

THESIS / THÈSE

MASTER IN COMPUTER SCIENCE

Improving the cognitive effectiveness of the KAOS requirements modelling language

Dupriez, Muriel

Award date:
2011

Awarding institution:
University of Namur

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Facultés Universitaires Notre-Dame de la Paix, Namur

Faculté d'informatique.

Année académique 2010-2011

Improving the Cognitive Effectiveness
of
the KAOS Requirements Modelling Language.

Muriel Dupriez

Mémoire présenté en vue de l'obtention du grade de master en Sciences Informatiques.

"Computers are magnificent tools for the realisation of our dreams, but no machine can replace the human spark of spirit, compassion, love, and understanding."

[Louis Gerstner](#)

Abstract

In this thesis we present the requirements engineering discipline and in particular goal-oriented modelling languages. They are mostly used to help the stakeholders to express their needs. During the requirements engineering process, these needs are transformed into goals. Then these goals will be themselves transformed into requirements that will be fulfilled by the future system. To facilitate the communication between users and developers, system modellers often use diagrams to graphically represent goals and requirements. They use this technique because it is commonly accepted that graphical representations are easier to understand than formal sentences. However, it is not as trivial as it looks. To be effective, diagrams have to be drawn following specific rules described in a graphical language evaluation theory called the Physics of Notation. If not, the risk is that they will become so complex that they fail to reach their aim.

We analyse in particular one goal modelling approach called KAOS. Then, we apply the principles for an effective communication which is measured by the speed, ease and accuracy with which the information content is understood. Following these principles the KAOS visual notation is evaluated, and finally we give some recommendations to improve it. These recommendations will be of 3 types: for novice users, for users that draw diagrams during a meeting and for software developers who are building a tool that implements KAOS.

Our recommendations are validated in a running example that consists of an online bookshop store. Our work leads to improvements to the KAOS visual notation to create more effective goal models.

Résumé

Dans ce mémoire, nous présentons la discipline de l'analyse des exigences et en particulier les langages orientés buts. Ils sont principalement utilisés pour aider les parties prenantes d'un nouveau système à exprimer leurs besoins. Durant le processus de l'analyse des exigences, les besoins sont transformés en buts à atteindre. Ces besoins seront eux-mêmes transformés en exigences qui devront être remplies par le futur système. Pour faciliter la communication entre les utilisateurs et les développeurs, les architectes du système utilisent le plus souvent des diagrammes pour représenter graphiquement les buts et les exigences. Ils utilisent cette technique car il est communément accepté que les représentations graphiques sont plus faciles à comprendre que du texte formel. Cependant, ce n'est pas aussi trivial que cela en a l'air. Pour être efficaces, les diagrammes doivent être dessinés suivant des règles spécifiques décrites dans une théorie d'évaluation des langages graphiques appelée la « Physics of Notations ». Si ce n'est pas le cas, le risque est qu'ils deviennent si complexes qu'ils n'atteignent pas leurs buts.

Nous analysons en particulier un langage de modélisation des buts appelé : KAOS. Ensuite, nous appliquons les principes d'une communication efficace qui peut être mesurée grâce à la vitesse, la facilité et l'efficacité à laquelle l'information est comprise. Suivant ces principes, la notation visuelle de KAOS est évaluée, et finalement nous formulerons des recommandations pour l'améliorer. Ces recommandations sont de 3 types: pour les utilisateurs débutants, pour les utilisateurs qui dessinent des diagrammes en réunion et pour les développeurs de logiciel qui implémentent KAOS.

Nos recommandations sont validées grâce à un exemple réel basé sur une librairie en ligne. Notre travail mène à des améliorations de la notation visuelle de KAOS pour créer des modèles de buts plus efficaces.

Acknowledgement

I would like to acknowledge the following persons for their support during the elaboration of this master thesis.

Professor P. Heymans, the supervisor of this work and his assistant N. Genon. They make me discover the discipline of the requirements engineering further. They also gave me the necessary support to fix the objectives of my research by providing me with the required documentation, by guiding the redaction of this text and by following the advancement of my project. Both showed me the importance of using accurate and precise terms to redact clearly this document and avoid any confusion.

The head of my service, Paul Marechal who encouraged me during the time of my studies and especially during the redaction of this thesis.

My colleagues, Chris Retsin, who gave me advices to improve this work, verified the coherence and the typography.

And finally my family and in particular my husband who supports me in everyday life and encourages me in the more difficult moments.

Table of Contents

Abstract	v
Résumé	vii
Acknowledgement.....	ix
Table of Contents	xi
List of Figures	xvii
List of Tables.....	xxi
Chapter 1 Introduction	1
1.1 Structure	2
1.2 Terminology.....	3
1.3 Text format.....	5
Part I Background.....	7
Chapter 2 Requirements Engineering	9
2.1 Introduction to requirements engineering	9
2.2 The WHY, the WHAT and the WHO dimensions.....	10
2.3 Categories of statements.....	10
2.4 Categories of requirements.....	11
2.5 Requirements lifecycle: processes, actors and products	12
2.6 Target qualities and defects to avoid.....	14
Chapter 3 Goal-oriented languages.....	17
3.1 What are goals?.....	17
3.2 Where are the goals coming from?	18
3.3 The granularity of goals and their relationship with requirements and assumptions	18

3.4	Goal types and categories.....	20
3.5	The central role of goals in the requirements engineering process	21
3.6	The choice of KAOS	23
Chapter 4 Principles of the Physics of Notations Theory		25
4.1	Introduction to graphical notations	25
4.2	Principles for Designing Effective Visual Notations	27
4.2.1	Principle of Semiotic Clarity	28
4.2.2	Principle of Perceptual Discriminability.....	29
4.2.3	Principle of Semantic Transparency	31
4.2.4	Principle of Manageable Complexity	32
4.2.5	Principle of Cognitive Integration	33
4.2.6	Principle of Visual Expressiveness.....	34
4.2.7	Principle of Dual Coding	35
4.2.8	Principle of Graphic Economy.....	36
4.2.9	Principle of Cognitive Fit.....	37
4.3	Interaction among Principles.....	38
Chapter 5 The KAOS Language		39
5.1	Introduction	39
5.2	Goal Model.....	39
5.3	Agent Model.....	45
5.4	Operation Model	48
5.5	Object Model.....	50
5.6	Behaviour Model.....	52
Part II Contribution		53
Chapter 6 The KAOS meta-model.....		55
6.1	Goal Model.....	55
6.2	Agent Model.....	57

6.3	Operation Model	57
6.4	Object Model.....	58
6.5	Behaviour Model.....	59
Chapter 7 Applying the Physics of Notations to KAOS.....		61
7.1	Principle of Semiotic Clarity.....	61
7.1.1	Analysis results	61
7.1.2	Recommendations to improve the semiotic clarity.....	63
7.2	Principle of Cognitive Fit.....	66
7.2.1	Analysis results	66
7.2.2	Recommendations to improve cognitive fit.....	66
7.3	Principle of Perceptual Discriminability.....	67
7.3.1	Analysis results	67
7.3.2	Recommendations to improve the perceptual discriminability	70
7.4	Principle of Semantic Transparency	71
7.4.1	Analysis results	71
7.4.2	Recommendations to improve the semantic transparency.....	71
7.5	Principle of Visual Expressiveness	73
7.5.1	Analysis results	73
7.5.2	Recommendations to improve the visual expressiveness	74
7.6	Principle of Dual Coding.....	75
7.6.1	Analysis results	75
7.6.2	Recommendations to improve dual coding.....	76
7.7	Principle of Graphic Economy.....	78
7.7.1	Analysis results	78
7.7.2	Recommendations to improve graphic economy.....	79
7.8	Principle of Manageable Complexity.....	80
7.8.1	Analysis results	80
7.8.2	Recommendations to improve the manageable complexity	81

7.9	Principle of Cognitive Integration.....	82
7.9.1	Analysis results	82
7.9.2	Recommendations to improve the cognitive integration	83
7.10	Summary.....	85
Chapter 8 Recommendations		89
8.1	General recommendations.....	90
8.2	Recommendations for language engineers to improve visual notation for novices 91	
8.3	Recommendations for meeting users	96
8.4	Recommendations for software developers	98
Part III Illustration.....		105
Chapter 9 An illustrative example		107
9.1	Context Description.....	107
9.1.1	Online discount bookstore	107
9.1.2	What are the different activities?	107
9.1.3	Who are the different stakeholders?	108
9.1.4	How does an order happen?.....	110
9.2	KAOS Analysis.....	110
9.2.1	Goal model.....	111
9.2.2	Agent and operation model.....	112
9.2.3	Object model.....	113
9.2.4	Behaviour model.....	114
9.3	Modified versions of diagrams.....	115
9.4	First Evaluation of Recommendations	120
9.5	Limitations	120
Part IV Conclusion		121
Chapter 10 Conclusion.....		123
10.1	Conclusion.....	123

10.2	Limitations.....	124
10.2.1	Self-criticism about visual notations.....	124
10.2.2	Self-criticism about the author.....	124
10.2.3	Self-criticism about the work.....	124
10.3	Future Works	125
	Glossary.....	127
	Bibliography.....	129
	Annex 1: Analysis of the meta-model concepts.....	135
	Annex 2: The modules of the running example.....	141

List of Figures

Figure 1-1 Thesis structure	3
Figure 1-2 The main definitions used in this document and the relationships that exist between them [Moody, et al., 2010]	3
Figure 1-3 The meta-model of KAOS, an example of model and the associated diagram ...	5
Figure 2-1 Spiral model for requirements engineering process [Lamsweerde, 2009].....	12
Figure 3-1 Goal statements hierarchical classification [Lamsweerde, 2009]	19
Figure 3-2 Goal categories hierarchical classification [Lamsweerde, 2009].....	20
Figure 4-1 The theory of diagrammatic communication [Moody, 2009]	25
Figure 4-2 The visual alphabet [Moody, 2009]	26
Figure 4-3 The human graphical information processing [Moody, 2009].....	26
Figure 4-4 The 9 principles of the Physics of Notations theory [Moody, 2009]	27
Figure 4-5 The anomalies of the semiotic clarity [Moody, 2009]	28
Figure 4-6 Poor visual distance between Actor, Agent and Role in i* [Moody, et al., 2010]	29
Figure 4-7 Improvement proposition of i* to distinguish actor, agent and role	29
Figure 4-8 Redundant coding: add colour to shapes to increase the visual distance.....	30
Figure 4-9 Textual differentiation in UML class diagram [Moody, 2009].....	30
Figure 4-10 The degrees of semantic transparency [Moody, et al., 2010]	31
Figure 4-11 The only icon which is frequently used is this one which represents the user.	31
Figure 4-12 Hierarchical Structuring in DFDs [Moody]	33
Figure 4-13 Duplicate elements are used to describe the meta-model on KAOS in [Lamsweerde, 2009]	33
Figure 4-14 Conceptual information (part A) and perceptual integration (part B).....	34
Figure 4-15 Visual expressiveness: differences between primary and secondary notations [Moody, 2009]	35

Figure 4-16 Example of dual coding using textual information [Moody, 2009].....	36
Figure 4-17 Cognitive fit is the result of a three-way interaction between the representation, task and problem solver [Moody, 2009]	37
Figure 4-18 Interactions between principles [Moody, 2009].....	38
Figure 5-1 A goal and its features.....	40
Figure 5-2 AND-refinement and complete AND-refinement.....	41
Figure 5-3 OR-refinement.....	42
Figure 5-4 Conflict among goals	42
Figure 5-5 An obstacle and its features.....	43
Figure 5-6 An obstacle and one of its possible obstruction	44
Figure 5-7 Obstacle analysis and goal model elaboration [Lamsweerde, 2009]	45
Figure 5-8 Environment agent and agent.....	46
Figure 5-9 Agent responsibility and performance	46
Figure 5-10 Agent capabilities [Lamsweerde, 2009].....	47
Figure 5-11 Agent dependencies [Lamsweerde, 2009]	47
Figure 5-12 Operation signature	48
Figure 5-13 Domain pre-conditions of an operation.....	49
Figure 5-14 Required conditions annotating operationalisations	49
Figure 5-15 An object and its features.....	51
Figure 6-1 Meta-model of KAOS	60
Figure 7-1 The KAOS visual vocabulary	62
Figure 7-2 Suggestion of new symbol for expectation	64
Figure 7-3 Suggestion to improve the differentiation between AND-refinement and OR-refinement	65
Figure 7-4 A Suggestion of (complete) OR-refinement relationship link	66
Figure 7-5 Use of shapes and textures in the KAOS notation	68
Figure 7-6 Elements that do not visually pop out.....	68
Figure 7-7 AND-refinement and OR-refinement are not easily discriminable	69
Figure 7-8 Suggestion of a new shape to represent obstacles.....	70

Figure 7-9 Different icons to represent the different concepts of KAOS.....	73
Figure 7-10 Simplified symbols to represent the different KAOS concepts	73
Figure 7-11 3D shapes for goal and domain hypothesis to increase the visual expressiveness.....	74
Figure 7-12 Add colour to shapes to increase the visual expressiveness	75
Figure 7-13 The intensity of the colour suggests the priority of the goal.....	75
Figure 7-14 A goal and its annotation.....	76
Figure 7-15 Goal, achieve goal and maintain goal can only be textually differentiated	76
Figure 7-16 Annotations do not need their own visual construct	76
Figure 7-17 Dual coding: each link is labelled	78
Figure 7-18 Using symbol deficit produces a smaller KAOS visual alphabet	79
Figure 7-19 Add colour to increase visual effectiveness in the aim of improving the graphic economy	80
Figure 7-20 Goal model of the running example, suggestion of modularisation	82
Figure 7-21 Conceptual integration - models used in the system.....	83
Figure 7-22 Cognitive integration for goal model of the running example.....	84
Figure 7-23 Perceptual integration mechanisms added to the sub-diagram 3.1	85
Figure 8-1 Recommendation to use modularisation	92
Figure 8-2 Recommendation to improve cognitive integration at system level	93
Figure 8-3 Improve semantic transparency by adding symbols inside the shapes	94
Figure 8-4 Combine colour and symbol to improve semantic transparency	94
Figure 8-5 Recommendation for meeting users - Increase Semantic transparency by using colours.....	96
Figure 8-6 Recommendation for meeting users - Increase Semantic transparency by using + and - signs.....	96
Figure 8-7 Recommendation for meeting users - Increase Semantic transparency by adding 1 or many times the sign that represents goals	97
Figure 8-8 Recommendation to use modularisation	99
Figure 8-9 Screenshot of the software that will implement KAOS	100
Figure 8-10 The software offers a view with symbols instead of the abstract figure.....	101

Figure 8-11 The software offers a view with 3D shapes	102
Figure 9-1 The stakeholders of the running example	109
Figure 9-2 The order process	110
Figure 9-3 Goal model of the system-to-be of the online store 'Oh my book'.....	111
Figure 9-4 Obstacle model of the system-to-be of the online store 'Oh my book'	112
Figure 9-5 Agent and Operation models of the system-to-be of the online store 'Oh my book'	113
Figure 9-6 Object model of the system-to-be of the online store 'Oh my book'	113
Figure 9-7 Sequence diagram of the order process.....	114
Figure 9-8 State machine diagram of an order.....	114
Figure 9-9 Integration map of the online bookstore system	115
Figure 9-10 Integration map at model level.....	116
Figure 9-11 Module 1.1 of the goal model of the running example	117
Figure 9-12 Module 2.1 of the goal model of the running example, it contains goals with different priority.....	118
Figure 9-13 Module 3.2 of the goal model of the running example, it contains goals with different priority.....	119
Figure A2-1 Module 1.1	141
Figure A2-2 Module 2.1	141
Figure A2-3 Module 3.1	142
Figure A2-4 Module 4.1	142
Figure A2-5 module 5.1	143
Figure A2-6 Module 2.2	144
Figure A2-7 Module 3.2	144

List of Tables

Table 3-1 Translation of terms used in requirements engineering languages into terms used in goal-oriented languages	19
Table 3-2 The distribution of the main roles in goal-oriented requirements engineering language	23
Table 5-1 Obstacle types and the obstructed goal types [Lamsweerde, 2009]	44
Table 6-1 Colours used to differentiate the different models in the meta-model	55
Table 6-2 Attributes of the goal meta-class and its subclasses	56
Table 6-3 Attributes of the refinement meta-class	56
Table 6-4 Attributes of the agent meta-class	57
Table 6-5 Attributes of the operation meta-class	58
Table 6-6 Attributes of the object meta-class	58
Table 6-7 Attributes of the domain description meta-class	59
Table 7-1 Semiotic clarity analysis of the KAOS visual notation	62
Table 7-2 Symbols overload analysis of relationships	64
Table 7-3 Recommendations grouped by principle of the Physics of Notation	86
Table 8-1 Summary of the general recommendation	90
Table 8-2 Summary of the recommendations for language engineers	95
Table 8-3 Summary of the recommendations for meeting users	97
Table 8-4 Summary of the recommendations for software developers	103
Table 9-1 Goal standardised names	115
Table A1-1 List of semantic constructs and their representation in the goal model	136
Table A1-2 Details of the semiotic equation	139

Chapter 1 Introduction

The success of software depends on degree of satisfaction of its users. Indeed, software is generally built to achieve certain needs expressed by users. And if the developed solution does not help to achieve the users' jobs, it will not be used. Unfortunately, misunderstandings between software designers and software users are very frequent. Consequently, developers build software that does not match the users' requirements.

The goal of the requirements engineering discipline is to reduce the number of inappropriate software. Its aim is to have a complete understanding of the stakeholders' needs and to address them at the software development stage. The requirements engineering process is composed of different steps such as identifying the stakeholders, their needs, their obligations, etc. This information is collected from different stakeholders who are, somehow or other, concerned with the future system. The greatest difficulty is to help the stakeholders to express their needs and their expectations about the new system (system-to-be). Usually, they are not familiar with requirements engineering. They are specialists in their domain and they know how to perform their work but they have difficulties to explain it in a more formal way.

To collect this information we have to interview the stakeholders. Requirements engineering techniques/frameworks can guide the software designers. One of the most well-known frameworks is UML [OMG_UML, 2011] that includes use case diagrams. Use case diagrams are used to describe the sequence of operations that the user will have to perform with the new software. Goal modelling is another technique whose objective is to understand the needs of the stakeholders, to translate them into goals and finally to model them in a graphical way. One of the advantages of this technique is to facilitate the communication between stakeholders and system modellers because graphical representations are generally considered easier to understand than textual descriptions. This is probably related to the idea that graphical representations are easier to understand than formal sentences and that they represent only essential information [Petre, 1995].

There are many goal-oriented languages such as i^* [Yu, 1997], NFR [Chung, et al., 2000], Tropos [Bresciani, et al., 2004] and KAOS [Lamsweerde, 2009]. Obviously, each of these languages has its own syntax and its own semantic. However, all of them will help the system modellers to follow the different steps of the requirements engineering process. They also share a common goal: highlight the user requirements in order to develop an appropriate software system.

