, but the two classes are ontologically identical: All things compose, so whenever we have two or more things, we can always talk about their composite. And when two or more things are coupled, their composite must be a system (by definition of system). So all coupled things are system components. (The converse — that all system components are coupled — is also true by definition of system.)

SystemEnvironmentThings *(subclass of AllThings)*

ProposedSystems*, aka ProposedSystemThings(*) (subclass of Systems)*

SubsystemsOfTheProposedSystem*(*) (subclass of Subsystems and ProposedSystems)*

ActiveThings *(subclass of CoupledThings)*

ThingsThatActOnTheProposedSystem *(subclass of SystemEnvironmentThings)*

ActedOnThings *(subclass of CoupledThings)*

***ExchangedThings**, aka CommunicatedThings (subclass of ActedOnThings)*

***ContainerThings** (subclass of AllThings)*

***ThingsThatTrackTime** (subclass of AllThings)*

***ThingsThatKnowTheTime**, aka ThingsThatKnowAbsoluteTime (subclass of AllThings)*

(To be elaborated and completed.)

Examples of Property Types (Unfinished)

This section provides examples of general property types, to aid filling in the *property entry* of the template. Remember that these are only examples. If none of them fit, you can propose your own property types, refining one or more of the existing ones. The list below is derived from the *characteristic (or defining) properties* of the example classes presented in the previous section.

Ability to associate

Ability to compose

Ability to juxtapose

Ability to track time

Content (preceded by Mutual property)

Knowledge of time

(To be elaborated and completed.)

Change List

Version 1.2 is the first separate version of this tutorial. Previous versions were integrated with versions 1.0 and 1.1 of the template itself.

Planned changes: There are two major refinements of the template to be done (though not necessarily in the course of INTEROP):

- *Make it XML-based.* This refinement should be done some time during 2005 and should produce version 2.0 of the template. It will require going through the definitions once more, but should not require any additional analysis. We will look for ways to make that job as simple as possible.
- *Make it mathematically formal.* This will require introducing (at least):
    - property *types* and *values* (and probably *defaults* too),
    - formal descriptions of static and dynamic laws (e.g., using 1. order predicate calculus, Z, OCL ort another formal notation. The parameters to such a formal expression will be the law property's sub-properties),
    - formal state descripitions,
    - formal descripitions of event conditions, triggers and actions.

  This task is maybe for 2006, or maybe it is beyond INTEROP.

Additional changes are:

- Formalise the *Presentation* section. The template work has so far focussed on the *Representation* part, which is believed to be most difficult. But the *Presentation* section must be formalised too, to the extent possible.
- Tool support must be provided.
- The layout and social conventions must be investigated in further detail.

References

[1] Andreas L. Opdahl and Brian Henderson-Sellers. A Template for Defining Enterprise Modelling Constructs. *Journal of Database Management (JDM)* 15(1). Idea Group Publishing, 2004.

[2] Andreas L. Opdahl and Brian Henderson-Sellers. Template-Based Definition of Information Systems and Enterprise Modelling Constructs. Chapter 6 in *Ontologies and Business System Analysis*, Peter Green and Michael Rosemann (eds.). Idea Group Publishing, 2005.

Appendix D

ARIS meta models

This appendix groups the meta models used for the analysis of ARIS. The explanations are inspired by [Sch99].

D.1 Function View

In ARIS concept, functions are regarded as individual views of business processes. The function structures and target strutures are depictied in the following meta model (Figure D.1).

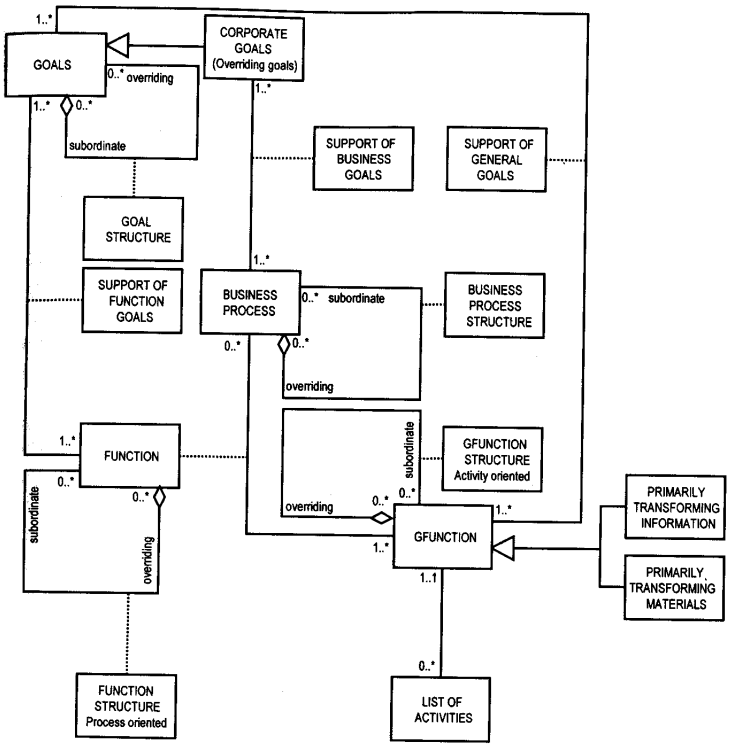


Figure D.1: Meta model depicting function structures and target structures (from [Sch99])

For our analysis, we focused on the left side of this meta model, Figure D.1. Functions are defined as operations applied to objects for the purpose of supporting one or more goals. Goals can be linked with one another, with a subordinate goal supporting several overriding goals. Thus the structure of goals linked with one another in a network results in a *:* association within the class *goals*. General functions belonging to the appropriate business processes are allocated to the latter by the association class *function*. The classes *function* and *goals* are associated together by a (1..*):(1..*) link.

D.2 Organization View

The business oriented organization view describes the hierarchical organization, i.e., the organizational units with their communication and reporting relationships among them. The hierarchical organization is depicting in the following meta model (Figure D.2).

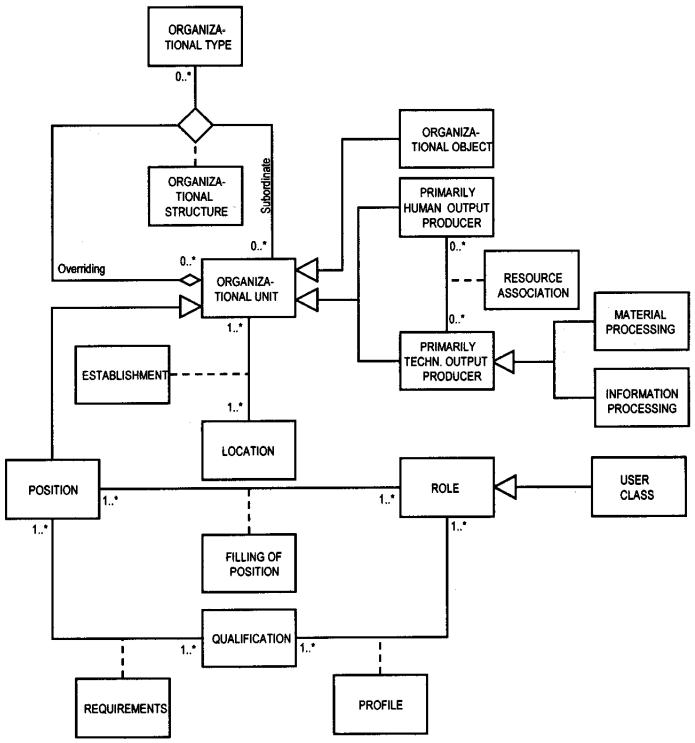


Figure D.2: Meta model of hierarchical organization (from [Sch99])

Organizational units to be modeled are in center of this meta model (Figure D.2) and keep all our attention. They make up the class *organizational unit*. Geographic distribution of organizational units is achieved by the relation *regional office* (*establishment*) in between the classes *location* and *organizational unit*. The smallest unit within the organizational structure is the "position", usually defined by the function amount that an individual employee can handle. The class *position* contains the individual positions in the form of instances. The classes *position* and *organizational unit* are linked by a specialization/generalization relation.

D.3 Data View

The data view includes the description of data objects, manipulated by functions. Various objects with varied granularity are deployed in the data view. Data objects consolidated in the data view have various roles (see Figure D.3). They describe the events and messages controlling the business process, i.e., the control flow. Furthermore, the environment status of the business process is illustrated by data objects. In functions processing information, function output, i.e., the deliverable, is represented by documents and thus by data. Due to the fact that output results in the ARIS output view are described individually, there is some overlapping.

Data that can be broken down into more detailed elements is called macro data. The macro data objects are illustrated in the meta model shown in Figure D.4. The class *data cluster* is a synonym for the modeling construct *Environmental data*.

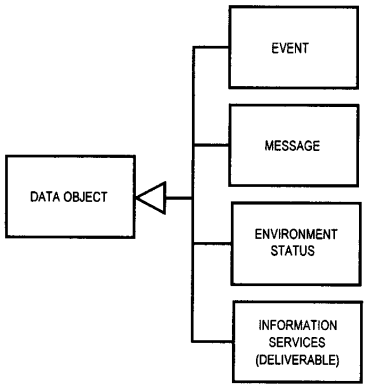


Figure D.3: Data object roles (from [Sch99])

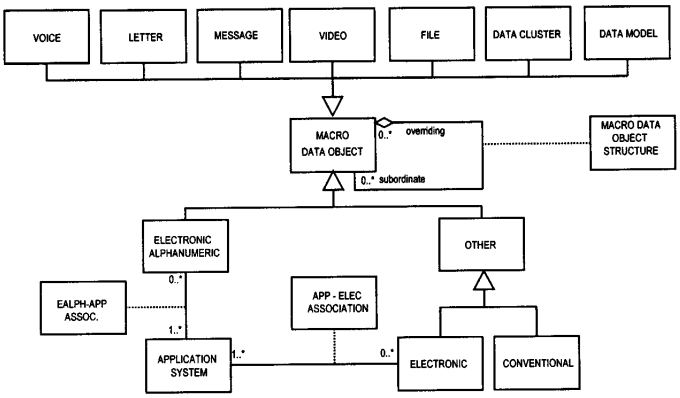


Figure D.4: Metal model of macro data objects (from [Sch99])

D.4 Output View

Output is the result of processes, with the demand for input driving the execution of processes. Output is heterogeneous and can be used at various levels of detail. They also use "output" in the same context as "product". The output view is depicting in the meta model shown in Figure D.5.

The outputs make up the class *output/input or product*, along with their respective sub-classes *material output/input* and *services*. The class *services* has two sub-classes, *information services* and *other services*. Information services are modeled as data objects and thus become a part of the ARIS data view, although it is necessary to define the status of the data object precisely if the output status is to be defined. Information services are identified in the output view as the input or output, respectively, of a function. There is some overlap between the output view and the data view.

D.5 Control View

This section describes the relationships between the views. We will introduce meta models for describing relationship pair between the views.

Relationships between Functions and Organization

Figure D.6 shows the basic relationship between a business function and an organizational unit. An organizational unit is responsible for one or more functions.

The functions are represented by the class *function* and the organizational unit by the class *organizational unit*. They are linked by a (1..*):(1..*) association. An organizational unit can be reponsible for several functions and a function can be under the responsibility of several organizational units. If

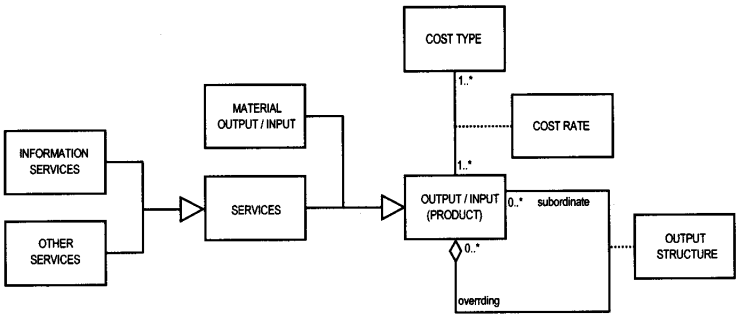


Figure D.5: Metal model of the output view (from [Sch99])

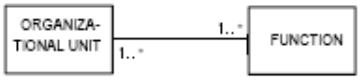


Figure D.6: Metal model of the relationship between organizational units and functions (adapted from [Sch99])

multiple organizational units are participating in a function, it is possible to detail the type of their participation by adding information as to which organizational units are responsible, actively involved or only associated.

Relationships between Functions and Data

When deriving the data view from the general ARIS business process model, two different kinds of relationships between functions and data can be distinguished :

- Functions process data by transforming input data into output data.
- Events are (data) status modifications, created by functions. Messages indicate that the status modifications (event) have been detected, pass them on to successive functions and then activate them.

Data are allocated to their respective functions. If functions are to be modeled independently of data, an *:* association is established by functions allocations. Thus, a function can be allocated to multiple data objects. Figure D.7 depicts the meta model of the function allocation diagram. The class *information object* represents the data's.

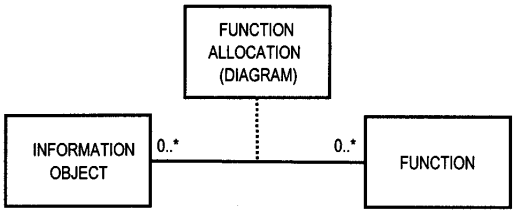


Figure D.7: Meta model function allocation diagram (from [Sch99])

The event and message control in an EPC is depicted by the following metal model (see Figure D.8).

In information systems, events are represented by data (updates). Therefore, in Figure D.8, the class *event* is shown as a sub-class of the class *information object*. The logical relationships between events are modeled by an association between *logical type of relationship* (such as *AND*, *OR*) and *event*. Direct relationships with a function, i.e., relationships not explicity included in message control, are displayed by the associations *event-function triggering* and *event-function generation*. *function* can be launched

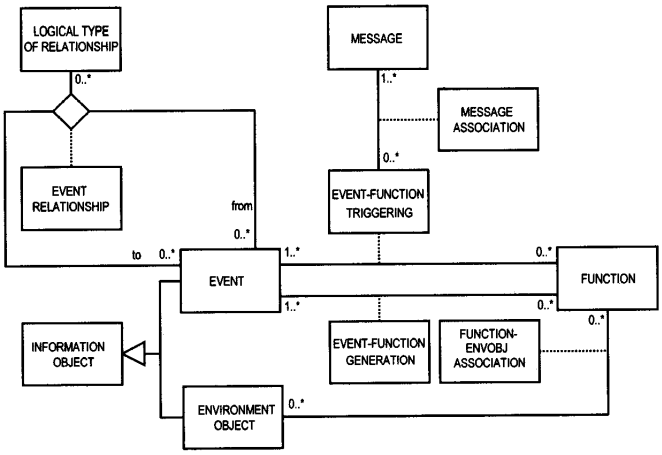


Figure D.8: Meta model of event and message control in an EPC (from [Sch99])

by one or multiple events. At the same time, one function can create multiple events. An event can be the result of multiple functions. The class *message* is linked by an association with the association structure *event-function triggering*. The messages illustrated as letter symbols can be attached with various properties. The meta model in Figure D.8 should be enhanced accordingly. We don't describe this enhanced meta model which is too detailed for being used for our analysis.

Relationships between Functions and Output

The term "output" encompasses material output and services, including information services. Relationships with the function view consist of functions transforming input into output by means of processing. Figure D.9 shows the meta model of the relationships between the functions and the output.

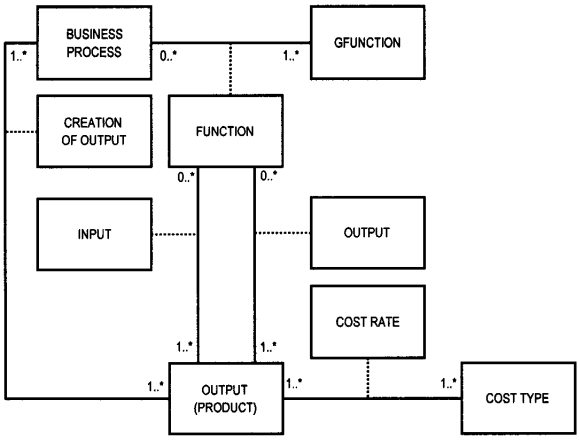


Figure D.9: Meta model for changing output types after function processing (from [Sch99])

The class *output* is associated to the class *function* by means of two associations *input* and *output*. A function can create (0:*) output(s) and can process (0:*) input(s).

Appendix E

BPMN meta models

This appendix groups the meta models used for the analysis of BPMN.

E.1 Meta model centered on the events

In the meta model E.1, the event can be specified into three types of constructs: the Start (SE), Intermediate (IE) and End (EE) events. Let us begin with the SE, it is the source of at least one Sequence Flow. One of this flow can only have one SE as source. This event can also be the target of multiple Message Flows, but this flow can only have a SE as source. The EE is the target of one or more Sequence Flow and a Sequence Flow can have a EE as target. This event can also be the source of multiple Message Flow and a Message Flow can have an EE as source. To close this part, we have to explain the IE relationships. Firstly, the IE can be the target of a Message Flow and a message flow can have a IE as target. It is exactly the same situation with the Sequence Flow except that the IE is the source of a Sequence Flow and a Sequence Flow can have an IE as source. Beside of it, an IE is attached to the boundary of an activity and an activity can have an IE attached to its boundary.

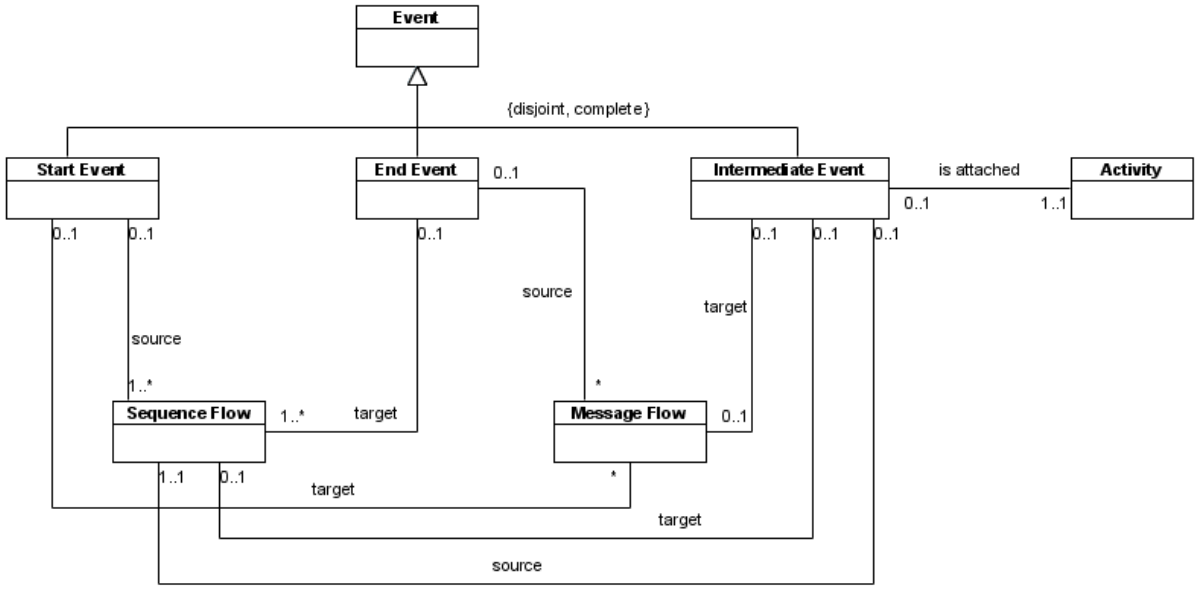


Figure E.1: Meta model centered on the events

E.2 Meta model centered on the artifacts

The last meta model concerns the artifacts (Figure E.2). We did not describe the Group construct because it is just a graphical notation that we did not use, not too much significant. In fact, it is just a graphical object that groups elements informally. Thus, this meta model describes the Data Object and the Text Annotation. A Data Object can be associated to a Sequence Flow or a Message Flow by an Association. A Data Object can also be the target/source of an Association linked to a Flow Object. As regards the Text Annotation, it can be linked to Flow Objects, Connecting Objects (Sequence Flow and Message Flow) and Swimlanes by an Association.

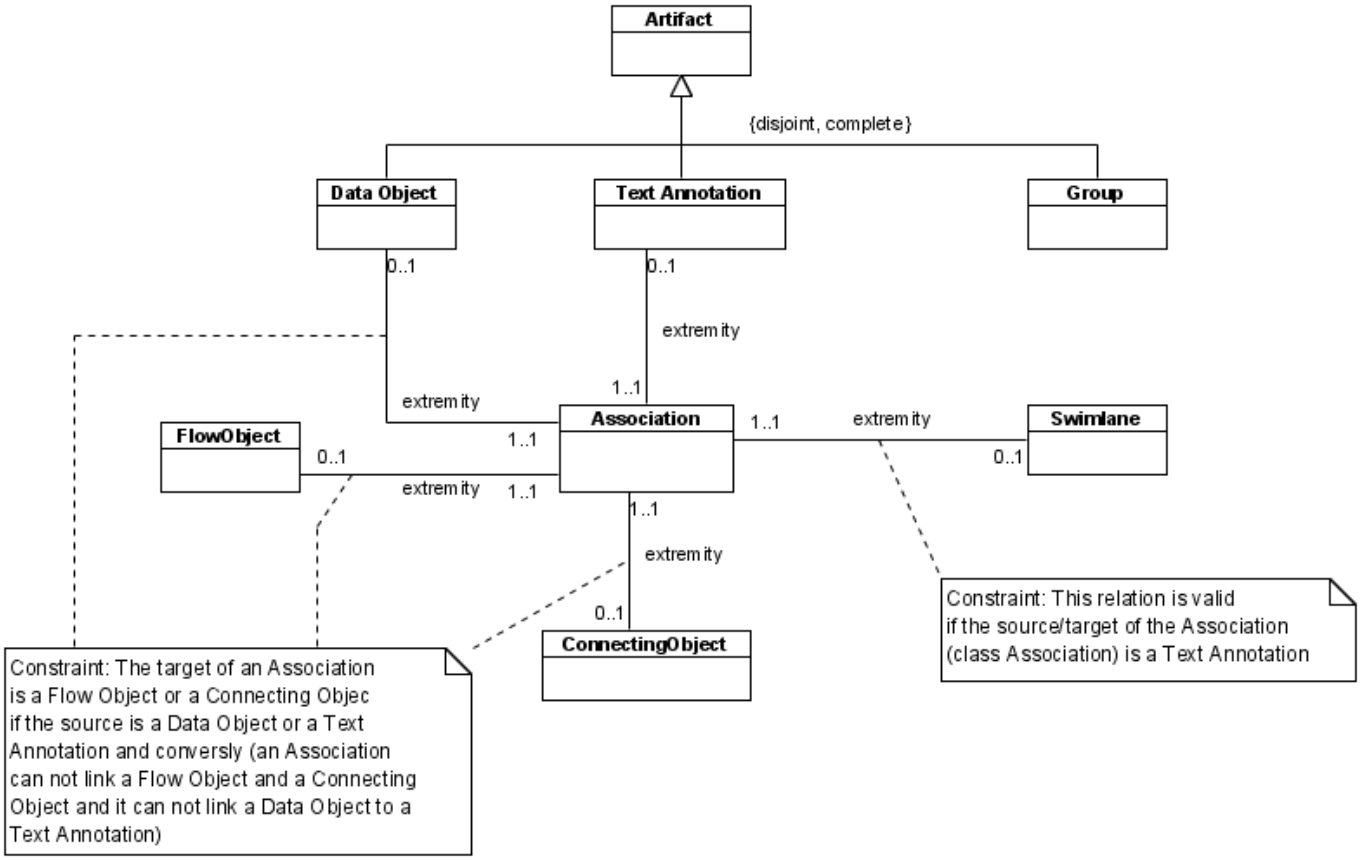


Figure E.2: Meta model centered on the artifacts

E.3 Meta model centered on the gateways

In Figure E.3, we can see that there are five types of gateways, the parallel (PG), the Data-Based Exclusive (DBEG), the Event-Based Exclusive (EBEG), the Inclusive (IG) and the Complex (CG) Gateways. A PG can be the source or the target of multiple Sequence Flows and a Sequence Flow can have a PG as source or as target. The situation is the same for the DBEG, the CG and the IG. For the EBEG, it is the source of at least two different Sequence Flows and a Sequence Flow can have a EBEG as source or as target. The EBEG can also be the target of multiple Sequence Flow and a Sequence Flow can have a EBEG as target. It is necessary to mention that a gateway cannot have several imputs and several outputs at the same time.

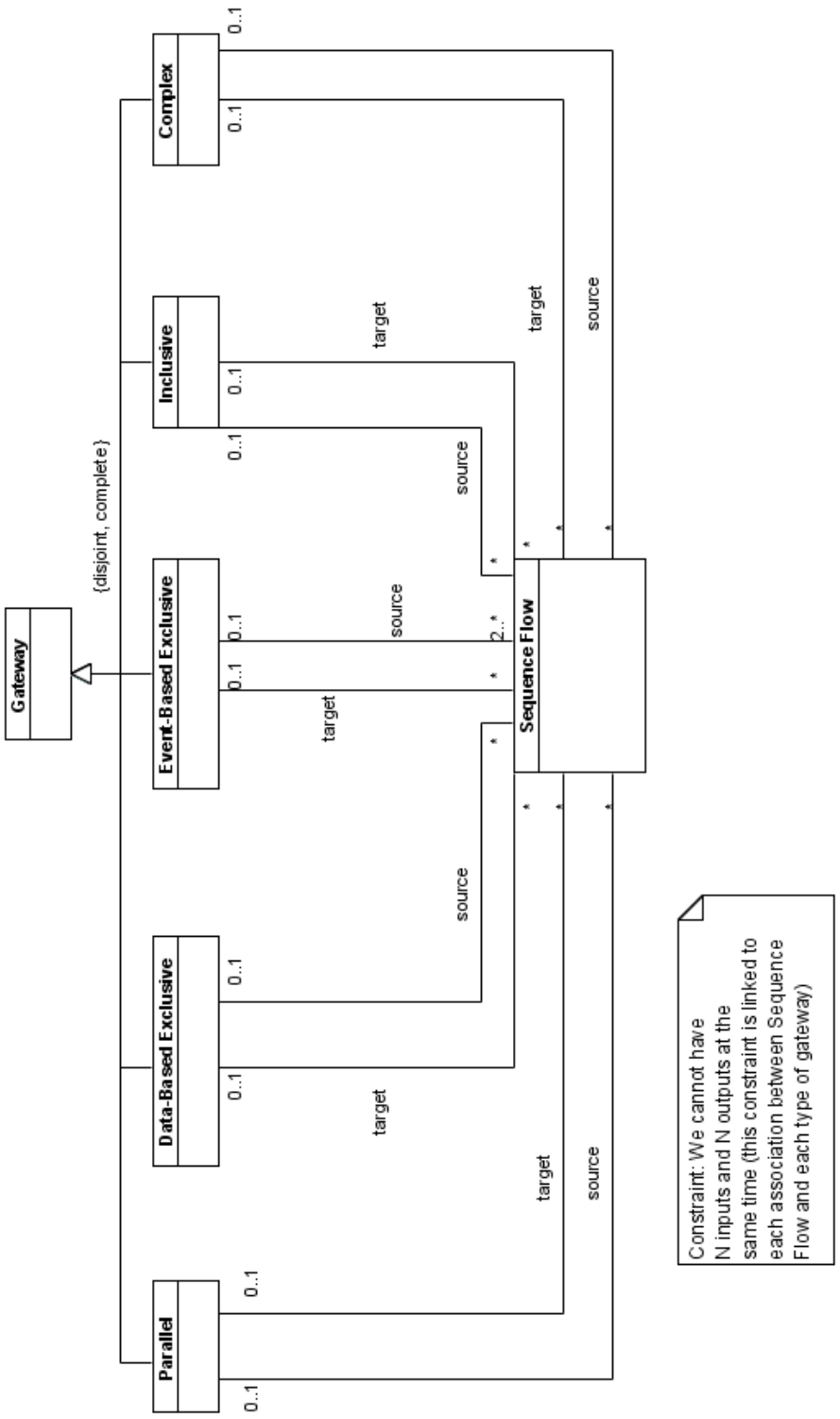


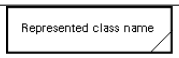
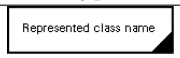
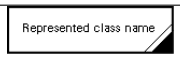
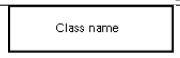
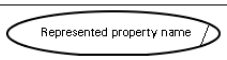
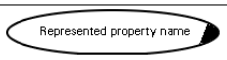
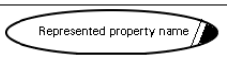
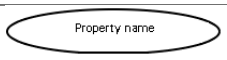
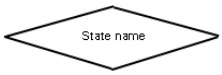
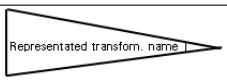
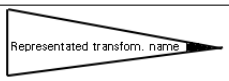
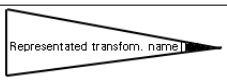
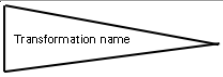
Figure E.3: Meta model centered on the gateways

Appendix F

UEML 2.0 graphical representation standard

The Table F.1 and Figure F.1 show the UEML 2.0 graphical representation standard. This notation is the legend for the Figures 8.2, 8.3, 8.4, 8.5, 9.2, 9.3, 9.4, 9.5, 11.1, 11.2, 13.1 and 13.2.

Table F.1: UEML 2.0 graphical representation standard

	Instance	Type	Both	Common ontology
Things				
Properties				
States				
Transformations				

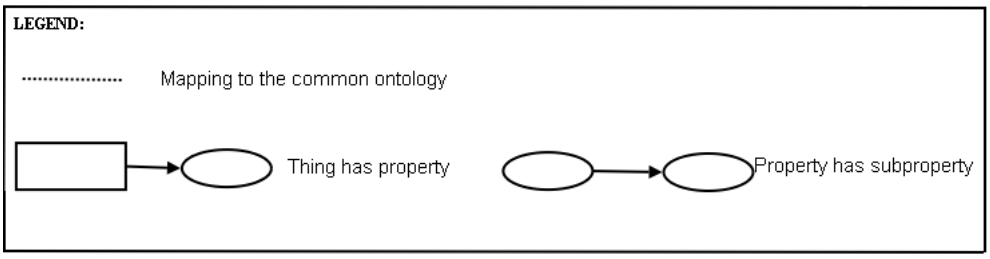


Figure F.1: Legend of Figures 8.2, 8.3, 8.4, 8.5, 9.2, 9.3, 9.4, 9.5, 11.1, 11.2, 13.1 and 13.2

Appendix G

BPMN Legend

Figure G.1 illustrates the legend of the BPMN constructs used for Figures 6.12, 12.7, 12.8, 12.9, 12.10, 12.11 and 12.12.