In this thesis, the focus is on the visual representation of goal diagrams and their understanding by stakeholders of the system-to-be. Consequently, it is important that these stakeholders correctly understand diagrams that represent their needs and the goals of the system. The fact that communication is easier with diagram is largely based on intuitions and slogans like "a picture worth a thousand words" [Moody, 2006]. But, in reality, diagram quality depends largely on the skills of the designer. However, Moody explains in [Moody, 2009] that, in practice diagrams may act "as a barrier rather than an aid to user-developer communication". To help remove this potential barrier, Moody has formulated 9 principles that form the 'Physics of Notations theory'. This theory explains how to make diagrams communicate effectively. The 9 principles were elaborated on a wide range of disciplines like cartography, cognitive integration, communication theory and also different aspects of psychology.

This thesis focuses on the diagram design with a specific goal-oriented modelling language called "KAOS". KAOS is a language introduced in the 1990's by Dardenne and van Lamsweerde in [Dardenne, et al., 1993]. It is provided with an abstract and concrete syntax and semantics.

Section 1.1 provides a detailed view of the structure of the document, as well as the relationships between the different chapters. Section 1.2 describes the terminology we use in this work.

1.1 Structure

This work is divided into 4 parts.

Part I introduces the background of the work. It consists of Chapter 2, that explains what is requirements engineering, and Chapter 3, that presents what are the existing goal-oriented languages are and why we have selected KAOS. In Chapter 4, we explain the different semantic constructs of KAOS and finally Chapter 5 introduces the Physics of Notations theory [Moody, 2009].

Part II focuses on our contribution. In Chapter 6, we present the KAOS meta-model that is used to apply one of the principles of the Physics of Notations: the semiotic clarity. It consists in verifying the correspondence between the visual notation of a language and its semantic constructs. Chapter 7 reports on the analysis of KAOS against the Physics of Notations. Finally Chapter 8 suggests recommendations for KAOS modellers, for using KAOS during a meeting and for software developers who would like to implement the visual notation of KAOS in a CASE tool.

Part III illustrates of the recommendations suggested in Part II. Chapter 9 introduces a running example based on an online bookshop store. This chapter is divided into 2 parts: the first one describes the example and its models with the actual visual representation of KAOS. In the second part, we represent the same goal model in taking into account the recommendations for the language engineers (see Chapter 8) that should improve the understanding of novices.

Part IV finally concludes our work and open perspectives of future work.

The complete structure and the flows between the different chapters are summarised in Figure 1-1.

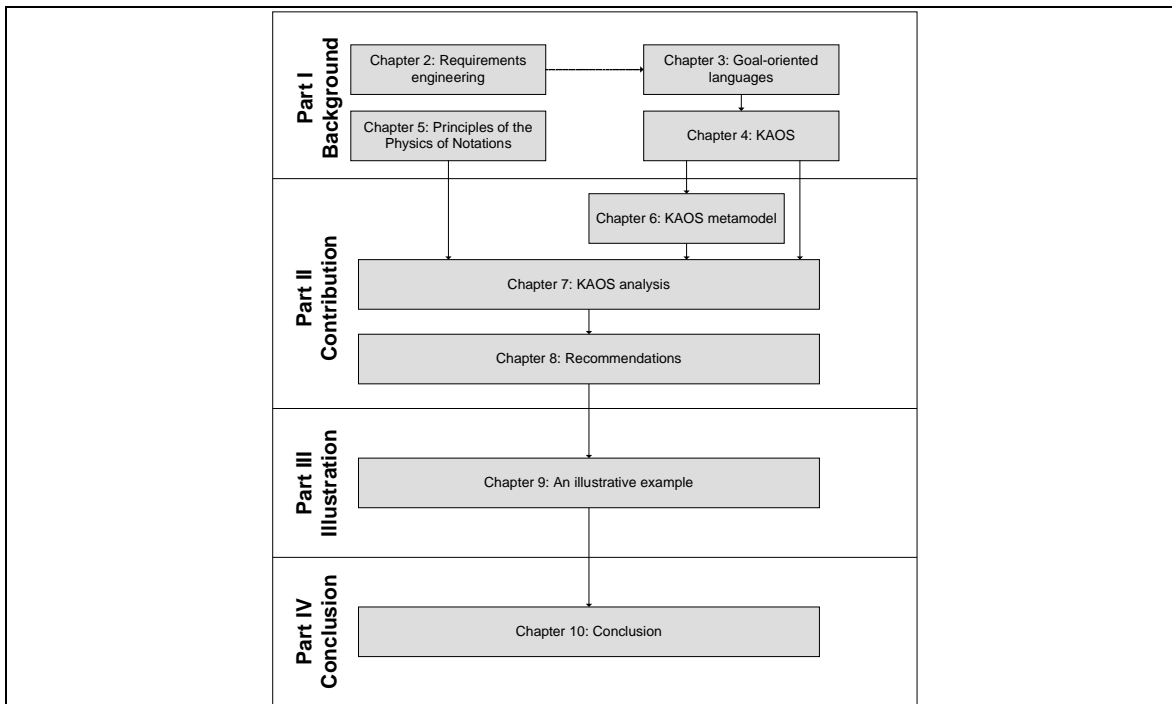


Figure 1-1 Thesis structure

1.2 Terminology

Most of the modelling terms used in the following document are usually shared with various disciplines or domains, but with distinct meanings. In order to avoid misunderstanding, we provide in this section the definitions for these terms. The definitions are based on documents from different authors and domains.

We use figure 1-2 as support to define the modelling terms and the links that exist between them. We start with the definition of **model**, then we explain its links with the concepts of **meta-model** and the **diagram**. Finally we define the concept of **visual representation**.

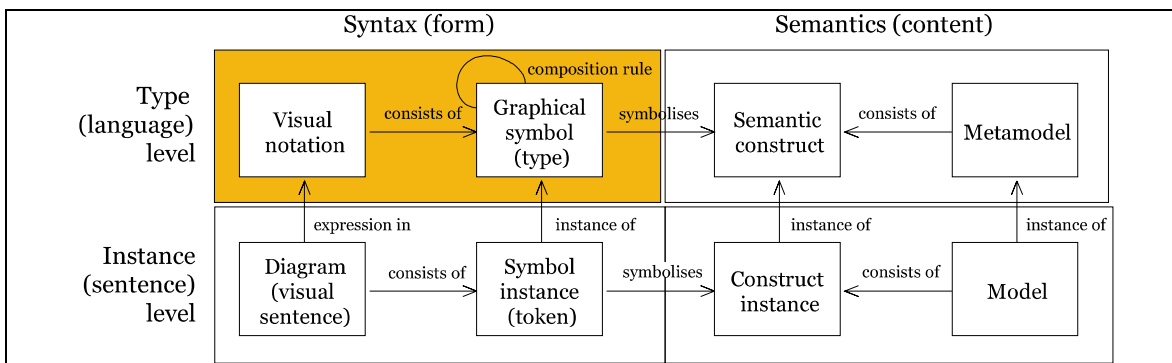


Figure 1-2 The main definitions used in this document and the relationships that exist between them [Moody, et al., 2010]

Figure 1-3 illustrates, through a concrete example, the differences between these terms. This example consists of an agent that has to fill the goal 'PreparePackageQuickly' and to perform an operation 'PreparePackage'. To fulfil this operation, he needs objects (package and book) that are used as input for the operation.

Model

Model is a term frequently used in various domains. Below, we gather the definitions that are meaningful in the requirements engineering domain:

- A model is an abstraction of a physical system, with a certain purpose (in the context of UML standard) [Selic, 2004]
- A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system [Bezivin, et al., 2001]
- A model is a set of statements about some system under study [Seidewitz, 2003]

We sum up these definitions in the following terms: "a model is an abstract form of a system. It is independent of the representation that is used to express it (i.e., a same model could be represented according to different visual notations)".

The model consists of **constructs instances** and they do not have a specific representation. To highlight this difference, we have used dashed boxes and dashed lines in the example depicted in figure 1-3.

Meta-model

A model is built using concepts defined in a **meta-model**. The meta-model represents the different **meta-concepts** of a language and the **meta-relationships** between them. The meta-concepts may also be called **semantic constructs**. They have to be instantiated in order to obtain the construct instances that composed the model.

The **meta-concepts** that are represented in figure 1-3 come from KAOS. These **meta-concepts** (e.g., Goal, Agent, Operation, Entity and Association) and their **meta-relationships** (e.g., Responsibility, Performance, Input and Link) are represented respectively as UML classes and UML relationships. Reader interested in the complete KAOS meta-model can refer to Chapter 5.

Diagram or visual representation

A **diagram** is the symbolisation of a model. When a modeller represents a model concretely, he has to choose a concrete syntax. There are different kinds of representation: graphical, textual or even acoustic.

As mentioned by Mackinlay in [Mackinlay, 1986], the term **visual representation** is a synonym of a **graphical representation**. To complete the list of synonyms, visual representations that symbolises a **model** is referred to as a **diagram**.

In [Moody, et al., 2010] the term **visual notation** is defined in this way: "A visual notation (or visual language, graphical notation, diagramming notation) consists of a set of **graphical symbols**, a set of compositional rules for how to form valid visual sentences, and definitions of their meanings (visual semantics). The set of symbols (visual vocabulary) and compositional rules (visual grammar) forms the visual (or concrete) syntax. Graphical symbols are used to symbolise (perceptually represent) **semantic constructs**, typically defined by a **meta-model**. The meanings of graphical symbols are defined by a mapping to the constructs they represent. "

The term **icon** is also used as a synonym of graphical symbol but we prefer to consider icons as a subcategory of symbols. Indeed, a symbol may resemble or not the semantic construct it symbolises, while an icon should directly suggest its meaning.

To complete the discussion on figure 1-2, diagrams contain **symbol instances** that are used to symbolise the **construct instances**. The symbol instances are instances of **graphical symbols** defined in the visual notation of the language. As illustrated in figure 1-3 each concept is "translated" into a specific symbol (e.g., the "OrderPicker" concept is translated into a hexagon with a sticky man).

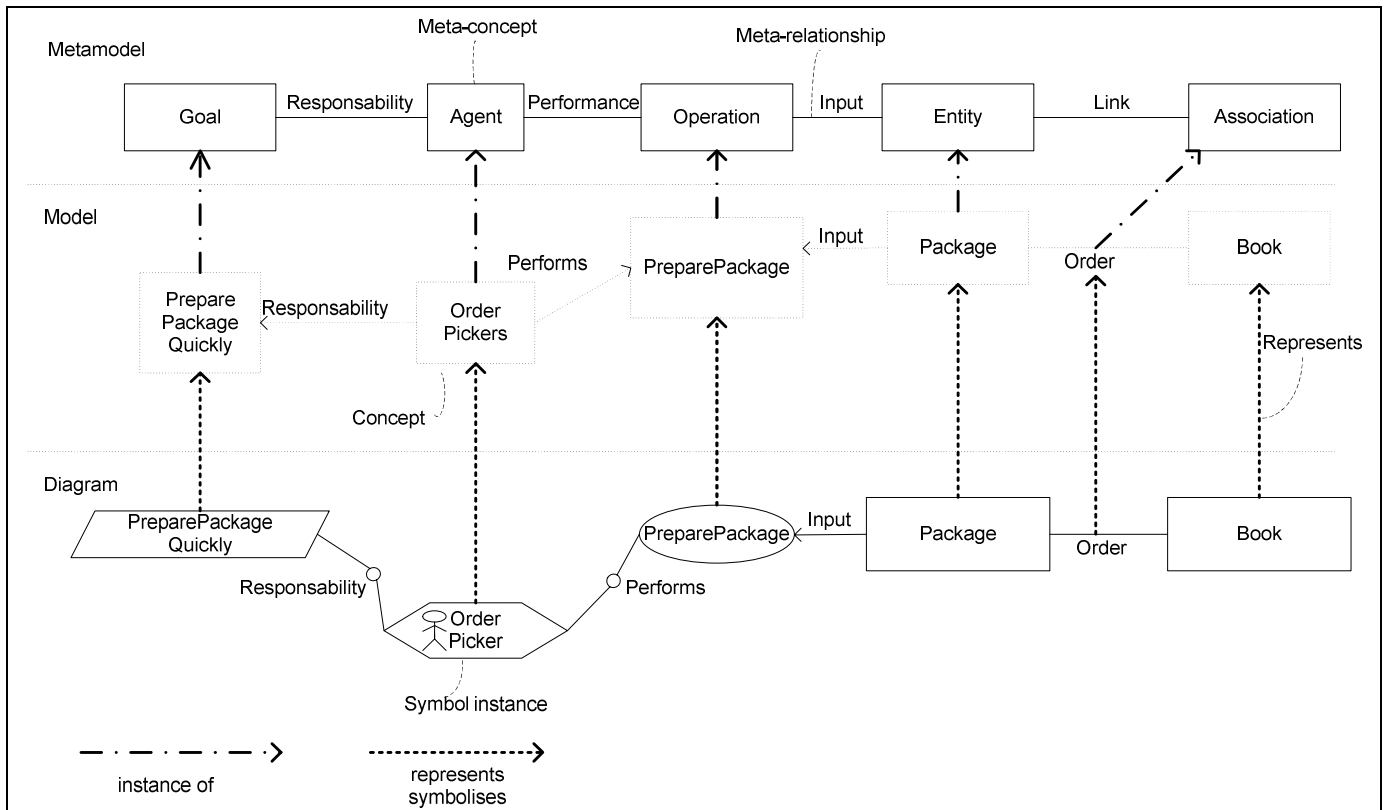


Figure 1-3 The meta-model of KAOS, an example of model and the associated diagram

1.3 Text format

For the readability of the text, we have formatted the text on different ways. Words or expressions are in **bold** when they are definitions. Words or expressions that belong to the running example are in this format.

Part I

Background

Chapter 2 Requirements Engineering

Requirements engineering is a discipline started in the 1990's [Nuseibeh, et al., 2000] and nowadays, it is considered as crucial part of software development. It allows avoiding conceptual errors in software which cost a lot of time and money to be solved. The goal of this discipline is to understand the problem as a whole and produce documents that will be used to develop the future software. The explanations that follow are based on [Lamsweerde, 2009], [Heymans, 2006-2007], [Nuseibeh, et al., 2000] and [Habra, 2009-2010].

2.1 Introduction to requirements engineering

The requirements engineering discipline is used to study the different aspects of a particular problem in the aim of developing a system. To solve a problem, it has to be clearly understood and defined. Even if it seems obvious; it is not always easy to figure out the right problem and its scope. Replying to this question will define *what* is the problem, *who* is involved in the responsibility of solving the problem and *why* the problem needs to be solved. The solution of the problem is called the *machine*.

Jackson in [Jackson, 1995] explains that the machine will be used in an environment that will interact with it. These interactions (aka. shared phenomenon) have to be taken into account to build the machine; otherwise it will not fulfil the requirements.

Requirements engineering describes the phenomena of the machine in its environment and the assumptions that are made about it. In other words, requirements engineering studies the environment phenomena including those which are shared with the machine. Conversely software design studies only machine phenomena.

One of the goals of requirements engineering is to model the machine to understand precisely which needs it has to fulfil and the environment that surrounds it.

During requirements engineering process, engineers will consider 2 versions of a same system:

- the system-as-is, the actual system (before building machine),
- the system-to-be, the future system (after building the machine).

The system-as-is informs us of the objectives, regulating laws, deficiencies and limitations that the system-to-be will encounter. The system-to-be gives information to build the new software according to assumptions and the hypothesis on the environment.

The machine that will be developed represents only a part of the system-to-be (it is called the software-to-be). It includes also other parts that belong to the surrounding world: departments of the company that will play a role in the system-to-be, devices that work under specific constraints and conformed to physical laws and finally foreign software that we will have to take into account.

As the environment will certainly evolve, the software-to-be should be developed keeping in mind that it will have to follow the evolution of the environment sooner or later.

As said before, the requirements engineering process has 3 dimensions: the *why*, the *what* and the *who*. These 3 dimensions are detailed in the next section.

2.2 The WHY, the WHAT and the WHO dimensions

The WHY dimension studies the contextual reasons of building a new (version of the) system (e.g., new regulation laws). It defines objectives that decreases or lower the limitations of the system-as-is and exploits the opportunities (e.g., doing more in less time). These objectives have to be described precisely and in details, the interactions with the environment will have to be studied carefully. This part of the process examines the domain in which the problem is situated. The domain might be complex in terms of concepts, regulating laws, procedures and terminology. Each part must be thoroughly explored and nothing left at random. Most of the time, objectives can be satisfied in different ways. These alternatives have to be studied separately to know their respective pros and cons, and then the most preferable one can be selected. Moreover, objectives can be defined by different sources that have different points of view and interests which can lead to conflicting goals. However, at the end, system engineers have to ensure that the set of objectives is coherent and corresponds to a good compromise between the needs of the different sources.

The WHAT dimension deals with the functional services that the system-to-be should provide to satisfy the objectives identified in the WHY dimension. These services are based on specific system assumptions and they have to meet constraints related to performance, security, usability, interoperability¹ and cost. Considering these elements will allow us to build scenarios that will simulate the system-to-be and to verify that system services, constraints and assumptions are identified correctly. The traceability between the system objectives and the scenarios must be described in documentation. It will be used later to check how and by who the objectives are satisfied. Scenarios have to be formulated in terms and languages that all concerned parties –of the system-as-is and the system-to-be– will understand. After reading and understanding them, the concerned parties will give their validation and the development team will be able to start working on the system.

The WHO dimension establishes the assignment of responsibilities for achieving the objectives, services and constraints among the components of the system-to-be. The responsibilities can be assigned to human, devices or software; sometimes there are alternative possibilities that have to be studied separately. We have to know the pros and the cons for each of them. The responsibilities will be assigned so that the risk of not achieving important system objectives is as low as possible. Indeed if a part of the system is not achieved properly, it is the whole system that will not work properly.

2.3 Categories of statements

During the requirements engineering process, engineers gather, write, analyse, correct and adapt statements given by the different persons involved in the building of the system-to-be. There are several kinds of statements: descriptive statements, prescriptive statements, system requirements, software requirements, domain properties, assumptions and definitions [Lamsweerde, 2009].

In [Lamsweerde, 2009], **descriptive statements** are defined as "state properties about the system that hold regardless of how the system behaves" and **prescriptive statements** are defined as "state desirable properties about the system that may hold or not depending on how the system behaves". To differentiate them easily, descriptive statements are generally written in the indicative mood (e.g., if train doors are open, they are not closed) and prescriptive statements are written in the optative mood (e.g., train doors shall always remain closed when the train is moving).

¹ These elements could correspond to a 4th dimension, called the HOW dimension. This dimension should describe HOW the requirements have to be fulfilled. Anyway, we will not explain it longer because it is not mentioned in [Lamsweerde, 2009].

The major distinction between descriptive statements and prescriptive statements is that the first ones cannot be discussed, modified or weakened while the second ones can be discussed, modified or weakened.