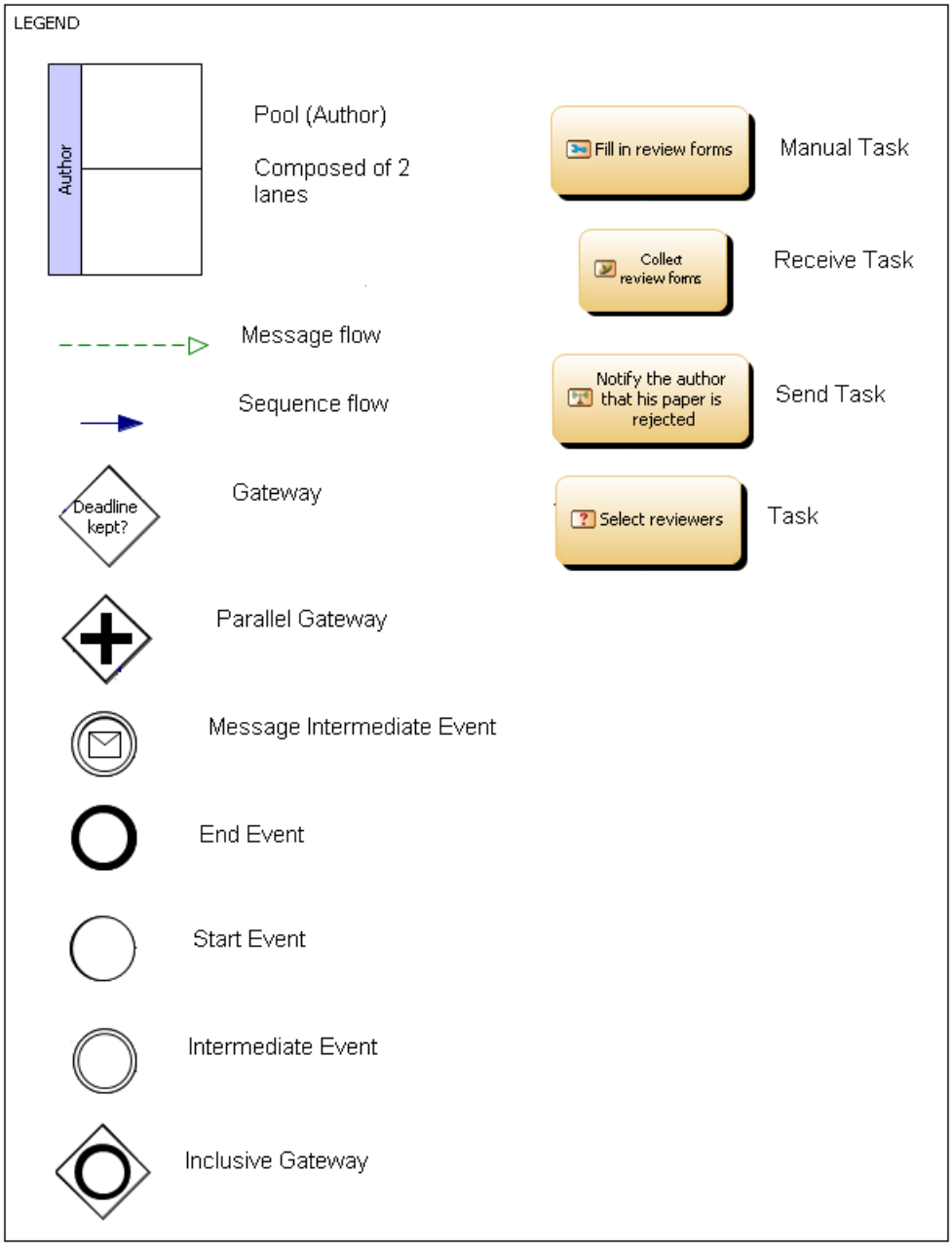


Figure G.1: Legend of Figures 6.12, 12.7, 12.8, 12.9, 12.10, 12.11 and 12.12

Appendix H

Analysis of ARIS

Here is grouped all the filled templates for each analysed modelling constructs of ARIS. The templates are the templates revised after the case study, i.e. the mappings are corrected according to the modifications presented in Chapter 12.

H.1 Organizational unit

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Organizational Unit
- **Alternative construct names :** Responsible entity
- **Related construct names :** None
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** An organizational unit is represented by an oval with a vertical line on the left and the name inside. The organizational unit is linked to the function by the organizational flow.

Figure H.1: Organizational Unit

- **User-definable attributes :**
  - Name : the name of the organizational unit
  - Type of participition : the type of participition of the organizational unit in the context of a function
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - can be responsible for 1..N function(s)
  - can be located at 0..N location(s)
  - can be composed of 0..N position(s)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Both, instance and type level
- **Classes of things:**
  - Participant representing the organizational unit.
    - Role name: "Organization unit"
    - Cardinality: 1-1
  - RoleHolder representing the role performed by individual people.
    - Role name: "Position"
    - Cardinality: 0-N
  - HumanOutput representing the human output
    - Role Name : "HumanOutput"
    - Cardinality : 0-N
- **Properties and relationships:**
  - Name representing the name of the organizational unit.
    - Role name: "Name"
    - Cardinality: 1-1
    - Belongs to each "Organizational unit". Cardinality 1:1. Reverse cardinality 1:1.
  - Name representing the name of the position.
    - Role name: "NamePosition"
    - Cardinality: 1-1
    - Belongs to each "Position". Cardinality 1:1. Reverse cardinality 1:1.
  - Name representing the name of the human output.
    - Role name: "NameHumanOutput"
    - Cardinality: 1-1
    - Belongs to each "HumanOutput". Cardinality 1:1. Reverse cardinality 1:1.
  - RegulerProperty representing the type of participation of the organizational unit
    - Role name: "Participation"
    - Cardinality: 0-1
    - Belongs to each "Organizational unit". Cardinality 1:1. Reverse cardinality 0:1.
  - FunctionLaw representing the function under the responsability of the organizational unit.
    - Role name: "Function"
    - Cardinality: 1-N
    - Belongs to each "Organizational unit". Cardinality 1:1. Reverse cardinality 1:N.
  - Location representing the location of the organizational unit.
    - Role name: "Location"
    - Cardinality: 0-N
    - Belongs to each "Organizational unit". Cardinality 1:1. Reverse cardinality 0:N.
  - PartWholeRelation representing the components of the organizational unit.
    - Role name: "RelationToPart"
    - Cardinality: 0-N
    - Belongs to each "Organizational unit" in role of whole. Cardinality 1:1. Reverse cardinality 0:N.

  - Belongs to each "Position" in role of part. Cardinality 1:1. Reverse cardinality 1:1.
* PartWholeRelation representing the components of the organizational unit.
  - Role name: "RelationToPartHumanOutput"
  - Cardinality: 0-N
  - Belongs to each "Organizational unit" in role of whole. Cardinality 1:1. Reverse cardinality 0:N.
  - Belongs to each "HumanOutput" in role of part. Cardinality 1:1. Reverse cardinality 1:1.

- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

H.2 Position

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Position
- **Alternative construct names :** None
- **Related construct names :** Organizational Unit
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** A Position is represented by a rectangle with a vertical line at the left side.

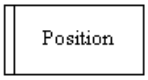


Figure H.2: Position

- **User-definable attributes :**
  - Name : the name of the position
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - can compose 1..N organizational unit(s)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Both, instance and type level
- **Classes of things:**
  - Participant representing the organizational unit.
    - Role name: "Organization unit"
    - Cardinality: 1-1
  - RoleHolder representing the role performed by individual people.
    - Role name: "Position"
    - Cardinality: 1-1
- **Properties and relationships:**
  - Name representing the name of the position.
    - Role name: "Name"
    - Cardinality: 1-1
    - Belongs to each "Position". Cardinality 1:1. Reverse cardinality 1:1.
  - Name representing the name of the organizational unit.
    - Role name: "NameOrganizationalUnit"
    - Cardinality: 1-1
    - Belongs to each "OrganizationalUnit". Cardinality 1:1. Reverse cardinality 1:1.
  - PartWholeRelation representing the composite of the position.
    - Role name: "RelationToWhole"
    - Cardinality: 1-1
    - Belongs to each "Organizational unit" in role of whole. Cardinality 1:1. Reverse cardinality 1:1.
    - Belongs to each "Position" in role of part. Cardinality 1:1. Reverse cardinality 1:1.
  - Responsability representing the responsability of the position (roleHolder).
    - Role name: "Responsability"
    - Cardinality: 1-1
    - Belongs "Position". Cardinality 1:1. Reverse cardinality 1:1.
- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

H.3 Function

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Function
- **Alternative construct names :** Activity, process, transformation, task
- **Related construct names :** None
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** A function is represented by a soft rectangle.

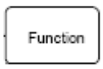


Figure H.3: Function

- **User-definable attributes :** Name : the name of the function
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - can be under responsibility of 1..N organizational unit(s)
  - can be executed by 0..N software(s)
  - can use 0..N machine resource(s)
  - can use 0..N computer hardware
  - can create 0..N output(s)
  - can process 0..N input(s)
  - can support 0..N goal(s) (can be controlled by 0..N goal(s))
  - can be processed by 0..N human output
  - can transform 0..N environmental data's
  - can be triggered by 1..N event(s) (by 1..N message(s) of the events)
  - can produce 1..N event(s)
- **Diagram layout conventions :** None
- **Other usage conventions :** None
- **Comments :** For the relations to other constructs, we take a general cardinality (0-N) because nothing was specified in [Sch98] and [Sch99].

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Both, instance and type level
- **Classes of things:**
  - Participant representing the organizational unit
    - Role name: "OrganizationalUnit"
    - Cardinality: 1-1
  - ExecutingThing representing the application software
    - Role name: "Software"
    - Cardinality: 0-N
  - MachineResource representing the machine resource
    - Role Name : "Machine"
    - Cardinality : 0-N
  - ComputerHardware representing the computer hardware
    - Role Name : "ComputerHardware"
    - Cardinality : 0-N
  - HumanOutput representing the human output
    - Role Name : "HumanOutput"
    - Cardinality : 0-N
  - ReactiveThing && InputOutputThing representing the environmental data
    - Role name: "EnvironmentalData"
    - Cardinality: 0-N
  - Repository && InputThing representing the output.
    - Role name: "TargetOutput"
    - Cardinality: 0-N
  - Repository && OutputThing representing the output.
    - Role name: "SourceOutput"
    - Cardinality: 0-N
- **Properties and relationships:**
  - FunctionLaw representing the law that makes the function happen
    - Role name: "FunctionLaw"
    - Cardinality: 1-1
    - Belongs to "OrganizationalUnit". Cardinality 1:N. Reverse cardinality 1:1.
  - IsActive representing the fact that the function is active or non-active
    - Role name: "IsActive"
    - Cardinality: 1-1
    - Subproperty of "FunctionLaw". Cardinality 1:1. Reverse cardinality 1:1.
  - ParticipationLaw representing the participation of the human in the creation of an output of the function
    - Role name : "Participation"
    - Cardinality : 1-1
    - Belongs to "HumanOutput". Cardinality : 1:1. ReverseCardinality : 1:1

    - SubProperty of "FunctionLaw". Cardinality : 1:1. ReverseCardinality : 1:N
- ApplicationLaw representing the fact that the application software executes a part of the function
    - Role name: "ApplicationLaw"
    - Cardinality: 1-1
    - Subproperty of "FunctionLaw". Cardinality 1:1. Reverse cardinality 1:N.
    - Belongs to "Software". Cardinality 1:1. Reverse cardinality 1:1.
- UseLaw representing the fact that a part of the function uses a computer hardware
    - Role name: "UseLawComputerHardware"
    - Cardinality: 1-1
    - Belongs to "ComputerHardware". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "FunctionLaw". Cardinality 1:1. Reverse cardinality 1:N.
- UseLaw representing the fact that a part of the function uses a machine resource
    - Role name: "UseLawMachine"
    - Cardinality: 1-1
    - Belongs to "Machine". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "FunctionLaw". Cardinality 1:1. Reverse cardinality 1:1.
- Law representing the goal
    - Cardinality : 0-N
    - Role Name : "Goal"
    - Subproperty of "FunctionLaw". Cardinality : 1:1. ReverseCardinality : 0:N
- InteractionRelation representing the information flow between the function and the environmental data
    - Cardinality : 1-1
    - Role Name : "InformationFlow"
    - Subproperty of "FunctionLaw". Cardinality : 1:1. ReverseCardinality : 1:1
    - Belongs to "EnvironmentalData". Cardinality : 1:1. ReverseCardinality : 1:1
- Flow representing the incoming flow of a function
    - Cardinality : 0-1
    - Role Name : "IncomingFlow"
    - Subproperty of "FunctionLaw". Cardinality : 1:1. ReverseCardinality : 0:1
- Flow representing the outgoing flow of a function
    - Cardinality : 0-1
    - Role Name : "OutgoingFlow"
    - Subproperty of "FunctionLaw". Cardinality : 1:1. ReverseCardinality : 0:1
    - Subproperty of "Goal". Cardinality : 1:1. ReverseCardinality : 1:1
- FlowContent representing the content of the flow of functions
    - Cardinality : 1-1
    - Role Name : "OutputFlow"
    - Belongs to "TargetOutput". Cardinality : 1:1. ReverseCardinality : 1:1
    - Subproperty of "OutgoingFlow". Cardinality : 1:1. ReverseCardinality : 1:1
- FlowContent representing the content of the flow of functions
    - Cardinality : 1-1
    - Role Name : "OutputFlowInput"
    - Belongs to "SourceOutput". Cardinality : 1:1. ReverseCardinality : 1:1
    - Subproperty of "IncomingFlow". Cardinality : 1:1. ReverseCardinality : 1:1

- **Behaviour :** Process
  Represented states :
  - "FunctioningState" played by ActiveState.
    - Defining property : "IsActive"
    - State Constraint : "IsActive == true"
  - "NonFunctioningState" played by InactiveState.
    - Defining property : "IsActive"
    - State Constraint : "IsActive == false"

  Represented transformations :
  - "TriggeringFunction" played by Triggering
    - **From state :** NonFunctioningState
    - **To state :** FunctioningState
    - **Trigger :** An event occurs (An input is created)
    - **Condition :** All inputs are available
    - **Action :** Function realisation **effected by** FunctionLaw (TriggeringLaw)
  - "TerminationFunction" played by Termination
    - **From state :** FunctioningState
    - **To state :** NonFunctioningState
    - **Trigger :** An event occurs (An output is created)
    - **Condition :** All outputs are available
    - **Action :** Function termination **effected by** FunctionLaw (TerminationLaw)
  - "NotAllInputAvailable" played by AnyTransformation
    - **From state :** NonFunctioningState
    - **To state :** NonFunctioningState
    - **Trigger :** An input is created
    - **Condition :** Not all inputs are available
    - **Action :** Function waits to have all inputs available **effected by** FunctionLaw
  - "NotAllOutputAvailable" played by AnyTransformation
    - **From state :** FunctioningState
    - **To state :** FunctioningState
    - **Trigger :** An output is created
    - **Condition :** Not all outputs are available
    - **Action :** Function continues to be executed to produce all outputs **effected by** FunctionLaw
- **Modality (permission, recommendation, . . .) :** Regular assertion

H.4 Logical operator "And"

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Logical operator "And"
- **Alternative construct names :** None
- **Related construct names :** Logical operator "Or","XOR"
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The logical operator "AND" is represented by a circle with the mathematical representation of the logical operator "AND" inside. The logical operator "AND" is linked to the event by the control flow.

Figure H.4: Logical operator "AND"

- **User-definable attributes :** None
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - can have 1..N input(s) (event or function or a logical operator "And" - "Or" - "XOR")
  - can have 1..N output(s) (event or function or a logical operator "And" - "Or" - "XOR")
- **Comments :** We can't have N inputs and N outputs at the same time.
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - CoupledThing representing the input of the logical operator "And".
    - Role name : "InputAnd"
    - Cardinality : 1-N
  - CoupledThing representing the output of the logical operator "And".
    - Role name : "OutputAnd"
    - Cardinality : 1-N
- **Properties and relationships:**
  - MutualLaw representing the logical operator "And".
    - Role name: "And"
    - Cardinality: 1-1
    - Belongs to :
      - "InputAnd". Cardinality 1:N. Reverse cardinality 1:1.
      - "OutputAnd". Cardinality 1:N. Reverse cardinality 1:1.
  - CouplingRelation representing the coupling (ending) between input and the logical operator "And".
    - Role name: "EndingInput"
    - Cardinality: 1-1
    - Belongs to "InputAnd". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "And". Cardinality 1:1. Reverse cardinality 1:N.
  - CouplingRelation representing the coupling (ending) between output and the logical operator "And".
    - Role name: "EndingOutput"
    - Cardinality: 1-1
    - Belongs to "OutputAnd". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "And". Cardinality 1:1. Reverse cardinality 1:N.
- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

H.5 Logical operator "Or"

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Logical operator "Or"
- **Alternative construct names :** None
- **Related construct names :** Logical operator "And", "XOR"
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The logical operator "OR" is represented by a circle with the mathematical representation of the logical operator "OR" inside. The logical operator "OR" is linked to the event by the control flow.

Figure H.5: Logical operator "OR"

- **User-definable attributes :** None
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - can have 1..N input(s) (event or function or a logical operator "And" - "Or" - "XOR")
  - can have 1..N output(s) (event or function or a logical operator "And" - "Or" - "XOR")
- **Comments :** We can't have N inputs and N outputs at the same time.
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - CoupledThing representing the input of the logical operator "Or".
    - Role name : "InputOr"
    - Cardinality : 1-N
  - CoupledThing representing the output of the logical operator "Or".
    - Role name : "OutputOr"
    - Cardinality : 1-N
- **Properties and relationships:**
  - MutualLaw representing the logical operator "Or".
    - Role name: "Or"
    - Cardinality: 1-1
    - Belongs to :
      - "InputOr". Cardinality 1:N. Reverse cardinality 1:1.
      - "OutputOr". Cardinality 1:N. Reverse cardinality 1:1.
  - CouplingRelation representing the coupling (ending) between input and the logical operator "Or".
    - Role name: "EndingInput"
    - Cardinality: 1-1
    - Belongs to "InputOr". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "Or". Cardinality 1:1. Reverse cardinality 1:N.
  - CouplingRelation representing the coupling (ending) between output and the logical operator "Or".
    - Role name: "EndingOutput"
    - Cardinality: 1-1
    - Belongs to "OutputOr". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "Or". Cardinality 1:1. Reverse cardinality 1:N.
- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

H.6 Logical operator "XOR"

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Logical operator "XOR"
- **Alternative construct names :** None
- **Related construct names :** Logical operator "And", "Or"
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The logical operator "XOR" is represented by a circle with the mathematical representation of the logical operator "XOR" inside. The logical operator "XOR" is linked to the event by the control flow.

Figure H.6: Logical operator "XOR"

- **User-definable attributes :** None
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - can have 1..N input(s) (event or function or a logical operator "And" - "Or" - "XOR")
  - can have 1..N output(s) (event or function or a logical operator "And" - "Or" - "XOR")
- **Comments :** We can't have N inputs and N outputs at the same time.
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - CoupledThing representing the input of the logical operator "XOR".
    - Role name : "InputXOR"
    - Cardinality : 1-N
  - CoupledThing representing the output of the logical operator "XOR".
    - Role name : "OutputXOR"
    - Cardinality : 1-N
- **Properties and relationships:**
  - MutualLaw representing the logical operator "XOR".
    - Role name: "XOR"
    - Cardinality: 1-1
    - Belongs to :
      - "InputXOR". Cardinality 1:N. Reverse cardinality 1:1.
      - "OutputXOR". Cardinality 1:N. Reverse cardinality 1:1.
  - CouplingRelation representing the coupling (ending) between input and the logical operator "Or".
    - Role name: "EndingInput"
    - Cardinality: 1-1
    - Belongs to "InputOr". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "Or". Cardinality 1:1. Reverse cardinality 1:N.
  - CouplingRelation representing the coupling (ending) between output and the logical operator "XOR".
    - Role name: "EndingOutput"
    - Cardinality: 1-1
    - Belongs to "OutputXOR". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "XOR". Cardinality 1:1. Reverse cardinality 1:N.
- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

H.7 Output

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Output
- **Alternative construct names :** Product
- **Related construct names :**
  - material output
  - services
  - information services
  - other services
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The output is represented by two boxes (rectangles) one included into the other

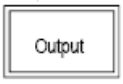


Figure H.7: Output

- **User-definable attributes :** Name : the name of the output.
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - is created by 1..N function(s)
  - can be input of 1..N function(s)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** type and instance level
- **Classes of things:**
  - Repository && InputThing representing the output.
    - Role name: "TargetOutput"
    - Cardinality: 0-1
  - Repository && OutputThing representing the output.
    - Role name: "SourceOutput"
    - Cardinality: 0-1
- **Properties and relationships:**
  - Name representing the name of the Output
    - Role name: "Name"
    - Cardinality: 1-1
    - Belongs to "TargetOutput". Cardinality 1:1. Reverse cardinality 1:1.
    - Belongs to "SourceOutput". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularProperty representing the attributes of the output
    - Role name: "Attributes"
    - Cardinality: 1-1
    - Belongs to "TargetOutput". Cardinality 1:1. Reverse cardinality 1:1.
    - Belongs to "SourceOutput". Cardinality 1:1. Reverse cardinality 1:1.
  - FunctionLaw representing the function
    - Role name: "FunctionLawInput"
    - Cardinality: 1-1
    - Belongs to "TargetOutput". Cardinality 1:1. Reverse cardinality 1:1.
  - FunctionLaw representing the function
    - Role name: "FunctionLawOutput"
    - Cardinality: 1-1
    - Belongs to "SourceOutput". Cardinality 1:1. Reverse cardinality 1:1.
  - Flow representing the incoming flow of a function
    - Cardinality : 1-1
    - Role Name : "Flow"
    - Subproperty of "FunctionLaw". Cardinality : 1:2. ReverseCardinality : 1:1
  - FlowContent representing the content of the flow of functions
    - Cardinality : 1-1
    - Role Name : "OutputFlow"
    - Subproperty of "Flow". Cardinality : 1:1. ReverseCardinality : 1:1
- **Behaviour :** State
  Represented state :
  - "OutputState" played by AnyState.
    - Defining property : "Attributes"
    - State Constraint : /
- **Modality (permission, recommendation, ...) :** Regular assertion

H.8 Material Output

Preamble section

- **Builds on :** Output construct
- **Built on by :** None
- **Construct name :** Material output
- **Alternative construct names :** None
- **Related construct names :**
  - Services
  - Other Services
  - Information Services
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The material output is represented by two boxes (rectangles) one included into the other

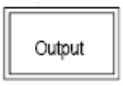


Figure H.8: Material Output

- **User-definable attributes :** Name : the name of the material output.
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - is created by 1..N function(s)
  - can be input of 1..N function(s)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** type and instance level
- **Classes of things:**
  - MaterialRepository && InputThing representing the material output.
    - Role name: "TargetMaterialOutput"
    - Cardinality: 0-1
  - MaterialRepository && OutputThing representing the material output.
    - Role name: "SourceMaterialOutput"
    - Cardinality: 0-1
- **Properties and relationships:**
  - Name representing the name of the material output
    - Role name: "Name"
    - Cardinality: 1-1
    - Belongs to "TargetMaterialOutput". Cardinality 1:1. Reverse cardinality 1:1.
    - Belongs to "SourceMaterialOutput". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularBooleanProperty representing the nature of the material output
    - Role name: "IsMaterial"
    - Cardinality: 1-1
    - Belongs to "TargetMaterialOutput". Cardinality 1:1. Reverse cardinality 1:1.
    - Belongs to "SourceMaterialOutput". Cardinality 1:1. Reverse cardinality 1:1.
  - FunctionLaw representing the function
    - Role name: "FunctionLawInput"
    - Cardinality: 1-1
    - Belongs to "TargetMaterialOutput". Cardinality 1:1. Reverse cardinality 1:1.
  - FunctionLaw representing the function
    - Role name: "FunctionLawOutput"
    - Cardinality: 1-1
    - Belongs to "SourceMaterialOutput". Cardinality 1:1. Reverse cardinality 1:1.
  - Flow representing the incoming flow of a function
    - Cardinality : 1-1
    - Role Name : "Flow"
    - Subproperty of "FunctionLaw". Cardinality : 1:2. ReverseCardinality : 1:1
  - MutualFlowContent representing the content of the flow of functions
    - Cardinality : 1-1
    - Role Name : "OutputFlow"
    - Subproperty of "Flow". Cardinality : 1:1. ReverseCardinality : 1:1
- **Behaviour :** State
  Represented state :
  - "MaterialOutputState" played by AnyState.
    - Defining property : "IsMaterial"
    - State Constraint : "IsMaterial == true"
- **Modality (permission, recommendation, ...) :** Regular assertion

H.9 Services

Preamble section

- **Builds on :** Output construct
- **Built on by :**
  - Information services construct
  - Other services construct
- **Construct name :** Services
- **Alternative construct names :** Non physical output
- **Related construct names :**
  - Material Output
  - Information Services
  - Other Services
- **Related terms :** As for <output>
- **Language :** As for <output>
- **Diagram types :** As for <output>

Presentation section

- **Builds on :** Output construct

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Both, instance and type level
- **Classes of things:**
  - Service && InputThing representing the service output.
    - Role name: "TargetService"
    - Cardinality: 0-1
  - Service && OutputThing representing the service output.
    - Role name: "SourceService"
    - Cardinality: 0-1
- **Properties and relationships:**
  - Name representing the name of the service
    - Role name: "Name"
    - Cardinality: 1-1
    - Belongs to "TargetService". Cardinality 1:1. Reverse cardinality 1:1.
    - Belongs to "SourceService". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularBooleanProperty representing the nature of the service output
    - Role name: "IsMaterial"
    - Cardinality: 1-1

  - Belongs to "TargetService". Cardinality 1:1. Reverse cardinality 1:1.
  - Belongs to "SourceService". Cardinality 1:1. Reverse cardinality 1:1.
* FunctionLaw representing the function
  - Role name: "FunctionLawInput"
  - Cardinality: 1-1
  - Belongs to "TargetService". Cardinality 1:1. Reverse cardinality 1:1.
* FunctionLaw representing the function
  - Role name: "FunctionLawOutput"
  - Cardinality: 1-1
  - Belongs to "SourceService". Cardinality 1:1. Reverse cardinality 1:1.
* Flow representing the incoming flow of a function
  - Cardinality : 1-1
  - Role Name : "Flow"
  - Subproperty of "FunctionLaw". Cardinality : 1:2. ReverseCardinality : 1:1
* RegularFlowContent representing the content of the flow of functions
  - Cardinality : 1-1
  - Role Name : "OutputFlow"
  - Subproperty of "Flow". Cardinality : 1:1. ReverseCardinality : 1:1

- **Behaviour :** State
  Represented state :
  - "ServiceState" played by AnyState.
    * Defining property : "IsMaterial"
    * State Constraint : "IsMaterial == false"
- **Modality (permission, recommendation, ...) :** As for <output>

H.10 Information services

Preamble section

- **Builds on :** Services construct
- **Built on by :** None
- **Construct name :** Information services
- **Alternative construct names :** Information services object
- **Related construct names :**
  - Other services
- **Related terms :** As for <Services>
- **Language :** As for <Services>
- **Diagram types :** As for <Services>

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** As for <Services>
- **User-definable attributes :** Name : the name of the information service
- **Relations to other constructs :** As for <Services>
- **Diagram layout conventions :** In a ARIS business process model, the Information Services are used as Output. Then, they have the same representation. In the information flow of the business process, we use the representation of the Information Services Object. The following representation represents the Information Services Object :
- **Other usage conventions :** As for <Services>

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** type and instance level
- **Classes of things:**
  - InformationService && InputThing representing the information service output.
    - Role name: "TargetInformationService"
    - Cardinality: 0-1
  - InformationService && OutputThing representing the information service output.
    - Role name: "SourceInformationService"
    - Cardinality: 0-1
- **Properties and relationships:**
  - Name representing the name of the information service

  - Role name: "Name"
  - Cardinality: 1-1
  - Belongs to each "TargetInformationService". Cardinality 1:1. Reverse cardinality 1:1.
  - Belongs to each "SourceInformationService". Cardinality 1:1. Reverse cardinality 1:1.
* RegularBooleanProperty representing the nature of the information service
  - Role name: "IsMaterial"
  - Cardinality: 1-1
  - Belongs to "TargetInformationService". Cardinality 1:1. Reverse cardinality 1:1.
  - Belongs to "SourceInformationService". Cardinality 1:1. Reverse cardinality 1:1.
* RegularBooleanProperty representing the nature of the information service
  - Role name: "IsInformation"
  - Cardinality: 1-1
  - Belongs to "TargetInformationService". Cardinality 1:1. Reverse cardinality 1:1.
  - Belongs to "SourceInformationService". Cardinality 1:1. Reverse cardinality 1:1.
* FunctionLaw representing the function
  - Role name: "FunctionLawInput"
  - Cardinality: 1-1
  - Belongs to each "TargetInformationService". Cardinality 1:1. Reverse cardinality 1:1.
* FunctionLaw representing the function
  - Role name: "FunctionLawOutput"
  - Cardinality: 1-1
  - Belongs to each "SourceInformationService". Cardinality 1:1. Reverse cardinality 1:1.
* Flow representing the incoming flow of a function
  - Cardinality : 1-1
  - Role Name : "Flow"
  - Subproperty of "FunctionLaw". Cardinality : 1:2. ReverseCardinality : 1:1
* RegularFlowContent representing the content of the flow of functions
  - Cardinality : 1-1
  - Role Name : "OutputFlow"
  - Subproperty of "Flow". Cardinality : 1:1. ReverseCardinality : 1:1

- **Behaviour :** State
  Represented state :
  - "InformationServiceState" played by AnyState.
    * Defining property : "IsMaterial"
    * Defining property : "IsInformation"
    * State Constraint :
      · "IsMaterial == false"
      · "IsInformation == true"
- **Modality (permission, recommendation, ...) :** As for <Services>

H.11 Other services

Preamble section

- **Builds on :** Services construct
- **Built on by :** None
- **Construct name :** Other services
- **Alternative construct names :** None
- **Related construct names :**
  - Information services
- **Related terms :** As for <Services>
- **Language :** As for <Services>
- **Diagram types :** As for <Services>

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** As for <Services>
- **User-definable attributes :** Name : the name of the other services
- **Relations to other constructs :** As for <Services>
- **Diagram layout conventions :** None
- **Other usage conventions :** As for <Services>

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** type and instance level
- **Classes of things:**
  - MaterialService && InputThing representing the other service output.
    - Role name: "TargetOtherService"
    - Cardinality: 0-1
  - MaterialService && OutputThing representing the other service output.
    - Role name: "SourceOtherService"
    - Cardinality: 0-1
- **Properties and relationships:**
  - Name representing the name of the other service
    - Role name: "Name"
    - Cardinality: 1-1
    - Belongs to each "TargetOtherService". Cardinality 1:1. Reverse cardinality 1:1.
    - Belongs to each "SourceOtherService". Cardinality 1:1. Reverse cardinality 1:1.

  - RegularBooleanProperty representing the nature of the other service
    - Role name: "IsMaterial"
    - Cardinality: 1-1
    - Belongs to "TargetOtherService". Cardinality 1:1. Reverse cardinality 1:1.
    - Belongs to "SourceOtherService". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularBooleanProperty representing the nature of the other service
    - Role name: "IsInformation"
    - Cardinality: 1-1
    - Belongs to "TargetOtherService". Cardinality 1:1. Reverse cardinality 1:1.
    - Belongs to "SourceOtherService". Cardinality 1:1. Reverse cardinality 1:1.
  - FunctionLaw representing the function
    - Role name: "FunctionLawInput"
    - Cardinality: 1-1
    - Belongs to each "TargetOtherService". Cardinality 1:1. Reverse cardinality 1:1.
  - FunctionLaw representing the function
    - Role name: "FunctionLawOutput"
    - Cardinality: 1-1
    - Belongs to each "SourceOtherService". Cardinality 1:1. Reverse cardinality 1:1.
  - Flow representing the incoming flow of a function
    - Cardinality : 1-1
    - Role Name : "Flow"
    - Subproperty of "FunctionLaw". Cardinality : 1:2. ReverseCardinality : 1:1
  - RegularFlowContent representing the content of the flow of functions
    - Cardinality : 1-1
    - Role Name : "OutputFlow"
    - Subproperty of "Flow". Cardinality : 1:1. ReverseCardinality : 1:1

- **Behaviour :** State
  Represented state :
  - "OtherServiceState" played by AnyState.
    - Defining property : "IsMaterial"
    - Defining property : "IsInformation"
    - State Constraint :
      - "IsMaterial == false"
      - "IsInformation == false"
- **Modality (permission, recommendation, ...) :** As for <Services>

H.12 Environmental data

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Environmental data
- **Alternative construct names :** Data cluster
- **Related construct names :**
  - Information services
  - Event
  - Message
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** An environmental data is represented by a rectangle and 2 vertical lines on each sides.

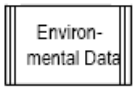


Figure H.9: Environmental data

- **User-definable attributes :** Name : the name of the environmental data
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - is transformed by 1..1 function (is input/output of 1..1 function)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** type and instance level
- **Classes of things:**
  - ReactiveThing && InputOutputThing representing the environmental data.
    - Role name: "EnvironmentalData"
    - Cardinality: 1-1
- **Properties and relationships:**
  - Name representing the name of the environmental data
    - Role name: "Name"
    - Cardinality: 1-1
    - Belongs to each "EnvironmentalData". Cardinality 1:1. Reverse cardinality 1:1
  - RegularMutableProperty representing the change of the environmental data.
    - Role name: "ChangingAttribute"
    - Cardinality: 1-1
    - Belongs to each "EnvironmentalData". Cardinality 1:1. Reverse cardinality 1:1
    - Subproperty of "InformationFlow". Cardinality : 1:1. ReverseCardinality : 1:1
  - FunctionLaw representing the function
    - Role name: "Function"
    - Cardinality: 1-1
    - Belongs to each "EnvironmentalData". Cardinality 1:1. Reverse cardinality 1:1
  - InteractionRelation representing the information flow between the function and the environmental data
    - Cardinality : 1-1
    - Role Name : "InformationFlow"
    - Subproperty of "Function". Cardinality : 1:1. ReverseCardinality : 1:1
    - Belongs to "EnvironmentalData". Cardinality : 1:1. ReverseCardinality : 1:1
- **Behaviour :** State **Represented states :**
  - "EnvDataState" played by MutableState.
    - Defining property : "ChangingAttribute"
    - State Constraint : /
- **Modality (permission, recommendation, . . .) :** None

H.13 Event

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Event
- **Alternative construct names :**
  - Trigger
  - Outcome
- **Related, but distinct construct names :**
  - Message
  - Environmental Data
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** A event is represented by a parallelepiped.

Figure H.10: Event

- **User-definable attributes :** Name : the name of the event
- **Relations to other constructs :**
  - is created by 1..N function(s)
  - triggers 1..N function(s)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Both level
- **Classes of things:**
  - Participant representing the organizational unit
    - Role name: "OrganizationalUnit"
    - Cardinality: 1-1
- **Properties and relationships:**
  - FunctionLaw representing the function
    - Role name: "FunctionLaw"
    - Cardinality: 1-1
  - Flow representing the flow between functions
    - Role name: "Flow"
    - Cardinality: 1-1
    - Subproperty of "FunctionLaw". Cardinality 1:1. Reverse cardinality 1:1.
  - FlowContent representing the event (change in the flow)
    - Cardinality : 1-1
    - Role Name : "Event"
    - Subproperty of "Flow". Cardinality : 1:1. ReverseCardinality : 1:1
  - RegularMutableProperty representing the number of flowContent (output)
    - Cardinality : 1-1
    - Role Name : "#FlowContent"
    - Subproperty of "Event". Cardinality : 1:1. ReverseCardinality : 1:1
- **Behaviour :** Process
  Represented states :
  - "PreState" played by AnyState
    - Defining property : "#FlowContent"
    - State Constraint: "#FlowContent == 0"
  - "PostState" played by AnyState
    - Defining property : "#FlowContent"
    - State Constraint: "#FlowContent == 0"
  - "ActivatedFlow" played by AnyState
    - Defining property : "#FlowContent"
    - State Constraint: "#FlowContent > 0"

  Represented transformations :
  - "InputEvent" played by AnyTransformation
    - **From state :** PreState
    - **To state :** ActivatedFlow
    - **Trigger :** The output is available
    - **Condition :** /
    - **Action :** Event occurs **effected by** TransformationLaw

  - "OutputEvent" played by AnyTransformation
    - **From state :** ActivatedFlow
    - **To state :** PostState
    - **Trigger :** The output is consumed
    - **Condition :** /
    - **Action :** Event occurs **effected by** TransformationLaw

- **Modality (permission, recommendation, ...) :** None

H.14 Message

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Message
- **Alternative construct names :** None
- **Related, but distinct construct names :**
  - Event
  - Environmental Data
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** A message is represented by a letter symbol.

Figure H.11: Message

- **User-definable attributes :**
  - Additional attributes : attributes transmitting special processing information to the function
- **Relations to other constructs :**
  - is created by 1..1 event
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Both
- **Classes of things:**
  - Participant representing the organizational unit
    - Role name: "OrganizationalUnit"
    - Cardinality: 1-1
- **Properties and relationships:**
  - FunctionLaw representing the function
    - Role name: "FunctionLaw"
    - Cardinality: 1-1
  - Flow representing the flow between functions
    - Role name: "Flow"
    - Cardinality: 1-1
    - Subproperty of "FunctionLaw". Cardinality 1:1. Reverse cardinality 1:1.
  - FlowContent representing the event (change in the flow)
    - Cardinality : 1-1
    - Role Name : "Event"
    - subproperty
    - Subproperty of "Flow". Cardinality : 1:1. ReverseCardinality : 1:1
    - Subproperty of "Message". Cardinality : 1:1. ReverseCardinality : 1:1
  - StateLaw representing the message
    - Role name: "Message"
    - Cardinality: 1-1
    - Subproperty of "FunctionLaw". Cardinality : 1:1. ReverseCardinality : 1:1
  - RegulerProperty representing the attribute of the message
    - Cardinality : 0:N
    - Role Name : "Attribute"
    - Subproperty of "Message". Cardinality : 1:1. ReverseCardinality : 0:N
- **Behaviour :** State **Represented state :**
  - "MessageState" played by AnyState.
    - Defining property : "Attribute"
    - Defining property : "Message"
    - State Constraint: /
- **Modality (permission, recommendation, . . .) :** None

H.15 Application software

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Application software
- **Alternative construct names :** Software, Application system type
- **Related, but distinct construct names :** None
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** An application software is represented by a rectangle with two vertical broken lines on each side.

Figure H.12: Application software

- **User-definable attributes :**
  - Name : the name of the application software.
  - Rules : the computer-aided processing rules of a function are defined in the application software
- **Relations to other constructs :**
  - executes 1..N function(s)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Both
- **Classes of things:**
  - ExecutingThing representing the application software
    - Cardinality : 1-1
    - Role Name : "Software"
- **Properties and relationships:**
  - Name representing the name of the application software
    - Cardinality : 1:1
    - Role Name : "Name"
    - Belongs to "Software". Cardinality : 1:1. ReverseCardinality : 1:1
  - RegulerProperty representing the computer-aided processing rules of a function in the application software
    - Cardinality : 0:N
    - Role Name : "Rule"
    - Belongs to "Software". Cardinality : 1:1. ReverseCardinality : 0:N
    - Subproperty of "FunctionLaw". Cardinality : 1:1. ReverseCardinality : 0:N
  - FunctionLaw representing the function
    - Cardinality : 1:N
    - Role Name : "FunctionLaw"
    - Belongs to "Software". Cardinality : 1:1. ReverseCardinality : 1:N
  - ApplicationLaw representing the fact that the application software executes a part of the function
    - Cardinality : 1:1
    - Role Name : "ApplicationLaw"
    - Belongs to "Software". Cardinality : 1:1. ReverseCardinality : 1:1
    - Subproperty of "FunctionLaw". Cardinality : 1:1. ReverseCardinality : 1:1
  - IsActive representing the fact that the software (ApplicationLaw) is active or non-active
    - Cardinality : 1:1
    - Role Name : "IsActive"
    - Subproperty of "ApplicationLaw". Cardinality : 1:1. ReverseCardinality : 1:1
- **Behaviour :** Process
  Represented states :
  - "IsActiveState" played by ActiveState.
    - Defining property : "IsActive"
    - State Constraint : "IsActive == true"
  - "IsNotActiveState" played by InactiveState.
    - Defining property : "IsActive"
    - State Constraint : "IsActive == false"

  Represented transformations :
  - "Triggering" played by Triggering

    - **From state :** IsNotActiveState
    - **To state :** IsActiveState
    - **Trigger :** The function is executed
    - **Condition :** /
    - **Action :** The application software executes a part of the function **effected by** ApplicationLaw (TriggeringLaw)
  - "Termination" played by Termination
    - **From state :** IsActiveState
    - **To state :** IsNotActiveState
    - **Trigger :** The part of the function exectuted by the application software is finished
    - **Condition :** /
    - **Action :** The application software is released **effected by** ApplicationLaw (TerminationLaw)

- **Modality (permission, recommendation, ...) :** None

H.16 Human output

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Human output
- **Alternative construct names :** None
- **Related, but distinct construct names :** None
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** An human output is represented by a rectangle.

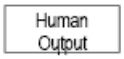


Figure H.13: Human output

- **User-definable attributes :**
  - Name : the name of the human output.
- **Relations to other constructs :**
  - processes 1..N function(s)
  - is allocated to 1..N organizational unit(s)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Both
- **Classes of things:**
  - HumanOutput representing the human output
    - Cardinality : 1-1
    - Role Name : "HumanOutput"
  - Participant representing the organizational unit.
    - Role name: "OrganizationalUnit"
    - Cardinality: 1-1
- **Properties and relationships:**
  - Name representing the name of the human output
    - Cardinality : 1:1
    - Role Name : "Name"
    - Belongs to "HumanOutput". Cardinality : 1:1. ReverseCardinality : 1:1
  - Name representing the name of the organizational unit
    - Cardinality : 1:1
    - Role Name : "NameOrganizationalUnit"
    - Belongs to "OrganizationalUnit". Cardinality : 1:1. ReverseCardinality : 1:1
  - PartWholeRelation representing the composite of the human output.
    - Role name: "RelationToWhole"
    - Cardinality: 1-1
    - Belongs to each "OrganizationalUnit" in role of whole. Cardinality 1:1. Reverse cardinality 1:1.
    - Belongs to each "HumanOutput" in role of part. Cardinality 1:1. Reverse cardinality 1:1.
  - FunctionLaw representing the law that makes the function happen
    - Role name: "FunctionLaw"
    - Cardinality: 1-1
    - Belongs to "OrganizationalUnit". Cardinality 1:1. Reverse cardinality 1:1.
  - ParticipationLaw representing the participation of the human in the creation of an output of the function
    - Role name : "Participation"
    - Cardinality : 1-1
    - Belongs to "HumanOutput". Cardinality : 1:1. ReverseCardinality : 1:1
    - SubProperty of "FunctionLaw". Cardinality : 1:1. ReverseCardinality : 1:1
- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** None

H.17 Goal

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Goal
- **Alternative construct names :** Objective
- **Related, but distinct construct names :** Corporate goal
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
  - **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** A goal is represented by a rectangle with a triangle for the up side.

Figure H.14: Goal

- **User-definable attributes :**
  - Name : the name of the goal.
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - controls/is supported by 1..N function(s)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Both
- **Classes of things:**
  - Participant representing the organizational unit
    - Role name: "OrganizationalUnit"
    - Cardinality: 1-1
- **Properties and relationships:**
  - FunctionLaw representing the function
    - Cardinality : 1-1
    - Role Name : "FunctionLaw"
  - Law representing the goal
    - Cardinality : 1-1
    - Role Name : "Goal"
    - Subproperty of "FunctionLaw". Cardinality : 1:1. ReverseCardinality : 1:N
  - Flow representing the outgoing flow of the function
    - Cardinality : 1-1
    - Role Name : "OutgoingFlow"
    - Subproperty of "FunctionLaw". Cardinality : 1:1. ReverseCardinality : 1:1
    - Subproperty of "Goal". Cardinality : 1:1. ReverseCardinality : 1:1
- **Comments :** We choose to say that the general Law represents the goal. Because ARIS doesn't make the distinction between the goals which represent a StateLaw or a TransformationLaw.
- **Behaviour :** Existence
- **Modality (permission, recommendation, . . .) :** Intention. The function is intented to achieve the finality that the goal represents.

H.18 Machine resource

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Machine resource
- **Alternative construct names :** Machine
- **Related, but distinct construct names :** None
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
  - **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** A machine resource is represented by a rectangle with a vertical broken line on the right side and a circle in.

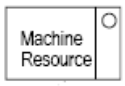


Figure H.15: Machine resource

- **User-definable attributes :**
  - Name : the name of the machine resource
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - is used by 1..N function(s)
  - is allocated to 1..N organizational unit(s)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Both
- **Classes of things:**
  - MachineResource representing the machine resource
    - Cardinality : 1-1
    - Role Name : "Machine"
  - Participant representing the organizational unit.
    - Role name: "OrganizationalUnit"
    - Cardinality: 1-1
- **Properties and relationships:**
  - Name representing the name of the machine resource
    - Cardinality : 1:1
    - Role Name : "Name"
    - Belongs to "Machine". Cardinality : 1:1. ReverseCardinality : 1:1
  - Name representing the name of the organizational unit
    - Cardinality : 1:1
    - Role Name : "NameOrganizationalUnit"
    - Belongs to "OrganizationalUnit". Cardinality : 1:1. ReverseCardinality : 1:1
  - Realization representing the realization of a work
    - Cardinality : 1:1
    - Role Name : "Realization"
    - Belongs to "Machine". Cardinality : 1:1. ReverseCardinality : 1:1
  - FunctionLaw representing the function
    - Cardinality : 1:1
    - Role Name : "FunctionLaw"
    - Belongs to "OrganizationalUnit". Cardinality : 1:1. ReverseCardinality : 1:1
  - UseLaw representing the fact that a part of the function uses a machine resource
    - Role name: "UseLaw"
    - Cardinality: 1-1
    - Subproperty of "FunctionLaw". Cardinality 1:1. Reverse cardinality 1:1.
    - Belongs to "Machine". Cardinality 1:1. Reverse cardinality 1:1.
  - IsActive representing the fact that the machine (UseLaw) is active or non-active
    - Cardinality : 1:1
    - Role Name : "IsActive"
    - Subproperty of "UseLaw". Cardinality : 1:1. ReverseCardinality : 1:1
  - MutualProperty representing the allocation of the machine resource to an organizational unit
    - Cardinality : 1:1
    - Role Name : "Allocation"
    - Belongs to "Machine". Cardinality : 1:1. ReverseCardinality : 1:1
    - Belongs to "OrganizationalUnit". Cardinality : 1:1. ReverseCardinality : 1:1
- **Behaviour :** Process
  Represented states :

- "IsActiveState" played by ActiveState.
  - Defining property : "IsActive"
  - State Constraint : "IsActive == true"
- "IsNotActiveState" played by InactiveState.
  - Defining property : "IsActive"
  - State Constraint : "IsActive == false"

Represented transformations :

- "Triggering" played by Triggering
  - **From state :** IsNotActiveState
  - **To state :** IsActiveState
  - **Trigger :** A part of the function uses a machine ressource or a computer hardware.
  - **Condition :** /
  - **Action :** A machine resource or a computer hardware is used **effected by** UseLaw (TriggeringLaw)
- "Termination" played by Termination
  - **From state :** IsActiveState
  - **To state :** IsNotActiveState
  - **Trigger :** A part of the function uses a machine resource or a computer hardware no more.
  - **Condition :** /
  - **Action :** A machine resource or a computer hardware is released **effected by** UseLaw (TerminationLaw)

- **Modality (permission, recommendation, . . .) :** None

H.19 Computer hardware

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Computer hardware
- **Alternative construct names :** Computer hardware resource, Hardware
- **Related, but distinct construct names :** None
- **Related terms :** None
- **Language :** "Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
  - **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** A computer hardware is represented by the following picture.

Figure H.16: Computer hardware

- **User-definable attributes :**
  - Name : the name of the computer hardware
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - is used by 1..N function(s)
  - is allocated to 1..N organizational unit(s)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Both
- **Classes of things:**
  - ComputerHardware representing the computer hardware
    - Cardinality : 1-1
    - Role Name : "ComputerHardware"
  - Participant representing the organizational unit.
    - Role name: "OrganizationalUnit"
    - Cardinality: 1-1
- **Properties and relationships:**
  - Name representing the name of the computer hardware
    - Cardinality : 1:1
    - Role Name : "Name"
    - Belongs to "ComputerHardware". Cardinality : 1:1. ReverseCardinality : 1:1
  - Name representing the name of the organizational unit
    - Cardinality : 1:1
    - Role Name : "NameOrganizationalUnit"
    - Belongs to "OrganizationalUnit". Cardinality : 1:1. ReverseCardinality : 1:1
  - FunctionLaw representing the function
    - Cardinality : 1:1
    - Role Name : "FunctionLaw"
    - Belongs to "OrganizationalUnit". Cardinality : 1:1. ReverseCardinality : 1:1
  - UseLaw representing the fact that a part of the function uses computer hardware
    - Role name: "UseLaw"
    - Cardinality: 1-1
    - Subproperty of "FunctionLaw". Cardinality 1:1. Reverse cardinality 1:1.
    - Belongs to "ComputerHardware". Cardinality 1:1. Reverse cardinality 1:1.
  - IsActive representing the fact that the computer hardware (UseLaw) is active or non-active
    - Cardinality : 1:1
    - Role Name : "IsActive"
    - Subproperty of "UseLaw". Cardinality : 1:1. ReverseCardinality : 1:1
  - MutualProperty representing the allocation of the computer hardware to a organizational unit
    - Cardinality : 1:1
    - Role Name : "Allocation"
    - Belongs to "ComputerHardware". Cardinality : 1:1. ReverseCardinality : 1:1
    - Belongs to "OrganizationalUnit". Cardinality : 1:1. ReverseCardinality : 1:1
- **Behaviour :** Process
  Represented states :
  - "IsActiveState" played by ActiveState.
    - Defining property : "IsActive"
    - State Constraint : "IsActive == true"

- "IsNotActiveState" played by InactiveState.
  - Defining property : "IsActive"
  - State Constraint : "IsActive == false"

Represented transformations :

- "Triggering" played by Triggering
  - **From state :** IsNotActiveState
  - **To state :** IsActiveState
  - **Trigger :** A part of the function uses a machine ressource or a computer hardware.
  - **Condition :** /
  - **Action :** A machine resource or a computer hardware is used **effected by** UseLaw (TriggeringLaw)
- "Termination" played by Termination
  - **From state :** IsActiveState
  - **To state :** IsNotActiveState
  - **Trigger :** A part of the function uses a machine resource or a computer hardware no more.
  - **Condition :** /
  - **Action :** A machine resource or a computer hardware is released **effected by** UseLaw (TerminationLaw)

- **Modality (permission, recommendation, . . .) :** None

H.20 Control Flow

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Control Flow
- **Alternative construct names :** None
- **Related construct names :** None
- **Related terms :** None
- **Language :**"Architecture of Integrated Information systems" (ARIS)
  - "ARIS - Business Process Frameworks", Second, completely revised and enlarged edition, A.-W. Scheer
  - "ARIS - Business Process Modeling", Third edition, A.-W. scheer
  - "Business Process Modelling with ARIS - A Practical Guide", Rob Davis.
- **Diagram types :** ARIS business process model

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** A Control Flow is represented by a single black line with a open arrowhead.

Figure H.17: Control Flow

- **User-definable attributes :** None
- **Relations to other constructs :**
  - belongs to 1..1 ARIS model
  - can connect 1..1 Function to 1..1 Event.
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - OutputThing representing the source of the Control Flow

    - Role name : "Source"
    - Cardinality : 1-1
  - InputThing representing the target of the Control Flow
    - Role name : "Target"
    - Cardinality : 1-1

- **Properties and relationships:**

  * Flow representing the Control Flow
    - Role name: "ControlFlow"
    - Cardinality: 1-1
    - Belongs to :
      - "Source". Cardinality 1:1. Reverse cardinality 1:1
      - "Target". Cardinality 1:1. Reverse cardinality 1:1

- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

Appendix I

Analysis of BPMN

Here is grouped all the filled templates for each analysed modelling constructs of BPMN. These templates are the corrected one according to the case study (Chapter 12).

I.1 Event

Preamble section

- **Builds on :** None
- **Built on by :** Start Event, Intermediate Event, End Event
- **Construct name :** Event
- **Alternative construct names :** None
- **Related construct names :** Start Event, Intermediate Event, End Event
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Event is represented by a small circle with open centers to allow internal markers to differentiate triggers or results.

Figure I.1: Event

- **User-definable attributes :**
  - Name : the text description of the Event
  - Assignments (0-n) : one or more assignment expressions (Set of attributes)
  - Pool : a Pool must be identified for the Event to identify its location
  - Lanes (0-n) : If the Pool has more than one Lane, then the Id of at least one Lane must be added
  - EventType : the type of the Event (Start, Intermediate, End)
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be the source for 0..N Sequence Flow (multiple outgoing Sequence Flow)
  - can be the target for 0..N Sequence Flow (multiple incoming Sequence Flow)
  - can be the target for 0..N Message Flow (0..N incoming Message Flow)
  - can be the source for 0..N Message Flow (multiple outgoing Message Flow)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - InputThing representing the target of the Sequence Flow.
    - Role name : "TargetSF"
    - Cardinality : 0-1
  - OutputThing representing the source of the Sequence Flow.
    - Role name : "SourceSF"
    - Cardinality : 0-1
  - InputThing representing the target of the Message Flow.
    - Role name : "TargetMF"
    - Cardinality : 0-1
  - OutputThing representing the source of the Message Flow.
    - Role name : "SourceMF"
    - Cardinality : 0-1
- **Properties and relationships:**
  - Flow representing the Sequence Flow
    - Role name: "SequenceFlow"
    - Cardinality: 0-1
    - Belongs to "TargetSF". Cardinality 0:1. Reverse cardinality 0:1.
    - Belongs to "SourceSF". Cardinality 0:1. Reverse cardinality 0:1.
  - Flow representing the Message Flow
    - Role name: "MessageFlow"
    - Cardinality: 0-1
    - Belongs to "TargetMF". Cardinality 0:1. Reverse cardinality 0:1.
    - Belongs to "SourceMF". Cardinality 0:1. Reverse cardinality 0:1.
  - FlowContent representing the parameters of the Event
    - Role name: "EventParameters"
    - Cardinality: 1-1
    - Subproperty of "SequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "MessageFlow". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularStringProperty representing the type of the Event
    - Role name: "Type"
    - Cardinality: 1-1
    - Subproperty of "EventParameters". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularStringProperty representing the description of the Event
    - Role name: "Name"
    - Cardinality: 1-1
    - Subproperty of "EventParameters". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularProperty representing the assignments of the Event
    - Role name: "Assignments"

  - Cardinality: 0-N
  - Subproperty of "EventParameters". Cardinality 1:1. Reverse cardinality 0:N.
* RegularProperty representing the Pool where is located the Event
  - Role name: "Pool"
  - Cardinality: 1-1
  - Subproperty of "EventParameters". Cardinality 1:1. Reverse cardinality 1:1.
* RegularProperty representing the Id of the lane where is located the Event
  - Role name: "Lane"
  - Cardinality: 0-N
  - Subproperty of "EventParameters". Cardinality 1:1. Reverse cardinality 0:N
* RegularMutableProperty representing the number of Tokens
  - Role name: "#Token"
  - Cardinality: 1-1
  - Subproperty of "EventParameters". Cardinality 1:1. Reverse cardinality 1:1
* TransformationLaw representing the law of the target of the Sequence Flow
  - Role name: "TLaw"
  - Cardinality: 1-1
  - Belongs to "TargetSF". Cardinality 1:1. Reverse cardinality 1:1
* TransformationLaw representing the law of the target of the Message Flow
  - Role name: "TLaw"
  - Cardinality: 1-1
  - Belongs to "TargetMF". Cardinality 1:1. Reverse cardinality 1:1

- **Behaviour :** Process

  Represented states :

  - "PreState" played by AnyState
    * Defining property : "#Token"
    * State Constraint: "#Token == 0"
  - "PostState" played by AnyState
    * Defining property : "#Token"
    * State Constraint: "#Token == 0"
  - "ActivatedFlow" played by AnyState
    * Defining property : "#Token"
    * State Constraint: "#Token > 0"

  Represented transformations :

  - "InputEvent" played by AnyTransformation
    * **From state :** PreState
    * **To state :** ActivatedFlow
    * **Trigger :** The token is available **OR** The trigger of the Start Event occurs **OR** A Token is consumed (arrived to the End Event)
    * **Condition :** /
    * **Action :** Event occurs **effected by** TransformationLaw
  - "OutputEvent" played by AnyTransformation
    * **From state :** ActivatedFlow
    * **To state :** PostState

  - **Trigger :** The token is consumed **OR** A Token is generated for each outgoing Sequence Flow from that event **OR** The result expected for the End Event occurs
  - **Condition :** /
  - **Action :** Event occurs **effected by** TransformationLaw

- **Modality (permission, recommendation, ...) :** Regular assertion

I.2 Start Event

Preamble section

- **Builds on :** Event
- **Built on by :** None
- **Construct name :** Start Event
- **Alternative construct names :** None
- **Related construct names :** Intermediate Event, End Event
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Start Event is represented by a small circle drawn with a thin line.

Figure I.2: Start Event

- **User-definable attributes :** "As for Event with adding"
  - Trigger : the type of trigger expected for the Start (default None)
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be the source for 1..N Sequence Flow (multiple outgoing Sequence Flow)
  - can be the target for 0..N Message Flow (0..N incoming Message Flow)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - InputThing representing the target of the outgoing Sequence Flow.
    - Role name : "TargetOSF"
    - Cardinality : 1-1
  - OutputThing representing the source of the incoming Message Flow.
    - Role name : "SourceIMF"
    - Cardinality : 0-1
- **Properties and relationships:** "As for Event with adding/modifying"
  - Flow representing the outgoing Sequence Flow
    - Role name: "OutgoingSequenceFlow"
    - Cardinality: 1-1
    - Belongs to "TargetOSF". Cardinality 1:1. Reverse cardinality 1:1.
  - Flow representing the incoming Message Flow
    - Role name: "IncomingMessageFlow"
    - Cardinality: 0-1
    - Belongs to "SourceIMF". Cardinality 1:1. Reverse cardinality 0:1.
  - FlowContent representing the parameters of the Start Event
    - Role name: "StartEventParameters"
    - Cardinality: 1-1
    - Subproperty of "OutgoingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "IncomingMessageFlow". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularStringProperty representing the type of the Start Event
    - Role name: "Trigger"
    - Cardinality: 1-1
    - Subproperty of "StartEventParameters". Cardinality 1:1. Reverse cardinality 1:1.
  - TransformationLaw representing the law of the target of the Sequence Flow
    - Role name: "TLaw"
    - Cardinality: 1-1
    - Belongs to "TargetOSF". Cardinality 1:1. Reverse cardinality 1:1
  - RegularMutableProperty representing the number of Tokens
    - Role name: "#Token"
    - Cardinality: 1-1
    - Subproperty of "StartEventParameters". Cardinality 1:1. Reverse cardinality 1:1
- **Behaviour :** Process

  Represented states :
  - "PreState" played by AnyState
    - Defining property : "#Token"
    - State Constraint: "#Token == 0"

- "PostState" played by AnyState
  - Defining property : "#Token"
  - State Constraint: "#Token == 0"
- "ActivatedFlow" played by AnyState
  - Defining property : "#Token"
  - State Constraint: "#Token > 0"

Represented transformations :

- "InputEvent" played by AnyTransformation
  - **From state :** PreState
  - **To state :** ActivatedFlow
  - **Trigger :** The trigger of the Start Event occurs
  - **Condition :** /
  - **Action :** Event occurs **effected by** TransformationLaw
- "OutputEvent" played by AnyTransformation
  - **From state :** ActivatedFlow
  - **To state :** PostState
  - **Trigger :** A Token is generated for each outgoing Sequence Flow from that event
  - **Condition :** /
  - **Action :** Event occurs **effected by** TransformationLaw

- **Modality (permission, recommendation, . . .) :** Regular assertion

I.3 Intermediate Event

Preamble section

- **Builds on :** Event
- **Built on by :** None
- **Construct name :** Intermediate Event
- **Alternative construct names :** None
- **Related construct names :** Start Event, End Event
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www. bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Intermediate Event is represented by a small circle drawn with a double thin black line.

Figure I.3: Intermediate Event

- **User-definable attributes :** "As for Event with adding"
  - Trigger : the type of trigger expected for the Intermediate Event (default Message)
  - Target (0-1) : it must be an activity (Sub-Process or Task). This means that the Intermediate Event is attached to the boundary of the activity and is used to signify an exception or compensation for that activity.
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be the source for 1..1 Sequence Flow (1:1 outgoing Sequence Flow)
  - can be the target for 0..1 Sequence Flow (0:1 incoming Sequence Flow)
  - can be the target for 0..1 Message Flow (Intermediate Event of type Message)
  - can be attached to the boundary of an 1..1 activity
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - InputThing representing the target of the outgoing Sequence Flow.
    - Role name : "TargetOSF"
    - Cardinality : 1-1
  - OutputThing representing the source of the incoming Sequence Flow.
    - Role name : "SourceISF"
    - Cardinality : 0-1
  - OutputThing representing the source of the incoming Message Flow.
    - Role name : "SourceIMF"
    - Cardinality : 0-1
- **Properties and relationships:** "As for Event with adding/modifying"
  - Flow representing the outgoing Sequence Flow
    - Role name: "OutgoingSequenceFlow"
    - Cardinality: 1-1
    - Belongs to "TargetOSF". Cardinality 1:1. Reverse cardinality 1:1.
  - Flow representing the incoming Sequence Flow
    - Role name: "IncomingSequenceFlow"
    - Cardinality: 0-1
    - Belongs to "SourceISF". Cardinality 1:1. Reverse cardinality 0:1.
  - Flow representing the incoming Message Flow
    - Role name: "IncomingMessageFlow"
    - Cardinality: 0-1
    - Belongs to "SourceIMF". Cardinality 1:1. Reverse cardinality 0:1.
  - FlowContent representing the parameters of the Intermediate Event
    - Role name : "IntermediateEventParameters"
    - Cardinality : 1-1
    - Subproperty of "OutgoingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "IncomingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "IncomingMessageFlow". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularStringProperty representing the type of the Intermediate Event
    - Role name: "Trigger"
    - Cardinality: 1-1
    - Subproperty of "IntermediateEventParameters". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularProperty representing the activity where the Intermediate Event is attached
    - Role name: "Target"
    - Cardinality: 0-1
    - Subproperty of "IntermediateEventParameters". Cardinality 1:1. Reverse cardinality 0:1
  - TransformationLaw representing the law of the target of the Sequence Flow
    - Role name: "TLaw"

  - Cardinality: 1-1
  - Belongs to "TargetOSF". Cardinality 1:1. Reverse cardinality 1:1
* RegularMutableProperty representing the number of Tokens
  - Role name: "#Token"
  - Cardinality: 1-1
  - Subproperty of "IntermediateEventParameters". Cardinality 1:1. Reverse cardinality 1:1

- **Behaviour :** Process

  Represented states :

  - "PreState" played by AnyState
    * Defining property : "#Token"
    * State Constraint: "#Token == 0"
  - "PostState" played by AnyState
    * Defining property : "#Token"
    * State Constraint: "#Token == 0"
  - "ActivatedFlow" played by AnyState
    * Defining property : "#Token"
    * State Constraint: "#Token > 0"

  Represented transformations :

  - "InputEvent" played by AnyTransformation
    * **From state :** PreState
    * **To state :** ActivatedFlow
    * **Trigger :** The token is available
    * **Condition :** /
    * **Action :** Event occurs **effected by** TransformationLaw
  - "OutputEvent" played by AnyTransformation
    * **From state :** ActivatedFlow
    * **To state :** PostState
    * **Trigger :** The token is consumed
    * **Condition :** /
    * **Action :** Event occurs **effected by** TransformationLaw

- **Modality (permission, recommendation, ...) :** Regular assertion

I.4 End Event

Preamble section

- **Builds on :** Event
- **Built on by :** None
- **Construct name :** End Event
- **Alternative construct names :** None
- **Related construct names :** Start Event, Intermediate Event
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The End Event is represented by a small circle drawn with a single thick black line.