In [Lamsweerde, 2009], **system requirements** (aka. user requirements) are described as: "prescriptive statements that are enforced by the software-to-be, sometimes in collaboration with the other system components and formulated in term of environment phenomena and that can be stated". They describe the usage of monitored and controlled variables. These variables are numeric information that the software monitors or controls through input/output devices.

In opposition, **software requirements** (aka. product requirements, specifications) are defined in [Lamsweerde, 2009] as prescriptive statements that are enforced solely by the software-to-be and formulated only in terms of phenomena shared between the software and the environment. They have to be written in terms of input/output variables of the software because they will be used by developers.

In [Lamsweerde, 2009], **domain properties** are "descriptive statements about the problem world. They correspond to physical laws that cannot be changed (e.g., a car is moving if and only if its speed is non-null). These properties do not vary regardless of how the system will behave and even regardless of whether there will be any software-to-be or not".

Assumptions are defined in [Lamsweerde, 2009] as "prescriptive statements that are satisfied by the environment and are formulated in terms of environmental phenomena" (e.g., a car's measured speed is non-null if and only if its speed is non-null).

Finally in [Lamsweerde, 2009], **definitions** are described as "sentences that allow domain concept and auxiliary terms to be given a precise, complete an agreed meaning that will be used by every one" (e.g., a patron is any person who manages a company).

All statements emerging from the requirements engineering should be written in the documentation, then anyone reading the documentation can directly know if a statement is a requirement, a domain property or a definition.

System requirements have to be translated into software requirements. It is not a simple mapping between the machine vocabulary and the software one. Domain properties and environmental assumptions, that can be used to confirm and to validate the correctness of the translation, are called **satisfaction argument**. Satisfaction arguments are used to manage the traceability among requirements and assumptions during the evolution phase of the requirements engineering process.

2.4 Categories of requirements

Requirements can be classified into 2 categories: the functional and the non-functional requirements.

Functional requirements define the functional needs that the software-to-be has to provide to its environment. They are a part of the WHAT dimension (e.g., the plane control software shall control the takeoff and the landing of all the system's planes). These functional needs will be the result of operations automated by the software. Functional requirements can also divide the work in units that will be supported by the software. The set of these units is the software-to-be.

Non-functional requirements define constraints on the way that the software-to-be has to satisfy to fulfil the functional requirements or on the way they should be developed (e.g., plane altimeter have to be refreshed every second). These constraints can be classified in many groups [Davis, 1993] [Robertson, et al., 1999] [Chung, et al., 2000]:

- Quality requirements (aka. quality attributes): based on quality properties that the functional effects of the software should have. This group can be fined-grained into: safety, security, reliability, performance and interface requirements.
- Compliance requirements: based on laws, norms or any legal rule.
- Architectural requirements: mandatory structural constraints that have to be applied to the software-to-be to be compliant with its environment. These requirements can be divided in distribution constraints and installation constraints.
- Development requirements: informed the developers on the way the software should be developed (cost, deadline, variability and maintainability).

Generally a requirement cannot belong to both categories at the same time. But, sometimes the boundary between them is very fuzzy and the requirement may be shared by the 2 categories (e.g., in a library software system, security requirements are non-functional requirements because stakeholders do not care about it, conversely in a firewall software system, security requirements are functional requirements because they are asked by stakeholders).

2.5 Requirements lifecycle: processes, actors and products

Actors of the requirement lifecycle are called **stakeholders**. This is a group or a person affected by the system-to-be and who may influence how the new system will be built and they play a very important role in the requirements engineering process. Stakeholders can be: managers of the company, future users and legal authorities. System-as-is stakeholders can be different from the system-to-be stakeholders because the new system will probably involve different persons. They are also responsible of the acceptance of the system-to-be.

The requirements engineering process consists of several activities that concern different products and actors. It is composed of 4 activities: (i) domain understanding & requirements elicitation activity, (ii) evaluation & agreement activity, (iii) specification & documentation activity and (iv) validation & verification. These activities are realised by system engineers and stakeholders.

The previous activities are called **phases** of the requirements engineering process. The output of one phase is generally used as input for the next phase. These phases are rarely performed sequentially. They are so intertwined that they overlap each other and that, sometimes, backtracking is needed. According to this view, the requirements engineering process can be seen as iteration on successive increments as described by the spiral model [Boehm, 1988]. This model involves that when a complete iteration of the steps is done, another one can started. The spiral model and the 4 phases are represented in figure 2-1.

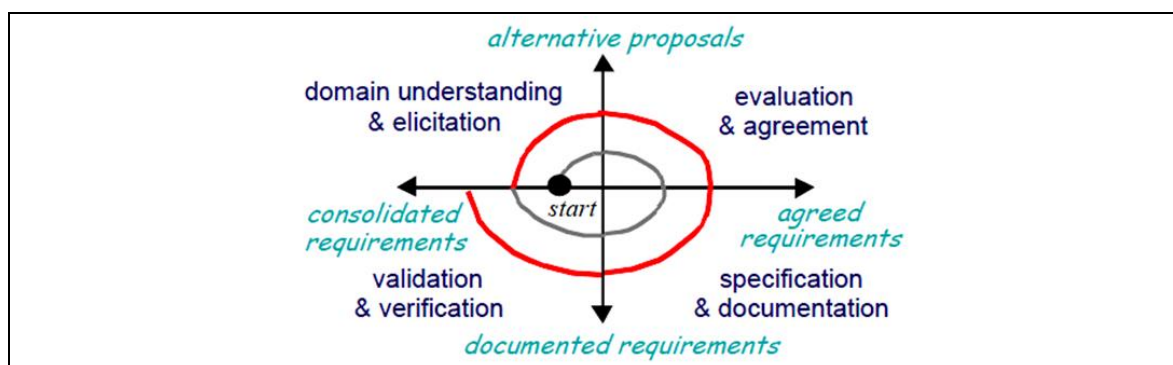


Figure 2-1 Spiral model for requirements engineering process [Lamsweerde, 2009]

1. The domain understanding & requirements elicitation activity

As soon as the stakeholders are defined, system engineers can start to study the system-as-is to understand how it works. This step is called the domain understanding. Its goal is to situate the domain in which the problem is rooted and what the roots of the problem are. This analysis includes learning of: the system-as-is organisation (organization, strategic objectives, business policies, roles played by the different units and the dependencies among them) and the system-as-is scope (the objectives, the involved parts and the concepts on which it relies) and finally the strengths and the weaknesses of the system-as-is seen by the stakeholders. This set of activities will produce a first draft proposal that describes the contextual aspects. It is also very useful to write a glossary of terms that contains the definitions of key concepts that everybody will agree. Thanks to this exercise, system engineers will understand the different points of view of the stakeholders (e.g., user view, developer view, manager view).

During the second part of this activity, stakeholders and system engineers will have to collaborate to make a list of the requirements of the system-to-be as well as the assumptions that will be used to build the latest. This list will allow them to study the weakness of the system-as-is and to improve the domain understanding. Requirements elicitation is one of the most important activities of the requirements engineering process.

Requirements of the system-to-be will have to meet the following objectives [Lamsweerde, 2009]:

- respect the organisational and technical constraints,
- improve the system-as-is,
- take into account new technologies and market conditions,
- evaluate alternatives of which processes can be automated and which should be left under the responsibility of the environment,
- describe scenarios with interaction between the software-to-be and its environment,
- take into account domain properties and assumptions about the environment that are necessary for the software-to-be.

The requirements and assumptions list of the system-to-be will be added to the first draft done during domain understanding step. This document will be used for the evaluation and agreement activity.

2. The evaluation & agreement activity

During the evaluation and agreement activity, considered decisions are taken, based on the elicitation requirements activity. Conflicting requirements have to be identified and resolved, the alternative options have to be evaluated and compared and if necessary a priority has to be given to requirements. Negotiations about these elements may be required to reach a consensus.

At the end of these 2 first steps, an agreement about requirements and assumptions of the system-to-be has to be found by all stakeholders of the project.

3. The specification & documentation activity

In the specification and documentation activity, the aim is to detail, structure and document the agreed characteristics of the system-to-be. These characteristics are documented, in what we call "the requirements document". This document contains also satisfaction arguments, a description of likely variants and revision, acceptance test data and cost figures. If some parts of the document concern specific parties (such as users or developers), they have to be written in an understandable form by this audience to receive their validation (i.e., documents have to be written with a formalism adapted to the readers).

4. The validation & verification activity

In the validation and verification activity, specifications have to be validated by stakeholders. If there are inadequacies between specifications and the stakeholder wishes, they have to be identified and solved before the software requirements are transmitted to the developers. This step offers a quality assurance for the software-to-be.

After this activity, a document that contains the consolidated requirements will be produced. A prototype for requirements validation and additional test data can also be produced to verify if the system-to-be meets the requirements. This last step allows building the model that will be used to communicate with the client and managing the project.

2.6 Target qualities and defects to avoid

Writing a good requirements document is not easy but it is crucial in order to build a software-to-be that will encounter a maximum of the stakeholder's needs. The way to reach this goal is long and full of traps. Below the main qualities of requirements are described followed by the most critical requirement errors and flaws.

Requirements qualities [Habra, 2009-2010],[IEEE-STD830, 1998]

Correctness: every requirement has to be asked by stakeholders.

Completeness: requirements, assumptions and domain properties have to be sufficient to ensure that the system-to-be will satisfy its objectives. The specification of the requirements has to be detailed enough for software development.

Unambiguous: requirements cannot have another interpretation.

Consistent: there can not be any conflicts between the requirements. There are many types of conflicts (e.g., about the behaviour of the system or the definitions of the vocabulary).

Verifiable: it should be possible to test if a requirement is fulfilled or not. In other words, a test to verify the requirement should be done.

Comprehensible: requirements have to be written in a language understandable by the stakeholders.

Modifiable: requirements have to be written in a style and structurally allowing any change in requirements to be reflected in a way that is simple, complete and consistent.

Traceable: requirements have to be written to facilitate references in the design document and test document.

Traced: the source of the requirements should be easily found.

Independent: requirements have to be independent of any architecture, algorithm or code.

Requirement errors and flaws

The 4 more critical kinds of errors are:

Omission: will result in the software failing to implement an unstated critical requirement, or failing to take into account an unstated critical assumption or domain property.

Contradiction: the requirement will solve the problem in an incompatible way.

Inadequacy: the requirement will not solve the problem in the most adequate way.

Ambiguity: the requirement can be interpreted in different ways.

These errors/flaws are the opposite of what a good requirement should be. Requirement should NOT be: *immeasurable, overspecified, unfeasible, unintelligible, poorly structured, poorly modifiable, opaque*.

They should not contain *noise* and *forward references*.

Now that the requirements engineering background is set, we will present the most common goal-oriented languages.

Chapter 3 Goal-oriented languages

The requirements engineering approach has shown some inadequacies. They are mainly situated at requirements level [DeMarco, 1978] [Ross, 1977] [Rumbaugh, et al., 1991] that focuses only on processes and data and do not take into account the aim of the system-to-be. The links between requirements and high level objectives cannot be done easily. To fill this gap, in goal-oriented requirements engineering languages, systems will be represented in terms of goals. Goals have to be reached to build a system that corresponds to the users' needs. Goal-oriented models will help to reason about systems in terms of the WHY and WHO dimensions of requirements engineering process. Like the previous chapter, this chapter is based on [Lamsweerde, 2009].

3.1 What are goals?

Goals-oriented requirements engineering implies that goals are used for the different steps of the requirements engineering process (elicitation, evaluation, negotiation, elaboration, structuring, documentation, analysis and evolution). We will now define 2 main terms used in goal-oriented requirements engineering process: goal and agent.

There are many manners to define the term **goal**. Van Lamsweerde in [Lamsweerde, 2009] defines it as "an objective that the system should achieve through cooperation of agents in the software-to-be in the environment. As a consequence, goals must be formulated in terms of actions (aka. operations) that will be shared among system agents; such actions will be realised by some agents and monitored by others". Anton [Anton, et al., 1994] states that goals are "high-level objectives of the business, organization or system; they capture the reasons why a system is needed and guide decisions at various levels within the enterprise." In practice goals are prescriptive statements like requirements (in opposition to domain properties that are descriptive). In fact, requirements "implement" goals much the same way as programs implement design specification [Lamsweerde, et al., 2003].

An **agent** is an active component of the system, playing a specific role in the satisfaction of the goals [Lamsweerde, 2009]. The set of system agents defines the scope of the system (a part of this scope will be the software-to-be). There are different types of system agents:

- human agents that play specific roles (operators or end-users),
- devices such as measurement instruments (sensors) or communication media,
- piece of existing software components such as foreign components in an open system,
- piece of new software components that will be used in the software-to-be.

A system agent can be an existing piece of software or a new piece of software. The term 'system' may refer to the system-as-is or the system-to-be.

3.2 Where are the goals coming from?

It is not an easy task to identify the goals. Some of them state explicitly as system objectives during goal elicitation but they are more often implicit and we need to 'extract' them from the preliminary document and from other pieces of information given by the stakeholders. Some goals come from the complaints about the system-as-is and others can be deduced from interviews done with stakeholders. They are formulated as sentences that are formed by keywords such as 'has to', 'shall', 'in order to' or 'so that'.

When goals are explicitly stated as system objectives, we have to ask 'HOW' questions to refine them into finer-grained goals and make them more explicit. In the requirements engineering process, the sooner a goal is defined and validated, the better.

3.3 The granularity of goals and their relationship with requirements and assumptions

In [Lamsweerde, 2009], goals can be stated at different levels of abstraction:

- at higher levels, there are general goals (aka. coarser-grained goals). They describe general objectives that have to be reached by the system-to-be (e.g., the plane speed shall be increased by 50%),
- at lower levels, there are specific goals (aka. finer-grained goals). They describe technical objectives that the software-to-be will have to fulfil (e.g., plane altimeter has to be refreshed every second).

As there are many levels and many granularities, modellers should create a specification-structuring mechanism based on **contribution links** among goals. This mechanism will help the stakeholders to evaluate the refinement of a goal. A coarser-grained goal can be **refined** into finer-grained goals. The set of these finer-grained goals will be used to reach the coarser-grained goal. The mechanism of refinement can be applied in the opposite way; in this case finer-grained goals can be **abstracted** towards coarser-grained goals.

The more a goal is refined, the better it is because the responsibility of the goal will be divided into many agents. Moreover the tasks that will be realised by these agents will be defined more precisely and more accurately.

When goals are refined at their maximum, they can be of 2 types. In [Lamsweerde, 2009],

- **requirement** is defined as "a goal under the responsibility of a single agent of the software-to-be",
- **expectation** is defined as "a goal under the responsibility of a single agent in the environment of the software-to-be".

In goal-oriented languages, the different terms are expressed as statements that can be of different types. In the table 3-1, the different terms of Chapter 2 (i.e., system requirements, software requirements, domain properties and assumptions) are translated in terms of statements.

Table 3-1 Translation of terms used in requirements engineering languages into terms used in goal-oriented languages

Requirements engineering languages	Goal-oriented languages
System requirement	- goal described by a prescriptive statement under the responsibility of multi-agent
System requirement that needs a single software agent to be fulfilled	- Requirement
Software requirement	- Requirement
Domain property	- Descriptive statement about the environment. It should be independent of the behaviour of the system (still called domain property)
Assumptions	- Prescriptive for environment agent (aka. expectation) - Descriptive for other agents
Environment assumptions	- Expectations (if it will be satisfied by a single environment agent) - Domain hypotheses (descriptive statements satisfied by the environment and subject to change).

Figure 3-1 describes the different types of statement in a hierarchical way. It is clear, there are 2 kinds of statement: prescriptive and descriptive. The prescriptive statements can be realised either by multi-agent (aka. multi-agent goal) or by a single-agent (aka. single-agent goal). In this last case, the goal can be called a requirement (if fulfilled by a software agent) or an expectation (if fulfilled by an environment agent). Among the descriptive statements, we can distinguish the domain properties and the domain hypothesis.

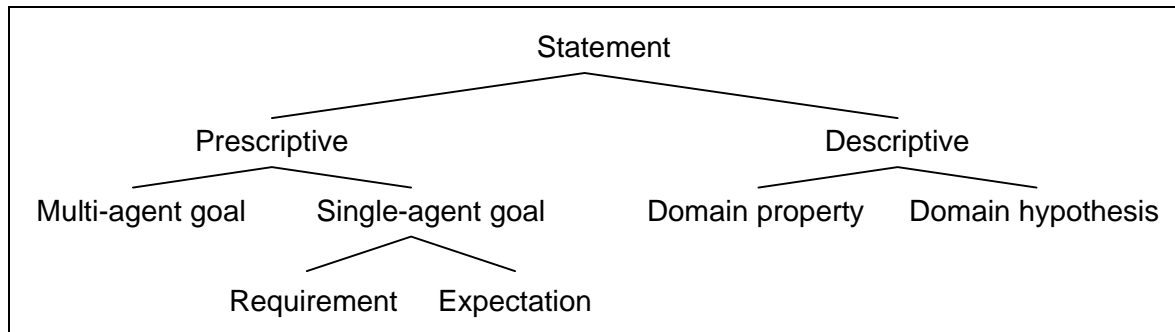


Figure 3-1 Goal statements hierarchical classification [Lamsweerde, 2009]

3.4 Goal types and categories

Goals can be classified along 2 dimensions: types and categories. The **type** of a goal depends if it fulfils a goal intended in the system behaviours or if it is a preference among alternative behaviours. The **category** of a goal is its functional or non-functional properties under a single abstraction, in other words it depends if it describes a functionality or a quality constraint.

Types of goals

There are 2 main types of goals: behavioural goals and soft goals. They do not overlap, i.e., a goal is either a behavioural goal or a soft goal but, in any case, not both at the same time.

Behavioural goals describe the behaviour of the system-to-be with declarative statements. The behavioural goal set of a system implicitly defines a maximal set of acceptable system behaviours. Behavioural goals can always be described in a clear-cut sense that will allow determining if a goal is satisfied or not. Behavioural goals can be classified into 3 subtypes: achieve goals, maintain goals and its opposite avoid goals. **Achieve goals** are defined in [Lamsweerde, 2009] as "prescribed intended behaviours where a target condition must sooner or later hold whenever some other condition holds in the current system state". To recognize this type of goal, the goal name will be preceded by Achieve [TargetCondition]. **Maintain goal** is defined in [Lamsweerde, 2009] as "a goal that prescribes intended behaviours where a 'good' condition must always hold (possibly under some other condition on the current state)". The name of this type of goal will be preceded by Maintain [GoodCondition] or its dual variant Avoid [BadCondition].

Soft goals are preferences among the different alternatives of system behaviours. A soft goal can not be written in a clear-cut sense but we might say that the system behaviour will be better reached by some alternatives and less by others. They are used as criteria for selecting one system option among many alternatives. Like Achieve and Maintain goals, their labels can be preceded by one of the following pattern: Improve [TargetCondition], Increase [TargetQuantity], Reduce [TargetQuantity], Maximize [ObjectiveFunction], Minimize [ObjectiveFunction].