Figure I.4: End Event

- **User-definable attributes :** "As for Event with adding"
  - Result : the type of result expected for the End (default None)
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be the target for 1..N Sequence Flow (multiple incoming Sequence Flow)
  - can be the source for 0..N Message Flow (multiple outgoing Message Flow)
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - InputThing representing the target of the outgoing Message Flow.
    - Role name : "TargetOMF"
    - Cardinality : 0-1
  - OutputThing representing the source of the incoming Sequence Flow.
    - Role name : "SourceISF"
    - Cardinality : 1-1
- **Properties and relationships:** "As for Event with adding/modifying"
  - Flow representing the incoming Sequence Flow
    - Role name: "IncomingSequenceFlow"
    - Cardinality: 1-1
    - Belongs to "SourceISF". Cardinality 1:1. Reverse cardinality 1:1.
  - Flow representing the outgoing Message Flow
    - Role name: "OutgoingMessageFlow"
    - Cardinality: 0-1
    - Belongs to "TargetOMF". Cardinality 1:1. Reverse cardinality 0:1.
  - Flow representing the parameters of the End Event (the end of a process)
    - Role name : "EndEventParameters"
    - Cardinality : 1-1
    - Subproperty of "IncomingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "OutgoingMessageFlow". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularStringProperty representing the type of the End Event
    - Role name: "Result"
    - Cardinality: 1-1
    - Subproperty of "EndEventParameters". Cardinality 1:1. Reverse cardinality 1:1.
  - TransformationLaw representing the law of the source of the Sequence Flow
    - Role name: "TLaw"
    - Cardinality: 1-1
    - Belongs to "SourceISF". Cardinality 1:1. Reverse cardinality 1:1
  - RegularMutableProperty representing the number of Tokens
    - Role name: "#Token"
    - Cardinality: 1-1
    - Subproperty of "EndEventParameters". Cardinality 1:1. Reverse cardinality 1:1
- **Behaviour :** Process

  Represented states :
  - "PreState" played by AnyState
    - Defining property : "#Token"
    - State Constraint: "#Token == 0"

- "PostState" played by AnyState
  - Defining property : "#Token"
  - State Constraint: "#Token == 0"
- "ActivatedFlow" played by AnyState
  - Defining property : "#Token"
  - State Constraint: "#Token > 0"

Represented transformations :

- "InputEvent" played by AnyTransformation
  - **From state :** PreState
  - **To state :** ActivatedFlow
  - **Trigger :** A Token is consumed (arrived to the End Event)
  - **Condition :** /
  - **Action :** Event occurs **effected by** TransformationLaw
- "OutputEvent" played by AnyTransformation
  - **From state :** ActivatedFlow
  - **To state :** PostState
  - **Trigger :** The result expected for the End Event occurs
  - **Condition :** /
  - **Action :** Event occurs **effected by** TransformationLaw

- **Modality (permission, recommendation, . . .) :** Regular assertion

I.5 Activity

Preamble section

- **Builds on :** None
- **Built on by :** Task, Sub-Process
- **Construct name :** Activity
- **Alternative construct names :** None
- **Related construct names :** Task, Sub-Process, Process
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Activity is represented by a rounded corner rectangle.

Figure I.5: Activity

- **User-definable attributes :**
  - Name : the text description of the Activity
  - Assignments (0-n) : one or more assignment expressions (Set of attributes)
  - Pool : a Pool must be identified for the Activity to identify its location
  - Lanes (0-n) : If the Pool has more than one Lane, then the Id of at least one Lane must be added
  - ActivityType : must be of type Task or Sub-Process.
  - Status : is determined when the activity is being executed by a process engine. The Status of an activity can be used within Assignment Expressions.
  - Properties (0-n) : are "local" to the activity and are only for use within the processing of the activity.
  - InputSets (0-n) : defines the data requirements for input to the activity.
  - OutputSets (0-n) : defines the data requirements for output from the activity.
  - IORules (0-n) : is an expression that defines the relationship between one InputSet and one OutputSet. That is, if the activity is instantiated with a specified InputSet, then the output of the activity must produce the specified OutputSet.

  - StartQuantity : defines the number of Tokens that must arrive from a single Sequence Flow before the activity can begin. The default value is 1. The value must not be less than 1.
  - LoopType : is an attribute and is by default None, but may be set to Standard or MultiInstance.

- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be the source for 0..N Sequence Flow
  - can be the target for 0..N Sequence Flow
  - can be the source for 0..N Message Flow
  - can be the target for 0..N Message Flow
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - Participant representing the participant in a process who performs or is responsible for an activity.
    * Role name : "Participant"
    * Cardinality : 1-1
- **Properties and relationships:**
  NB : create a property ActivityLaw preceded by TransformationLaw
  * ActivityLaw representing the Activity
    - Role name: "Activity"
    - Cardinality: 1-1
    - Belongs to "Participant". Cardinality 1:1. Reverse cardinality 1:1.
  * RegularStringProperty representing the description of the Activity
    - Role name: "Name"
    - Cardinality: 1-1
    - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 1:1.
  * RegularProperty representing the assignments of the Activity
    - Role name: "Assignments"
    - Cardinality: 0-N
    - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 0:N.
  * RegularProperty representing the Pool where is located the Activity
    - Role name: "Pool"
    - Cardinality: 1-1
    - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 1:1.
  * RegularProperty representing the Id of the lane where is located the Activity
    - Role name: "Lane"
    - Cardinality: 0-N

  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 0:N
* RegularStringProperty representing the type of the Activity
  - Role name: "ActivityType"
  - Cardinality: 1-1
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 1:1.
* RegularStringProperty representing the status of the activity being executed by a process engine
  - Role name: "Status"
  - Cardinality: 1-1
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 1:1.
* RegularProperty representing the properties of the activity
  - Role name: "Properties"
  - Cardinality: 0-N
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 0:N.
* Law representing the InputOuputRules of the activity
  - Role name: "IORules"
  - Cardinality: 0-N
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 0:N.
* Law representing the InputSets of the activity
  - Role name: "InputSets"
  - Cardinality: 0-N
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 0:N.
  - Subproperty of "IORules". Cardinality 1:1. Reverse cardinality 1:1
* Law representing the OutputSets of the activity
  - Role name: "OutputSets"
  - Cardinality: 0-N
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 0:N.
  - Subproperty of "IORules". Cardinality 1:1. Reverse cardinality 1:1.
* RegularNaturalProperty representing the number of Tokens that must arrive from a single Sequence Flow before the activity can begin
  - Role name: "StartQuantity"
  - Cardinality: 0-N
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 0:N.
* RegularStringProperty representing the Looptype of the activity
  - Role name: "Looptype"
  - Cardinality: 1-1
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 1:1.
* Flow representing the incoming Sequence Flow
  - Role name: "IncomingSequenceFlow"
  - Cardinality: 0-N
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 0:N.
* Flow representing the outgoing Sequence Flow
  - Role name: "OutgoingSequenceFlow"
  - Cardinality: 0-N
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 0:N.
* Flow representing the outgoing Message Flow

  - Role name: "OutgoingMessageFlow"
  - Cardinality: 0-N
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 0:N.
  - Subproperty of "OutputSets". Cardinality 1:1. Reverse cardinality 0:N.
* Flow representing the incoming Message Flow
  - Role name: "IncomingMessageFlow"
  - Cardinality: 0-N
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 0:N.
  - Subproperty of "InputSets". Cardinality 1:1. Reverse cardinality 0:N.
* FlowContent representing the content of the incoming Sequence Flow
  - Role name: "IncomingSequenceFlowContent"
  - Cardinality: 1-1
  - Subproperty of "IncomingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
* FlowContent representing the content of the outgoing Sequence Flow
  - Role name: "OutgoingSequenceFlowContent"
  - Cardinality: 1-1
  - Subproperty of "OutgoingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
* FlowContent representing the content of the incoming Message Flow
  - Role name: "IncomingMessageFlowContent"
  - Cardinality: 1-1
  - Subproperty of "IncomingMessageFlow". Cardinality 1:1. Reverse cardinality 1:1.
* FlowContent representing the content of the outgoing Message Flow
  - Role name: "OutgoingMessageFlowContent"
  - Cardinality: 1-1
  - Subproperty of "OutgoingMessageFlow". Cardinality 1:1. Reverse cardinality 1:1.
* IsActive representing if the Activity is active or non-active
  - Role name: "IsActive"
  - Cardinality: 1-1
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 1:1.
* RegularMutableProperty representing the number of Tokens of a single (incoming) Sequence Flow that are already arrived to the Activity
  - Role name: "Token"
  - Cardinality: 1-1
  - Subproperty of "Activity". Cardinality 1:1. Reverse cardinality 1:1.

- **Behaviour :** Process

  Represented states :

  - "ActiveState" played by ActiveState.
    * Defining property : "IsActive"
    * State Constraint : "IsActive == true"
  - "InactiveState" played by InactiveState.
    * Defining property : "IsActive"
    * State Constraint : "IsActive == false"

  Represented transformations :

  - "TriggeringActivity" played by Triggering

    - **From state :** InactiveState
    - **To state :** ActiveState
    - **Trigger :** A Token arrives from a (incoming) Sequence Flow
    - **Condition :** StartQuantity == NbrToken
    - **Action :** Activity realisation **effected by** ActivityLaw (TriggeringLaw)
  - "TerminationActivity" played by Termination
    - **From state :** ActiveState
    - **To state :** InactiveState
    - **Trigger :** A Token is generated for a (outgoing) Sequence Flow
    - **Condition :** All Tokens must be generated for each (outgoing) Sequence Flow
    - **Action :** Activity termination **effected by** ActivityLaw (TerminationLaw)
  - "NotAllTokenAvailable" played by AnyTransformation
    - **From state :** ActiveState
    - **To state :** ActiveState
    - **Trigger :** A Token is is generated for a (outgoing) Sequence Flow
    - **Condition :** Not all Tokens are generated for each (outgoing) Sequence Flow
    - **Action :** Activity continues to be executed to generate all Token **effected by** Activity-Law

- **Modality (permission, recommendation, . . .) :** Regular assertion

I.6 Task

Preamble section

- **Builds on :** Activity
- **Built on by :** None
- **Construct name :** Task
- **Alternative construct names :** None
- **Related construct names :** Sub-Process, Process
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Task is represented by a rounded corner rectangle drawn with a single thin black line.

Figure I.6: Task

- **User-definable attributes :** "As for Activity but with adding"
  - TaskType : the type of the Task (default Service)
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be the target for 0..N Sequence Flow (multiple incoming Sequence Flow)
  - can be the source for 0..N Sequence Flow (multiple outgoing Sequence Flow)
  - can be the source for 0..N Message Flow (0..N outgoing Message Flow)
  - can be the target for 0..1 Message Flow (0..1 incoming Message Flow)
- **Diagram layout conventions :** BPMN specifies three types of markers for Task : a Loop Marker or a Multiple Instance Marker and a Compensation Marker. A Task may have one or two of these markers.
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - Participant representing the participant in a process who performs or is responsible for the Task
    - Role name: "Participant"
    - Cardinality: 1-1
  - System representing the Process
    - Role name : "Process"
    - Cardinality : 1-1
- **Properties and relationships:** "As for Activity with adding/modifying"
  NB : Create an ActivityLaw : preceded by TransformationLaw
  - TransformationLaw representing the law that makes the Process happen
    - Role name: "ProcessLaw"
    - Cardinality: 1-1
    - Belongs to "Process". Cardinality 1:1. Reverse cardinality 1:1.
  - ActivityLaw representing the Task
    - Role name: "Task"
    - Cardinality: 1-1
    - Belons to "Participant". Cardinality 1:1. Reverse cardinality 1:1
    - Subproperty of "ProcessLaw". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularStringProperty representing the type of the Task
    - Role name: "TaskType"
    - Cardinality: 1-1
    - Subproperty of "Task". Cardinality 1:1. Reverse cardinality 1:1.
  - Flow representing the incoming Sequence Flow
    - Role name: "IncomingSequenceFlow"
    - Cardinality: 0-N
    - Subproperty of "Task". Cardinality 1:1. Reverse cardinality 0:N
  - Flow representing the outgoing Sequence Flow
    - Role name: "OutgoingSequenceFlow"
    - Cardinality: 0-N
    - Subproperty of "Task". Cardinality 1:1. Reverse cardinality 0:N
  - Flow representing the outgoing Message Flow
    - Role name: "OutgoingMessageFlow"
    - Cardinality: 0-N
    - Subproperty of "Task". Cardinality 1:1. Reverse cardinality 0:N
  - Flow representing the incoming Message Flow
    - Role name: "IncomingMessageFlow"
    - Cardinality: 0-1
    - Subproperty of "Task". Cardinality 1:1. Reverse cardinality 0:1
  - FlowContent representing the content of the incoming Sequence Flow

  - Role name: "IncomingSequenceFlowContent"
  - Cardinality: 1-1
  - Subproperty of "IncomingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
* FlowContent representing the content of the outgoing Sequence Flow
  - Role name: "OutgoingSequenceFlowContent"
  - Cardinality: 1-1
  - Subproperty of "OutgoingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
* FlowContent representing the content of the incoming Message Flow
  - Role name: "IncomingMessageFlowContent"
  - Cardinality: 1-1
  - Subproperty of "IncomingMessageFlow". Cardinality 1:1. Reverse cardinality 1:1.
* FlowContent representing the content of the outgoing Message Flow
  - Role name: "OutgoingMessageFlowContent"
  - Cardinality: 1-1
  - Subproperty of "OutgoingMessageFlow". Cardinality 1:1. Reverse cardinality 1:1.
* IsActive representing if the Task is active or non-active
  - Role name: "IsActive"
  - Cardinality: 1-1
  - Subproperty of "Task". Cardinality 1:1. Reverse cardinality 1:1.
* RegularMutableProperty representing the number of Tokens of a single (incoming) Sequence Flow that are already arrived to the Task
  - Role name: "Token"
  - Cardinality: 1-1
  - Subproperty of "Task". Cardinality 1:1. Reverse cardinality 1:1.

- **Behaviour :** Process

  Represented states :

  - "ActiveState" played by ActiveState.
    * Defining property : "IsActive"
    * State Constraint : "IsActive == true"
  - "InactiveState" played by InactiveState.
    * Defining property : "IsActive"
    * State Constraint : "IsActive == false"

  Represented transformations :

  - "TriggeringTask" played by Triggering
    * **From state :** InactiveState
    * **To state :** ActiveState
    * **Trigger :** A Token arrives from a (incoming) Sequence Flow
    * **Condition :** StartQuantity == NbrToken
    * **Action :** Task realisation **effected by** ActivityLaw (TriggeringLaw)
  - "TerminationTask" played by Termination
    * **From state :** ActiveState
    * **To state :** InactiveState
    * **Trigger :** A Token is generated for a (outgoing) Sequence Flow
    * **Condition :** All Tokens must be generated for each (outgoing) Sequence Flow

    - **Action :** Task termination **effected by** ActivityLaw (TerminationLaw)
  - "NotAllTokenAvailable" played by AnyTransformation
    - **From state :** ActiveState
    - **To state :** ActiveState
    - **Trigger :** A Token is is generated for a (outgoing) Sequence Flow
    - **Condition :** Not all Tokens are generated for each (outgoing) Sequence Flow
    - **Action :** Task continues to be executed to generate all Token **effected by** ActivityLaw

- **Modality (permission, recommendation, ...) :** Regular assertion

I.7 Process

Preamble section

- **Builds on :** Activity
- **Built on by :** None
- **Construct name :** Process
- **Alternative construct names :** None
- **Related construct names :** Task, Sub-Process
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Process is depicted as a graph of Flow Objects, which are a set of other activities and the controls that sequence them.
- **User-definable attributes :**
  - Id : a unique Id that identifies the object from other objects within the Diagram.
  - Name : the text description of the Process
  - ProcessType : provides information about which lower-level language the Pool will be mapped (default None)
  - Status : is determined when the Process is being executed by a process engine. The Status of a Process can be used within Assignment Expressions.
  - GraphicalElements (0-n) : identifies all of the objects (e.g., Events, Activities, Gateways, and Artifacts) that are contained within the Business Process.
  - Assignments (0-n) : shall be performed as defined by the AssignTime attribute.
  - Properties (0-n) : are "local" to the Process.
  - AdHoc : specifies whether the Process is Ad Hoc or not (default false). The activities within an Ad Hoc Process are not controlled or sequenced in a particular order, their performance is determined by the performers of the activities. If set to True, then the Ad Hoc marker shall be placed at the bottom center of the Process or the Sub-Process shape for Ad Hoc Processes.
  - SuppressJoinFailure : is included for mapping to BPEL4WS. This specifies whether or not a BPEL4WS joinFailure fault will be suppressed for all activities in the BPEL4WS process. (Default false)
  - EnableInstanceCompensation : is included for mapping to BPEL4WS. It specifies whether or not a compensation can be performed after the Process has completed normally. (Default false)
  - Categories (0-n) : used for purposes such as reporting and analysis.
  - Documentation (0-1) : text documentation about the Process.

- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be composed of 0..N Flow Objects
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - System representing the Process
    - Role name : "Process"
    - Cardinality : 1-1
  - Component representing the Flow Object composing the Process
    - Role name : "FlowObject"
    - Cardinality : 0-N
- **Properties and relationships:**
  - TransformationLaw representing the law that makes the Process happen
    - Role name: "ProcessLaw"
    - Cardinality: 1-1
    - Belongs to "Process". Cardinality 1:1. Reverse cardinality 1:1.
  - PartWholeRelation representing the composition of the Process
    - Role name: "RelationToPart"
    - Cardinality: 0-N
    - Belongs to :
      - "Process". Cardinality 1:1. Reverse cardinality 0:N.
      - "FlowObject". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularStringProperty representing the description of the Process
    - Role name: "Name"
    - Cardinality: 1-1
    - Belongs to "Process". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularProperty representing the Id of the Process
    - Role name: "Id"
    - Cardinality: 1-1
    - Belongs to "Process". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularStringProperty representing the type of the Process
    - Role name: "Type"
    - Cardinality: 1-1
    - Belongs to "Process". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularStringProperty representing the status of the Process
    - Role name: "Status"

  - Cardinality: 1-1
  - Belongs to "Process". Cardinality 1:1. Reverse cardinality 1:1.
* RegularProperty representing the assignments of the Activity
  - Role name: "Assignments"
  - Cardinality: 0-N
  - Belongs to "Process". Cardinality 1:1. Reverse cardinality 0:N.
* RegularProperty representing the properties of the Process
  - Role name: "Properties"
  - Cardinality: 0-N
  - Belongs to "Process". Cardinality 1:1. Reverse cardinality 0:N.
* RegularBooleanProperty representing whether the Process is Ad Hoc or not
  - Role name: "AdHoc"
  - Cardinality: 1-1
  - Belongs to "Process". Cardinality 1:1. Reverse cardinality 1:1.
* RegularBooleanProperty representing whether or not a BPEL4WS joinFailure fault will be suppressed for all activities in the BPEL4WS process
  - Role name: "SuppressJoinFailure"
  - Cardinality: 1-1
  - Belongs to "Process". Cardinality 1:1. Reverse cardinality 1:1.
* RegularBooleanProperty representing whether or not a compensation can be performed after the Process has completed normally
  - Role name: "EnableInstanceCompensation"
  - Cardinality: 1-1
  - Belongs to "Process". Cardinality 1:1. Reverse cardinality 1:1.
* RegularStringProperty representing the categories of the Process
  - Role name: "Categories"
  - Cardinality: 0-N
  - Belongs to "Process". Cardinality 1:1. Reverse cardinality 0:N
* RegularStringProperty representing the documentation of the Process
  - Role name: "Documentation"
  - Cardinality: 0-1
  - Belongs to "Process". Cardinality 1:1. Reverse cardinality 0:1
* IsActive representing if the Process is active or non-active
  - Role name: "IsActive"
  - Cardinality: 1-1
  - Subproperty of "ProcessLaw". Cardinality 1:1. Reverse cardinality 1:1.

- **Behaviour :** Process

  Represented states :

  - "ActiveState" played by ActiveState.
    * Defining property : "IsActive"
    * State Constraint : "IsActive == true"
  - "InactiveState" played by InactiveState.
    * Defining property : "IsActive"
    * State Constraint : "IsActive == false"

  Represented transformations :

  - "TriggeringProcess" played by Triggering
    - **From state :** InactiveState
    - **To state :** ActiveState
    - **Trigger :** A Start Event starts the Process or the Process is instantiated (no Start Event)
    - **Condition :** /
    - **Action :** Process realisation **effected by** TransformationLaw (TriggeringLaw)
  - "TerminationProcess" played by Termination
    - **From state :** ActiveState
    - **To state :** InactiveState
    - **Trigger :** A End Event ends the Process or all parallel paths in the Process have completed - Flow Objects that do not have any outgoing Sequence Flow (no End Event)
    - **Condition :** /
    - **Action :** Process termination **effected by** TransformationLaw (TerminationLaw)
  - "NotAllPathCompleted" played by AnyTransformation
    - **From state :** ActiveState
    - **To state :** ActiveState
    - **Trigger :** A parallel path in the Process is completed
    - **Condition :** Not all parallel paths in the Process have completed
    - **Action :** Process continues running to complete the incompleted paths **effected by** TransformationLaw (TerminationLaw)

- **Modality (permission, recommendation, . . .) :** Regular assertion

I.8 Sub-Process

Preamble section

- **Builds on :** Activity
- **Built on by :** None
- **Construct name :** Sub-Process
- **Alternative construct names :** None
- **Related construct names :** Task, Process
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Sub-Process is represented by a rounded corner rectangle drawn with a single thin black line.

Figure I.7: Sub-Process

- **User-definable attributes :** "As for Activity with adding"
  - SubProcessType : the type of the Sub-Procces. It defines whether the Sub-Process details are embedded within the higher level Process or refers to another, re-usable Process (default Embedded)
  - IsATransaction : determines whether or not the behavior of the Sub-Process will follow the behavior of a Transaction
  - Transaction (0-1) : If the Transaction attribute is False, then a Transaction must not be identified. If the Transaction attribute is True, then a Transaction must be identified.
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be the target for 0..N Sequence Flow (multiple incoming Sequence Flow)
  - can be the source for 0..N Sequence Flow (multiple outgoing Sequence Flow)
  - can be the source for 0..N Message Flow (0..N outgoing Message Flow)
  - can be the target for 0..N Message Flow (0..N incoming Message Flow)
- **Diagram layout conventions :**

- The Sub-Process can be in a collapsed view that hides its details or a Sub-Process can be in an expanded view that shows its details within the view of the Process in which it is contained. In the collapsed form, the Sub-Process object uses a marker to distinguish it as a Sub-Process, rather than a Task. The Sub-Process marker must be a small square with a plus sign (+) inside. The square must be positioned at the bottom center of the shape.

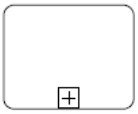


Figure I.8: Collapsed Sub-Process

- BPMN specifies five types of standard markers for Sub-Processes. The (Collapsed) Sub-Process Marker can be combined with four other markers: a Loop Marker or a Parallel Marker, a Compensation Marker, and an Ad Hoc Marker. A collapsed Sub-Process may have one to three of these other markers, in all combinations except that Loop and Multiple Instance cannot be shown at the same time.

- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - Participant representing the participant in a process who performs or is responsible for the Sub-Process
    - Role name: "Participant"
    - Cardinality: 1-1
  - System representing the Process
    - Role name : "Process"
    - Cardinality : 1-1
- **Properties and relationships:** "As for Activity with adding"
  NB : Create an ActivityLaw : preceded by TransformationLaw
  - TransformationLaw representing the law that makes the Process happen
    - Role name: "ProcessLaw"
    - Cardinality: 1-1
    - Belongs to "Process". Cardinality 1:1. Reverse cardinality 1:1.
  - ActivityLaw representing the Sub-Process
    - Role name: "SubProcess"
    - Cardinality: 1-1
    - Belons to "Participant". Cardinality 1:1. Reverse cardinality 1:1
    - Subproperty of "ProcessLaw". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularStringProperty representing the type of the Sub-Process
    - Role name: "SubProcessType"
    - Cardinality: 1-1

  - Subproperty of "SubProcess". Cardinality 1:1. Reverse cardinality 1:1.
* Flow representing the incoming Sequence Flow
  - Role name: "IncomingSequenceFlow"
  - Cardinality: 0-N
  - Subproperty of "SubProcess". Cardinality 1:1. Reverse cardinality 0:N
* Flow representing the outgoing Sequence Flow
  - Role name: "OutgoingSequenceFlow"
  - Cardinality: 0-N
  - Subproperty of "SubProcess". Cardinality 1:1. Reverse cardinality 0:N
* Flow representing the outgoing Message Flow
  - Role name: "OutgoingMessageFlow"
  - Cardinality: 0-N
  - Subproperty of "SubProcess". Cardinality 1:1. Reverse cardinality 0:N
* Flow representing the incoming Message Flow
  - Role name: "IncomingMessageFlow"
  - Cardinality: 0-N
  - Subproperty of "SubProcess". Cardinality 1:1. Reverse cardinality 0:N
* FlowContent representing the content of the incoming Sequence Flow
  - Role name: "IncomingSequenceFlowContent"
  - Cardinality: 1-1
  - Subproperty of "IncomingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
* FlowContent representing the content of the outgoing Sequence Flow
  - Role name: "OutgoingSequenceFlowContent"
  - Cardinality: 1-1
  - Subproperty of "OutgoingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
* FlowContent representing the content of the incoming Message Flow
  - Role name: "IncomingMessageFlowContent"
  - Cardinality: 1-1
  - Subproperty of "IncomingMessageFlow". Cardinality 1:1. Reverse cardinality 1:1.
* FlowContent representing the content of the outgoing Message Flow
  - Role name: "OutgoingMessageFlowContent"
  - Cardinality: 1-1
  - Subproperty of "OutgoingMessageFlow". Cardinality 1:1. Reverse cardinality 1:1.
* IsActive representing if the Sub-Process is active or non-active
  - Role name: "IsActive"
  - Cardinality: 1-1
  - Subproperty of "SubProcess". Cardinality 1:1. Reverse cardinality 1:1.
* RegularMutableProperty representing the number of Tokens of a single (incoming) Sequence Flow that are already arrived to the Sub-Process
  - Role name: "Token"
  - Cardinality: 1-1
  - Subproperty of "SubProcess". Cardinality 1:1. Reverse cardinality 1:1.

- **Behaviour :** Process

  Represented states :

  - "ActiveState" played by ActiveState.

    - Defining property : "IsActive"
    - State Constraint : "IsActive == true"
  - "InactiveState" played by InactiveState.
    - Defining property : "IsActive"
    - State Constraint : "IsActive == false"

  Represented transformations :

  - "TriggeringSubProcess" played by Triggering
    - **From state :** InactiveState
    - **To state :** ActiveState
    - **Trigger :** A Token arrives from a (incoming) Sequence Flow
    - **Condition :** StartQuantity == NbrToken
    - **Action :** Sub-Process realisation **effected by** ActivityLaw (TriggeringLaw)
  - "TerminationSubProcess" played by Termination
    - **From state :** ActiveState
    - **To state :** InactiveState
    - **Trigger :** A Token is generated for a (outgoing) Sequence Flow
    - **Condition :** All Tokens must be generated for each (outgoing) Sequence Flow
    - **Action :** Sub-Process termination **effected by** ActivityLaw (TerminationLaw)
  - "NotAllTokenAvailable" played by AnyTransformation
    - **From state :** ActiveState
    - **To state :** ActiveState
    - **Trigger :** A Token is is generated for a (outgoing) Sequence Flow
    - **Condition :** Not all Tokens are generated for each (outgoing) Sequence Flow
    - **Action :** Sub-Process continues to be executed to generate all Token **effected by** ActivityLaw

- **Modality (permission, recommendation, . . .) :** Regular assertion

I.9 Gateway

Preamble section

- **Builds on :** None
- **Built on by :** Data-Based Exclusive Gateway, Event-Based Exclusive Gateway, Inclusive Gateway, Complex Gateway, Parallel Gateway
- **Construct name :** Gateway
- **Alternative construct names :** None
- **Related construct names :** Data-Based Exclusive Gateway, Event-Based Exclusive Gateway, Inclusive Gateway, Complex Gateway, Parallel Gateway
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Gateways is represented by a diamond drawn with a single thin black line.

Figure I.9: Gateway

- **User-definable attributes :**
  - Name : the text description of the Gateway
  - Assignments (0-n) : one or more assignment expressions (Set of attributes)
  - Pool : a Pool must be identified for the Gateway to identify its location
  - Lanes (0-n) : If the Pool has more than one Lane, then the Id of at least one Lane must be added
  - GatewayType : the type of the Gateway (default XOR). The GatewayType will determine the behavior of the Gateway, both for incoming and outgoing Sequence Flow, and will determine the internal indicator.
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be the source for 0..N Sequence Flow
  - can be the target for 0..N Sequence Flow

- **Diagram layout conventions :** Each type of Gateway will have an internal indicator or marker to show the type of Gateway that is being used. The internal marker associated with the Gateway must be placed inside the shape, in any size or location.
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - CoupledThing representing the input of the Gateway
    - Role name : "InputGateway"
    - Cardinality : 0-N
  - CoupledThing representing the output of the Gateway
    - Role name : "OutputGateway"
    - Cardinality : 0-N
- **Properties and relationships:**
  - MutualLaw representing the Gateway
    - Role name: "Gateway"
    - Cardinality: 1-1
    - Belongs to :
      - "InputGateway". Cardinality 0:N. Reverse cardinality 1:1.
      - "OutputGateway". Cardinality 0:N. Reverse cardinality 1:1.
  - RegularStringProperty representing the description of the Gateway
    - Role name: "Name"
    - Cardinality: 1-1
    - Subproperty of "Gateway". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularProperty representing the assignments of the Gateway
    - Role name: "Assignments"
    - Cardinality: 0-N
    - Subproperty of "Gateway". Cardinality 1:1. Reverse cardinality 0:N.
  - RegularProperty representing the Pool where is located the Gateway
    - Role name: "Pool"
    - Cardinality: 1-1
    - Subproperty of "Gateway". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularProperty representing the Id of the lane where is located the Gateway
    - Role name: "Lane"
    - Cardinality: 0-N
    - Subproperty of "Gateway". Cardinality 1:1. Reverse cardinality 0:N
  - RegularStringProperty representing the type of the Gateway
    - Role name: "GatewayType"
    - Cardinality: 1-1
    - Subproperty of "Gateway". Cardinality 1:1. Reverse cardinality 1:1.
  - Flow representing the incoming Sequence Flow of the Gateway

  - Role name: "IncomingSequenceFlow"
  - Cardinality: 0-N
  - Subproperty of "Gateway". Cardinality 1:1. Reverse cardinality 0:N.
* Flow representing the outgoing Sequence Flow of the Gateway
  - Role name: "OutgoingSequenceFlow"
  - Cardinality: 0-N
  - Subproperty of "Gateway". Cardinality 1:1. Reverse cardinality 0:N.
* CouplingRelation representing the coupling between the incoming Sequence Flow and the inputs of the Gateway
  - Role name: "InputCoupling"
  - Cardinality: 1-1
  - Belongs to "InputGateway". Cardinality 1:1. Reverse cardinality 1:1.
  - Subproperty of "IncomingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
* CouplingRelation representing the coupling between the outgoing Sequence Flow and the outputs of the Gateway
  - Role name: "OutputCoupling"
  - Cardinality: 1-1
  - Belongs to "OutputGateway". Cardinality 1:1. Reverse cardinality 1:1.
  - Subproperty of "OutgoingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.

- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

I.10 Data-Based Exclusive Gateway

Preamble section

- **Builds on :** Gateway
- **Built on by :** None
- **Construct name :** Data-Based Exclusive Gateway
- **Alternative construct names :** XOR
- **Related construct names :** Parallel Gateway, Complex Gateway, Inclusive Gateway, Event-Based Exclusive Gateway
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :**
  - The Data-Based Exclusive Gateway may be represented by a diamond drawn with a single thin black line

Figure I.10: Data-Based Exclusive Gateway

  - The Data-Based Exclusive Gateway may use a marker that is shaped like an "X" and is placed within the Gateway diamond

Figure I.11: Data-Based Exclusive Gateway

- **User-definable attributes :** "As for Gateway with adding"
  - XORType : the type of the XOR (default Data)
  - MarkerVisible : determines if the XOR Marker is displayed in the center of the Gateway diamond (an "X"). The marker is displayed if the attribute is True and it is not displayed if the attribute is False.

  - Gates (0-n) : Zero Gates are allowed if the Gateway is last object in a Process flow and there are no Start or End Events for the Process. If there are zero or only one incoming Sequence Flow (i.e, the Gateway is acting as a Decision), then there must be at least one Gate. In this case, if there is no DefaultGate, then there must be at least two Gates.
  - DefaultGate (0-1) : A Default Gate may be specified.

- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be the source for 0..N Sequence Flow
  - can be the target for 0..N Sequence Flow
- **Comments :** We can't have N inputs and N outputs at the same time. We can have 1 input and N outputs : Data-Based Exclusive Decision. Or we can have N inputs and 1 output : Data-Based Exclusive Merge.
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - CoupledThing representing the input of the Data-Based Exclusive Gateway
    - Role name : "InputGateway"
    - Cardinality : 0-N
  - CoupledThing representing the output of the Data-Based Exclusive Gateway
    - Role name : "OutputGateway"
    - Cardinality : 0-N
- **Properties and relationships:** "As for Gateway with adding"
  - MutualLaw representing the Data-Based Exclusive Gateway
    - Role name: "DataBasedExclusiveGateway"
    - Cardinality: 1-1
    - Belongs to :
      - "InputGateway". Cardinality 0:N. Reverse cardinality 1:1.
      - "OutputGateway". Cardinality 0:N. Reverse cardinality 1:1.
  - RegularStringProperty representing the type of the Data-Based Exclusive Gateway
    - Role name: "XORType"
    - Cardinality: 1-1
    - Subproperty of "DataBasedExclusiveGateway". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularBooleanProperty representing if the XOR Marker is displayed in the center of the Data-Based Exclusive Gateway diamond (an "X").
    - Role name: "MarkerVisible"
    - Cardinality: 1-1
    - Subproperty of "DataBasedExclusiveGateway". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularProperty representing the gates of the Data-Based Exclusive Gateway

  - Role name: "Gates"
  - Cardinality: 0-N
  - Subproperty of "DataBasedExclusiveGateway". Cardinality 1:1. Reverse cardinality 0:N.
* RegularProperty representing the default gate
  - Role name: "DefaultGate"
  - Cardinality: 0-1
  - Subproperty of "DataBasedExclusiveGateway". Cardinality 1:1. Reverse cardinality 0:1.
* Flow representing the incoming Sequence Flow of the Data-Based Exclusive Gateway
  - Role name: "IncomingSequenceFlow"
  - Cardinality: 0-N
  - Subproperty of "DataBasedExclusiveGateway". Cardinality 1:1. Reverse cardinality 0:N
* Flow representing the outgoing Sequence Flow of the Data-Based Exclusive Gateway
  - Role name: "OutgoingSequenceFlow"
  - Cardinality: 0-N
  - Subproperty of "DataBasedExclusiveGateway". Cardinality 1:1. Reverse cardinality 0:N
* CouplingRelation representing the coupling between the incoming Sequence Flow and the inputs of the Data-Based Exclusive Gateway
  - Role name: "DataBasedExclusiveInputCoupling"
  - Cardinality: 1-1
  - Belongs to "InputGateway". Cardinality 1:1. Reverse cardinality 1:1.
  - Subproperty of "IncomingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
* CouplingRelation representing the coupling between the outgoing Sequence Flow and the outputs of the Data-Based Exclusive Gateway
  - Role name: "DataBasedExclusiveOutputCoupling"
  - Cardinality: 1-1
  - Belongs to "OutputGateway". Cardinality 1:1. Reverse cardinality 1:1.
  - Subproperty of "OutgoingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.

- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

I.11 Event-Based Exclusive Gateway

Preamble section

- **Builds on :** Gateway
- **Built on by :** None
- **Construct name :** Event-Based Exclusive Gateway
- **Alternative construct names :** XOR
- **Related construct names :** Parallel Gateway, Complex Gateway, Inclusive Gateway, Data-Based Exclusive Gateway
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Event-Based Exclusive Gateway must use a marker that is black star and is placed within the Event-Based Exclusive Gateway diamond.

Figure I.12: Event-Based Exclusive Gateway

- **User-definable attributes :** "As for Gateway with adding"
  - XORType : the type of the XOR (default Data)
  - Instantiate : Event-Based Gateways can be defined as the instantiation mechanism for the Process with the Instantiate attribute. This attribute may be set to true if the Gateway is the first element after the Start Event or a starting Gateway if there is no Start Event (i.e., there are no incoming Sequence Flow).
  - Gates (2-n) : There must be two or more Gates.
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - is the source for 2..N Sequence Flow
  - is the target for 0..N Sequence Flow
- **Comments :** We can have N inputs and N outputs at the same time. This type of Gateway does not act only as a Merge - it is always a Decision, at least.
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - CoupledThing representing the input of the Event-Based Exclusive Gateway
    - Role name : "InputGateway"
    - Cardinality : 0-N
  - CoupledThing representing the output of the Event-Based Exclusive Gateway
    - Role name : "OutputGateway"
    - Cardinality : 2-N
- **Properties and relationships:** "As for Gateway with adding"
  - MutualLaw representing the Event-Based Exclusive Gateway
    - Role name: "EventBasedExclusiveGateway"
    - Cardinality: 1-1
    - Belongs to :
      - "InputGateway". Cardinality 0:N. Reverse cardinality 1:1.
      - "OutputGateway". Cardinality 2:N. Reverse cardinality 1:1.
  - RegularStringProperty representing the type of the Event-Based Exclusive Gateway
    - Role name: "XORType"
    - Cardinality: 1-1
    - Subproperty of "EventBasedExclusiveGateway". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularBooleanProperty representing the Event-Based Exclusive Gateway as the instantiation mechanism for the Process
    - Role name: "InstantiateFalse"
    - Cardinality: 1-1
    - Subproperty of "EventBasedExclusiveGateway". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularProperty representing the gates of the Event-Based Exclusive Gateway
    - Role name: "Gates"
    - Cardinality: 2-N
    - Subproperty of "EventBasedExclusiveGateway". Cardinality 1:1. Reverse cardinality 2:N.
  - Flow representing the incoming Sequence Flow of the Event-Based Exclusive Gateway
    - Role name: "IncomingSequenceFlow"
    - Cardinality: 0-N
    - Subproperty of "EventBasedExclusiveGateway". Cardinality 1:1. Reverse cardinality 0:N
  - Flow representing the outgoing Sequence Flow of the Event-Based Exclusive Gateway
    - Role name: "OutgoingSequenceFlow"
    - Cardinality: 2-N
    - Subproperty of "EventBasedExclusiveGateway". Cardinality 1:1. Reverse cardinality 2:N
  - CouplingRelation representing the coupling between the incoming Sequence Flow and the inputs of the Event-Based Exclusive Gateway
    - Role name: "EventBasedExclusiveInputCoupling"
    - Cardinality: 1-1

  - Belongs to "InputGateway". Cardinality 1:1. Reverse cardinality 1:1.
  - Subproperty of "IncomingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
* CouplingRelation representing the coupling between the outgoing Sequence Flow and the outputs of the Event-Based Exclusive Gateway
  - Role name: "EventBasedExclusiveOutputCoupling"
  - Cardinality: 1-1
  - Belongs to "OutputGateway". Cardinality 1:1. Reverse cardinality 1:1.
  - Subproperty of "OutgoingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.

- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

I.12 Inclusive Gateway

Preamble section

- **Builds on :** Gateway
- **Built on by :** None
- **Construct name :** Inclusive Gateway
- **Alternative construct names :** OR
- **Related construct names :** Data-Based Exclusive Gateway, Event-Based Exclusive Gateway, Parallel Gateway, Complex Gateway
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Inclusive Gateway is represented by a diamond drawn with a thin black line. A marker is black circle and is placed within the diamond.

Figure I.13: Inclusive Gateway

- **User-definable attributes :** "As for Gateway with adding"
  - Gates (0-n) : Zero Gates are allowed if the Gateway is last object in a Process flow and there are no Start or End Events for the Process. If there are zero or only one incoming Sequence Flow (i.e, the Gateway is acting as a Decision), then there must be at least two Gates.
  - DefaultGate (0-1) : A Default Gate may be specified.
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be the source for 0..N Sequence Flow
  - can be the target for 0..N Sequence Flow
- **Comments :** We can't have N inputs and N outputs at the same time. We can have 1 input and N outputs : Inclusive Decision. Or we can have N inputs and 1 output : Inclusive Merge.
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - CoupledThing representing the input of the Inclusive Decision
    - Role name : "InputInclusiveGateway"
    - Cardinality : 0-N
  - CoupledThing representing the output of the Inclusive Decision
    - Role name : "OutputInclusiveGateway"
    - Cardinality : 0-N
- **Properties and relationships:** "As for Gateway with adding"
  - MutualLaw representing the Inclusive Gateway
    - Role name: "InclusiveGateway"
    - Cardinality: 1-1
    - Belongs to :
      - "InputInclusiveGateway". Cardinality 0:N. Reverse cardinality 1:1.
      - "OutputInclusiveGateway". Cardinality 0:N. Reverse cardinality 1:1.
  - RegularProperty representing the gate of the Inclusive Decision
    - Role name: "Gates"
    - Cardinality: 0-N
    - Subproperty of "InclusiveDecision". Cardinality 1:1. Reverse cardinality 0:N.
  - RegularProperty representing the default gate
    - Role name: "DefaultGate"
    - Cardinality: 0-1
    - Subproperty of "InclusiveDecision". Cardinality 1:1. Reverse cardinality 0:1.
  - Flow representing the incoming Sequence Flow of the Inclusive Gateway
    - Role name: "IncomingSequenceFlow"
    - Cardinality: 0-N
    - Subproperty of "InclusiveGateway". Cardinality 1:1. Reverse cardinality 0:N
  - Flow representing the outgoing Sequence Flow of the Inclusive Gateway
    - Role name: "OutgoingSequenceFlow"
    - Cardinality: 0-N
    - Subproperty of "InclusiveGateway". Cardinality 1:1. Reverse cardinality 0:N
  - CouplingRelation representing the coupling between the incoming Sequence Flow and the inputs of the Inclusive Gateway
    - Role name: "InclusiveInputCoupling"
    - Cardinality: 1-1
    - Belongs to "InputInclusiveGateway". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "IncomingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
  - CouplingRelation representing the coupling between the outgoing Sequence Flow and the outputs of the Inclusive Gateway
    - Role name: "InclusiveOutputCoupling"
    - Cardinality: 1-1

  - Belongs to "OutputInlusiveGateway". Cardinality 1:1. Reverse cardinality 1:1.
  - Subproperty of "OutgoingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.

- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

I.13 Complex Gateway

Preamble section

- **Builds on :** Gateway
- **Built on by :** None
- **Construct name :** Complex Gateway
- **Alternative construct names :** None
- **Related construct names :** Data-Based Exclusive Gateway, Event-Based Exclusive Gateway, Inclusive Gateway, Parallel Gateway
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Complex Gateway is represented by a diamond drawn with a thin black line. A marker is in the shape of an asterisk and is placed within the diamond.

Figure I.14: Complex Gateway

- **User-definable attributes :** "As for <Gateway> with adding"
  - Gates (0-n) : Zero Gates are allowed if the Gateway is last object in a Process flow and there are no Start or End Events for the Process. If there are zero or only one incoming Sequence Flow, then there must be at least two Gates.
  - IncomingCondition (0-1) : If there are multiple incoming Sequence Flow, an IncomingCondition expression must be set by the modeler. This will consist of an expression that can reference Sequence Flow names and/or Process Properties (Data).
  - OutgoingCondition (0-1) : If there are multiple outgoing Sequence Flow, an OutgoingCondition expression must be set by the modeler. This will consist of an expression that can reference (outgoing) Sequence Flow Ids and/or Process Properties (Data).
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be the source for 0..N Sequence Flow
  - can be the target for 0..N Sequence Flow

- **Comments :** We can't have N inputs and N outputs at the same time. We can have 1 input and N outputs : Complex Decision. Or we can have N inputs and 1 output : Complex Merge.
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - CoupledThing representing the input of the Complex Gateway
    - Role name : "InputComplexGateway"
    - Cardinality : 0-N
  - CoupledThing representing the output of the Complex Gateway
    - Role name : "OutputComplexGateway"
    - Cardinality : 0-N
- **Properties and relationships:** "As for Gateway with adding/modifying"
  - MutualLaw representing the Complex Gateway
    - Role name: "ComplexGateway"
    - Cardinality: 1-1
    - Belongs to :
      - "InputComplexGateway". Cardinality 0:N. Reverse cardinality 1:1.
      - "OutputComplexGateway". Cardinality 0:N. Reverse cardinality 1:1.
  - RegularProperty representing the gate of the Complex Gateway
    - Role name: "Gate"
    - Cardinality: 0-N
    - Subproperty of "ComplexGateway". Cardinality 1:1. Reverse cardinality 0:N
  - RegularProperty representing the incoming condition of the Complex Gateway
    - Role name: "IncomingCondition"
    - Cardinality: 0-1
    - Subproperty of "ComplexGateway". Cardinality 1:1. Reverse cardinality 0:1
  - RegularProperty representing the outgoing condition of the Complex Gateway
    - Role name: "OutgoingCondition"
    - Cardinality: 0-1
    - Subproperty of "ComplexGateway". Cardinality 1:1. Reverse cardinality 0:1
  - Flow representing the incoming Sequence Flow of the Complex Gateway
    - Role name: "IncomingSequenceFlow"
    - Cardinality: 0-N
    - Subproperty of "ComplexGateway". Cardinality 1:1. Reverse cardinality 0:N
  - Flow representing the outgoing Sequence Flow of the Complex Gateway
    - Role name: "OutgoingSequenceFlow"
    - Cardinality: 0-N
    - Subproperty of "ComplexGateway". Cardinality 1:1. Reverse cardinality 0:N

  - CouplingRelation representing the coupling between the incoming Sequence Flow and the inputs of the Complex Gateway
    - Role name: "InputCoupling"
    - Cardinality: 1-1
    - Belongs to "InputComplexGateway". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "IncomingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
  - CouplingRelation representing the coupling between the outgoing Sequence Flow and the outputs of the Complex Gateway
    - Role name: "OutputCoupling"
    - Cardinality: 1-1
    - Belongs to "OutputComplexGateway". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "OutgoingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.

- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

I.14 Parallel Gateway

Preamble section

- **Builds on :** Gateway
- **Built on by :** None
- **Construct name :** Parallel Gateway
- **Alternative construct names :** AND
- **Related construct names :** Data-Based Exclusive Gateway, Event-Based Exclusive Gateway, Inclusive Gateway, Complex Gateway
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Parallel Gateway is represented by a diamond drawn with a thin black line. A marker is in the shape of an plus sign and is placed within the diamond.

Figure I.15: Parallel Gateway

- **User-definable attributes :** "As for <Gateway> with adding"
  - Gates (0-n) : Zero Gates are allowed if the Gateway is last object in a Process flow and there are no Start or End Events for the Process. If there are zero or only one incoming Sequence Flow (i.e., the Gateway is acting as a fork), then there must be at least two Gates.
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be the source for 0..N Sequence Flow
  - can be the target for 0..N Sequence Flow
- **Comments :** We can't have N inputs and N outputs at the same time. We can have 1 input and N outputs : Forking. Or we can have N inputs and 1 output : Joining.
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - CoupledThing representing the input of the Parallel Gateway
    - Role name : "InputParallelGateway"
    - Cardinality : 0-N
  - CoupledThing representing the output of the Parallel Gateway
    - Role name : "OutputParallelGateway"
    - Cardinality : 0-N
- **Properties and relationships:** "As for Gateway with adding/modifying"
  - MutualLaw representing the Parallel Gateway
    - Role name: "ParallelGateway"
    - Cardinality: 1-1
    - Belongs to :
      - "InputParallelGateway". Cardinality 0:N. Reverse cardinality 1:1.
      - "OutputParallelGateway". Cardinality 0:N. Reverse cardinality 1:1.
  - RegularProperty representing the gate of the Parallel Gateway
    - Role name: "Gate"
    - Cardinality: 0-N
    - Subproperty of "ParallelGateway". Cardinality 1:1. Reverse cardinality 0:N
  - Flow representing the incoming Sequence Flow of the Parallel Gateway
    - Role name: "IncomingSequenceFlow"
    - Cardinality: 0-N
    - Subproperty of "ParallelGateway". Cardinality 1:1. Reverse cardinality 0:N
  - Flow representing the outgoing Sequence Flow of the Parallel Gateway
    - Role name: "OutgoingSequenceFlow"
    - Cardinality: 0-N
    - Subproperty of "ParallelGateway". Cardinality 1:1. Reverse cardinality 0:N
  - CouplingRelation representing the coupling between the incoming Sequence Flow and the inputs of the Parallel Gateway
    - Role name: "ParralelInputCoupling"
    - Cardinality: 1-1
    - Belongs to "InputParallelGateway". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "IncomingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
  - CouplingRelation representing the coupling between the outgoing Sequence Flow and the outputs of the Parallel Gateway
    - Role name: "ParallelOutputCoupling"
    - Cardinality: 1-1
    - Belongs to "OutputParallelGateway". Cardinality 1:1. Reverse cardinality 1:1.
    - Subproperty of "OutgoingSequenceFlow". Cardinality 1:1. Reverse cardinality 1:1.
- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

I.15 Pool

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Pool
- **Alternative construct names :** None
- **Related construct names :** Lane
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** A Pool is represented by a square-cornered rectangle drawn with a solid single black line.

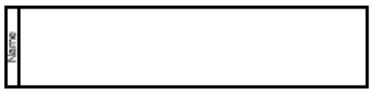


Figure I.16: Pool

- **User-definable attributes :**
  - Name : the text description of the Pool
  - Process (0-1) : defines the Process that is contained within the Pool.
  - Participant : can be either a Role or an Entity. This defines the role that a particular Entity or Role the Pool will play in a Diagram that includes collaboration.
  - Lanes (1-n) : If there is only one Lane, then that Lane shares the name of the Pool and only the Pool name is displayed. If there is more than one Lane, then each Lane has to have its own name and all names are displayed.
  - BoundaryVisible : defines if the rectangular boundary for the Pool is visible. Only one Pool in the Diagram may have the attribute set to False.
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can be composed of 1..N Lanes
  - can be the source for 0..N Message Flow
  - can be the target for 0..N Message Flow
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - Participant representing the Pool implied in the process
    - Role name : "Pool"
    - Cardinality : 1-1
  - ActiveThing representing the Lane implied in the process
    - Role name : "Lane"
    - Cardinality : 1-N
- **Properties and relationships:**
  - RegularStringProperty representing the description of the Pool
    - Role name: "Name"
    - Cardinality: 1-1
    - Belongs to "Pool". Cardinality 1:1. Reverse cardinality 1:1.
  - Flow representing the incoming Message Flow of the Pool
    - Role name: "IncomingMessageFlow"
    - Cardinality: 0-N
    - Belongs to "Pool". Cardinality 1:1. Reverse cardinality 0:N
  - Flow representing the outgoing Message Flow of the Pool
    - Role name: "OutgoingMessageFlow"
    - Cardinality: 0-N
    - Belongs to "Pool". Cardinality 1:1. Reverse cardinality 0:N
  - RegularProperty representing the process that is contained within the Pool.
    - Role name: "Process"
    - Cardinality: 0-1
    - Belongs to "Pool". Cardinality 1:1. Reverse cardinality 0:1
  - RegularProperty representing the participant for a Pool.
    - Role name: "Participant"
    - Cardinality: 1-1
    - Belongs to "Pool". Cardinality 1:1. Reverse cardinality 1:1
  - PartWholeRelation representing the lanes composing the Pool.
    - Role name: "Lanes"
    - Cardinality: 1-N
    - Belongs to :
      - "Pool". Cardinality 1:1. Reverse cardinality 1:N
      - "Lane". Cardinality 1:1. Reverse cardinality 1:1
  - RegularBooleanProperty defines if the rectangular boundary for the Pool is visible.
    - Role name: "BoundaryVisible"
    - Cardinality: 1-1
    - Belongs to "Pool". Cardinality 1:1. Reverse cardinality 1:1
- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

I.16 Lane

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Lane
- **Alternative construct names :** None
- **Related construct names :** Pool
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Lane is represented by a rectangle box.

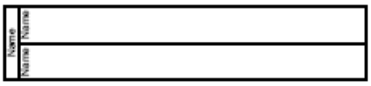


Figure I.17: Lane

- **User-definable attributes :**
  - Name : the text description of the Lane
  - ParentPool : the parent Pool of the Lane.
  - ParentLane (0-1) : is used if the Lane is nested within another Lane. Nesting can be multi-level, but only the immediate parent is specified.
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can compose 1..1 Pool
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - ActiveThing representing the Lane
    - Role name : "Lane"
    - Cardinality : 1-1
  - Participant representing the Pool
    - Role name : "Pool"
    - Cardinality : 1-1
- **Properties and relationships:**
  - RegularStringProperty representing the description of the Lane
    - Role name: "Name"
    - Cardinality: 1-1
    - Belongs to "Lane". Cardinality 1:1. Reverse cardinality 1:1.
  - PartWholeRelation representing the Parent Pool of the Lane
    - Role name: "ParentPool"
    - Cardinality: 1-1
    - Belongs to :
      - "Lane". Cardinality 1:1. Reverse cardinality 1:1.
      - "Pool". Cardinality 1:1. Reverse cardinality 1:N
  - PartWholeRelation representing the Parent Lane of the Lane
    - Role name: "ParentLane"
    - Cardinality: 0-1
    - Belongs to "Lane". Cardinality 2:2. Reverse cardinality 0:1.
- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

I.17 Sequence Flow

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Sequence Flow
- **Alternative construct names :** None
- **Related construct names :** Message Flow
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** A Sequence Flow is represented by a line with a solid arrowhead drawn with a solid single line.

Figure I.18: Sequence Flow

- **User-definable attributes :**
  - Name (0-1) : the text description of the Sequence Flow
  - Source : identifies which Flow Object the Connecting Object is connected *from*
  - Target : identifies which Flow Object the Connecting Object is connected *to*
  - ConditionType : By default, the ConditionType of a Sequence Flow is None. This means that there is no evaluation at runtime to determine whether or not the Sequence Flow will be used. Once a Token is ready to traverse the Sequence Flow , then the Token will do so. A None ConditionType must not be used if the Source of the Sequence Flow is an Exclusive Data-Based or Inclusive Gateway.
    The ConditionType attribute may be set to Expression if the Source of the Sequence Flow is a Task, a Sub-Process, or a Gateway of type Exclusive-Data-Based or Inclusive. If the ConditionType attribute is set to Expression, then a condition marker shall be added to the line if the Sequence Flow is outgoing from an activity. However, a condition indicator must not be added to the line if the Sequence Flow is outgoing from a Gateway. An Expression ConditionType must not be used if the Source of the Sequence Flow is an Event-Based Exclusive Gateway, a Complex Gateway, a Parallel Gateway, a Start Event, or an Intermediate Event. In addition, an Expression ConditionType must not be used if the Sequence Flow is associated with the Default Gate of a Gateway.

The ConditionType attribute may be set to Default only if the Source of the Sequence Flow is an activity or an Exclusive Data-Based Gateway. If the ConditionType is Default, then the Default marker shall be displayed.

- Quantity : defines the number of Tokens that will be generated down the Sequence Flow. The default value is 1. The value must not be less than 1.

- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can connect 1..1 Flow Object (Events, Activities and Gateways) to 1..1 Flow Object.
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - OutputThing representing the source of the Sequence Flow
    - Role name : "Source"
    - Cardinality : 1-1
  - InputThing representing the target of the Sequence Flow
    - Role name : "Target"
    - Cardinality : 1-1
- **Properties and relationships:**
  - Flow representing the Sequence Flow
    - Role name: "SequenceFlow"
    - Cardinality: 1-1
    - Belongs to :
      - "Source". Cardinality 1:1. Reverse cardinality 1:1
      - "Target". Cardinality 1:1. Reverse cardinality 1:1
  - RegularStringProperty representing the condition type of the Sequence Flow
    - Role name: "ConditionType"
    - Cardinality: 1-1
    - Subproperty of "SequenceFlow". Cardinality 1:1. Reverse cardinality 1:1
  - RegularNaturalProperty representing the number of Tokens that will be generated down the Sequence Flow
    - Role name: "Quantity"
    - Cardinality: 1-1
    - Subproperty of "SequenceFlow". Cardinality 1:1. Reverse cardinality 1:1
  - RegularStringProperty representing the description of the Sequence Flow
    - Role name: "Name"
    - Cardinality: 0-1
    - Subproperty of "SequenceFlow". Cardinality 1:1. Reverse cardinality 0:1
- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

I.18 Message Flow

Preamble section

- **Builds on :** None
- **Built on by :** None
- **Construct name :** Message Flow
- **Alternative construct names :** None
- **Related construct names :** Sequence Flow
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** The Message Flow is represented by a dashed single black line with a open arrowhead.

Figure I.19: Message Flow

- **User-definable attributes :**
  - Name (0-1) : the text description of the Message Flow
  - Source : identifies which Flow Object the Connecting Object is connected *from*
  - Target : identifies which Flow Object the Connecting Object is connected *to*
  - Message (0-1) : identifies the Message that is being sent
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - can connect two Pools, either to the Pools themselves or to Flow Objects within the Pools. (They cannot connect two objects within the same Pool).
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - OutputThing representing the source of the Message Flow
    - Role name : "Source"
    - Cardinality : 1-1
  - InputThing representing the target of the Message Flow
    - Role name : "Target"
    - Cardinality : 1-1
- **Properties and relationships:**
  - Flow representing the Message Flow
    - Role name: "MessageFlow"
    - Cardinality: 1-1
    - Belongs to :
      - "Source". Cardinality 1:1. Reverse cardinality 1:1.
      - "Target". Cardinality 1:1. Reverse cardinality 1:1.
  - RegularStringProperty representing the description of the Message Flow
    - Role name: "Name"
    - Cardinality: 0-1
    - Subproperty of "MessageFlow". Cardinality 1:1. Reverse cardinality 0:1
  - RegularProperty representing the message of the Message Flow
    - Role name: "Message"
    - Cardinality: 0-1
    - Subproperty of "MessageFlow". Cardinality 1:1. Reverse cardinality 0:1
- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

I.19 Artifact

Preamble section

- **Builds on :** None
- **Built on by :** Data Object
- **Construct name :** Artifact
- **Alternative construct names :** None
- **Related construct names :** None
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** A Artifact is represented by different ways according to the nature of the Artifact (Data Object, Text Annotation, Group).
- **User-definable attributes :**
  - ArtifactType : The ArtifactType may be set to DataObject, Group, or Annotation. The ArtifactType list may be extended to include new types.
  - Pool (0-1) : A Pool may be added to identify its location.
  - Lane (0-n) : If the Pool has been specified and it has more than one Lane, then a LaneName must be added.
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - is linked to 1..1 Flow Object (Events, Activities and Gateways) by an Association.
- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - Artifact representing the artifact
    - Role name : "Artifact"

    - Cardinality : 1-1
  - FlowObject representing the Flow object
    - Role name : "FlowObject"
    - Cardinality : 1-1

- **Properties and relationships:**
  - RegularStringProperty representing the type of the Artifact
    - Role name: "ArtifactType"
    - Cardinality: 1-1
    - Belongs to "Artifact". Cardinality 1:1. Reverse cardinality 1:1
  - RegularProperty representing the Pool where is located the Artifact
    - Role name: "Pool"
    - Cardinality: 0-1
    - Belongs to "Artifact". Cardinality 1:1. Reverse cardinality 0:1
  - RegularProperty representing the name of the Lane
    - Role name: "Lane"
    - Cardinality: 0-N
    - Belongs to "Artifact". Cardinality 1:1. Reverse cardinality 0:N
  - Flow representing the association that links the Artifact to the flow object
    - Role name: "Association"
    - Cardinality: 1-1
    - Belongs to "Artifact". Cardinality 1:1. Reverse cardinality 1:1
    - Belongs to "FlowObject". Cardinality 1:1. Reverse cardinality 1:1
- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

I.20 Data Object

Preamble section

- **Builds on :** Artifact
- **Built on by :** None
- **Construct name :** Data Object
- **Alternative construct names :** None
- **Related construct names :** None
- **Related terms :** None
- **Language :** "Business Process Modelling Notation", version 1.0 (BPMN 1.0)
  - "Tutorial - Business Process Modelling Language", Polytechnic University of Valencia
  - "Business Process Modelling Specification", OMG
  - <www.bpmn.org>
- **Diagram types :** BPD - Business Process Diagram

Presentation section

- **Builds on :** None
- **Built on by :** None
- **Icon, linestyle, text :** A Data object is represented by a portrait-oriented rectangle that has its upper-right corner folded over that is drawn with a solid single black line.

Figure I.20: Data Object

- **User-definable attributes :**
  - Name : The name of the Data Object.
  - State (0-1) : A state indicates the impact the Process has had on the Data Object.
  - Properties (0-n) : The properties of the Data Object.
  - RequiredForStart : The default value for this attribute is True. This means that the Input is required for an activity to start. If set to False, then the activity may start within the input, but may accept the input (more than once) after the activity has started.
  - ProducedAtCompletion : The default value for this attribute is True. This means that the Output will be produced when an activity has been completed. If set to False, then the activity may produce the output (more than once) before it has completed.
- **Relations to other constructs :**
  - belongs to 1..1 BPD
  - is linked to 1..1 Flow Object (Events, Activities and Gateways) by an Association.

  - is linked to 1..1 Sequence Flow by an Association.
  - is linked to 1..1 Message Flow by an Association.