Categories of goals

Goals can also be classified into 2 categories: functional or non-functional according to their purpose. **Functional goals** state as purpose underlying a system service (e.g., payments must be secure). A **non-functional goal** describes a quality or constraint during the development of the system-to-be (e.g., the system must be cheap and efficient). Each of these categories can be refined into subcategories as shown on figure 3-2. And conversely to goal types, goal categories can overlap.

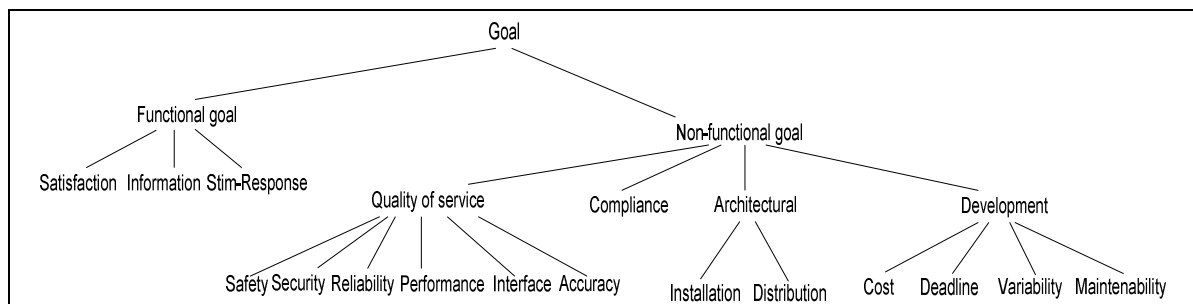


Figure 3-2 Goal categories hierarchical classification [Lamsweerde, 2009]

Main functional goal categories are:

- Satisfaction goal that fulfils agent requests.
- Information goal that sends data to agents to inform them about the system state.
- Stimulus-response goal that describes an action that occurs when a specific event happens.

Main non-functional goal categories are:

- Accuracy goal that needs to know the state of variables controlled by the software to translate accurately the state of the corresponding quantities controlled by environment agent.
- Security goal that describes the different types of agent protections against unexpected behaviours (those can be malicious or accidental).
- Performance goal that describes the expected performance of the system-to-be.
- Cost goal that defines the budget of the stakeholder to build the system-to-be.
- Other non-functional goal types that have the same description as in section 2.4.

Difference between goal types and goal categories

As said before, types and categories are 2 orthogonal dimensions to classify goals (i.e., each goal can be classified in the 2 dimension). These 2 dimensions are completely independent: to find the type of goal, we have to consider the semantic of the goal (does it satisfy system behaviour in a clear-cut sense or not?) and to find its category, we have to balance its pragmatic sense (does it describe a functionality of the system-to-be or a quality of a constraint?). Then, we should not confuse soft goals with the non-functional goal.

3.5 The central role of goals in the requirements engineering process

There are multiple reasons described in [Lamsweerde, 2009] and [Lapouchnian, 2005] for goals being so important in the requirements engineering process:

- Goal refinement provides an intuitive mechanism for structuring hierarchically complex specifications. Goals may be refined into sub-goals until they reach software requirements or expectations. When the set of sub-goals is fulfilled, its parent is also fulfilled.
- Goals provide rationale for requirements. When a goal is unclear, we can browse the goal refinement diagrams to find the goals to which it contributes and in the same time explain the requirement and its rationale to the stakeholders.
- Conversely we can identify which requirements have to be fulfilled to reach a specific goal. When a goal appears during the elicitation or evaluation phases of the requirements engineering process, we can study the different ways to achieve it and find which requirements are needed to contribute to this goal.
- Goals provide an accurate criterion to check the completeness of requirements. A set of requirements matches completely with a set of goals if and only if all the goals are satisfied when all the requirements are satisfied and if environment assumptions and domain properties are taken into account.
- Goals provide an accurate criterion to check the pertinence of requirements. A requirement is pertinent if it satisfy at least one goal in a set of goals.

- Goals provide bases to do risk analysis. A risk is defined as a lack of completion of some objective. Risks can be identified as obstacles that will have to be overcome to fulfil the goal. Modellers can draw risk trees using the refinement method. Then, during the evaluation and agreement activity, stakeholders will have to find goals that will prevent or reduce the occurrence of the identified risks.
- Goals can be used to manage conflicts among requirements. Conflicts between requirements are generally the consequence of conflicts between the underpinning goals to which the requirements are dedicated. Conflicting goals are expressed by stakeholders that have different points of view and concerns. To resolve goal conflict, we need to detect them, study the different possibilities of resolution, select the best one based on soft goals and diffuse the resolution until the requirement level.
- Goals are used to delimit the scope of the system. The system scope is specified by a set of goals that has to be fulfilled by the collaboration of 'good' agents. 'Bad' agents prevent to reach some of these goals. The set of 'good' and 'bad' agents delimits the scope of the system.
- Goals are used as basis for reasoning about alternative possibilities. A goal diagram can be refined into many alternative combinations of sub-goals which will define different possibilities to reach the main goal. Doing this will allow studying different solutions by assigning the responsibility of the goals between alternative agents. Incidental or malicious menaces of a goal can be avoided with alternative goals. Conflicts among goals can be resolved through alternative resolution goals. These alternative goals will allow making different system designs.
- Goals are used to keep the traceability. We do not need any extra mechanism to find the chains of satisfaction arguments as they are available in the goal-oriented requirements engineering process. The traceability can be seen from top-down level and conversely.
- Goals are useful for evolution of the system-to-be. When the system is built, a goal is fulfilled by requirements that are selected during the evaluation phase but the requirement can evolve towards another way of achieving the same goal. The same reasoning can be done about sub-goals of a goal. Sub-goals can evolve to fulfil the main goal but the higher level is the goal, the more it is stable through successive revisions. Even if a system is reviewed many times, it will generally reuse a common set of higher-level goals while the lower-level ones will change.

Finally, as said in [Lamsweerde, 2009]: "we can say that requirements 'implement' goals much the same way as programs implement design specifications. Without a specification, we cannot develop the correct program that meets the specification. Without goals, we cannot engineer complete, consistent, pertinent and adequate requirements that meet them."

3.6 The choice of KAOS

In [Lapouchnian, 2005], the best known goal-oriented languages are enumerated: Non-Functional Requirements (NFR) framework, i*/Tropos, KAOS and Goal-Based Requirements Analysis Method (GBRAM).

The **NRF framework** is described in [Chung, et al., 2000]. It focuses on the modelling and analysis of non-functional requirements. The goal of the framework is to elicit NFR of the system-to-be, decompose them and if possible identifying the NFR operations, manage conflict between NFR, prioritise them and highlight the dependencies between them. This framework suggests using 3 types of soft goals:

- NFR soft goals that have to be taken into account in the system-to-be.
- operationalising soft goals can be considered as software requirements and have to satisfy the NFR soft goals.
- claim soft goals that pinpoint the justification for soft goal refinements or soft goals prioritisation.

In [Yu, 1997], **i*/Tropos** is defined as an agent-oriented modelling framework. This framework has many goals: requirements engineering, business process reengineering, organizational impact analysis and finally software process modelling. The main role in this model is given to agents. It defines agents as concrete actors, system or human, with specific capacities. Each actor plays a specific role that defines his responsibilities. In this language, there are 2 models:

- the strategic dependency model that shows the dependencies between the different agents.
- the strategic rationale model is used to explore the justification of the process in the system-to-be.

Tropos is based on i* and it is a requirements-driven agent-oriented development methodology [Castro, et al., 2002]. It is used for the development of agent-based systems. The added value of this language is a formal specification language called Formal Tropos [Fuxman, et al., 2001].

KAOS methodology is a goal-oriented requirements engineering approach that uses many formal analysis techniques. In [Lamsweerde, et al., 2003], it is described as a multi-model framework that uses different levels of expression and reasoning: a semi-formal language is used to communicate with stakeholders while a formal language is used to do accurate reasoning. KAOS offers different views of a system as it will be explained in Chapter 5.

And finally the **GBRAM method** described in [Anton, 1996], [Anton, 1997] is based on goals which are identified and abstracted from various sources of information

Table 3-2 The distribution of the main roles in goal-oriented requirements engineering language

Goal-oriented requirements engineering languages	Main roles	Languages
NRF	Soft goals	
i*/Tropos	Agents	
Tropos	Agents	Formal language
KAOS	Multi roles	Semi-formal as well as formal languages
GBRAM	Goals	

We have chosen to study KAOS which is the richer method due to the fact that it analyses the system under different views. A description of the semantics and the syntax of KAOS are given in Chapter 5. Then we have elaborated and discussed its meta-model in Chapter 6.

Chapter 4 Principles of the Physics of Notations

Theory

In this chapter, we will introduce the different concepts of the graphical communication, and then we describe the 9 principles of the Physics of Notations explained in [Moody, 2009]. These principles will help us to identify design flaws of software engineering notations and help us to give practical suggestions for improving them. They will be applied to the KAOS visual notation in the Chapter 7.

4.1 Introduction to graphical notations

Thanks to graphical notations, software designers can communicate effectively with end users and customers. It is often easier to explain technical information and precise descriptions with a schema to non-technical persons [Avison, et al., 2003]. They are also very useful to improve the internal communication between the development team members as well as a mean to support for design and problem solving. But, if graphical notations are really useful and powerful, they have to be used perspicaciously; otherwise their usage can be counterproductive [Cheng, et al., 2001].

The communication theory

Moody in [Moody, 2009] explains the communication theory on this way: “A **diagram creator** (source) encodes **information** (message) in the form of a **diagram** (signal) and sends it to the **diagram user** (receiver). This one will decode the signal. The diagram is encoded using a visual notation (code). The **channel** (medium) is the physical form in which the diagram is represented (e.g., paper, whiteboard or computer screen). **Noise** represents any variations of the signal which can interfere with communication. The **communication effectiveness** is the difference between the attended message and the received message. The bigger the difference, the smaller the effectiveness”. Figure 4-1 illustrates the main concepts of the diagrammatic communication theory.

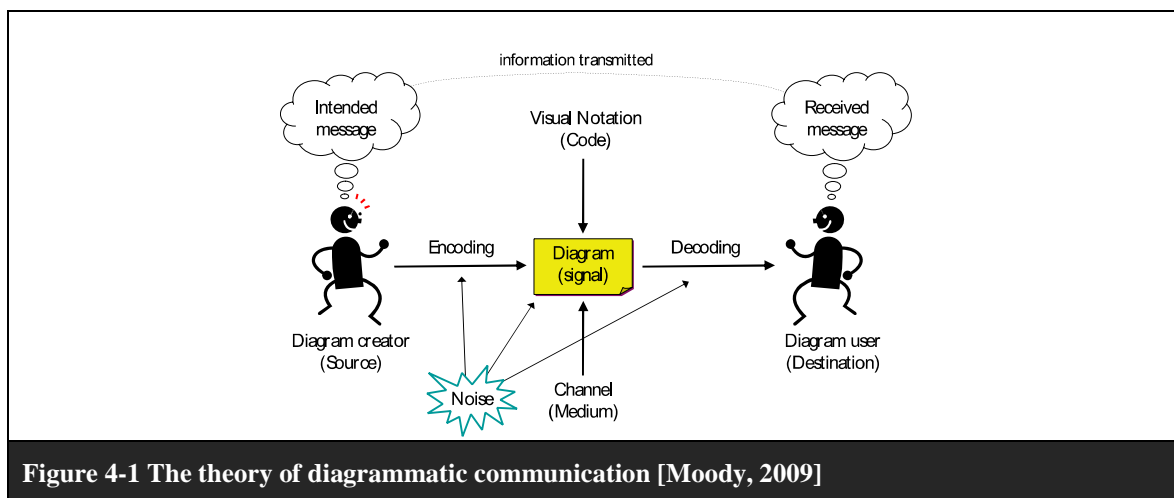


Figure 4-1 The theory of diagrammatic communication [Moody, 2009]

Following this theory, communication has 2 complementary processes: encoding (expression) and decoding (interpretation). To optimize the communication process, we have to work on both sides.

The graphic design space (encoding Side)

The **graphic design space** is composed of **8 visual variables** [Bertin, 1983] to encode graphically the information (see figure 4-2). These variables are called the dimensions of the graphic design space. They are divided in 2 subgroups: planar variables and retinal variables.

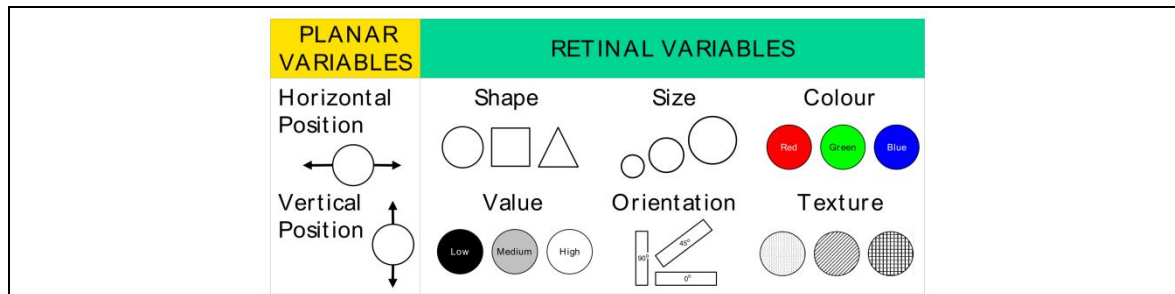


Figure 4-2 The visual alphabet [Moody, 2009]

Each of these variables has a set of values (limited for the shape value, unlimited in the other cases). Each value of a variable can be combined with other values of other variables to create any visual notation. Then, the combination of the values of the variables creates an unlimited set of possibilities. This set can be used as an alphabet by notation designers to create graphical representations. However some variables are more suitable to encode some type of information, some are more suitable than other to describe data. For example, colour can be used to encode nominal data but not ordinal or ratio data because this is not psychologically ordered [Kosslyn, 1989].

Even if each variable has an infinite number of variations, when they are used in a graphic representation, the difference between them has to be significant for the understanding of the diagram user.

In a diagram, there are the primary and the secondary notations. The **primary notation** is the official syntax of the visual notation of a language. The **secondary notation** refers to the use of visual variables not formally described in the language. The secondary notation is mostly used to reinforce the meaning [Moody, 2009].

The human information processing (Decoding Side)

Diagrams have to be optimally designed to be processed as well as possible by the human mind. The perceptual process is automatic, very fast and executed in parallel in opposition of the cognitive process that is a slow process requiring conscious control of attention. Figure 4-3 illustrates how the human graphical information process when decoding a message. It happens in 2 phases:

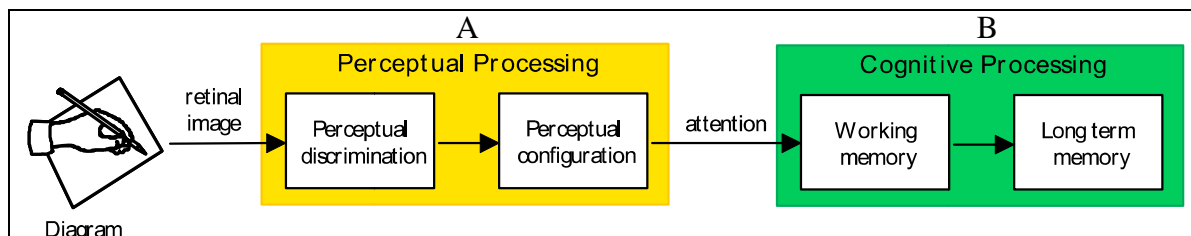


Figure 4-3 The human graphical information processing [Moody, 2009]

- Figure 4-3 part A describes the **perceptual processing** (seeing) which is decomposed in:
 - **Perceptual discrimination:** features of the retinal image (colour, shape, etc) are detected by specialised feature detectors. Based on this, the diagram is parsed into discrete elements and separated from the background [Palmer, et al., 1994]
 - **Perceptual configuration:** following ergonomic laws, the elements are grouped into perceptual units. The construction of these units is based on visual characteristics of the elements.
- Figure 4-3 part B describes the **cognitive processing** (understanding) which is decomposed in:
 - **Working memory:** this is a storage area which reflects the current focus of attention. This is used for active processing and to synchronise rapid perceptual process with slower cognitive processes. However it has very limited capacity and duration. It is a known bottleneck in graphical information processing [Kosslyn, 1989] [Lohse, 1997].
 - **Long term memory:** to be understood, information from the diagram must be integrated with prior knowledge stored in long term memory. Long term memory is a permanent storage area which has unlimited capacity and duration but is relatively slow [Kosslyn, 1985].

4.2 Principles for Designing Effective Visual Notations

In [Moody, 2009], Moody presents the 9 principles he defined to build "good" diagrams (i.e., diagrams that communicate effectively). The communication (or cognitive) effectiveness of a diagram depends on the speed, accuracy and ease required to understand the information presented in this diagram [Moody, 2009].

This section defines the set of principles for designing cognitively effective visual notations. Figure 4-4 represents each principle by a hexagon. The set of all principles represents a honeycomb. This structure has been chosen because it is modular, supports modifications and extensions.

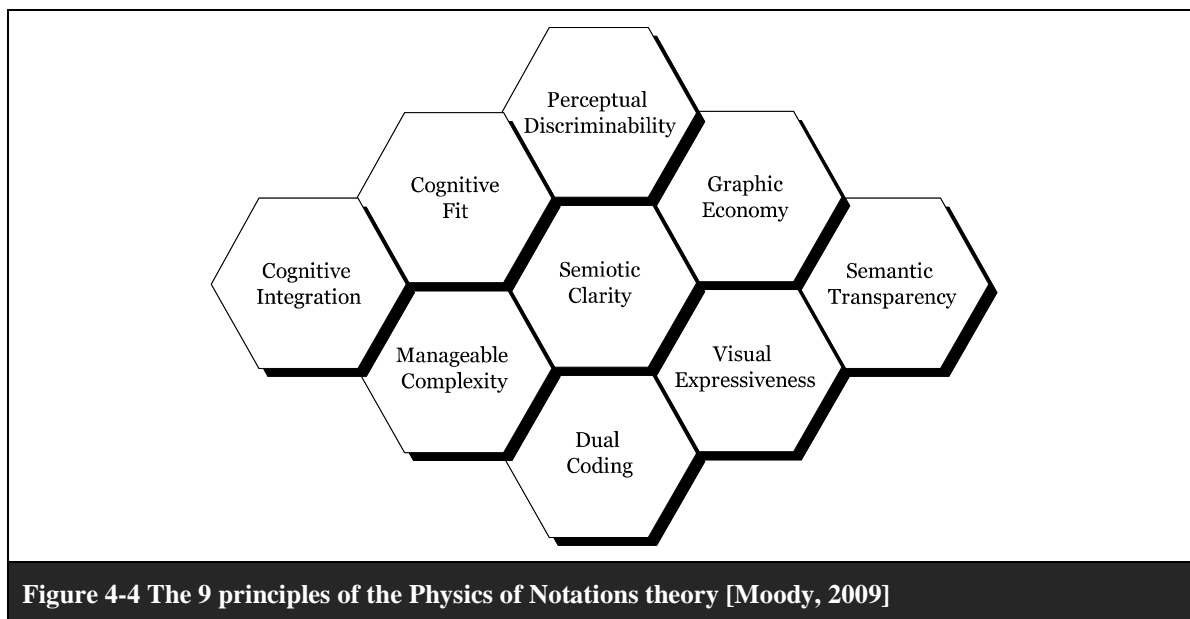


Figure 4-4 The 9 principles of the Physics of Notations theory [Moody, 2009]

The 9 principles are: semiotic clarity, perceptual discriminability, semantic transparency, complexity management, cognitive integration, visual expressiveness, dual coding, graphic economy and cognitive fit.