- **Diagram layout conventions :** None
- **Other usage conventions :** None

Representation section

- **Builds on :** None
- **Built on by :** None
- **Instantiation level :** Type and instance level
- **Classes of things:**
  - DataObject representing the Data Object
    - Role name : "DataObject"
    - Cardinality : 1-1
  - FlowObject representing the Flow object
    - Role name : "FlowObject"
    - Cardinality : 1-1
- **Properties and relationships:**
  - RegularStringProperty representing the name of the Data Object
    - Role name: "Name"
    - Cardinality: 1-1
    - Belongs to "DataObject". Cardinality 1:1. Reverse cardinality 1:1
  - RegularStringProperty representing the state of the Data Object
    - Role name: "State"
    - Cardinality: 0-1
    - Belongs to "DataObject". Cardinality 1:1. Reverse cardinality 0:1
  - RegularProperty representing the properties of the Data Object
    - Role name: "Properties"
    - Cardinality: 0-N
    - Belongs to "DataObject". Cardinality 1:1. Reverse cardinality 0:N
  - RegularBooleanProperty representing the RequiredForStart of the Data Object
    - Role name: "RequiredForStart"
    - Cardinality: 1-1
    - Belongs to "DataObject". Cardinality 1:1. Reverse cardinality 1:1
  - RegularBooleanProperty representing the ProducedAtCompletion of the Data Object
    - Role name: "ProducedAtCompletion"
    - Cardinality: 1-1
    - Belongs to "DataObject". Cardinality 1:1. Reverse cardinality 1:1
  - Flow representing the association that links the Data Object to the Flow Object
    - Role name: "AssociationFO"
    - Cardinality: 0-1
    - Belongs to "DataObject". Cardinality 1:1. Reverse cardinality 0:1
    - Belongs to "FlowObject". Cardinality 1:1. Reverse cardinality 0:1
  - Flow representing the Sequence Flow which is linked to the Data Object

  - Role name: "SequenceFlow"
  - Cardinality: 1-1
* Flow representing the Message Flow which is linked to the Data Object
  - Role name: "MessageFlow"
  - Cardinality: 1-1
* Flow representing the association that links the Data Object to the Sequence Flow
  - Role name: "AssociationSF"
  - Cardinality: 0-1
  - Belongs to "DataObject". Cardinality 1:1. Reverse cardinality 0:1
  - Subproperty of "SequenceFlow". Cardinality 1:1. Reverse cardinality 0:1
* Flow representing the association that links the Data Object to the Message Flow
  - Role name: "AssociationMF"
  - Cardinality: 0-1
  - Belongs to "DataObject". Cardinality 1:1. Reverse cardinality 0:1
  - Subproperty of "MessageFlow". Cardinality 1:1. Reverse cardinality 0:1

- **Behaviour :** Existence
- **Modality (permission, recommendation, ...) :** Regular assertion

Appendix J

Results of the UEML Validator's application

Here is grouped the results of the UEML Validator's application. It consists of all the mistakes identified concerning our analyses.

J.1 Bad rules mistakes

- No generalisation relationship from ontology subclass ActiveThing to superclass ProActiveThing.
- Ontology subproperty FunctionLaw does not belong to the same ontology class as its superproperty.
- Ontology subproperty MutualLaw does not belong to the same ontology class as its superproperty.
- Construct ARIS_ApplicationSoftware represents state ARIS_ApplicationSoftwareStateRole_IsNotActiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct ARIS_ApplicationSoftware represents state ARIS_ApplicationSoftwareState_IsActiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct ARIS_ComputerHardware represents state ARIS_ComputerHardwareStateRole_IsActiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct ARIS_ComputerHardware represents state ARIS_ComputerHardwareStateRole_IsNotActiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct ARIS_EnvironmentalData represents state ARIS_EnvironmentalDataStateRole_EnvDataState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct ARIS_EnvironmentalData represents state ARIS_EnvironmentalDataStateRole_EnvDataState defined by ontology property MutableProperty, but the construct does not represent this property.
- Construct ARIS_Event represents state ARIS-EventStateRole_PostState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct ARIS_Event represents state ARIS_EventStateRole_ActivatedFlow defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct ARIS_Event represents state ARIS_EventStateRole_PreState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct ARIS_Function represents state ARIS_FunctionStateRole_FunctioningState defined by ontology property IsActive, but the construct does not represent this property.
- Construct ARIS_Function represents state ARIS_FunctionStateRole_NonFunctioningState defined by ontology property IsActive, but the construct does not represent this property.

- Construct ARIS_InformationService represents state ARIS_InformationServiceStateRole_InformationService defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct ARIS_MachineRessource represents state ARIS_MachineResourceStateRole_IsActiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct ARIS_MachineRessource represents state ARIS_MachineResourceStateRole_IsNotActiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct ARIS_MaterialOutput represents state ARIS_MaterialOutputStateRole_MaterialOutput defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct ARIS_Message represents state ARIS_MessageStateRole_MessageState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct ARIS_OtherService represents state ARIS_OtherServiceStateRole_OtherServiceState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct ARIS_Output represents state ARIS_OutputStateRole_Output defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct ARIS_Service represents state ARIS_ServiceStateRole_Service defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct BPMN_Activity represents state BPMN_ActivityStateRole_ActiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct BPMN_Activity represents state BPMN_ActivityStateRole_InactiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct BPMN_EndEvent represents state BPMN_EndEventStateRole_ActivatedState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct BPMN_EndEvent represents state BPMN_EndEventStateRole_PostState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct BPMN_EndEvent represents state BPMN_EndEventStateRole_PreState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct BPMN_Event represents state BPMN_EventStateRole_ActivatedFlow defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct BPMN_Event represents state BPMN_EventStateRole_PostState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct BPMN_Event represents state BPMN_EventStateRole_PreState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct BPMN_StartEvent represents state BPMN_StartEventStateRole_PostState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct BPMN_StartEvent represents state BPMN_StartEventStateRole_PreState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct BPMN_SubProcess represents state BPMN_SubProcessStateRole_ActiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct BPMN_SubProcess represents state BPMN_SubProcessStateRole_InactiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct BPMN_Task represents state BPMN_TaskStateRole_ActiveState defined by ontology property IsActive, but the construct does not represent this property.

- Construct BPMN_Task represents state BPMN_TaskStateRole_InactiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct BPMN_IntermediateEvent represents state BPMN_IntermediateEventStateRole_ActivatedState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct BPMN_IntermediateEvent represents state BPMN_IntermediateEventStateRole_PostState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct BPMN_IntermediateEvent represents state BPMN_IntermediateEventStateRole_PreState defined by ontology property AnyProperty, but the construct does not represent this property.
- Construct BPMN_Process represents state BPMN_ProcessStateRole_ActiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct BPMN_Process represents state BPMN_ProcessStateRole_InactiveState defined by ontology property IsActive, but the construct does not represent this property.
- Construct BPMN_StartEvent represents state BPMN_StartEventStateRole_ActivatedFlow defined by ontology property AnyProperty, but the construct does not represent this property.

J.2 Generated mistakes

- **Describe any constructs:** The entry *describedConstruct* is empty for all represented phenomenon. It's an automatic generated thing.
  - Represented phenomenon _ARIS_ApplicationSoftwareClassRole_Software_ARIS_ApplicationSoftwarePropertyRole_Rule_RepresentedClassPropertyRelation does not describe any constructs.
  - Represented phenomenon _ARIS_ApplicationSoftwareClassRole_Software_ARIS_ApplicationSoftwareTLawRole_ApplicationLaw_RepresentedClassPropertyRelation does not describe any constructs.
  - Represented phenomenon _ARIS_FunctionClassRole_OrganizationalUnit_ARIS_FunctionPropertyRole_Name_RepresentedClassPropertyRelation does not describe any constructs.
  - Represented phenomenon _BPMN_ActivityClassRole_Participant_BPMN_ActivityTLawRole_Activity_RepresentedClassPropertyRelation does not describe any constructs.
  - Represented phenomenon _BPMN_ComplexGatewayClassRole_InputComplexGateway_BPMN_ComplexGatewayPropertyRole_InputCoupling_RepresentedClassPropertyRelation does not describe any constructs.
  - Represented phenomenon _BPMN_ComplexGatewayClassRole_InputComplexGateway_BPMN_ComplexGatewayTLaw_ComplexGateway_RepresentedClassPropertyRelation does not describe any constructs.
  - ... For all Represented phenomenon
- **Map onto any ontology phenomena:** The entry *represent* is empty for all represented class property relation. It's an automatic generated thing.
  - Represented phenomenon BPMN_LaneClassRole_Lane is not mapped onto any ontology phenomena.
  - Represented phenomenon ARIS_MessageSLaw_Message is not mapped onto any ontology phenomena.
  - Represented phenomenon ARIS_FunctionTLawRole_Function is not mapped onto any ontology phenomena.

- Represented phenomenon ARIS_FunctionTLawRole_Goal is not mapped onto any ontology phenomena.
- Represented phenomenon ARIS_GoalTLaw_Goal is not mapped onto any ontology phenomena.
- Represented phenomenon BPMN_ActivityTLawRole_Activity is not mapped onto any ontology phenomena.
- Represented phenomenon BPMN_SubProcessTLaw_SubProcess is not mapped onto any ontology phenomena.
- Represented phenomenon BPMN_TaskTLaw_ActivityLaw is not mapped onto any ontology phenomena.

J.3 Correctable mistakes

J.3.1 Non-filled entry

- **Entry *possessesProperty relation***
  - Ontology class ComputerHardware is not characterised by any ontology properties.
  - Ontology class Equipment is not characterised by any ontology properties.
  - Ontology class InformationResource is not characterised by any ontology properties.
  - Ontology class InformationService is not characterised by any ontology properties.
  - Ontology class MaterialService is not characterised by any ontology properties.
  - Ontology class ReactiveThing is not characterised by any ontology properties.
  - Ontology class Service is not characterised by any ontology properties.
- **Entry *represents***
  - Represented property ARIS_AndPropertyRole_And does not represent any ontology properties.
  - Represented property ARIS_EnvironmentalDataPropertyRole_ChangingAttribute does not represent any ontology properties.
  - Represented property ARIS_FunctionPropertyRole_ApplicationLaw does not represent any ontology properties.
- **Entry *roleName***
  - Represented phenomenon ARIS_MaterialOutputClassRole_SourceMaterialOutput plays no role.
- **Entry *isTypeOrValueOrNot***
  - Represented property ARIS_MessageSLaw_Message has no indication of type versus value.
  - Represented property ARIS_ApplicationSoftwareTLawRole_ApplicationLaw has no indication of type versus value.
  - Represented property ARIS_ApplicationSoftwareTLawRole_FunctionLaw has no indication of type versus value.
  - Represented property ARIS_ComputerHardwareTLawRole_Function has no indication of type versus value.
  - Represented property ARIS_ComputerHardwareTLawRole_UseLaw has no indication of type versus value.
  - Represented property ARIS_EnvironmentalDataTLawRole_FunctionLaw has no indication of type versus value.

- Represented property ARIS_EventTLawRole_FunctionLaw has no indication of type versus value.
- Represented property ARIS_FunctionTLawRole_Function has no indication of type versus value.
- Represented property ARIS_FunctionTLawRole_FunctionLaw has no indication of type versus value.
- Represented property ARIS_FunctionTLawRole_Goal has no indication of type versus value.
- Represented property ARIS_FunctionTLawRole_Participation has no indication of type versus value.
- Represented property ARIS_FunctionTLawRole_UseLaw has no indication of type versus value.
- Represented property ARIS_GoalTLaw_FunctionLaw has no indication of type versus value.
- Represented property ARIS_GoalTLaw_Goal has no indication of type versus value.
- Represented property ARIS_HumanOutputTLaw_FunctionLaw has no indication of type versus value.
- Represented property ARIS_HumanOutputTLaw_Participation has no indication of type versus value.
- Represented property ARIS_InformationServiceTLawRole_FunctionLawInput has no indication of type versus value.
- Represented property ARIS_InformationServiceTLawRole_FunctionLawOutput has no indication of type versus value.
- Represented property ARIS_MachineResourceTLaw_FunctionLaw has no indication of type versus value.
- Represented property ARIS_MachineResourceTLaw_UseLaw has no indication of type versus value.
- Represented property ARIS_MaterialOutputTLawRole_FunctionLawInput has no indication of type versus value.
- Represented property ARIS_MaterialOutputTLawRole_FunctionLawOutput has no indication of type versus value.
- Represented property ARIS_MessageTLaw_FunctionLaw has no indication of type versus value.
- Represented property ARIS_OrganizationUnitTLawRole_Function has no indication of type versus value.
- Represented property ARIS_OtherServiceTLawRole_FunctionLawInput has no indication of type versus value.
- Represented property ARIS_OtherServiceTLawRole_FunctionLawOutput has no indication of type versus value.
- Represented property ARIS_OutputTLawRole_FunctionLawInput has no indication of type versus value.
- Represented property ARIS_OutputTLawRole_FunctionLawOutput has no indication of type versus value.
- Represented property ARIS_ServiceTLawRole_FunctionLawInput has no indication of type versus value.
- Represented property ARIS_ServiceTLawRole_FunctionLawOutput has no indication of type versus value.
- Represented property BPMN_ActivityTLawRole_Activity has no indication of type versus value.
- Represented property BPMN_ComplexGatewayTLaw_ComplexGateway has no indication of type versus value.

  - Represented property BPMN_DataBasedExclusiveGatewayTLawRole_ DataBasedExclusiveGateway has no indication of type versus value.
  - Represented property BPMN_EndEventTLawRole_Tlaw has no indication of type versus value.
  - Represented property BPMN_EventBasedExclusiveGatewayTLawRole_ EventBasedExclusiveGateway has no indication of type versus value.
  - Represented property BPMN_EventTLawRole_TLaTargetMF has no indication of type versus value.
  - Represented property BPMN_EventTLawRole_TLawTargetSF has no indication of type versus value.
  - Represented property BPMN_GatewayTLaw_Gateway has no indication of type versus value.
  - Represented property BPMN_InclusiveGatewayTLawRole_InclusiveGateway has no indication of type versus value.
  - Represented property BPMN_IntermediateEventTLawRole_TLaw has no indication of type versus value.
  - Represented property BPMN_ParallelGatewayTLaw_ParallelGateway has no indication of type versus value.
  - Represented property BPMN_ProcessTLaw_ProcessLaw has no indication of type versus value.
  - Represented property BPMN_StartEventTLawRole_TLaw has no indication of type versus value.
  - Represented property BPMN_SubProcessTLaw_ProcessLaw has no indication of type versus value.
  - Represented property BPMN_SubProcessTLaw_SubProcess has no indication of type versus value.
  - Represented property BPMN_TaskTLaw-ProcessLaw has no indication of type versus value.
  - Represented property BPMN_TaskTLaw_ActivityLaw has no indication of type versus value.

- **Entry *languageVersion***
  - Language ARIS has no name and/or version.

- **Entry *effectsRepTransformation***
  - Ontology transformation law ActivityLaw does not effect any ontology transformations.
  - Ontology transformation law ApplicationLaw does not effect any ontology transformations.
  - Ontology transformation law FunctionLaw does not effect any ontology transformations.
  - Ontology transformation law ParticipationLaw does not effect any ontology transformations.
  - Ontology transformation law UseLaw does not effect any ontology transformations.
  - Represented transformation law ARIS_ApplicationSoftwareTLawRole_ApplicationLaw does not effect any represented transformations.
  - Represented transformation law ARIS_ApplicationSoftwareTLawRole_FunctionLaw does not effect any represented transformations.
  - Represented transformation law ARIS_ComputerHardwareTLawRole_Function does not effect any represented transformations.
  - Represented transformation law ARIS_ComputerHardwareTLawRole_UseLaw does not effect any represented transformations.
  - Represented transformation law ARIS_EnvironmentalDataTLawRole_FunctionLaw does not effect any represented transformations.

- Represented transformation law ARIS_EventTLawRole_FunctionLaw does not effect any represented transformations.
- Represented transformation law ARIS_FunctionTLawRole_Function does not effect any represented transformations.
- Represented transformation law ARIS_FunctionTLawRole_FunctionLaw does not effect any represented transformations.
- Represented transformation law ARIS_FunctionTLawRole_Goal does not effect any represented transformations.
- Represented transformation law ARIS_FunctionTLawRole_Participation does not effect any represented transformations.
- Represented transformation law ARIS_FunctionTLawRole_UseLaw does not effect any represented transformations.
- Represented transformation law ARIS_GoalTLaw_FunctionLaw does not effect any represented transformations.
- Represented transformation law ARIS_GoalTLaw_Goal does not effect any represented transformations.
- Represented transformation law ARIS_HumanOutputTLaw_FunctionLaw does not effect any represented transformations.
- Represented transformation law ARIS_HumanOutputTLaw_Participation does not effect any represented transformations.
- Represented transformation law ARIS_InformationServiceTLawRole_FunctionLawInput does not effect any represented transformations.
- Represented transformation law ARIS_InformationServiceTLawRole_FunctionLawOutput does not effect any represented transformations.
- Represented transformation law ARIS_MachineResourceTLaw_FunctionLaw does not effect any represented transformations.
- Represented transformation law ARIS_MachineResourceTLaw_UseLaw does not effect any represented transformations.
- Represented transformation law ARIS_MaterialOutputTLawRole_FunctionLawInput does not effect any represented transformations.
- Represented transformation law ARIS_MaterialOutputTLawRole_FunctionLawOutput does not effect any represented transformations.
- Represented transformation law ARIS_MessageTLaw_FunctionLaw does not effect any represented transformations.
- Represented transformation law ARIS_OrganizationUnitTLawRole_Function does not effect any represented transformations.
- Represented transformation law ARIS_OtherServiceTLawRole_FunctionLawInput does not effect any represented transformations.
- Represented transformation law ARIS_OtherServiceTLawRole_FunctionLawOutput does not effect any represented transformations.
- Represented transformation law ARIS_OutputTLawRole_FunctionLawInput does not effect any represented transformations.
- Represented transformation law ARIS_OutputTLawRole_FunctionLawOutput does not effect any represented transformations.
- Represented transformation law ARIS_ServiceTLawRole_FunctionLawInput does not effect any represented transformations.
- Represented transformation law ARIS_ServiceTLawRole_FunctionLawOutput does not effect any represented transformations.

- Represented transformation law BPMN_ActivityTLawRole_Activity does not effect any represented transformations.
- Represented transformation law BPMN_ComplexGatewayTLaw_ComplexGateway does not effect any represented transformations.
- Represented transformation law BPMN_DataBasedExclusiveGatewayTLawRole_ DataBasedExclusiveGateway does not effect any represented transformations.
- Represented transformation law BPMN_EndEventTLawRole_Tlaw does not effect any represented transformations.
- Represented transformation law BPMN_EventBasedExclusiveGatewayTLawRole_ EventBasedExclusiveGateway does not effect any represented transformations.
- Represented transformation law BPMN_EventTLawRole_TLaTargetMF does not effect any represented transformations.
- Represented transformation law BPMN_EventTLawRole_TLawTargetSF does not effect any represented transformations.
- Represented transformation law BPMN_GatewayTLaw_Gateway does not effect any represented transformations.
- Represented transformation law BPMN_InclusiveGatewayTLawRole_InclusiveGateway does not effect any represented transformations.
- Represented transformation law BPMN_IntermediateEventTLawRole_TLaw does not effect any represented transformations.
- Represented transformation law BPMN_ParallelGatewayTLaw_ParallelGateway does not effect any represented transformations.
- Represented transformation law BPMN_ProcessTLaw_ProcessLaw does not effect any represented transformations.
- Represented transformation law BPMN_StartEventTLawRole_TLaw does not effect any represented transformations.
- Represented transformation law BPMN_SubProcessTLaw_ProcessLaw does not effect any represented transformations.
- Represented transformation law BPMN_SubProcessTLaw_SubProcess does not effect any represented transformations.
- Represented transformation law BPMN_TaskTLaw-ProcessLaw does not effect any represented transformations.
- Represented transformation law BPMN_TaskTLaw_ActivityLaw does not effect any represented transformations.

- **Entry *belongsToRepClass relation***
  - Represented property ARIS_MessageSLaw_Message does not belong to any represented classes.
  - Represented property ARIS_EventTLawRole_FunctionLaw does not belong to any represented classes.
  - Represented property ARIS_FunctionTLawRole_Function does not belong to any represented classes.
  - Represented property ARIS_FunctionTLawRole_Goal does not belong to any represented classes.
  - Represented property ARIS_GoalTLaw_FunctionLaw does not belong to any represented classes.
  - Represented property ARIS_GoalTLaw_Goal does not belong to any represented classes.
  - Represented property ARIS_MessageTLaw_FunctionLaw does not belong to any represented classes.

- **Entry *belongsToClass relation* not complete**
  - Complex property FunctionLaw has subproperty ApplicationLaw, but FunctionLaw does not belong to any class that is the same as or a subclass of a class that ApplicationLaw belongs to.
  - Complex property FunctionLaw has subproperty ParticipationLaw, but FunctionLaw does not belong to any class that is the same as or a subclass of a class that ParticipationLaw belongs to.
  - Complex property FunctionLaw has subproperty UseLaw, but FunctionLaw does not belong to any class that is the same as or a subclass of a class that UseLaw belongs to.
  - Complex property FunctionLaw has subproperty ApplicationLaw, but FunctionLaw does not belong to any class that is the same as or a subclass of a class that ApplicationLaw belongs to.
  - Complex property FunctionLaw has subproperty ParticipationLaw, but FunctionLaw does not belong to any class that is the same as or a subclass of a class that ParticipationLaw belongs to.
  - Complex property FunctionLaw has subproperty UseLaw, but FunctionLaw does not belong to any class that is the same as or a subclass of a class that UseLaw belongs to.
  - Represented property ARIS_ApplicationSoftwareTLawRole_ApplicationLaw belongs to represented class AIRS_ApplicationSoftwareClassRole_Software, but the corresponding ontology property ApplicationLaw does not belong to ontology class ExecutingThing.
  - Represented property ARIS_ApplicationSoftwareTLawRole_FunctionLaw belongs to represented class AIRS_ApplicationSoftwareClassRole_Software, but the corresponding ontology property FunctionLaw does not belong to ontology class ExecutingThing.
  - Represented property ARIS_ComputerHardwareTLawRole_Function belongs to represented class ARIS_ComputerHardwareClassRole_OrganizationalUnit, but the corresponding ontology property FunctionLaw does not belong to ontology class OrganizationalUnit.
  - Represented property ARIS_ComputerHardwareTLawRole_UseLaw belongs to represented class ARIS_ComputerHardwareClassRole_ComputerHardware, but the corresponding ontology property UseLaw does not belong to ontology class ComputerHardware.
  - Represented property ARIS_EnvironmentalDataTLawRole_FunctionLaw belongs to represented class ARIS_EnvironmentalDataClassRole_EnvironmentalData, but the corresponding ontology property FunctionLaw does not belong to ontology class InputOutputThing.
  - Represented property ARIS_EnvironmentalDataTLawRole_FunctionLaw belongs to represented class ARIS_EnvironmentalDataClassRole_EnvironmentalData, but the corresponding ontology property FunctionLaw does not belong to ontology class ReactiveThing.
  - Represented property ARIS_FunctionTLawRole_FunctionLaw belongs to represented class ARIS_FunctionClassRole_OrganizationalUnit, but the corresponding ontology property FunctionLaw does not belong to ontology class OrganizationalUnit.
  - Represented property ARIS_FunctionTLawRole_Participation belongs to represented class ARIS_FunctionClassRole_HumanOutput, but the corresponding ontology property ParticipationLaw does not belong to ontology class HumanOutput.
  - Represented property ARIS_FunctionTLawRole_UseLaw belongs to represented class ARIS_FunctionClassRole_ComputerHardware, but the corresponding ontology property UseLaw does not belong to ontology class ComputerHardware.
  - Represented property ARIS_FunctionTLawRole_UseLaw belongs to represented class ARIS_FunctionClassRole_Machine, but the corresponding ontology property UseLaw does not belong to ontology class MachineResource.
  - Represented property ARIS_HumanOutputTLaw_FunctionLaw belongs to represented class ARIS_HumanOutputClassRole_OrganizationalUnit, but the corresponding ontology property FunctionLaw does not belong to ontology class OrganizationalUnit.

- Represented property ARIS_HumanOutputTLaw_Participation belongs to represented class ARIS_HumanOutputClassRole_HumanOutput, but the corresponding ontology property ParticipationLaw does not belong to ontology class HumanOutput.
- Represented property ARIS_InformationServiceTLawRole_FunctionLawInput belongs to represented class ARIS_InformationServiceClassRole_TargetInformationService, but the corresponding ontology property FunctionLaw does not belong to ontology class InformationService.
- Represented property ARIS_InformationServiceTLawRole_FunctionLawInput belongs to represented class ARIS_InformationServiceClassRole_TargetInformationService, but the corresponding ontology property FunctionLaw does not belong to ontology class InputThing.
- Represented property ARIS_InformationServiceTLawRole_FunctionLawOutput belongs to represented class ARIS_InformationServiceClassRole_SourceInformationService, but the corresponding ontology property FunctionLaw does not belong to ontology class InformationService.
- Represented property ARIS_InformationServiceTLawRole_FunctionLawOutput belongs to represented class ARIS_InformationServiceClassRole_SourceInformationService, but the corresponding ontology property FunctionLaw does not belong to ontology class OutputThing.
- Represented property ARIS_MachineResourceTLaw_FunctionLaw belongs to represented class ARIS_MachineResourceClassRole_OrganizationalUnit, but the corresponding ontology property FunctionLaw does not belong to ontology class OrganizationalUnit.
- Represented property ARIS_MachineResourceTLaw_UseLaw belongs to represented class ARIS_MachineResourceClassRole_Machine, but the corresponding ontology property UseLaw does not belong to ontology class MachineResource.
- Represented property ARIS_MaterialOutputTLawRole_FunctionLawInput belongs to represented class ARIS_MaterialOutputClassRole_TargetMaterialOutput, but the corresponding ontology property FunctionLaw does not belong to ontology class InputThing.
- Represented property ARIS_MaterialOutputTLawRole_FunctionLawInput belongs to represented class ARIS_MaterialOutputClassRole_TargetMaterialOutput, but the corresponding ontology property FunctionLaw does not belong to ontology class MaterialRepository.
- Represented property ARIS_MaterialOutputTLawRole_FunctionLawOutput belongs to represented class ARIS_MaterialOutputClassRole_SourceMaterialOutput, but the corresponding ontology property FunctionLaw does not belong to ontology class MaterialRepository.
- Represented property ARIS_MaterialOutputTLawRole_FunctionLawOutput belongs to represented class ARIS_MaterialOutputClassRole_SourceMaterialOutput, but the corresponding ontology property FunctionLaw does not belong to ontology class OutputThing.
- Represented property ARIS_OrganizationUnitTLawRole_Function belongs to represented class ARIS_OrganizationalUnitClassRole_OrganizationalUnit, but the corresponding ontology property FunctionLaw does not belong to ontology class OrganizationalUnit.
- Represented property ARIS_OtherServiceTLawRole_FunctionLawInput belongs to represented class ARIS_OtherServiceClassRole_TargetOtherService, but the corresponding ontology property FunctionLaw does not belong to ontology class InputThing.
- Represented property ARIS_OtherServiceTLawRole_FunctionLawInput belongs to represented class ARIS_OtherServiceClassRole_TargetOtherService, but the corresponding ontology property FunctionLaw does not belong to ontology class MaterialService.
- Represented property ARIS_OtherServiceTLawRole_FunctionLawOutput belongs to represented class ARIS_OtherServiceClassRole_SourceOtherService, but the corresponding ontology property FunctionLaw does not belong to ontology class MaterialService.
- Represented property ARIS_OtherServiceTLawRole_FunctionLawOutput belongs to represented class ARIS_OtherServiceClassRole_SourceOtherService, but the corresponding ontology property FunctionLaw does not belong to ontology class OutputThing.

- Represented property ARIS_OutputTLawRole_FunctionLawInput belongs to represented class ARIS_OutputClassRole_TargetOutput, but the corresponding ontology property FunctionLaw does not belong to ontology class InputThing.
- Represented property ARIS_OutputTLawRole_FunctionLawInput belongs to represented class ARIS_OutputClassRole_TargetOutput, but the corresponding ontology property FunctionLaw does not belong to ontology class Repository.
- Represented property ARIS_OutputTLawRole_FunctionLawOutput belongs to represented class ARIS_OutputClassRole_SourceOutput, but the corresponding ontology property FunctionLaw does not belong to ontology class OutputThing.
- Represented property ARIS_OutputTLawRole_FunctionLawOutput belongs to represented class ARIS_OutputClassRole_SourceOutput, but the corresponding ontology property FunctionLaw does not belong to ontology class Repository.
- Represented property ARIS_ServiceTLawRole_FunctionLawInput belongs to represented class ARIS_ServiceClassRole_TargetService, but the corresponding ontology property FunctionLaw does not belong to ontology class InputThing.
- Represented property ARIS_ServiceTLawRole_FunctionLawInput belongs to represented class ARIS_ServiceClassRole_TargetService, but the corresponding ontology property FunctionLaw does not belong to ontology class Service.
- Represented property ARIS_ServiceTLawRole_FunctionLawOutput belongs to represented class ARIS_ServiceClassRole_SourceService, but the corresponding ontology property FunctionLaw does not belong to ontology class OutputThing.
- Represented property ARIS_ServiceTLawRole_FunctionLawOutput belongs to represented class ARIS_ServiceClassRole_SourceService, but the corresponding ontology property FunctionLaw does not belong to ontology class Service.
- Represented property BPMN_ComplexGatewayTLaw_ComplexGateway belongs to represented class BPMN_ComplexGatewayClassRole_InputComplexGateway, but the corresponding ontology property MutualLaw does not belong to ontology class CoupledThing.
- Represented property BPMN_ComplexGatewayTLaw_ComplexGateway belongs to represented class BPMN_ComplexGatewayClassRole_OutputComplxGateway, but the corresponding ontology property MutualLaw does not belong to ontology class CoupledThing.
- Represented property BPMN_DataBasedExclusiveGatewayTLawRole_DataBasedExclusiveGateway belongs to represented class BPMN_DataBasedExclusiveGatewayClassRole_InputGateway, but the corresponding ontology property MutualLaw does not belong to ontology class CoupledThing.
- Represented property BPMN_DataBasedExclusiveGatewayTLawRole_DataBasedExclusiveGateway belongs to represented class BPMN_DataBasedExclusiveGatewayClassRole_OutputGateway, but the corresponding ontology property MutualLaw does not belong to ontology class CoupledThing.
- Represented property BPMN_EndEventTLawRole_Tlaw belongs to represented class BPMN_EndEventClassRole_SourceISF, but the corresponding ontology property TransformationLaw does not belong to ontology class OutputThing.
- Represented property BPMN_EventBasedExclusiveGatewayTLawRole_EventBasedExclusiveGateway belongs to represented class BPMN_EventBasedExclusiveGatewayClassRole_InputGateway, but the corresponding ontology property MutualLaw does not belong to ontology class CoupledThing.
- Represented property BPMN_EventBasedExclusiveGatewayTLawRole_EventBasedExclusiveGateway belongs to represented class BPMN_EventBasedExclusiveGatewayClassRole_OutputGateway, but the corresponding ontology property MutualLaw does not belong to ontology class CoupledThing.
- Represented property BPMN_EventTLawRole_TLaTargetMF belongs to represented class BPMN_EventClassRole_TargetMF, but the corresponding ontology property TransformationLaw does not belong to ontology class InputThing.

- Represented property BPMN_EventTLawRole_TLawTargetSF belongs to represented class BPMN_EventClassRole_TargetSF, but the corresponding ontology property TransformationLaw does not belong to ontology class InputThing.
- Represented property BPMN_GatewayTLaw_Gateway belongs to represented class BPMN_GatewayClassRole_InputGateway, but the corresponding ontology property MutualLaw does not belong to ontology class CoupledThing.
- Represented property BPMN_GatewayTLaw_Gateway belongs to represented class BPMN_GatewayClassRole_OutputGateway, but the corresponding ontology property MutualLaw does not belong to ontology class CoupledThing.
- Represented property BPMN_InclusiveGatewayTLawRole_InclusiveGateway belongs to represented class BPMN_InclusiveGatewayClassRole_InputGateway, but the corresponding ontology property MutualLaw does not belong to ontology class CoupledThing.
- Represented property BPMN_InclusiveGatewayTLawRole_InclusiveGateway belongs to represented class BPMN_InclusiveGatewayClassRole_OutputInclusiveGateway, but the corresponding ontology property MutualLaw does not belong to ontology class CoupledThing.
- Represented property BPMN_IntermediateEventTLawRole_TLaw belongs to represented class BPMN_IntermediateEventClassRole_TargetOSF, but the corresponding ontology property TransformationLaw does not belong to ontology class InputThing.
- Represented property BPMN_ParallelGatewayTLaw_ParallelGateway belongs to represented class BPMN_ParallelGatewayClassRole_InputParallelGateway, but the corresponding ontology property MutualLaw does not belong to ontology class CoupledThing.
- Represented property BPMN_ParallelGatewayTLaw_ParallelGateway belongs to represented class BPMN_ParallelGatewayClassRole_OutputParallelGateway, but the corresponding ontology property MutualLaw does not belong to ontology class CoupledThing.
- Represented property BPMN_ProcessTLaw_ProcessLaw belongs to represented class BPMN_ProcessClassRole_Process, but the corresponding ontology property TransformationLaw does not belong to ontology class System.
- Represented property BPMN_StartEventTLawRole_TLaw belongs to represented class BPMN_StartEventClassRole_TargetOSF, but the corresponding ontology property TransformationLaw does not belong to ontology class InputThing.
- Represented property BPMN_SubProcessTLaw_ProcessLaw belongs to represented class BPMN_SubProcessClassRole_Process, but the corresponding ontology property TransformationLaw does not belong to ontology class System.
- Represented property BPMN_TaskTLaw-ProcessLaw belongs to represented class BPMN_TaskClassRole_Process, but the corresponding ontology property TransformationLaw does not belong to ontology class System.

- **Entry *possessesProperty relation* not complete**
  - Represented class AIRS_ApplicationSoftwareClassRole_Software possesses represented property AIRS_ApplicationSoftwarePropertyRole_Rule, but the corresponding ontology class ExecutingThing does not possess ontology property RegularProperty.
  - Represented class AIRS_ApplicationSoftwareClassRole_Software possesses represented property ARIS_ApplicationSoftwareTLawRole_ApplicationLaw, but the corresponding ontology class ExecutingThing does not possess ontology property ApplicationLaw.
  - Represented class AIRS_ApplicationSoftwareClassRole_Software possesses represented property ARIS_ApplicationSoftwareTLawRole_FunctionLaw, but the corresponding ontology class ExecutingThing does not possess ontology property FunctionLaw.
  - Represented class AIRS_ApplicationSoftwareClassRole_Software possesses represented property ARIS_ApplicationsoftwarePropertyRole_Name, but the corresponding ontology class ExecutingThing does not possess ontology property Name.

- Represented class ARIS_AndClassRole_InputAnd possesses represented property ARIS_AndPropertyRole_And, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class ARIS_AndClassRole_InputAnd possesses represented property ARIS_AndPropertyRole_EndingInput, but the corresponding ontology class CoupledThing does not possess ontology property Flow.
- Represented class ARIS_AndClassRole_OutputAnd possesses represented property ARIS_AndPropertyRole_And, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class ARIS_AndClassRole_OutputAnd possesses represented property ARIS_AndPropertyRole_EndingOutput, but the corresponding ontology class CoupledThing does not possess ontology property Flow.
- Represented class ARIS_ComputerHardwareClassRole_ComputerHardware possesses represented property ARIS_ComputerHardwarePropertyRole_Allocation, but the corresponding ontology class ComputerHardware does not possess ontology property MutualProperty.
- Represented class ARIS_ComputerHardwareClassRole_ComputerHardware possesses represented property ARIS_ComputerHardwarePropertyRole_Name, but the corresponding ontology class ComputerHardware does not possess ontology property Name.
- Represented class ARIS_ComputerHardwareClassRole_ComputerHardware possesses represented property ARIS_ComputerHardwareTLawRole_UseLaw, but the corresponding ontology class ComputerHardware does not possess ontology property UseLaw.
- Represented class ARIS_ComputerHardwareClassRole_OrganizationalUnit possesses represented property ARIS_ComputerHardwarePropertyRole_Allocation, but the corresponding ontology class OrganizationalUnit does not possess ontology property MutualProperty.
- Represented class ARIS_ComputerHardwareClassRole_OrganizationalUnit possesses represented property ARIS_ComputerHardwarePropertyRole_NameOrganizationalUnit, but the corresponding ontology class OrganizationalUnit does not possess ontology property Name.
- Represented class ARIS_ComputerHardwareClassRole_OrganizationalUnit possesses represented property ARIS_ComputerHardwareTLawRole_Function, but the corresponding ontology class OrganizationalUnit does not possess ontology property FunctionLaw.
- Represented class ARIS_EnvironmentalDataClassRole_EnvironmentalData possesses represented property ARIS_EnvironmentalDataPropertyRole_ChangingAttribute, but the corresponding ontology class InputOutputThing does not possess ontology property RegularMutableProperty.
- Represented class ARIS_EnvironmentalDataClassRole_EnvironmentalData possesses represented property ARIS_EnvironmentalDataPropertyRole_ChangingAttribute, but the corresponding ontology class InputOutputThing does not possess ontology property _AttributedThing_RegularProperty_OntologyClassPropertyRelation.
- Represented class ARIS_EnvironmentalDataClassRole_EnvironmentalData possesses represented property ARIS_EnvironmentalDataPropertyRole_ChangingAttribute, but the corresponding ontology class ReactiveThing does not possess ontology property RegularMutableProperty.
- Represented class ARIS_EnvironmentalDataClassRole_EnvironmentalData possesses represented property ARIS_EnvironmentalDataPropertyRole_ChangingAttribute, but the corresponding ontology class ReactiveThing does not possess ontology property _AttributedThing_RegularProperty_OntologyClassPropertyRelation.
- Represented class ARIS_EnvironmentalDataClassRole_EnvironmentalData possesses represented property ARIS_EnvironmentalDataPropertyRole_InformationFlow, but the corresponding ontology class InputOutputThing does not possess ontology property InteractionRelation.

- Represented class ARIS_EnvironmentalDataClassRole_EnvironmentalData possesses represented property ARIS_EnvironmentalDataPropertyRole_InformationFlow, but the corresponding ontology class ReactiveThing does not possess ontology property InteractionRelation.
- Represented class ARIS_EnvironmentalDataClassRole_EnvironmentalData possesses represented property ARIS_EnvironmentalDataPropertyRole_Name, but the corresponding ontology class InputOutputThing does not possess ontology property Name.
- Represented class ARIS_EnvironmentalDataClassRole_EnvironmentalData possesses represented property ARIS_EnvironmentalDataPropertyRole_Name, but the corresponding ontology class ReactiveThing does not possess ontology property Name.
- Represented class ARIS_EnvironmentalDataClassRole_EnvironmentalData possesses represented property ARIS_EnvironmentalDataTLawRole_FunctionLaw, but the corresponding ontology class InputOutputThing does not possess ontology property FunctionLaw.
- Represented class ARIS_EnvironmentalDataClassRole_EnvironmentalData possesses represented property ARIS_EnvironmentalDataTLawRole_FunctionLaw, but the corresponding ontology class ReactiveThing does not possess ontology property FunctionLaw.
- Represented class ARIS_FunctionClassRole_ComputerHardware possesses represented property ARIS_FunctionTLawRole_UseLaw, but the corresponding ontology class ComputerHardware does not possess ontology property UseLaw.
- Represented class ARIS_FunctionClassRole_HumanOutput possesses represented property ARIS_FunctionTLawRole_Participation, but the corresponding ontology class HumanOutput does not possess ontology property ParticipationLaw.
- Represented class ARIS_FunctionClassRole_Machine possesses represented property ARIS_FunctionTLawRole_UseLaw, but the corresponding ontology class MachineResource does not possess ontology property UseLaw.
- Represented class ARIS_FunctionClassRole_OrganizationalUnit possesses represented property ARIS_FunctionPropertyRole_Name, but the corresponding ontology class OrganizationalUnit does not possess ontology property Name.
- Represented class ARIS_FunctionClassRole_OrganizationalUnit possesses represented property ARIS_FunctionTLawRole_FunctionLaw, but the corresponding ontology class OrganizationalUnit does not possess ontology property FunctionLaw.
- Represented class ARIS_FunctionClassRole_Software possesses represented property ARIS_FunctionPropertyRole_ApplicationLaw, but the corresponding ontology class ExecutingThing does not possess ontology property ApplicationLaw.
- Represented class ARIS_FunctionClassRole_SourceOutput possesses represented property ARIS_FunctionPropertyRole_OutputFlowInput, but the corresponding ontology class OutputThing does not possess ontology property FlowContent.
- Represented class ARIS_FunctionClassRole_SourceOutput possesses represented property ARIS_FunctionPropertyRole_OutputFlowInput, but the corresponding ontology class Repository does not possess ontology property FlowContent.
- Represented class ARIS_FunctionClassRole_TargetOutput possesses represented property ARIS_FunctionPropertyRole_OutputFlow, but the corresponding ontology class InputThing does not possess ontology property FlowContent.
- Represented class ARIS_FunctionClassRole_TargetOutput possesses represented property ARIS_FunctionPropertyRole_OutputFlow, but the corresponding ontology class Repository does not possess ontology property FlowContent.
- Represented class ARIS_HumanOutputClassRole_HumanOutput possesses represented property ARIS_HumanOutputPropertyRole_Name, but the corresponding ontology class HumanOutput does not possess ontology property Name.
- Represented class ARIS_HumanOutputClassRole_HumanOutput possesses represented property ARIS_HumanOutputPropertyRole_RelationToWhole, but the corresponding ontology class HumanOutput does not possess ontology property PartWholeRelation.

- Represented class ARIS_HumanOutputClassRole_HumanOutput possesses represented property ARIS_HumanOutputTLaw_Participation, but the corresponding ontology class HumanOutput does not possess ontology property ParticipationLaw.
- Represented class ARIS_HumanOutputClassRole_OrganizationalUnit possesses represented property ARIS_HumanOutputPropertyRole_NameOrganizationalUnit, but the corresponding ontology class OrganizationalUnit does not possess ontology property Name.
- Represented class ARIS_HumanOutputClassRole_OrganizationalUnit possesses represented property ARIS_HumanOutputPropertyRole_RelationToWhole, but the corresponding ontology class OrganizationalUnit does not possess ontology property PartWholeRelation.
- Represented class ARIS_HumanOutputClassRole_OrganizationalUnit possesses represented property ARIS_HumanOutputTLaw_FunctionLaw, but the corresponding ontology class OrganizationalUnit does not possess ontology property FunctionLaw.
- Represented class ARIS_InformationServiceClassRole_SourceInformationService possesses represented property ARIS_InformationServicePropertyRole_IsInformation, but the corresponding ontology class InformationService does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_InformationServiceClassRole_SourceInformationService possesses represented property ARIS_InformationServicePropertyRole_IsInformation, but the corresponding ontology class OutputThing does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_InformationServiceClassRole_SourceInformationService possesses represented property ARIS_InformationServicePropertyRole_IsMaterial, but the corresponding ontology class InformationService does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_InformationServiceClassRole_SourceInformationService possesses represented property ARIS_InformationServicePropertyRole_IsMaterial, but the corresponding ontology class OutputThing does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_InformationServiceClassRole_SourceInformationService possesses represented property ARIS_InformationServicePropertyRole_Name, but the corresponding ontology class InformationService does not possess ontology property Name.
- Represented class ARIS_InformationServiceClassRole_SourceInformationService possesses represented property ARIS_InformationServicePropertyRole_Name, but the corresponding ontology class OutputThing does not possess ontology property Name.
- Represented class ARIS_InformationServiceClassRole_SourceInformationService possesses represented property ARIS_InformationServiceTLawRole_FunctionLawOutput, but the corresponding ontology class InformationService does not possess ontology property FunctionLaw.
- Represented class ARIS_InformationServiceClassRole_SourceInformationService possesses represented property ARIS_InformationServiceTLawRole_FunctionLawOutput, but the corresponding ontology class OutputThing does not possess ontology property FunctionLaw.
- Represented class ARIS_InformationServiceClassRole_TargetInformationService possesses represented property ARIS_InformationServicePropertyRole_IsInformation, but the corresponding ontology class InformationService does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_InformationServiceClassRole_TargetInformationService possesses represented property ARIS_InformationServicePropertyRole_IsInformation, but the corresponding ontology class InputThing does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_InformationServiceClassRole_TargetInformationService possesses represented property ARIS_InformationServicePropertyRole_IsMaterial, but the corresponding ontology class InformationService does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_InformationServiceClassRole_TargetInformationService possesses represented property ARIS_InformationServicePropertyRole_IsMaterial, but the corresponding ontology class InputThing does not possess ontology property RegularBooleanProperty.

- Represented class ARIS_InformationServiceClassRole_TargetInformationService possesses represented property ARIS_InformationServicePropertyRole_Name, but the corresponding ontology class InformationService does not possess ontology property Name.
- Represented class ARIS_InformationServiceClassRole_TargetInformationService possesses represented property ARIS_InformationServicePropertyRole_Name, but the corresponding ontology class InputThing does not possess ontology property Name.
- Represented class ARIS_InformationServiceClassRole_TargetInformationService possesses represented property ARIS_InformationServiceTLawRole_FunctionLawInput, but the corresponding ontology class InformationService does not possess ontology property FunctionLaw.
- Represented class ARIS_InformationServiceClassRole_TargetInformationService possesses represented property ARIS_InformationServiceTLawRole_FunctionLawInput, but the corresponding ontology class InputThing does not possess ontology property FunctionLaw.
- Represented class ARIS_MachineResourceClassRole_Machine possesses represented property ARIS_MachineResourcePropertyRole_Allocation, but the corresponding ontology class MachineResource does not possess ontology property MutualProperty.
- Represented class ARIS_MachineResourceClassRole_Machine possesses represented property ARIS_MachineResourcePropertyRole_Name, but the corresponding ontology class MachineResource does not possess ontology property Name.
- Represented class ARIS_MachineResourceClassRole_Machine possesses represented property ARIS_MachineResourceTLaw_UseLaw, but the corresponding ontology class MachineResource does not possess ontology property UseLaw.
- Represented class ARIS_MachineResourceClassRole_OrganizationalUnit possesses represented property ARIS_MachineResourcePropertyRole_Allocation, but the corresponding ontology class OrganizationalUnit does not possess ontology property MutualProperty.
- Represented class ARIS_MachineResourceClassRole_OrganizationalUnit possesses represented property ARIS_MachineResourcePropertyRole_NameOrganizationalUnit, but the corresponding ontology class OrganizationalUnit does not possess ontology property Name.
- Represented class ARIS_MachineResourceClassRole_OrganizationalUnit possesses represented property ARIS_MachineResourceTLaw_FunctionLaw, but the corresponding ontology class OrganizationalUnit does not possess ontology property FunctionLaw.
- Represented class ARIS_MaterialOutputClassRole_SourceMaterialOutput possesses represented property ARIS_MaterialOutputPropertyRole_IsMaterial, but the corresponding ontology class MaterialRepository does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_MaterialOutputClassRole_SourceMaterialOutput possesses represented property ARIS_MaterialOutputPropertyRole_IsMaterial, but the corresponding ontology class OutputThing does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_MaterialOutputClassRole_SourceMaterialOutput possesses represented property ARIS_MaterialOutputPropertyRole_Name, but the corresponding ontology class MaterialRepository does not possess ontology property Name.
- Represented class ARIS_MaterialOutputClassRole_SourceMaterialOutput possesses represented property ARIS_MaterialOutputPropertyRole_Name, but the corresponding ontology class OutputThing does not possess ontology property Name.
- Represented class ARIS_MaterialOutputClassRole_SourceMaterialOutput possesses represented property ARIS_MaterialOutputTLawRole_FunctionLawOutput, but the corresponding ontology class MaterialRepository does not possess ontology property FunctionLaw.
- Represented class ARIS_MaterialOutputClassRole_SourceMaterialOutput possesses represented property ARIS_MaterialOutputTLawRole_FunctionLawOutput, but the corresponding ontology class OutputThing does not possess ontology property FunctionLaw.
- Represented class ARIS_MaterialOutputClassRole_TargetMaterialOutput possesses represented property ARIS_MaterialOutputPropertyRole_IsMaterial, but the corresponding ontology class InputThing does not possess ontology property RegularBooleanProperty.

- Represented class ARIS_MaterialOutputClassRole_TargetMaterialOutput possesses represented property ARIS_MaterialOutputPropertyRole_IsMaterial, but the corresponding ontology class MaterialRepository does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_MaterialOutputClassRole_TargetMaterialOutput possesses represented property ARIS_MaterialOutputPropertyRole_Name, but the corresponding ontology class InputThing does not possess ontology property Name.
- Represented class ARIS_MaterialOutputClassRole_TargetMaterialOutput possesses represented property ARIS_MaterialOutputPropertyRole_Name, but the corresponding ontology class MaterialRepository does not possess ontology property Name.
- Represented class ARIS_MaterialOutputClassRole_TargetMaterialOutput possesses represented property ARIS_MaterialOutputTLawRole_FunctionLawInput, but the corresponding ontology class InputThing does not possess ontology property FunctionLaw.
- Represented class ARIS_MaterialOutputClassRole_TargetMaterialOutput possesses represented property ARIS_MaterialOutputTLawRole_FunctionLawInput, but the corresponding ontology class MaterialRepository does not possess ontology property FunctionLaw.
- Represented class ARIS_OrganizationalUnitClassRole_HumanOutput possesses represented property ARIS_OrganizationalUnitPropertyRole_NameHumanOutput, but the corresponding ontology class HumanOutput does not possess ontology property Name.
- Represented class ARIS_OrganizationalUnitClassRole_HumanOutput possesses represented property ARIS_OrganizationalUnitPropertyRole_RelationToPartHumanOutput, but the corresponding ontology class HumanOutput does not possess ontology property PartWholeRelation.
- Represented class ARIS_OrganizationalUnitClassRole_OrganizationalUnit possesses represented property ARIS_ComputerHardwarePropertyRole_NameOrganizationalUnit, but the corresponding ontology class OrganizationalUnit does not possess ontology property Name.
- Represented class ARIS_OrganizationalUnitClassRole_OrganizationalUnit possesses represented property ARIS_OrganizationUnitTLawRole_Function, but the corresponding ontology class OrganizationalUnit does not possess ontology property FunctionLaw.
- Represented class ARIS_OrganizationalUnitClassRole_OrganizationalUnit possesses represented property ARIS_OrganizationalUnitPropertyRole_Location, but the corresponding ontology class OrganizationalUnit does not possess ontology property Location.
- Represented class ARIS_OrganizationalUnitClassRole_OrganizationalUnit possesses represented property ARIS_OrganizationalUnitPropertyRole_Name, but the corresponding ontology class OrganizationalUnit does not possess ontology property Name.
- Represented class ARIS_OrganizationalUnitClassRole_OrganizationalUnit possesses represented property ARIS_OrganizationalUnitPropertyRole_Participation, but the corresponding ontology class OrganizationalUnit does not possess ontology property RegularProperty.
- Represented class ARIS_OrganizationalUnitClassRole_OrganizationalUnit possesses represented property ARIS_OrganizationalUnitPropertyRole_RelationToPart, but the corresponding ontology class OrganizationalUnit does not possess ontology property PartWholeRelation.
- Represented class ARIS_OrganizationalUnitClassRole_OrganizationalUnit possesses represented property ARIS_OrganizationalUnitPropertyRole_RelationToPartHumanOutput, but the corresponding ontology class OrganizationalUnit does not possess ontology property PartWholeRelation.
- Represented class ARIS_OrganizationalUnitClassRole_Position possesses represented property ARIS_OrganizationalUnitPropertyRole_NamePosition, but the corresponding ontology class RoleHolder does not possess ontology property Name.
- Represented class ARIS_OrganizationalUnitClassRole_Position possesses represented property ARIS_OrganizationalUnitPropertyRole_RelationToPart, but the corresponding ontology class RoleHolder does not possess ontology property PartWholeRelation.

- Represented class ARIS_OtherServiceClassRole_SourceOtherService possesses represented property ARIS_OtherServicePropertyRole_IsInformation, but the corresponding ontology class MaterialService does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_OtherServiceClassRole_SourceOtherService possesses represented property ARIS_OtherServicePropertyRole_IsInformation, but the corresponding ontology class OutputThing does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_OtherServiceClassRole_SourceOtherService possesses represented property ARIS_OtherServicePropertyRole_IsMaterial, but the corresponding ontology class MaterialService does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_OtherServiceClassRole_SourceOtherService possesses represented property ARIS_OtherServicePropertyRole_IsMaterial, but the corresponding ontology class OutputThing does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_OtherServiceClassRole_SourceOtherService possesses represented property ARIS_OtherServicePropertyRole_Name, but the corresponding ontology class MaterialService does not possess ontology property Name.
- Represented class ARIS_OtherServiceClassRole_SourceOtherService possesses represented property ARIS_OtherServicePropertyRole_Name, but the corresponding ontology class OutputThing does not possess ontology property Name.
- Represented class ARIS_OtherServiceClassRole_SourceOtherService possesses represented property ARIS_OtherServiceTLawRole_FunctionLawOutput, but the corresponding ontology class MaterialService does not possess ontology property FunctionLaw.
- Represented class ARIS_OtherServiceClassRole_SourceOtherService possesses represented property ARIS_OtherServiceTLawRole_FunctionLawOutput, but the corresponding ontology class OutputThing does not possess ontology property FunctionLaw.
- Represented class ARIS_OtherServiceClassRole_TargetOtherService possesses represented property ARIS_OtherServicePropertyRole_IsInformation, but the corresponding ontology class InputThing does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_OtherServiceClassRole_TargetOtherService possesses represented property ARIS_OtherServicePropertyRole_IsInformation, but the corresponding ontology class MaterialService does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_OtherServiceClassRole_TargetOtherService possesses represented property ARIS_OtherServicePropertyRole_IsMaterial, but the corresponding ontology class InputThing does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_OtherServiceClassRole_TargetOtherService possesses represented property ARIS_OtherServicePropertyRole_IsMaterial, but the corresponding ontology class MaterialService does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_OtherServiceClassRole_TargetOtherService possesses represented property ARIS_OtherServicePropertyRole_Name, but the corresponding ontology class InputThing does not possess ontology property Name.
- Represented class ARIS_OtherServiceClassRole_TargetOtherService possesses represented property ARIS_OtherServicePropertyRole_Name, but the corresponding ontology class MaterialService does not possess ontology property Name.
- Represented class ARIS_OtherServiceClassRole_TargetOtherService possesses represented property ARIS_OtherServiceTLawRole_FunctionLawInput, but the corresponding ontology class InputThing does not possess ontology property FunctionLaw.
- Represented class ARIS_OtherServiceClassRole_TargetOtherService possesses represented property ARIS_OtherServiceTLawRole_FunctionLawInput, but the corresponding ontology class MaterialService does not possess ontology property FunctionLaw.
- Represented class ARIS_OutputClassRole_SourceOutput possesses represented property ARIS_OutputPropertyRole_Attributes, but the corresponding ontology class OutputThing does not possess ontology property RegularProperty.

- Represented class ARIS_OutputClassRole_SourceOutput possesses represented property ARIS_OutputPropertyRole_Attributes, but the corresponding ontology class Repository does not possess ontology property RegularProperty.
- Represented class ARIS_OutputClassRole_SourceOutput possesses represented property ARIS_OutputPropertyRole_Name, but the corresponding ontology class OutputThing does not possess ontology property Name.
- Represented class ARIS_OutputClassRole_SourceOutput possesses represented property ARIS_OutputPropertyRole_Name, but the corresponding ontology class Repository does not possess ontology property Name.
- Represented class ARIS_OutputClassRole_SourceOutput possesses represented property ARIS_OutputTLawRole_FunctionLawOutput, but the corresponding ontology class OutputThing does not possess ontology property FunctionLaw.
- Represented class ARIS_OutputClassRole_SourceOutput possesses represented property ARIS_OutputTLawRole_FunctionLawOutput, but the corresponding ontology class Repository does not possess ontology property FunctionLaw.
- Represented class ARIS_OutputClassRole_TargetOutput possesses represented property ARIS_OutputPropertyRole_Attributes, but the corresponding ontology class InputThing does not possess ontology property RegularProperty.
- Represented class ARIS_OutputClassRole_TargetOutput possesses represented property ARIS_OutputPropertyRole_Attributes, but the corresponding ontology class Repository does not possess ontology property RegularProperty.
- Represented class ARIS_OutputClassRole_TargetOutput possesses represented property ARIS_OutputPropertyRole_Name, but the corresponding ontology class InputThing does not possess ontology property Name.
- Represented class ARIS_OutputClassRole_TargetOutput possesses represented property ARIS_OutputPropertyRole_Name, but the corresponding ontology class Repository does not possess ontology property Name.
- Represented class ARIS_OutputClassRole_TargetOutput possesses represented property ARIS_OutputTLawRole_FunctionLawInput, but the corresponding ontology class InputThing does not possess ontology property FunctionLaw.
- Represented class ARIS_OutputClassRole_TargetOutput possesses represented property ARIS_OutputTLawRole_FunctionLawInput, but the corresponding ontology class Repository does not possess ontology property FunctionLaw.
- Represented class ARIS_PositionClassRole_OrganizationUnit possesses represented property ARIS_PositionPropertyRole_NameOrgnanizationalUnit, but the corresponding ontology class OrganizationalUnit does not possess ontology property Name.
- Represented class ARIS_PositionClassRole_OrganizationUnit possesses represented property ARIS_PositionPropertyRole_RelationToWhole, but the corresponding ontology class OrganizationalUnit does not possess ontology property PartWholeRelation.
- Represented class ARIS_PositionClassRole_Position possesses represented property ARIS_PositionPropertyRole_Name, but the corresponding ontology class RoleHolder does not possess ontology property Name.
- Represented class ARIS_PositionClassRole_Position possesses represented property ARIS_PositionPropertyRole_RelationToWhole, but the corresponding ontology class RoleHolder does not possess ontology property PartWholeRelation.
- Represented class ARIS_ServiceClassRole_SourceService possesses represented property ARIS_ServicePropertyRole_IsMaterial, but the corresponding ontology class OutputThing does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_ServiceClassRole_SourceService possesses represented property ARIS_ServicePropertyRole_IsMaterial, but the corresponding ontology class Service does not possess ontology property RegularBooleanProperty.

- Represented class ARIS_ServiceClassRole_SourceService possesses represented property ARIS_ServicePropertyRole_Name, but the corresponding ontology class OutputThing does not possess ontology property Name.
- Represented class ARIS_ServiceClassRole_SourceService possesses represented property ARIS_ServicePropertyRole_Name, but the corresponding ontology class Service does not possess ontology property Name.
- Represented class ARIS_ServiceClassRole_SourceService possesses represented property ARIS_ServiceTLawRole_FunctionLawOutput, but the corresponding ontology class OutputThing does not possess ontology property FunctionLaw.
- Represented class ARIS_ServiceClassRole_SourceService possesses represented property ARIS_ServiceTLawRole_FunctionLawOutput, but the corresponding ontology class Service does not possess ontology property FunctionLaw.
- Represented class ARIS_ServiceClassRole_TargetService possesses represented property ARIS_ServicePropertyRole_IsMaterial, but the corresponding ontology class InputThing does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_ServiceClassRole_TargetService possesses represented property ARIS_ServicePropertyRole_IsMaterial, but the corresponding ontology class Service does not possess ontology property RegularBooleanProperty.
- Represented class ARIS_ServiceClassRole_TargetService possesses represented property ARIS_ServicePropertyRole_Name, but the corresponding ontology class InputThing does not possess ontology property Name.
- Represented class ARIS_ServiceClassRole_TargetService possesses represented property ARIS_ServicePropertyRole_Name, but the corresponding ontology class Service does not possess ontology property Name.
- Represented class ARIS_ServiceClassRole_TargetService possesses represented property ARIS_ServiceTLawRole_FunctionLawInput, but the corresponding ontology class InputThing does not possess ontology property FunctionLaw.
- Represented class ARIS_ServiceClassRole_TargetService possesses represented property ARIS_ServiceTLawRole_FunctionLawInput, but the corresponding ontology class Service does not possess ontology property FunctionLaw.
- Represented class BPMN_ComplexGatewayClassRole_InputComplexGateway possesses represented property BPMN_ComplexGatewayPropertyRole_InputCoupling, but the corresponding ontology class CoupledThing does not possess ontology property CouplingRelation.
- Represented class BPMN_ComplexGatewayClassRole_InputComplexGateway possesses represented property BPMN_ComplexGatewayTLaw_ComplexGateway, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class BPMN_ComplexGatewayClassRole_OutputComplxGateway possesses represented property BPMN_ComplexGatewayPropertyRole_OutputCoupling, but the corresponding ontology class CoupledThing does not possess ontology property CouplingRelation.
- Represented class BPMN_ComplexGatewayClassRole_OutputComplxGateway possesses represented property BPMN_ComplexGatewayTLaw_ComplexGateway, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class BPMN_DataBasedExclusiveGatewayClassRole_InputGateway possesses represented property BPMN_DataBasedExclusiveGatewayPropertyRole_ DataBasedExclusiveInputCoupling, but the corresponding ontology class CoupledThing does not possess ontology property CouplingRelation.
- Represented class BPMN_DataBasedExclusiveGatewayClassRole_InputGateway possesses represented property BPMN_DataBasedExclusiveGatewayTLawRole_DataBasedExclusiveGateway, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.