These principles are desirable and measurable properties of a visual notation. It means that a visual notation will be cognitively effective if these principles are respected. In the following sections we will describe them in a more detailed way.

4.2.1 Principle of Semiotic Clarity

The principle of **semiotic clarity** means that there should be a one-to-one correspondence between semantic constructs and graphical symbols.

According to Goodman's theory of symbols [Goodman, 1968], a notation satisfies the requirements of a notational system, if there is a one-to-one correspondence between symbols and their referent concepts. Notational languages have to follow the principle of semiotic clarity to be more accurate (by eliminating symbol overload), expressive (by eliminating symbol deficit) and parsimonious (by eliminating symbol redundancy and excess). If they do not respect it, one or many anomalies can occur. Figure 4-5 illustrates the different types of anomalies.

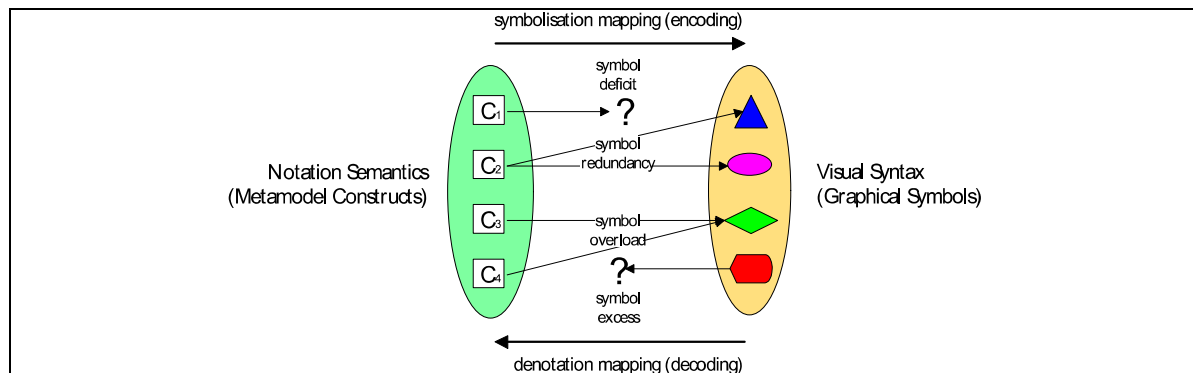


Figure 4-5 The anomalies of the semiotic clarity [Moody, 2009]

Symbol deficit occurs when there are semantic constructs that are not represented by any graphical symbol. This anomaly is not necessarily a problem because it reduces the number of signs on a graphic and it can be useful for the clarity.

Symbol redundancy occurs when multiple graphical symbols can be used to represent the same semantic construct. This phenomenon is called synograph (the graphical equivalent of synonyms). When drawing a diagram, the writer can choose the sign it will use to represent a particular semantic construct. Consequently the reader will have to keep in mind all the different representations of this construct to understand the diagram. And, in the worst case, a designer could use the different graphical representations of a semantic construct on a diagram. It will confuse completely the reader who can not understand why different graphical representations have the same meaning.

Symbol overload occurs when different constructs can be represented by the same graphical symbol. These are called homographs (the graphical equivalent of homonyms). According to Goodman [Goodman, 1968], this is the worst type of anomaly as it leads to ambiguity and misinterpretation.

Symbol excess occurs when graphical symbols are included in the visual notation of a language and they do not correspond to any semantic construct. It increases the graphic complexity and the reader does not know the meaning of the introduced symbol. It has to guess it.

To evaluate the semiotic clarity of a language, we have to proceed to a mapping between the concepts of the meta-model of the language and its visual vocabulary (the set of the symbols used in the language). Each semantic construct should be represented by a symbol and conversely.

4.2.2 Principle of Perceptual Discriminability

Perceptual discriminability is defined in [Moody, 2009] as “the ease and accuracy with which graphical symbols can be differentiated from each other”. This relates to the first phase of human visual information processing: perceptual discrimination (see figure 4-3).

Perceptual discriminability depends on these characteristics: visual distance, primacy of shapes, the perceptual popout, the redundant coding, the textual differentiation and the visual semantic congruence.

Visual distance

The discriminability between symbols is determined by their **visual distance**. According to Moody in [Moody, 2009]: “It is measured by the number of visual variables on which they differ and the size of these differences”. The more differences there are, the easier it is to distinguish 2 symbols. If the differences are too subtle, interpretation errors can occur.

For example, in i^* diagrams, it is difficult to differentiate symbols used for actor, agent and role. They use both a circle (one visual variable) as the basic symbol which is not enough (figure 4-6) [Moody, et al., 2010].

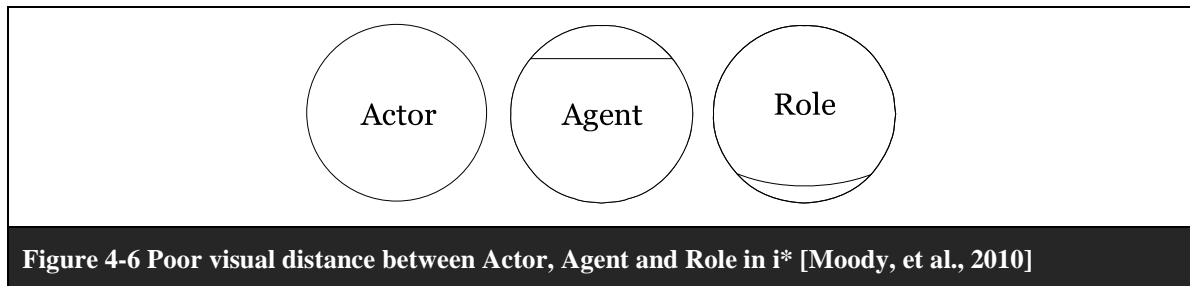


Figure 4-6 Poor visual distance between Actor, Agent and Role in i^* [Moody, et al., 2010]

Primacy of shape

Shapes play the first role in diagrams. It is the primary basis on which we classify the objects in the real world [Moody, 2009]. That is why the shape is the visual variable we mainly use to differentiate between symbols.

In the i^* language, it would be easier to distinguish actor, agent and role if they have a different shapes as proposed in figure 4-7.

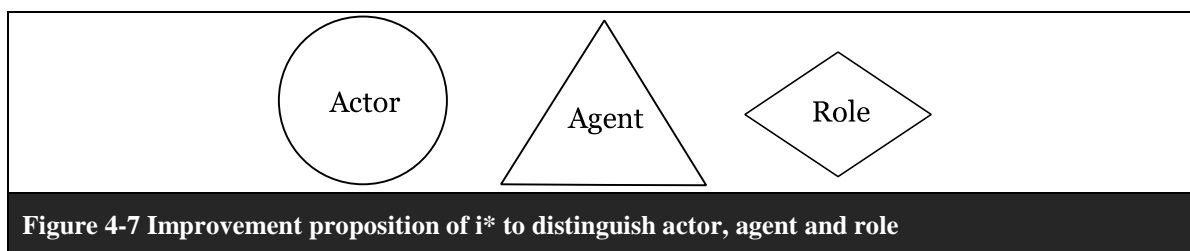


Figure 4-7 Improvement proposition of i^* to distinguish actor, agent and role

Perceptual popout

According to the feature integration theory, visual elements with unique values for at least one visual variable can be detected pre-attentively and in parallel across the visual field [Quinlan, 2003] [Treisman, et al., 1980]. Such elements appear to "pop out" from the drawing without effort. Elements that use many combination values (conjunctions) required serial search which is much slower and decrease the cognitive effectiveness.

Redundant coding

Moody [Moody, 2009] defines **redundant coding** as “using multiple visual variables to increase the visual distance between the symbols”. This technique is used to improve cognitive effectiveness and reduce errors. In other words, redundant coding is representing information with different visual variables to give the reader many possibilities to access to the information (some visual variables are not easy to understand by certain user -e.g., colour-blind cannot interpret colours). In figure 4-8, we have added some colours to the shapes of actor, agent and role which increase the visual distance between the symbols and now it is more cognitively effective.

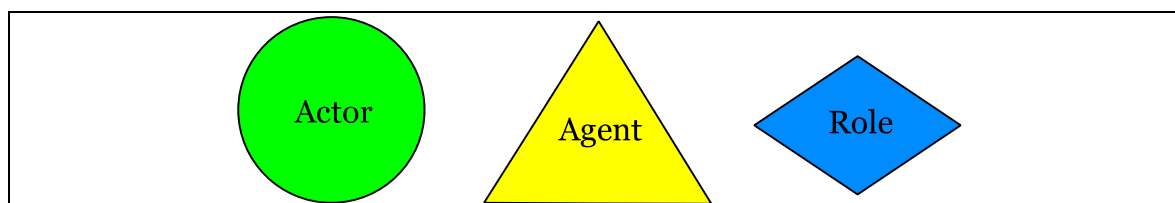


Figure 4-8 Redundant coding: add colour to shapes to increase the visual distance

Textual differentiation

Software engineering sometimes relies on text to distinguish symbols. Symbols that differ only on textual characteristics are technically homograph, as they have zero visual distance (Semiotic clarity) [Moody, 2009].

This kind of differentiation is commonly used but it is cognitively inefficient because the person who will decode the graphic has to read the text to differentiate the symbol what is a very slow process.

In UML class diagrams, there are many examples of textual differentiation. In figure 4-9, the relationship type "substitute" and "import" can only be distinguished by their textual labels.

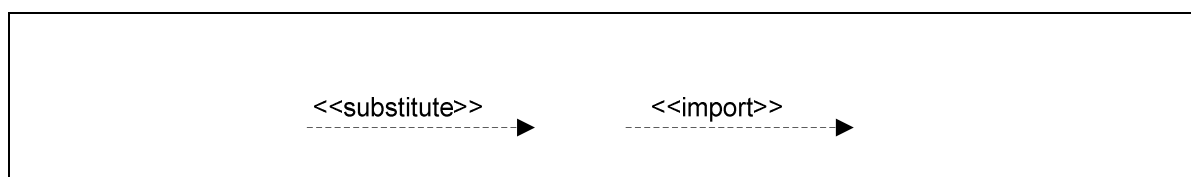


Figure 4-9 Textual differentiation in UML class diagram [Moody, 2009]

Visual-semantic congruence [Moody]

In general, the visual distance between symbols should be congruent to the semantic distance between the constructs they represent: constructs which are very different in meaning (large semantic distance) should have very different symbols (large visual distance), while constructs which are similar in meaning should have similar symbols.

4.2.3 Principle of Semantic Transparency

Semantic transparency involves the use of graphical representations whose appearance suggests their meanings.

As presented in figure 4-7, while the perceptual discrimination implies that symbols have to be different from each other to be recognised, **semantic transparency** involves that symbols should give a cue to their meanings. Using semantically transparent symbols has 2 advantages. Firstly, it decreases the cognitive load because users can use mnemonics to remind their meaning and secondly, they are easier to learn.

Figure 4-10 illustrates the degree of association between form and content. If the meaning of the concept can be inferred from the appearance of the form, it is positive. A novice reader could easily guess the meaning of the symbol. If the form has been chosen arbitrary and has no particular meaning, then the symbol is said to be **semantically opaque**. The reader will have to be informed of the signification of the symbol (e.g., rectangles in ER diagrams). And, in the worst case, the reader could understand a different or an opposite meaning because the symbol has not been well chosen. This case is called **semantic perversity**. Between semantic immediacy and semantic perversity, there are different **degrees of opacity**. Opaque symbols are more or less an aid to the memory but in all cases they require prior explanations.

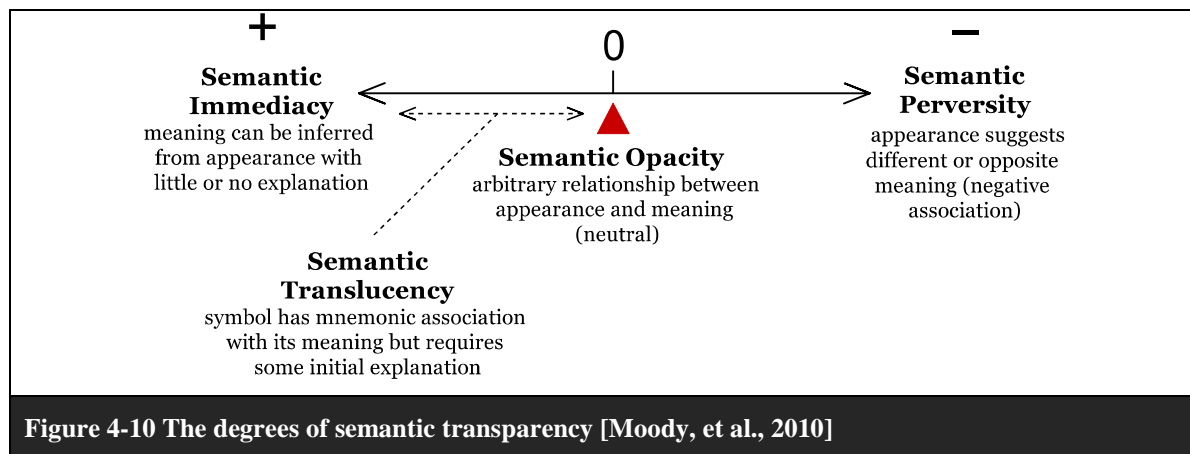


Figure 4-10 The degrees of semantic transparency [Moody, et al., 2010]

Icons (perceptual resemblance)

Icons are symbols that perceptually resemble the concepts they represent [Peirce, 1998]. They increase the cognitive effectiveness and help beginners to understand easily a visual notation. They make diagrams more visually appealing: people prefer real objects to abstract shapes [Petre, 1995] [Bar, et al., 2006].

Icons are very often used in cartography but rarely in software engineering visual notations. Most of the software engineering visual notations use nearly always abstract shapes. The only icon frequently seen is the sticky figure that means 'users' (illustrated in figure 4-11).

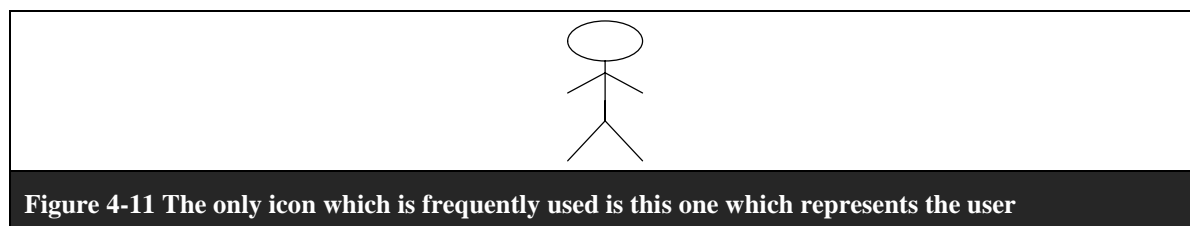


Figure 4-11 The only icon which is frequently used is this one which represents the user

Semantically transparent relationship

The concept of the semantic transparency can also be applied to relationships. The position of the symbols on the paper can also influence the reader to an interpretation of the relationship. For example, left-to-right arrangement of objects suggests causality or sequence while placing objects inside other objects suggests class membership [Moody, 2009].

4.2.4 Principle of Manageable Complexity

Manageable complexity is the principle that allows presenting large amounts of information without overloading the human mind [Moody, 2009]. This principle includes explicit complexity management mechanisms. Unlike textual representations, which can extend over as many pages as required, diagrams become difficult to comprehend, navigate and edit once they exceed a certain size [Citrin, 1996].

The perceptual and the cognitive abilities of humans are limited. This is the reason why it is difficult to manage a large amount of information. These limitations are:

- **Perceptual limits:** the ability to discriminate the different diagram elements increase quadratically with diagram size [Patrignani, 2003]
- **Cognitive limits:** The working memory capacity can understand a limited number of diagram elements at a time (7 plus or minus 2 elements) [Miller, 1956]. If there are more elements, a state of cognitive overload ensues and the comprehension degrades rapidly.

To reduce the complexity, there are 2 techniques: modularisation and hierarchical structuring. Moreover, we will also explain why designers should avoid having duplicate elements on their diagrams.

Modularisation (decomposition)

The most effective way of reducing complexity of large systems is to divide them into smaller subsystems, sub-diagrams or modules: this is called **modularisation** [Baldwin, et al., 2000] or decomposition [Simon, 1996]. Each module, to be cognitively manageable, should contain 7 plus or minus 2 elements [Miller, 1956].

Modularisation requires the existence of specific semantic constructs: either a general "module" constructs (e.g., packages in UML class diagram) or recursively defined (decomposable) constructs in the notation itself (e.g., state charts in UML activity diagram) [Moody, 2009]. But for the clarity of the diagram, the designer has to build a general "map" to make a link between each sub-diagram (see section 4.2.5 principle of cognitive integration) and some graphical conventions have to be defined.

Hierarchical structuring

Repeated application of modularisation will result in a hierarchy of diagrams at different levels of abstraction, with the number of levels depending on the complexity of the underlying model [Moody, 2009]. Elements at the top of the diagram are decomposed into sub-elements that will detail the first one following the principle of recursive decomposition. Data Flow Diagram (DFDs) uses this technique as demonstrated figure 4-12, the top element "Order system" is decomposed at the next level into sub-elements called "Check Credit", "Fill Order" and "Generate Invoice" (level 0). Then the element 'Fill Order' is himself decomposed at the next level into 'Check inventory', 'Fill order' and 'Create backorder'.

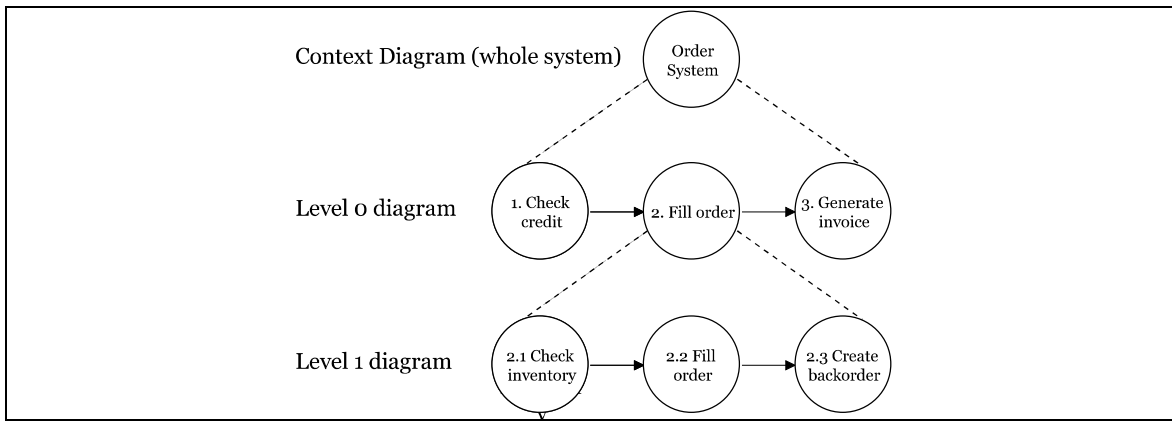


Figure 4-12 Hierarchical Structuring in DFDs [Moody]

Avoid duplicate elements

In practice, duplicate elements are often used to reduce line crossings on complex diagrams [Moody]. Some designers do it by themselves and invent methods to reduce the complexity of the diagram. For example, in figure 4-13, Axel van Lamsweerde uses dashed lines for duplicate elements (e.g., "Event" and "Agent") in its meta-model of KAOS realised according the UML class diagram notation. In the whole diagram of the meta-model, this construction simplifies the understanding of the reader by reducing the number of crossing lines. But once again, it means that there are as many solutions as designers.

Moreover, this technique tackles the symptom of the problem rather than the cause, it would be better to modularise the initial diagram.

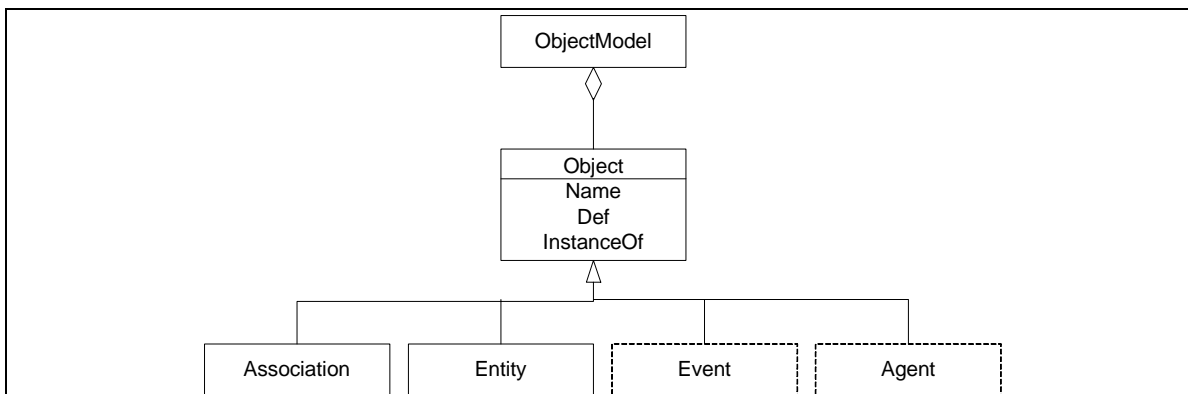


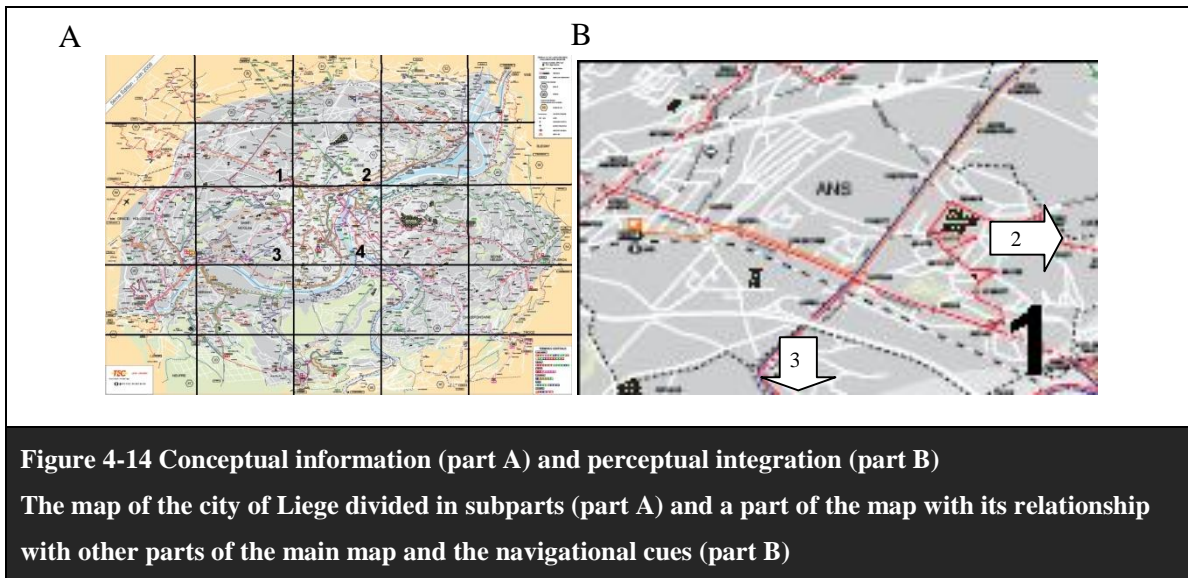
Figure 4-13 Duplicate elements are used to describe the meta-model on KAOS in [Lamsweerde, 2009]

4.2.5 Principle of Cognitive Integration

The principle of cognitive integration is used when a system is represented by multiple diagrams. This increases the cognitive load of the reader to integrate the information from the different diagrams. The reader needs to keep track of where he is. To solve this problem, Kim et al. [Hahn, et al., 1996] [Kim, et al., 2000] have developed a theory for multi-diagram representations called 'cognitive integration of diagram'. This theory includes 2 mechanisms to represent multi-diagram representations in a cognitively effective manner: the conceptual integration and the perceptual integration.

Conceptual integration mechanism enables the reader to integrate information distributed across different diagrams into a coherent mental representation of the system. However if this technique is largely used in cartography, it is not often used in software engineering languages.

This technique consists of a summary of the different sub-diagrams represented in a general one. Each sub-diagram contains contextual information showing its relationships to adjacent sub-diagrams, which is done by including all related elements from other sub-diagrams as foreign element [Moody, 2009]. Figure 4-14 part A shows a map of Liege [TECLiege, 2010] divided in rectangles. Some rectangles are identified by a number which references a more detailed map (figure 4-14 part B).



Conceptual information will allow top-down understanding as well as bottom-up understanding. The number inside each part of the main map helps the user to find the corresponding sub-map (top-down) and the number in each sub-maps (the number in the bottom right corner) helps the user can locate this piece of map on the main map (bottom up).

Perceptual integration provides perceptual cues to assist navigation and transitions between diagrams. It is composed of elements that help the user. These could be: clear labelling of diagrams, hierarchical numbering and locator maps [Lynch, 1960].

The map in figure 4-14 part B contains arrows that allow making links with other sub-maps. The arrow on the right refers to part 2 of the main map while the descending arrow refers to the part 3.

Both of these mechanisms can be applied equally to diagrams of the same type (homogeneous integration) and diagrams of different types (heterogeneous integration).

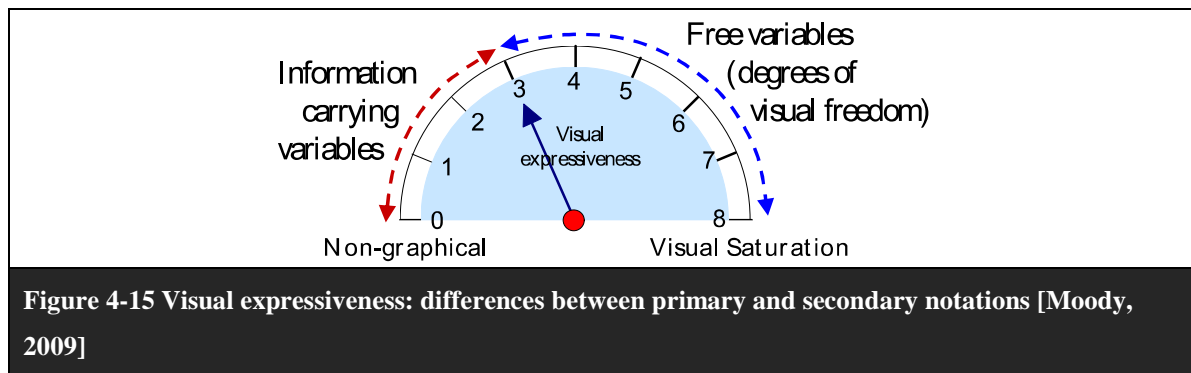
4.2.6 Principle of Visual Expressiveness

Visual expressiveness is the number of different visual variables used in a visual notation and the range of values used for each: this measure the utilisation of the graphic design space [Moody, 2009]. The goal of this principle is to use the different visual variables on the best way to improve cognitive effectiveness. Combination of these variables is used to exploit multiple, parallel channels of communication. Thanks to them, the reader can simplify the problem solving by separating the different parts of the problem in his mind [Cheng, 2004].

The concept of visual expressiveness divides the variables into 2 categories:

- **Information carrying variables:** this information has a specific meaning and is used to encode information. These variables define the primary notation.
- **Free variables:** these variables do not have a meaning defined formally. These variables can be used by the modeller to create the secondary notation.

Figure 4-15 defines the concept of visual expressiveness. As explained in the section 4.1, there are 8 visual variables (see figure 4-2): horizontal position, vertical position, shape, size, colour, value, orientation and texture. Then the visual expressiveness can vary from 0 to 8. If it is equal to 0, it means that no visual variable is used and there is no visual representation. If it is equal to 8, all variables are used and the draw is certainly saturated. Between these 2 extremities, the degree of visual freedom is equal to the number of free variables and varies inversely with visual effectiveness.



As explained in section 4.1, the choice of visual variables to use in a notation should not be arbitrary but should be based on the nature of the information to be represented [Bertin, 1983]. In other words, it means that some variables are more suitable to encode certain types of information. For example, colour can only be used for nominal data as it is not psychology ordered [Kosslyn, 1989].

Colour and spatial location are the most powerful visual variable because they are highly cognitively effective. The human visual system is highly sensitive to colour and can quickly and accurately distinguish between different colours [Mackinlay, 1986] [Winn, 1993] and spatial location can be used to encode all types of information [Moody, 2009]. But, even if there are effective, there are nearly not used in software engineering visual notations.

Another misuse of the visual variables is that software engineering notations use only a small part of the design visual space which is though unlimited. In example, modellers use mainly quadrilateral shapes: rectangle, parallelogram, diamond, and square. Nevertheless, studies have shown that curved shapes, 3 dimensional shapes and mimetic shapes are preferred by users [Bar, et al., 2006].

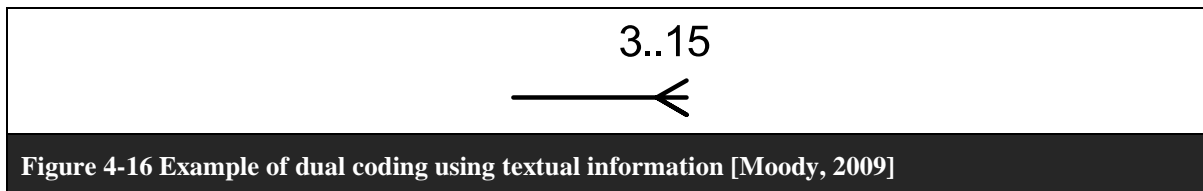
4.2.7 Principle of Dual Coding

Principle of dual coding goes against the principle of perceptual discriminability and visual expressiveness that advice not using textual information. However, text has its place in information encoding and it is not mutually exclusive with graphical information. Using both to encode the information is more effective than use them separately. When information is presented graphically and textually, representation of this information is encoded in separate verbal and visual systems within working memory and referential connections between the two are strengthened [Moody, 2009].

There are 2 ways to do dual coding: using annotations and adding textual information.

Annotations play the same role as comments in software programs. They improve the understanding of the diagram. When diagrams are annotated, they are more self-explaining and by consequence more readable for the user. If diagram documentations are placed on another document, it can cause problems of cognitive interpretation. In conclusion, textual information will facilitate the interpretation of the graphic.

Figure 4-16 represents a relationship link between 2 elements (in an imaginary visual representation). **Textual information** has been added to give more information about the cardinality of this relationship link. Without it, we can only say that this is a relation one-to-many. With it, we can say that the relation is one from 3 to 15. In this case, the encoded information gives more information.



4.2.8 Principle of Graphic Economy

Graphic complexity is defined by the number of different symbol types in a notation. It is the size of its visual vocabulary [Nordbotten, et al., 1999]. It is different from the diagrammatic complexity (see principle of Complexity Management), as it is applied at the syntax level and not at the diagram level [Moody, et al., 2010].

If the symbol number in the visual notation is huge, mnemonic and facilities have to be offered to novice users. A legend can be supplied but if it is frequently referenced, the user will spend a lot of time to understand the diagram.

The human ability to discriminate between perceptually distinct alternatives is around 6 categories [Miller, 1956]: this defines an effective upper limit for graphic complexity. In most of software engineering languages, this limit is exceeded. In visual notations, symbol numbers grow quickly because designers always add new symbols to increase the semantic expressiveness: each new construct requires a new symbol. Anyway the 2 most common languages (DFD and ER) follow the recommendation of Miller and this is maybe one of the reason why there are so popular.

Moody [Moody, 2009] describes 3 strategies to deal with graphic complexity:

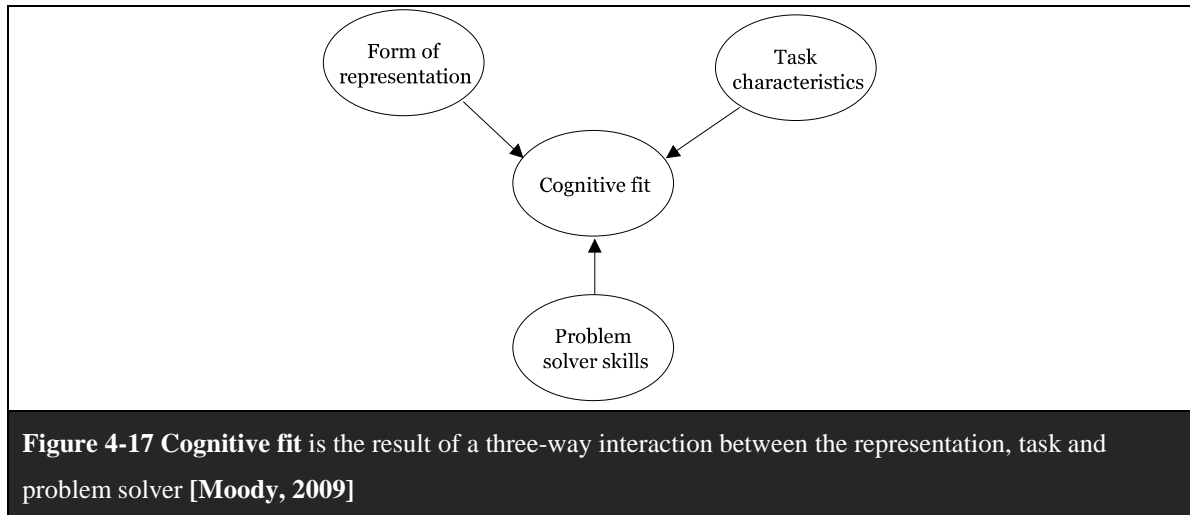
1. Reduce semantic complexity: usually the number of semantic constructs is proportional to the graphic complexity (following the principle of semiotic clarity). To reduce it, the notation can offer different views of the problem. These different views are represented by different diagrams. UML uses this technique, its meta-model is divided into different diagram types (e.g., class diagram, use case diagram, component diagram, sequence diagram and object diagram)

2. Introduce symbol deficit: graphic complexity can be reduced by introducing graphic deficit. It implies not to show all information on the graph and to write them textually in documentation. Diagrams are done to study high-level abstractions of problems rather than fully detailed specifications. Designers have to find the right balance between textual, graphical encoding and information that can be specified off diagram.

3. Increase visual expressiveness: increasing human discrimination can be done by increasing the number of visual variables that are used in the notation. Indeed the limit of 6 categories only applies if a single visual variable is used (which is true for most software engineering notations, which use shape as the sole information-carrying variable) [Moody, 2009].

4.2.9 Principle of Cognitive Fit

Most requirements engineering notations use the same visual notation for all readers and all usages. The cognitive fit theory, which is widely accepted in the computer scientist domain and used in many steps of software engineering, suggests that one single visual representation for all purpose is inappropriate [Shaft, et al., 2006] [Vessey, 1991] [Vessey, et al., 1992]. According to this theory, the visual notation of a language should be adapted to the form of representation, tasks characteristics and problem solver skills for which it is intended. Figure 4-17 represent the principle of cognitive fit as the middle between these 3 points.

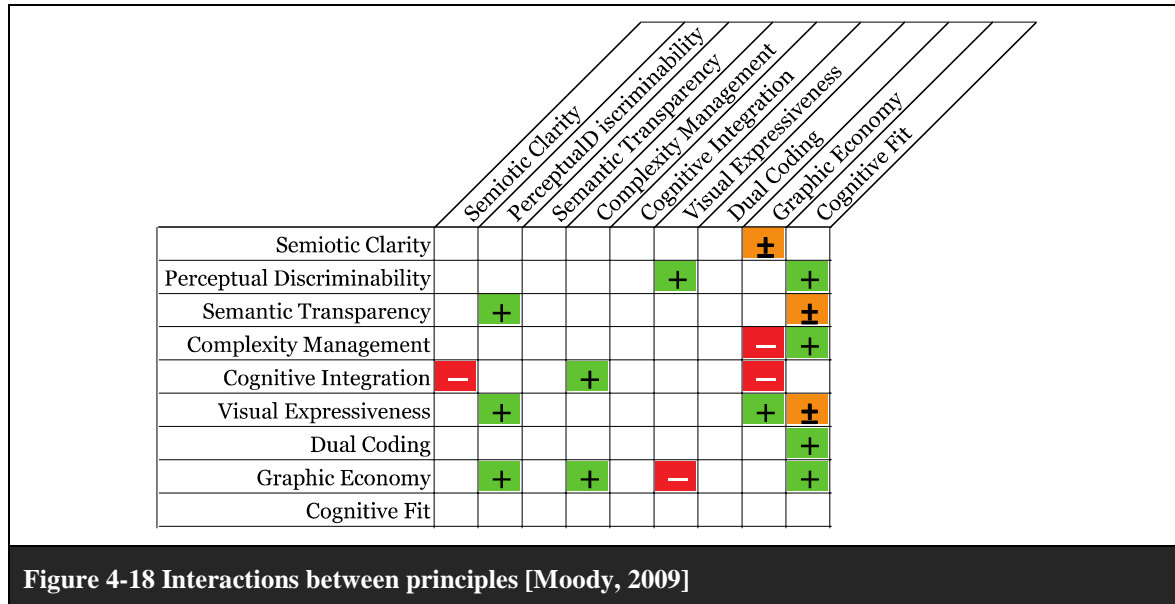


Usually, in most technical domains, graphical designs are used by experts. They know the meaning of each symbol and they frequently read them. But in software engineering, visual notations are used by both technical experts and business experts. Designers could use the "small common denominator" between them to build visual notations that could increase the cognitive effectiveness but it is rejected by the cognitive load theory. Visual notations for the novices should use clearly distinguishable symbols (perceptual discriminability), mnemonic conventions (perceptual immediacy), clarifying text (dual coding) and simplified visual vocabularies (graphic economy) [Moody, 2009]. However if notations are optimised for novices, their effectiveness may be reduced for experts and vice versa.