- Represented class BPMN_DataBasedExclusiveGatewayClassRole_OutputGateway possesses represented property BPMN_DataBasedExclusiveGatewayPropertyRole_ DataBasedExclusiveOutputCoupling, but the corresponding ontology class CoupledThing does not possess ontology property CouplingRelation.
- Represented class BPMN_DataBasedExclusiveGatewayClassRole_OutputGateway possesses represented property BPMN_DataBasedExclusiveGatewayTLawRole_DataBasedExclusiveGateway, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class BPMN_EndEventClassRole_SourceISF possesses represented property BPMN_EndEventPropertyRole_IncomingSequenceFlow, but the corresponding ontology class OutputThing does not possess ontology property Flow.
- Represented class BPMN_EndEventClassRole_SourceISF possesses represented property BPMN_EndEventTLawRole_Tlaw, but the corresponding ontology class OutputThing does not possess ontology property TransformationLaw.
- Represented class BPMN_EndEventClassRole_TargetOMF possesses represented property BPMN_EndEventPropertyRole_OutgoingMessageFlow, but the corresponding ontology class InputThing does not possess ontology property Flow.
- Represented class BPMN_EventBasedExclusiveGatewayClassRole_InputGateway possesses represented property BPMN_EventBasedExclusiveGatewayPropertyRole_ EventBasedExclusiveInputCoupling, but the corresponding ontology class CoupledThing does not possess ontology property CouplingRelation.
- Represented class BPMN_EventBasedExclusiveGatewayClassRole_InputGateway possesses represented property BPMN_EventBasedExclusiveGatewayTLawRole_EventBasedExclusiveGateway, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class BPMN_EventBasedExclusiveGatewayClassRole_OutputGateway possesses represented property BPMN_EventBasedExclusiveGatewayPropertyRole_ EventBasedExclusiveOutputCoupling, but the corresponding ontology class CoupledThing does not possess ontology property CouplingRelation.
- Represented class BPMN_EventBasedExclusiveGatewayClassRole_OutputGateway possesses represented property BPMN_EventBasedExclusiveGatewayTLawRole_EventBasedExclusiveGateway, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class BPMN_EventClassRole_SourceMF possesses represented property BPMN_EventPropertyRole_MessageFlow, but the corresponding ontology class OutputThing does not possess ontology property Flow.
- Represented class BPMN_EventClassRole_SourceSF possesses represented property BPMN_EventPropertyRole_SequenceFlow, but the corresponding ontology class OutputThing does not possess ontology property Flow.
- Represented class BPMN_EventClassRole_TargetMF possesses represented property BPMN_EventPropertyRole_MessageFlow, but the corresponding ontology class InputThing does not possess ontology property Flow.
- Represented class BPMN_EventClassRole_TargetMF possesses represented property BPMN_EventTLawRole_TLaTargetMF, but the corresponding ontology class InputThing does not possess ontology property TransformationLaw.
- Represented class BPMN_EventClassRole_TargetSF possesses represented property BPMN_EventPropertyRole_SequenceFlow, but the corresponding ontology class InputThing does not possess ontology property Flow.
- Represented class BPMN_EventClassRole_TargetSF possesses represented property BPMN_EventTLawRole_TLawTargetSF, but the corresponding ontology class InputThing does not possess ontology property TransformationLaw.

- Represented class BPMN_GatewayClassRole_InputGateway possesses represented property BPMN_GatewayPropertyRole_InputCoupling, but the corresponding ontology class CoupledThing does not possess ontology property CouplingRelation.
- Represented class BPMN_GatewayClassRole_InputGateway possesses represented property BPMN_GatewayTLaw_Gateway, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class BPMN_GatewayClassRole_OutputGateway possesses represented property BPMN_GatewayPropertyRole_OutputCoupling, but the corresponding ontology class CoupledThing does not possess ontology property CouplingRelation.
- Represented class BPMN_GatewayClassRole_OutputGateway possesses represented property BPMN_GatewayTLaw_Gateway, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class BPMN_InclusiveGatewayClassRole_InputGateway possesses represented property BPMN_InclusiveGatewayPropertyRole_InclusiveInputCoupling, but the corresponding ontology class CoupledThing does not possess ontology property CouplingRelation.
- Represented class BPMN_InclusiveGatewayClassRole_InputGateway possesses represented property BPMN_InclusiveGatewayTLawRole_InclusiveGateway, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class BPMN_InclusiveGatewayClassRole_OutputInclusiveGateway possesses represented property BPMN_InclusiveGatewayPropertyRole_InclusiveOutputCoupling, but the corresponding ontology class CoupledThing does not possess ontology property CouplingRelation.
- Represented class BPMN_InclusiveGatewayClassRole_OutputInclusiveGateway possesses represented property BPMN_InclusiveGatewayTLawRole_InclusiveGateway, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class BPMN_IntermediateEventClassRole_SourceIMF possesses represented property BPMN_IntermediateEventPropertyRole_IncomingMessageFlow, but the corresponding ontology class OutputThing does not possess ontology property Flow.
- Represented class BPMN_IntermediateEventClassRole_SourceISF possesses represented property BPMN_IntermediateEventPropertyRole_IncomingSequenceFlow, but the corresponding ontology class OutputThing does not possess ontology property Flow.
- Represented class BPMN_IntermediateEventClassRole_TargetOSF possesses represented property BPMN_IntermediateEventPropertyRole_OutgoingSequenceFlow, but the corresponding ontology class InputThing does not possess ontology property Flow.
- Represented class BPMN_IntermediateEventClassRole_TargetOSF possesses represented property BPMN_IntermediateEventTLawRole_TLaw, but the corresponding ontology class InputThing does not possess ontology property TransformationLaw.
- Represented class BPMN_LaneClassRole_Pool possesses represented property BPMN_LanePropertyRole_ParentPool, but the corresponding ontology class ActiveThing does not possess ontology property PartWholeRelation.
- Represented class BPMN_MessageFlowClassRole_Source possesses represented property BPMN_MessageFlowPropertyRole_MessageFlow, but the corresponding ontology class OutputThing does not possess ontology property Flow.
- Represented class BPMN_MessageFlowClassRole_Target possesses represented property BPMN_MessageFlowPropertyRole_MessageFlow, but the corresponding ontology class InputThing does not possess ontology property Flow.
- Represented class BPMN_ParallelGatewayClassRole_InputParallelGateway possesses represented property BPMN_ParallelGatewayPropertyRole_ParallelInputCoupling, but the corresponding ontology class CoupledThing does not possess ontology property CouplingRelation.

- Represented class BPMN_ParallelGatewayClassRole_InputParallelGateway possesses represented property BPMN_ParallelGatewayTLaw_ParallelGateway, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class BPMN_ParallelGatewayClassRole_OutputParallelGateway possesses represented property BPMN_ParallelGatewayPropertyRole_ParallelOutputCoupling, but the corresponding ontology class CoupledThing does not possess ontology property CouplingRelation.
- Represented class BPMN_ParallelGatewayClassRole_OutputParallelGateway possesses represented property BPMN_ParallelGatewayTLaw_ParallelGateway, but the corresponding ontology class CoupledThing does not possess ontology property MutualLaw.
- Represented class BPMN_PoolClassRole_Lanes possesses represented property BPMN_PoolPropertyRole_Lanes, but the corresponding ontology class ActiveThing does not possess ontology property PartWholeRelation.
- Represented class BPMN_PoolClassRole_Pool possesses represented property BPMN_PoolPropertyRole_BoundaryVisible, but the corresponding ontology class ActiveThing does not possess ontology property RegularBooleanProperty.
- Represented class BPMN_PoolClassRole_Pool possesses represented property BPMN_PoolPropertyRole_IncomingMessageFlow, but the corresponding ontology class ActiveThing does not possess ontology property Flow.
- Represented class BPMN_PoolClassRole_Pool possesses represented property BPMN_PoolPropertyRole_Lanes, but the corresponding ontology class ActiveThing does not possess ontology property PartWholeRelation.
- Represented class BPMN_PoolClassRole_Pool possesses represented property BPMN_PoolPropertyRole_Name, but the corresponding ontology class ActiveThing does not possess ontology property RegularStringProperty.
- Represented class BPMN_PoolClassRole_Pool possesses represented property BPMN_PoolPropertyRole_OutgoingMessageFlow, but the corresponding ontology class ActiveThing does not possess ontology property Flow.
- Represented class BPMN_PoolClassRole_Pool possesses represented property BPMN_PoolPropertyRole_Participant, but the corresponding ontology class ActiveThing does not possess ontology property RegularProperty.
- Represented class BPMN_PoolClassRole_Pool possesses represented property BPMN_PoolPropertyRole_Process, but the corresponding ontology class ActiveThing does not possess ontology property RegularProperty.
- Represented class BPMN_ProcessClassRole_FlowObject possesses represented property BPMN_ProcessPropertyRole_RelationToPart, but the corresponding ontology class Component does not possess ontology property PartWholeRelation.
- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessPropertyRole_AdHoc, but the corresponding ontology class System does not possess ontology property RegularBooleanProperty.
- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessPropertyRole_Assignements, but the corresponding ontology class System does not possess ontology property RegularProperty.
- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessPropertyRole_Categories, but the corresponding ontology class System does not possess ontology property RegularStringProperty.
- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessPropertyRole_Documentation, but the corresponding ontology class System does not possess ontology property RegularStringProperty.
- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessPropertyRole_EnableInstanceCompensation, but the corresponding ontology class System does not possess ontology property RegularBooleanProperty.

- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessPropertyRole_Id, but the corresponding ontology class System does not possess ontology property RegularProperty.
- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessPropertyRole_Name, but the corresponding ontology class System does not possess ontology property RegularStringProperty.
- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessPropertyRole_Properties, but the corresponding ontology class System does not possess ontology property RegularProperty.
- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessPropertyRole_RelationToPart, but the corresponding ontology class System does not possess ontology property PartWholeRelation.
- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessPropertyRole_Status, but the corresponding ontology class System does not possess ontology property RegularStringProperty.
- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessPropertyRole_SuppressJoinFailure, but the corresponding ontology class System does not possess ontology property RegularBooleanProperty.
- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessPropertyRole_Type, but the corresponding ontology class System does not possess ontology property RegularStringProperty.
- Represented class BPMN_ProcessClassRole_Process possesses represented property BPMN_ProcessTLaw_ProcessLaw, but the corresponding ontology class System does not possess ontology property TransformationLaw.
- Represented class BPMN_SequenceFlowClassRole_Source possesses represented property BPMN_SequenceFlowPropertyRole_SequenceFlow, but the corresponding ontology class OutputThing does not possess ontology property Flow.
- Represented class BPMN_SequenceFlowClassRole_Target possesses represented property BPMN_SequenceFlowPropertyRole_SequenceFlow, but the corresponding ontology class InputThing does not possess ontology property Flow.
- Represented class BPMN_StartEventClassRole_SourceIMF possesses represented property BPMN_StartEventPropertyRole_IncomingMessageFlow, but the corresponding ontology class OutputThing does not possess ontology property Flow.
- Represented class BPMN_StartEventClassRole_TargetOSF possesses represented property BPMN_StartEventTLawRole_TLaw, but the corresponding ontology class InputThing does not possess ontology property TransformationLaw.
- Represented class BPMN_StartEventClassRole_TargetOSF possesses represented property BPMN_StartEvent_OutgoingSequenceFlow, but the corresponding ontology class InputThing does not possess ontology property Flow.
- Represented class BPMN_SubProcessClassRole_Process possesses represented property BPMN_SubProcessTLaw_ProcessLaw, but the corresponding ontology class System does not possess ontology property TransformationLaw.
- Represented class BPMN_TaskClassRole_Process possesses represented property BPMN_TaskTLaw-ProcessLaw, but the corresponding ontology class System does not possess ontology property TransformationLaw.

J.3.2 Duplication of classes

- Ontology classes Component and OrganizationalUnit possess exactly the same ontology properties.
- Ontology classes Component and RoleHolder possess exactly the same ontology properties.
- Ontology classes Composite and RoleHolder possess exactly the same ontology properties.

- Ontology classes RoleHolder and Component possess exactly the same ontology properties.
- Ontology classes RoleHolder and Composite possess exactly the same ontology properties.

J.3.3 Recognizable superfluous relation

- Ontology property InformationRepositoryLaw precedes FunctionLaw, which in turn precedes TransformationLaw, so there is no need for a precedence relationship between the first and last.

J.3.4 Precedence of the properties and their relation to the class

- Ontology class ActiveThing possesses property ActivityLaw and also its precedent TransformationLaw.
- Ontology class Anything possesses property FunctionLaw and also its precedent TransformationLaw.
- Ontology class CoupledThing possesses property MutualLaw and also its precedent TransformationLaw.
- Ontology class ExecutingThing possesses property ApplicationLaw and also its precedent TransformationLaw.
- Ontology class ExecutingThing possesses property ExecutionLaw and also its precedent TransformationLaw.
- Ontology class HumanOutput possesses property ParticipationLaw and also its precedent TransformationLaw.
- Ontology class MachineResource possesses property UseLaw and also its precedent TransformationLaw.
- Ontology class OrganizationalUnit possesses property FunctionLaw and also its precedent TransformationLaw.

J.3.5 Property belongs to any class

- Construct ARIS_Event describes represented state ARIS-EventStateRole_PostState but not any represented class whose properties define ARIS-EventStateRole_PostState.
- Construct ARIS_Event describes represented state ARIS_EventStateRole_ActivatedFlow but not any represented class whose properties define ARIS_EventStateRole_ActivatedFlow.
- Construct ARIS_Event describes represented state ARIS_EventStateRole_PreState but not any represented class whose properties define ARIS_EventStateRole_PreState.
- Construct ARIS_Message describes represented state ARIS_MessageStateRole_MessageState but not any represented class whose properties define ARIS_MessageStateRole_MessageState.

J.3.6 Duplication of represented phenomenon

- Represented phenomenon DUPLICATE_1_OF__AIRS_ApplicationSoftwareClassRole_Software_AIRS_ApplicationSoftwarePropertyRole_Rule_RepresentedClassPropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__AIRS_ApplicationSoftwareClassRole_Software_ARIS_ApplicationSoftwareTLawRole_FunctionLaw_RepresentedClassPropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__AIRS_ApplicationSoftwareClassRole_Software_ARIS_ApplicationsoftwarePropertyRole_Name_RepresentedClassPropertyRelation does not describe any constructs.

- Represented phenomenon DUPLICATE_1_OF__ARIS_EnvironmentalDataClassRole _EnvironmentalData_ARIS_EnvironmentalDataPropertyRole_ChangingAttribute _RepresentedClassPropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__ARIS_EnvironmentalDataClassRole _EnvironmentalData_ARIS_EnvironmentalDataPropertyRole_InformationFlow _RepresentedClassPropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__ARIS_EnvironmentalDataClassRole _EnvironmentalData_ARIS_EnvironmentalDataPropertyRole_Name_RepresentedClassPropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__ARIS_EnvironmentalDataClassRole _EnvironmentalData_ARIS_EnvironmentalDataTLawRole_FunctionLaw_RepresentedClassPropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__ARIS_MachineResourceClassRole _Machine_ARIS_MachineResourcePropertyRole_Allocation_RepresentedClassPropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__ARIS_MachineResourceClassRole _Machine_ARIS_MachineResourcePropertyRole_Name_RepresentedClassPropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__ARIS_MachineResourceClassRole _Machine_ARIS_MachineResourceTLaw_UseLaw_RepresentedClassPropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__ARIS_MachineResourceClassRole _OrganizationalUnit_ARIS_MachineResourcePropertyRole_Allocation_RepresentedClassPropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__ARIS_GoalTLaw _FunctionLaw_ARIS_GoalPropertyRole_OutgoingFlow_RepresentedPropertySubpropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__ARIS_MessagePropertyRole _Event_ARIS_MessageSLaw_Message_RepresentedPropertySubpropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__ARIS_MessageSLaw _Message_ARIS_MessagePropertyRole_Attribute_RepresentedPropertySubpropertyRelation does not describe any constructs.
- Represented phenomenon DUPLICATE_1_OF__AIRS_ApplicationSoftwareClassRole _Software_AIRS_ApplicationSoftwarePropertyRole_Rule_RepresentedClassPropertyRelation is not mapped onto any ontology phenomena.
- Represented phenomenon DUPLICATE_1_OF__AIRS_ApplicationSoftwareClassRole _Software_ARIS_ApplicationSoftwareTLawRole_FunctionLaw_RepresentedClassPropertyRelation is not mapped onto any ontology phenomena.
- Represented phenomenon DUPLICATE_1_OF__AIRS_ApplicationSoftwareClassRole _Software_ARIS_ApplicationsoftwarePropertyRole_Name_RepresentedClassPropertyRelation is not mapped onto any ontology phenomena.
- Represented phenomenon DUPLICATE_1_OF__ARIS_EnvironmentalDataClassRole _EnvironmentalData_ARIS_EnvironmentalDataPropertyRole_ChangingAttribute _RepresentedClassPropertyRelation is not mapped onto any ontology phenomena.

- Represented phenomenon DUPLICATE_1_OF__ARIS_EnvironmentalDataClassRole _EnvironmentalData_ARIS_EnvironmentalDataPropertyRole_InformationFlow _RepresentedClassPropertyRelation is not mapped onto any ontology phenomena.
- Represented phenomenon DUPLICATE_1_OF__ARIS_EnvironmentalDataClassRole _EnvironmentalData_ARIS_EnvironmentalDataPropertyRole_Name_RepresentedClassPropertyRelation is not mapped onto any ontology phenomena.
- Represented phenomenon DUPLICATE_1_OF__ARIS_EnvironmentalDataClassRole _EnvironmentalData_ARIS_EnvironmentalDataTLawRole_FunctionLaw_RepresentedClassPropertyRelation is not mapped onto any ontology phenomena.
- Represented phenomenon DUPLICATE_1_OF__ARIS_MachineResourceClassRole _Machine_ARIS_MachineResourcePropertyRole_Allocation_RepresentedClassPropertyRelation is not mapped onto any ontology phenomena.
- Represented phenomenon DUPLICATE_1_OF__ARIS_MachineResourceClassRole _Machine_ARIS_MachineResourcePropertyRole_Name_RepresentedClassPropertyRelation is not mapped onto any ontology phenomena.
- Represented phenomenon DUPLICATE_1_OF__ARIS_MachineResourceClassRole _OrganizationalUnit_ARIS_MachineResourcePropertyRole_Allocation_RepresentedClassPropertyRelation is not mapped onto any ontology phenomena.
- Represented phenomenon DUPLICATE_1_OF__ARIS_GoalTLaw _FunctionLaw_ARIS_GoalPropertyRole_OutgoingFlow_RepresentedPropertySubpropertyRelation is not mapped onto any ontology phenomena.
- Represented phenomenon DUPLICATE_1_OF__ARIS_MessagePropertyRole _Event_ARIS_MessageSLaw_Message_RepresentedPropertySubpropertyRelation is not mapped onto any ontology phenomena.
- Represented phenomenon DUPLICATE_1_OF__ARIS_MessageSLaw_Message _ARIS_MessagePropertyRole_Attribute_RepresentedPropertySubpropertyRelation is not mapped onto any ontology phenomena.

Appendix K

Languages Comparison Tables

Here is grouped the comparisons between ARIS constructs and BPMN constructs by means of tables.

K.1 ARIS.Position - BPMN.Lane

Table K.1: Comparison between Organizational unit and Pool

ARIS	**BPMN**	**Common ontology**
OrganizationalUnit	Pool	*Participant*
Position		*RoleHolder*
	Lanes	*ActiveThing*
RelationToWhole	ParentPool	*PartWholeRelation*
Name	Name	*Name*
NameOrganizationalUnit	-	*Name*
-	NamePool	*RegularStringProperty*
Responsability	-	*Responsability*
-	ParentLane	*PartWholeRelation*

K.2 ARIS.Function - BPMN.Task

Table K.2: Comparison between Function and Task

ARIS	**BPMN**	**Common Ontology**
Organizational Unit	Participant	*Participant*
IsActive	IsActive	*IsActive*
FunctionningState	ActiveState	*ActiveState*
NonFunctionningState	InactiveState	*InactiveState*
TriggeringFunction	TriggeringTask	*Triggering*
NotAllInputAvailable	NotAllTokenAvailable	*AnyTransformation*
TerminationFunction	TerminationTask	*Termination*
IncomingFlow	IncomingMessageFlow	*Flow*
	IncomingSequenceFlow	*Flow*
OutgoingFlow	OutgoingMessageFlow	*Flow*
	OutgoingSequenceFlow	*Flow*
OutputFlow	OutgoingSequenceFlowContent	*FlowContent*
	OutgoingMessageFlowContent	*FlowContent*

Continued on next page

ARIS	BPMN	Common Ontology
OutputFlowInput	IncomingSequenceFlowContent	*FlowContent*
	IncomingMessageFlowContent	*FlowContent*
Function Law	-	*FunctionLaw*
NotAllOutputAvailable	-	*AnyTransformation*
UseLaw	-	*UseLaw*
ComputerHardware	-	*ComputerHardware*
Machine	-	*MachineResource*
Goal	-	*Law*
InformationFlow	-	*InteractionRelation*
EnvironmentalData	-	*ReactiveThing*
ApplicationLaw	-	*ApplicationLaw*
Software	-	*ExecutingThing*
Participation	-	*ParticipationLaw*
HumanOutput	-	*HumanOutput*
SourceOutput	-	*Repository & OutputThing*
TargetOutput	-	*Repository & InputThing*
-	Task	*ActivityLaw*
-	Process	*System*
-	ProcessLaw	*TransformationLaw*
-	TaskType	*RegularStringProperty*
-	Token	*RegularMutableProperty*

K.3 ARIS.Function - BPMN.SubProcess

Table K.3: Comparison between Function and Sub-Process

ARIS	BPMN	Common Ontology
Organizational Unit	Participant	*Participant*
IsActive	IsActive	*IsActive*
FunctionningState	ActiveState	*ActiveState*
NonFunctionningState	InactiveState	*InactiveState*
TriggeringFunction	TriggeringSubProcess	*Triggering*
NotAllInputAvailable	NotAllTokenAvailable	*AnyTransformation*
TerminationFunction	TerminationSubProcess	*Termination*
IncomingFlow	IncomingMessageFlow	*Flow*
	IncomingSequenceFlow	*Flow*
OutgoingFlow	OutgoingMessageFlow	*Flow*
	OutgoingSequenceFlow	*Flow*
OutputFlow	OutgoingSequenceFlowContent	*FlowContent*
	OutgoingMessageFlowContent	*FlowContent*
OutputFlowInput	IncomingSequenceFlowContent	*FlowContent*
	IncomingMessageFlowContent	*FlowContent*
Function Law	-	*FunctionLaw*
NotAllOutputAvailable	-	*AnyTransformation*
UseLaw	-	*UseLaw*
ComputerHardware	-	*ComputerHardware*
Machine	-	*MachineResource*
Goal	-	*Law*
InformationFlow	-	*InteractionRelation*
EnvironmentalData	-	*ReactiveThing*

Continued on next page

ARIS	BPMN	Common Ontology
ApplicationLaw	-	*ApplicationLaw*
Software	-	*ExecutingThing*
Participation	-	*ParticipationLaw*
HumanOutput	-	*HumanOutput*
SourceOutput	-	*Repository & OutputThing*
TargetOutput	-	*Repository & InputThing*
-	SubProcess	*ActivityLaw*
-	Process	*System*
-	ProcessLaw	*TransformationLaw*
-	SubProcessType	*RegularStringProperty*
-	Token	*RegularMutableProperty*

K.4 ARIS.And - BPMN.Gateway

Table K.4: Comparison between the logical operator "And" and Gateway

ARIS	BPMN	Common ontology
InputAnd, OutputAnd	InputGateway, OutputGateway	*CoupledThing*
And	Gateway	*MutualLaw*
EndingInput, EndingOutput	OutputCoupling ,InputCoupling	*CouplingRelation*
-	IncomingSequenceFlow, OutgoingSequenceFlow	*Flow*
-	GatewayType, Name	*RegularStringProperty*
-	Lane, Pool, Assignements	*RegularProperty*

K.5 ARIS.And - BPMN.ParallelGateway

Table K.5: Comparison between the logical operator "And" and Parallel Gateway

ARIS	BPMN	Common ontology
InputAnd, OutputAnd	InputParallelGateway, OutputParallelGateway	*CoupledThing*
And	ParallelGateway	*MutualLaw*
EndingInput, EndingOutput	ParallelOutputCoupling, ParallelInputCoupling	*CouplingRelation*
-	Gate	*RegularProperty*
-	IncomingSequenceFlow, OutgoingSequenceFlow	*Flow*

K.6 ARIS.Or - BPMN.Gateway

Table K.6: Comparison between the logical operator "Or" and Gateway

ARIS	**BPMN**	**Common ontology**
InputOr, OutputOr	InputGateway, OutputGateway	*CoupledThing*
Or	Gateway	*MutualLaw*
EndingInput, EndingOutput	OutputCoupling ,InputCoupling	*CouplingRelation*
-	IncomingSequenceFlow, OutgoingSequenceFlow	*Flow*
-	GatewayType, Name	*RegularStringProperty*
-	Lane, Pool, Assignements	*RegularProperty*

K.7 ARIS.Or - BPMN.InclusiveGateway

Table K.7: Comparison between the logical operator "Or" and Inclusive Gateway

ARIS	**BPMN**	**Common Ontology**
InputOr	InputInclusiveGateway	*CoupledThing*
OutputOr	OutputInclusiveGateway	*CoupledThing*
Or	InclusiveGateway	*MutualLaw*
EndingInput	InclusiveInputCoupling	*CouplingRelation*
EndingOutput	InclusiveOutputCoupling	*CouplingRelation*
-	IncomingSequenceFlow	*Flow*
-	OutgoingSequenceFlow	*Flow*
-	Gates	*RegularProperty*
-	DefaultGate	*RegularProperty*

K.8 ARIS.XOR - BPMN.Gateway

Table K.8: Comparison between the logical operator "XOR" and Gateway

ARIS	**BPMN**	**Common ontology**
InputXor, OutputXor	InputGateway, OutputGateway	*CoupledThing*
Xor	Gateway	*MutualLaw*
EndingInput, EndingOutput	OutputCoupling ,InputCoupling	*CouplingRelation*
-	IncomingSequenceFlow, OutgoingSequenceFlow	*Flow*
-	GatewayType, Name	*RegularStringProperty*

Continued on next page

ARIS	BPMN	Common ontology
-	Lane, Pool, Assignements	*RegularProperty*

K.9 ARIS.XOR - BPMN.ExclusiveGateway

Table K.9: Comparison between the logical operator "XOR" and Exclusive Gateway

ARIS	BPMN	Common ontology
InputXor, OutputXor	InputExclusiveGateway, OutputExclusiveGateway	*CoupledThing*
Xor	ExclusiveGateway	*MutualLaw*
EndingInput, EndingOutput	OutputCoupling ,InputCoupling	*CouplingRelation*
-	IncomingSequenceFlow, OutgoingSequenceFlow	*Flow*

K.10 ARIS.XOR - BPMN.EventBasedExclusiveGateway

Table K.10: Comparison between the logical operator "XOR" and Event-Based Exclusive Gateway

ARIS	BPMN	Common ontology
InputXor, OutputXor	InputGateway, OutputGateway	*CoupledThing*
Xor	EventBasedExclusiveGateway	*MutualLaw*
EndingInput, EndingOutput	EventBasedExclusiveOutputCoupling, EventBasedExclusiveInputCoupling	*CouplingRelation*
-	IncomingSequenceFlow, OutgoingSequenceFlow	*Flow*
-	Gates	*RegularProperty*
-	XORType	*RegularStringProperty*
-	InstantiateFalse	*RegularBooleanProperty*

K.11 ARIS.XOR - BPMN.DataBasedExclusiveGateway

Table K.11: Comparison between the logical operator "XOR" and Data-Based Exclusive Gateway

ARIS	BPMN	Common ontology
InputXor, OutputXor	InputGateway, OutputGateway	*CoupledThing*
Xor	DataBasedExclusiveGateway	*MutualLaw*

Continued on next page

ARIS	BPMN	Common ontology
EndingInput, EndingOutput	DataBasedExclusiveOutput-Coupling, DataBasedExclusiveInputCoupling	*CouplingRelation*
-	IncomingSequenceFlow, OutgoingSequenceFlow	*Flow*
-	Gates, DefaultGate	*RegularProperty*
-	XORType	*RegularStringProperty*
-	MarkerVisible	*RegularBooleanProperty*

K.12 ARIS.ControlFlow - BPMN.SequenceFlow

Table K.12: Comparison between Control Flow and Sequence Flow

ARIS	BPMN	Common ontology
Source	Source	*OutputThing*
Target	Target	*InputThnig*
ControlFlow	SequenceFlow	*Flow*
EndingInput, EndingOutput	EventBasedExclusiveOutput-Coupling, EventBasedExclusiveInputCoupling	*CouplingRelation*
-	Name, ConditionType	*RegularStringProperty*
-	Quantity	*RegularNaturalProperty*

K.13 ARIS.Event - BPMN.Event

Table K.13: Comparison between ARIS.Event and BPMN.Event

ARIS	BPMN	Common Ontology
Flow	SequenceFlow	*Flow*
	MessageFlow	*Flow*
Event	EventParameters	*FlowContent*
NBFlowContent	Token	*RegularMutableProperty*
PreState	PreState	*AnyState*
PostState	PostState	*AnyState*
ActivatedFlow	ActivatedFlow	*AnyState*
InputEvent	InputEvent	*AnyTransformation*
OutputEvent	OutputEvent	*AnyTransformation*
Organizational Unit	-	*Participant*
Function Law	-	*FunctionLaw*
-	TargetSF	*InputThing*
-	SourceSF	*OutputThing*
-	TargetMF	*InputThing*
-	SourceMF	*OutputThing*
-	TLawTargetMF	*TransformationLaw*
-	TLawTargetSF	*TransformationLaw*

Continued on next page

ARIS	BPMN	Common Ontology
-	Type	*RegularStringProperty*
-	Name	*RegularStringProperty*
-	Pool	*RegularProperty*
-	Lane	*RegularProperty*
-	Assignements	*RegularProperty*

K.14 ARIS.Event - BPMN.StartEvent

Table K.14: Comparison between ARIS.Event and BPMN.StartEvent

ARIS	BPMN	Common Ontology
Flow	OutgoingSequenceFlow	*Flow*
	IncomingMessageFlow	*Flow*
Event	StartEventParameters	*FlowContent*
NBFlowContent	Token	*RegularMutableProperty*
PreState	PreState	*AnyState*
PostState	PostState	*AnyState*
ActivatedFlow	ActivatedFlow	*AnyState*
InputEvent	InputEvent	*AnyTransformation*
OutputEvent	OutputEvent	*AnyTransformation*
Organizational Unit	-	*Participant*
Function Law	-	*FunctionLaw*
-	TargetOSF	*InputThing*
-	SourceIMF	*OutputThing*
-	TLaw	*TransformationLaw*
-	Trigger	*RegularStringProperty*

K.15 ARIS.Event - BPMN.IntermediateEvent

Table K.15: Comparison between ARIS.Event and BPMN.IntermediateEvent

ARIS	BPMN	Common Ontology
Flow	OutgoingSequenceFlow	*Flow*
	IncomingSequenceFlow	*Flow*
	IncomingMessageFlow	*Flow*
Event	IntermediateEventParameters	*FlowContent*
NBFlowContent	Token	*RegularMutableProperty*
PreState	PreState	*AnyState*
PostState	PostState	*AnyState*
ActivatedFlow	ActivatedFlow	*AnyState*
InputEvent	InputEvent	*AnyTransformation*
OutputEvent	OutputEvent	*AnyTransformation*
Organizational Unit	-	*Participant*
Function Law	-	*FunctionLaw*

Continued on next page

ARIS	BPMN	Common Ontology
-	TargetOSF	*InputThing*
-	SourceIMF	*OutputThing*
-	SourceISF	*OutputThing*
-	TLaw	*TransformationLaw*
-	Trigger	*RegularStringProperty*
-	Target	*RegularProperty*

K.16 ARIS.Event - BPMN.EndEvent

Table K.16: Comparison between ARIS.Event and BPMN.EndEvent

ARIS	BPMN	Common Ontology
Flow	IncomingSequenceFlow	*Flow*
	OutgoingMessageFlow	*Flow*
Event	IntermediateEventParameters	*FlowContent*
NBFlowContent	Token	*RegularMutableProperty*
PreState	PreState	*AnyState*
PostState	PostState	*AnyState*
ActivatedFlow	ActivatedFlow	*AnyState*
InputEvent	InputEvent	*AnyTransformation*
OutputEvent	OutputEvent	*AnyTransformation*
Organizational Unit	-	*Participant*
Function Law	-	*FunctionLaw*
-	TargetOMF	*InputThing*
-	SourceISF	*OutputThing*
-	TLaw	*TransformationLaw*
-	Result	*RegularStringProperty*