Software engineering visual notations are usually monolinguist; it means that the same symbols are used for all users. Differences can be found in only 2 visual languages: ORM [Halpin, 2005] and Oracle data Modelling [Barker, 1990]. They have specific designs for end users that allow them to clearly explain some concepts.

4.3 Interaction among Principles

Figure 4-18 summarises the interactions among the visual notation design principles. These interactions are not symmetrical. Visual notation designers should exploit maximally interactions that will have synergies between them and find the best compromise between principles that interact against each other.



- Semiotic clarity could have positive or negative effects on graphic economy. Symbol excess and redundancy increase the number of symbols (decrease of the graphic economy but increase graphic complexity) while overload and deficit reduce it (increase of the graphic economy but decrease graphic complexity) [Moody, 2009].

- Graphic economy reinforces the cognitive fit, the perceptual discriminability and the complexity management but it decrease the visual expressiveness if the number of symbols is decreased.

- Perceptual discriminability increases the cognitive fit and the visual expressiveness because it uses more visual variables.

- Dual coding reinforces cognitive fit.

- Visual expressiveness decreases graphic economy and improves the perceptual discriminability. It is due to the fact that this principle suggests using more symbols, many visual variables and a greater range of values to draw the diagrams.

- Complexity management improve cognitive fit but increase the number of graphical symbol, thus it disadvantages the graphic economy.

- Cognitive integration improves the complexity management but increases graphic complexity because symbols or diagram types are added to create links between them. Finally it creates excess symbol anomalies which are against the principle of semiotic clarity.

- Improving the principle of semantic transparency will also improve the principle of perceptual discriminability but it will advantage and disadvantage the principle of cognitive fit.

Chapter 5 The KAOS Language

5.1 Introduction

In this chapter, we will explain the different semantic constructs of the KAOS language and their visual notations. The described version of KAOS is this one presented in [Lamsweerde, 2009]. But we have to keep in mind that there could be other versions, in particular of the visual notation (e.g., Objectiver [Objectiver, 2007], a requirements engineering software that implements some parts of the KAOS visual notation). Figures in this chapter are drawn with a diagram creation software (e.g., Dia [Dia, 2011] or MS Visio) because we did not find any software that follows exactly the visual notation described in [Lamsweerde, 2009].

KAOS belongs to the goal-oriented requirements engineering methods described in Chapter 3. Like all of these methods, it is based on goals expressed by stakeholders.

KAOS stands for *Knowledge Acquisition in autOmedated Specification or Keep All Objects Satisfied* [KAOS, 2010]. This methodology helps analysts to build requirements models based on documents described by stakeholders or based on interviews given by stakeholders. After elaborating the requirements model, analysts will be able to produce requirements documents from KAOS models. KAOS offers many views of a system in term of WHO and WHY dimensions. The HOW dimension is also approached. This language uses semi-formal and formal expressions depending on the needs: semi-formal for modelling and structuring goals and formal when reasoning has to be more accurate.

KAOS is comprised of 5 models that are linked to each other: goal model, agent model, operation model, object model and behaviour model. These 5 models cover the whole system and not only the software part of it. They are described in the following sections. However, we will not describe in details the 2 last ones borrowed from the UML language.

The figures used in this chapter are based on a running example that describes an online bookstore (the example is more detailed in Chapter 9).

5.2 Goal Model

The goal model covers the WHY-dimension of a system. It contains goals that are prescriptive statements and can be functional or non-functional (see Chapter 3). Each goal will contribute to fulfil requirements expressed by the stakeholders and help to define requirements for the software-to-be. One of the advantages of this model is it will help stakeholders to choose between different alternatives, they can also see possible conflicts among goals. The second advantage is to show the whole system and not only the part of the system we need to build (i.e., the software-to-be).

The goal model allows seeing inter-model relationships such as responsibility links between goals and system agent, obstruction links between goals and obstacles, reference links from goals to conceptual objects or operationalisation links between goals and system operations.

Goals can be high-level or lower level. High level goals refer to strategic objectives that the system should fulfil, lower-level goals represent technical prescriptions.

A goal is represented by a parallelogram. Inside it, there is the goal's name, possibly prefixed by its type.

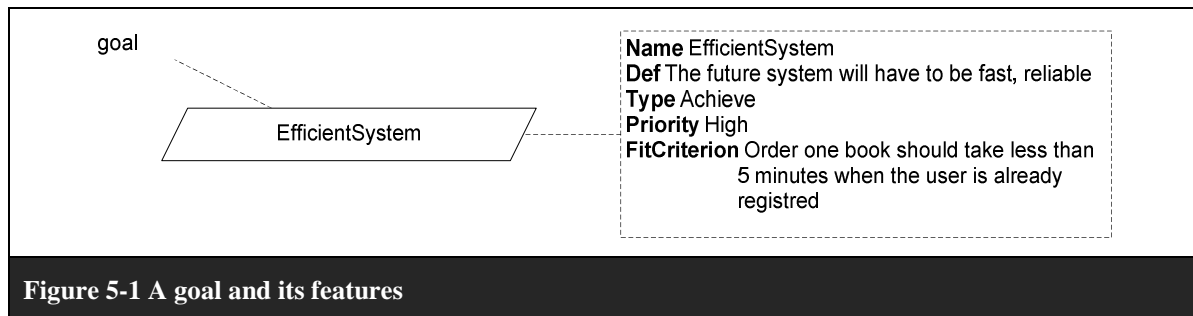
A goal has always a name and a specification. These 2 characteristics are mandatory:

- **Name:** unique in the whole system model.
- **Specification (Spec)²:** explains in natural language the behaviour in terms of phenomena that are monitorable and controllable in the system.

There are also optional features like:

- **Type:** indicates which class of prescribed or preferred behaviour the goal refers to. There are 3 goal types :
 - **Maintain:** behavioural goal with a specification pattern: **[if CurrentCondition then] always GoodCondition.**
 - **Achieve:** behavioural goal with a specification pattern: **[if CurrentCondition then] sooner-or-later TargetCondition.**
 - **Soft goal:** evaluating alternative.
- **Category:** indicates the taxonomic category of the goal.
- **Source:** gives the origin(s) of the goal (e.g., a stakeholder).
- **Priority:** allows the user to prioritise the goal for comparison with competing goals.
- **Fit criterion (FitCriterion):** annotates a soft goal, allows an evaluation of the alternative options against it.
- **Formal specification (FormalSpec):** annotates a behavioural goal to formalize its informal specification.

Figure 5-1 represents a goal called 'EfficientSystem' and the features of this goal such as the definition, the type, the priority and the fit criterion.



² The word in parenthesis represents the short name of the characteristics. This short name is used in diagram, in annotations and in the meta-model in Chapter 6.

Goal refinement

The goal model is represented by a **refinement graph** showing how higher-level goals are refined into lower-level goals and conversely. Goals can be refined on 2 ways: either by an AND-refinement or by an OR-refinement. The graph is thus an AND/OR graph, and can also be called **goal diagram**.

An **AND-refinement** link relates a goal to a set of **sub-goals**. Each of them contributes to satisfy the parent goal and all sub-goals are to be satisfied for satisfying the parent goal. This refinement link is represented by a small circle connecting the main goal and its sub-goals. The main goal is the target of the link.

An AND-refinement link is **complete** if the goal set of the AND-refinement is sufficient to fulfil the parent goal. In other words, a goal is completely refined if and only if all sub-goals of the AND-refinement are sufficient to satisfy the main goal. To represent it, the small circle of the relationship is coloured in black.

In an AND-refinement of a goal G, a sub-goal may itself be AND-refined, and so on recursively. G is thus the parent of a tree, the leaves of the tree are goals that cannot be refined. They represent software requirements or environment assumptions. Assumptions are generally prescriptive statements that are satisfied by the environment and are formulated in terms of environmental phenomena.

All leaf goals are needed to fulfil the parent goal. **Leaf goals** can be differentiated from the other goals by their bold border. They have to be under the responsibility of a single software agent (as requirements) or a single environment agent (as expectations or assumptions).

Figure 5-2 depicts a goal diagram where the goal 'SystemMeetsMaximumFunctionalRequirements' is refined into 3 sub-goals: 'SecurePayment', 'SellBooksAtBestPrice' and 'HighAvailability'. The goal 'SellBooksAtBestPrice' is itself refined into 3 other sub-goals: 'SellALotOfBooks', 'GoodInventoryManagement' and 'RobustOnlineStore'. These 3 sub-goals are sufficient to fulfil their parent goal. In this case, the refinement is said 'complete' that is graphically represented by the black circle on the refinement relationship link.

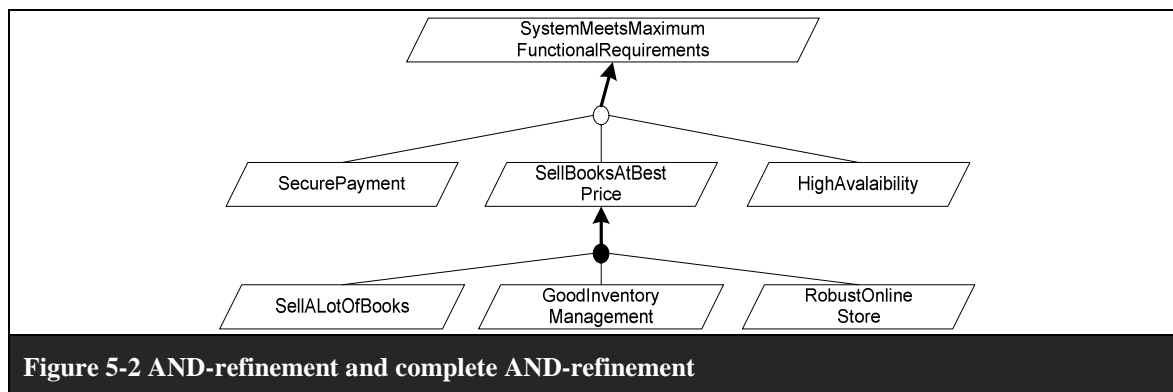


Figure 5-2 AND-refinement and complete AND-refinement

A goal can be satisfied in different ways, i.e., by different groups of AND-refined goals. Each AND-refinement is called an **alternative refinement**. A group of AND-refinements attached to the same parent is called an **OR-refinement**.

A refinement has the following features that can be documented by annotations:

- **Name:** gives a unique identifier to a refinement to avoid confusion.
- **System reference (SysRef):** indicates which alternative is chosen for a version of the system.
- **Tactic:** explains how a refinement was found.

Figure 5-3 depicts an example of an OR-refinement. The goal 'EasyToUseSystem' can be achieved if the goal 'HelpUserByPhone' is fulfilled or if the goal 'HelpUserByMail' is fulfilled. One of them is sufficient to fulfil the parent goal. In [Lamsweerde, 2009], there is no particular symbol to express that a goal is OR-refined.

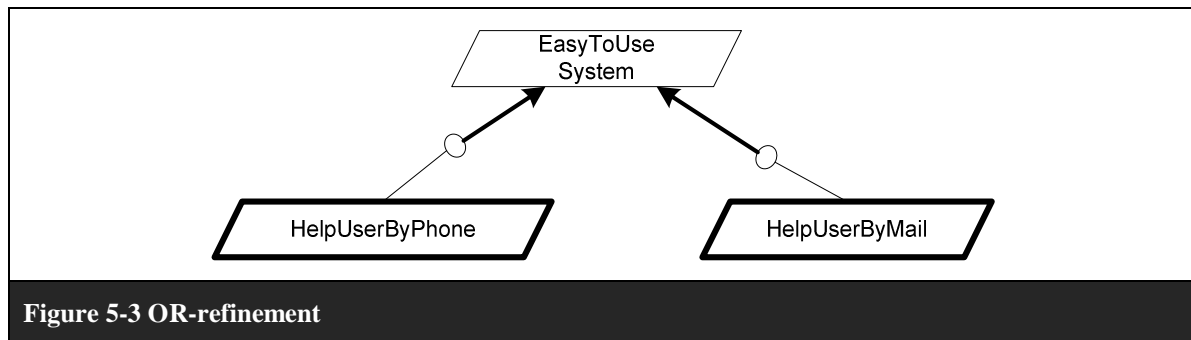


Figure 5-3 OR-refinement

Conflicts among goals

Sub-goals can contribute positively to a goal but, at the same time, they can also be opposed to another goal. Another possibility of conflict is when we have a **boundary condition**; it occurs when statements are not satisfied together under some condition. It happens in particular combination of circumstances that makes the statements strongly conflicting when it becomes true. It implies that under some boundary conditions, some goals become logically inconsistent in the considered domain. It often happens when the goals come from different sources.

Graphically a conflict between goals is represented by a link with a 'flash' icon on it.

Figure 5-4 shows that the goal 'RobustOnlineStore' is used to fulfil the goal 'SellBooksAtBestPrice' but it is in conflict with the goal 'CheapSystem' which is necessary to fulfil the parent goal 'SystemMeetsMaximumNonFunctionalRequirements'.

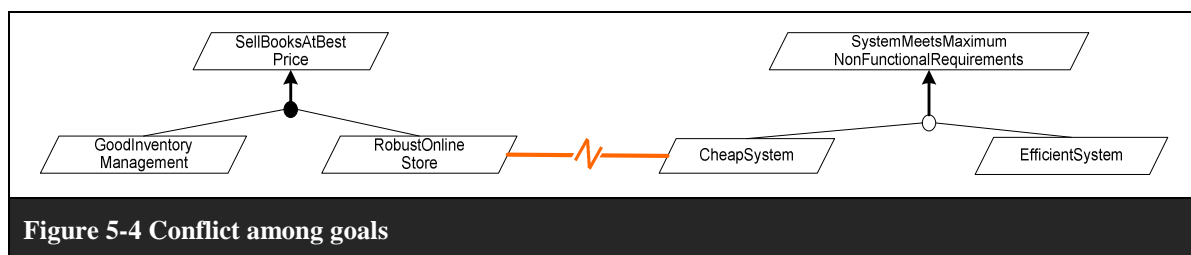


Figure 5-4 Conflict among goals

Obstacles

An ideal system should behave as expected. But this is not the case in the real world because there are always a lot of risks and unexpected events that can lead to loose the satisfaction of some objectives. These risks and unexpected events are called **obstacles**, they have to be studied in the goal model and thwarted as much as possible. The first step is to identify them and, in a second step, we have to find goals to prevent that they happen.

An obstacle will usually avoid that an assertion will be satisfied but it can also avoid that a domain property or a hypothesis will be satisfied.

Graphically, an obstacle is represented by a 'reverse' parallelogram (a left-oriented parallelogram) labelled by its name.

An obstacle always has a name and a specification (these characteristics are mandatory):

- **Name:** unique in the whole system model.
- **Specification (Spec):** explains in natural language the behaviour in terms of phenomena that are monitorable and controllable in the system.

The boundary condition from where the obstacle occurs has optional features like:

- **Category:** indicates the taxonomic category of the obstacle.
- **Likelihood:** estimates how likely the situation captured by the obstacle condition is. This estimation is used for risk assessment.
- **Criticality:** estimates how severe are the consequences of the situation captured by the obstacle condition are (e.g., catastrophic, severe, moderate, low).
- **Formal specification (FormalSpec):** annotates an obstacle to formalize its formal specification.

Figure 5-5 represents the obstacle 'BookNotDeliveredQuickly' and its characteristics: the name, the definition, the category, the likelihood and the criticality.

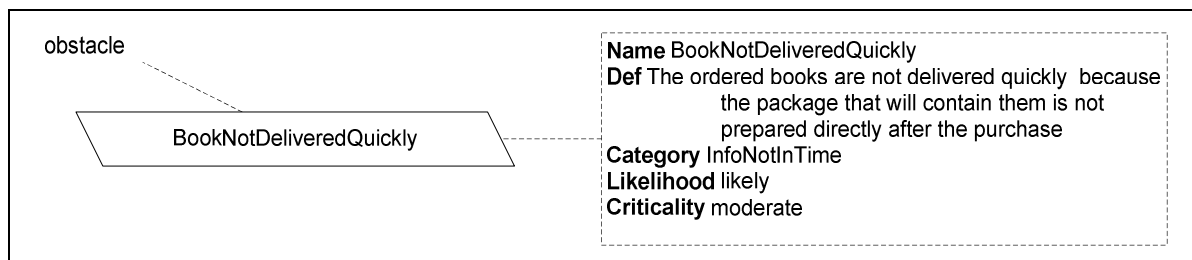


Figure 5-5 An obstacle and its features

Like goals, obstacles are organised in a graph that are called the **obstacle diagram**. The root of this tree is the assertion not G (where G is the goal that we want to study); it is the root obstacle; it is linked to the goal G by an obstruction link. The latter can be refined in an OR-refinement tree or in an AND-refinement tree. The leaves of the tree are elementary obstacles whose satisfiability, likelihood and resolution can be determined easily.

Figure 5-6 shows that the goal 'QuickDelivery' could be prevented by the obstacle 'BookNotDeliveredQuickly'.

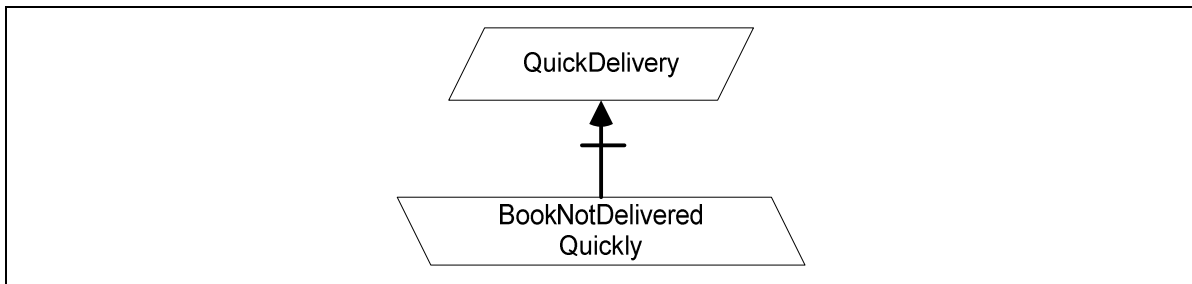


Figure 5-6 An obstacle and one of its possible obstruction

There are many categories of obstacles that obstruct some categories of goals as described in the table 5-1.

Table 5-1 Obstacle types and the obstructed goal types [Lamsweerde, 2009]

Obstacle	Goal obstructed
hazard	safety
threat	security
disclosure	confidentiality
corruption	integrity
denial-of-service	availability
dissatisfaction	satisfaction
nonSatisfaction	
partialSatisfaction	
tooLateSatisfaction	
misInformation	information
nonInformation	
wrongInformation	
tooLateInformation	
unusability	usability

Obstacle analysis and goal model elaboration

Figure 5-7 explains how to elaborate a goal model and analyse the obstacles. Firstly, the goal model has to be drawn. In this model, goals have to be refined at most. Then these leaf goals are preferred to find the obstacle following the iteration Identify-Access-Control cycle.

- **Identify:** select goals and identify obstacles for each of them.
- **Assess:** evaluate the probability and criticality of each identified obstacle.
- **Control:** resolve each obstacle according to its probability and its criticality to produce new goals as countermeasures in the goal model.

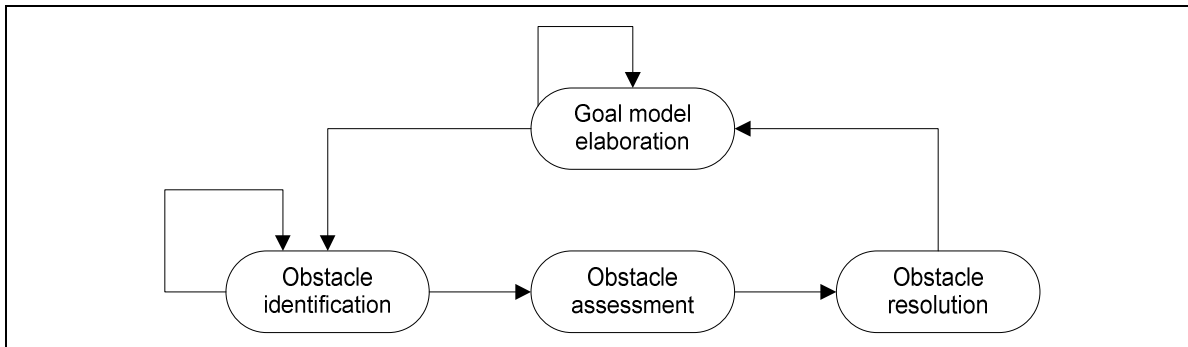


Figure 5-7 Obstacle analysis and goal model elaboration [Lamsweerde, 2009]

5.3 Agent Model

The agent model covers the WHO-dimension explained in the Chapter 2. It allows replying to the question: "Who is doing what and why?". This model shows how the agents of the system will have to fulfil the different sub-goals of the goal model. Then, the responsibilities and the workload of every agent (human or device) of the system can be deduced and used for load analysis.

System agents

System agents are responsible to satisfy leaf goals of the goal model. They have to do it according to their capabilities. The capabilities of an agent are defined in terms of object attributes and associations of the object model that the agent can monitor or control. As any object of the system, agents are defined by their name and their specification.

On one hand, agents have the responsibility of fulfilling a leaf goal but the responsibility can also be divided into several agents. And like goals and obstacles, agents can be decomposed into finer-grained ones. On the other hand, agents can also express their wishes for some goals.

Agents can be divided into 4 categories:

- **New software agent:** it has to be developed as software controllers, information managers or web services.
- **Existing software agent:** foreign or legacy component with which the new software will have to operate.
- **Devices:** sensors, measurement instruments...
- **Human agent:** it plays a specific role such as organisational units, end-user.

The system scope could also include malevolent agents, in other word, agents that will try to break the system goals to satisfy their own malicious goals (e.g., hackers that will try to steal private information).

Graphically, system agents are represented by hexagons. To identify environment agent, a 'sticky man' icon is added inside it. These representations can be seen on figure 5-8 where 'OrderPickers' is an environment agent and 'BookOrderSoft' a system agent –an existing software used to order books at the wholesaler.

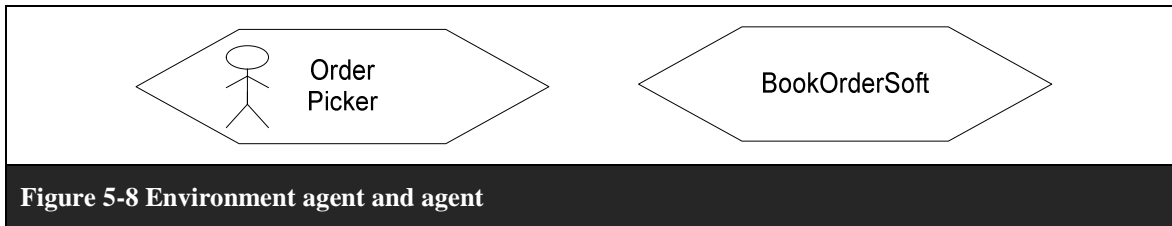


Figure 5-8 Environment agent and agent

To fulfil goals, agents will perform some operations (section 4.4 Operation Model). It is represented by a relationship between the agent and the operation. Graphically represented by a line with an adorned circle in the middle. The same graphical representation is used to show the responsibility of the agent fulfilling a goal that he has to satisfy.

Figure 5-9 expresses that the goal 'PreparePackageQuickly' is under the responsibility of the environment agent 'OrderPickers'. This agent will have to perform the operation 'PreparePackage'.

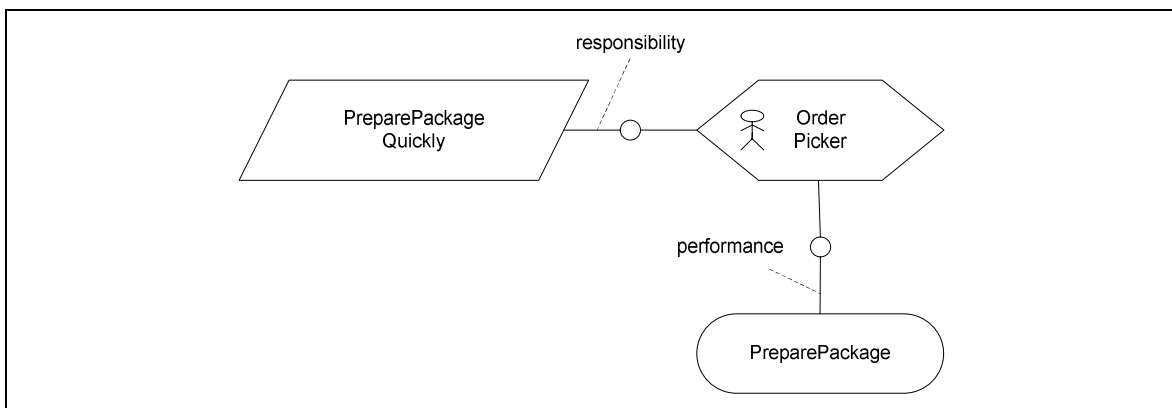


Figure 5-9 Agent responsibility and performance

An agent has always a name and a specification (these 2 characteristics are mandatory):

- **Name:** unique in the whole system model.
- **Specification (Spec):** explains in natural language what the agent has to do.

There are also optional features like:

- **Load:** work load supported by this agent.
- **Category:** indicates the category of the agent (new software agent, existing software agent, devices, human agent).

To fulfil a leaf goal, multiple agents can be envisaged even if at the end of the process the goal will have to be fulfilled by only one of them. Each alternative link is called an **assignment**. The same kind of design is used to represent it.

Agent capabilities

The capabilities of an agent are determined in terms of monitoring links and control links to objects in the object model. Agent can control or monitor either objects or associations.

- "An agent **monitors** an attribute of an object if its instances can get the values of this attribute from object instances. An agent monitors an association if its instances can evaluate whether this association holds between object instances." [Lamsweerde, 2009]
- "An agent **controls** an attribute of an object if its instances can set values for this attribute on object instances. The agent controls an association if its instances can create or delete association instances." [Lamsweerde, 2009]

When agents are instantiated, the variables that are monitored or controlled are also instantiated as state variables.

Figure 5-10 shows the graphical representation of monitored and controlled attributes of an object. The agent Ag monitors the attribute Att1 of the object Ob1. It is represented by an ingoing arrow from the object to the agent. The name of the attribute monitored is adorned to the arrow. On the same figure the agent Ag controls the attribute Att2 of the object Ob2. It is represented by an outgoing arrow from the agent to the object. The name of the attribute controlled is adorned to the arrow.

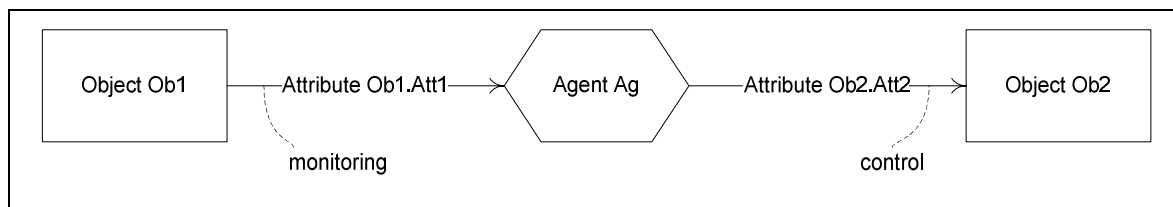


Figure 5-10 Agent capabilities [Lamsweerde, 2009]

Agent dependencies

"An agent A is said "to depend on" an agent B for a goal G, under the responsibility of B, if B's failure to get G satisfied can result in A's failure to get one of its assigned goals satisfied" [Lamsweerde, 2009]. Figure 5-11 shows that agent Ag2 depends on the responsibility of agent Ag1 to fulfil the goal G. This dependency relationship is depicted as an (oriented) link which is adorned with a 'D' character.

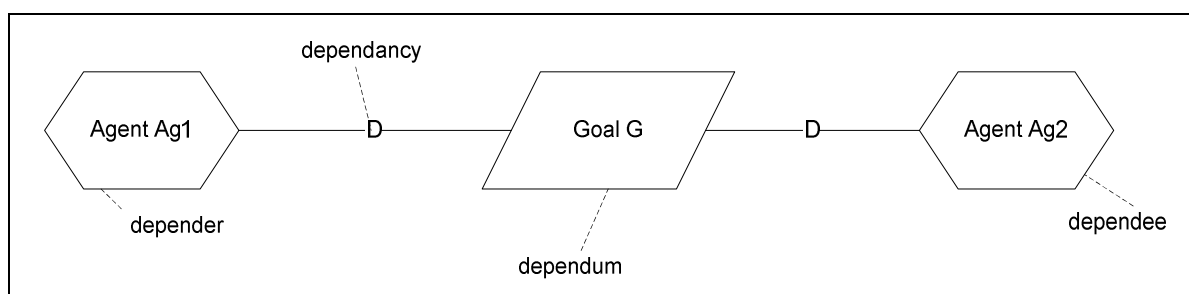


Figure 5-11 Agent dependencies [Lamsweerde, 2009]

5.4 Operation Model

The operation model covers the WHAT-dimension of the requirements engineering (see Chapter 2). It describes the services that the system should offer to fulfil the goals. An operation model represents the system operation in terms of their individual features and their links to the goal, objects and agent models. From this model, we can derive elements of the software architecture like external specifications of functional components, black-box test data and we can also define development work units.

An **operation** is performed by an agent; it executes work on objects (defined in the object model). It can create objects, trigger object state transitions and activate other operations (by sending an event).

Operations have a set of input variables and a set of output variables that define its signature. An operation is performed when it receives input variables corresponding to its running conditions. Thanks to signatures, links between the operation model and the object model are introduced.

There are different kinds of operations:

- **Software operations** (or services or functional features): realised by software agent. Their specifications have to be sent to the software engineer.
- **Environment operations**: realised by human agents, devices or existing software agents in the environment of the software-to-be.

An operation has the following features:

- **Name**: unique in the whole system model.
- **Specification (Spec)**: explains in natural language which goal the operation has to fulfil.
- **Category**: indicates the category of the operation (software operation or environment operation), this information is optional.
- **Signature**: specifies the input/output variables of the relation. A variable is declared by the name of the object it belongs followed by its name (e.g., Customer.DeliveryAddress). Objects can be an entity, an association, an agent or an event. Signature can be described graphically (figure 5-12 part A) or textually (figure 5-12 part B).

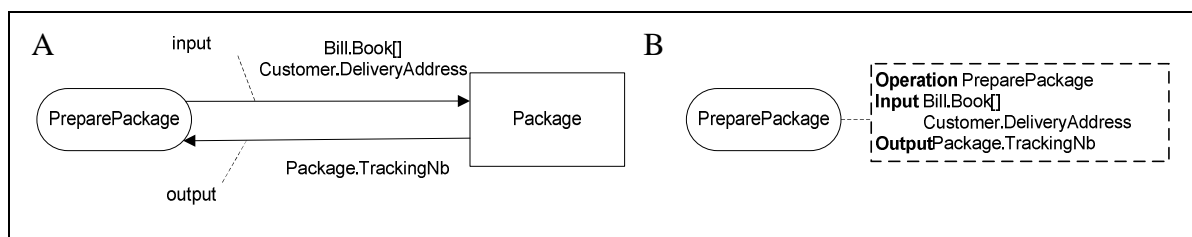


Figure 5-12 Operation signature

(part A shows graphical notation, part B shows textual notation)

- **Domain:** specifies the domain pre-conditions (DomPre) and post-conditions (DomPost) in which the operation will be realised. These conditions describe the set of state transitions defined by applications for the operation (domain pre-conditions define the class of input states when the operation is applied, domain post-conditions define the class of output states when the operation has been executed). Domain conditions are descriptive as they capture what the operation intrinsically means in the domain, they can be a requirement or not. If it is a requirement, it implies that the condition is a necessary condition to execute the operation. Figure 5-13 describes the domain conditions for the operation 'PreparePackage'. The precondition domain is: “the package has to be paid before being prepared” and the postcondition domain is “the package is ready to be delivery”.

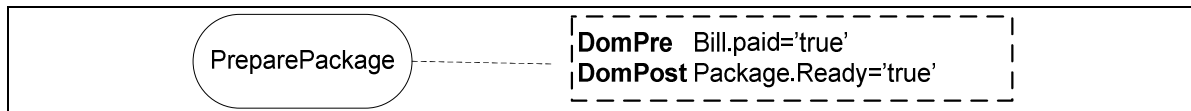


Figure 5-13 Domain pre-conditions of an operation

Operationalisation is the action that consists of mapping an operation and a leaf goal under the responsibility of a single agent. This operation has to be performed under required domain conditions.

The complete specification of the operation is obtained by combining all its required conditions on the input states with its domain pre-condition, and all its required conditions on the output states with its domain post-condition.

There are 3 types of required conditions:

- **Required pre-condition (ReqPre):** necessary condition on the operation's input states for the satisfaction of this goal by any application of the operation. It captures a permission.
- **Required trigger condition (ReqTrig):** sufficient condition on the operation's input states for the satisfaction of this goal by any application of the operation. It captures an obligation.
- **Required post-condition (ReqPost):** condition on the operation's output states for the satisfaction of this goal by any application of the operation.

Figure 5-14 depicts that operation 'PreparePackage' has a pre-requirement before being operationalised: the bill of the purchase has to be paid.

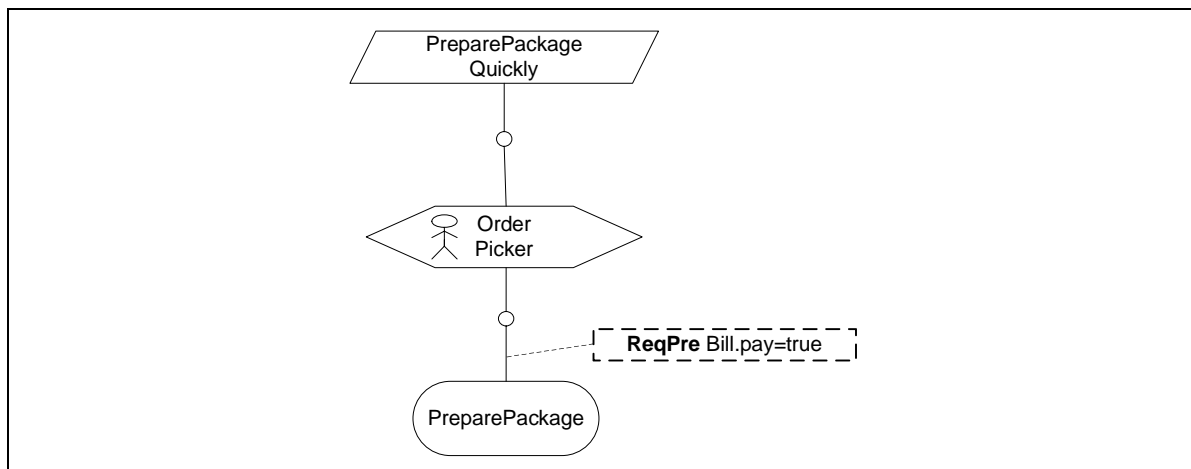


Figure 5-14 Required conditions annotating operationalisations

5.5 Object Model

The object model offers a structural view of the system. In this view, the conceptual **objects** manipulated in the system-as-is or system-to-be are defined, characterised through individual features and inter-related with each other through links. Objects belong to a domain that we have to describe. This domain can have domain invariants and domain hypothesis.

When developers start to develop the software, this model could be used to generate a database schema (if needed) and elaborate the software architecture. When the system is ready to run, objects are instantiated into features that belong to the 'real' world.

This diagram is represented by an entity-relationship diagram using the UML class diagram representation.

All object instances evolve individually from state to state depending on changes in the values of their state variables. Object instances can be:

- **Entity:** autonomous and passive object, its instance exists in the system independently of other objects.
- **Association:** conceptual object depends on other objects that it links, each link plays a specific role in the association. An association can be reflexive if the same object appears at multiple positions under different roles.
- **Agent:** autonomous and active object, it has individual behaviours, can control and monitor other objects (thanks to its associations and attributes).
- **Event:** instantaneous object, can be either internal or external.

The domain concepts will be represented by conceptual objects. These objects are sets of instances of a domain-specific concept. These instances are distinctly identifiable, can be counted in any system state, share similar features and may differ from each other in their individual states and state transitions.

Each object is characterised by:

- **Name:** unique in the whole system model.
- **Type:** declares the type of the object (entity, association, agent or event).
- **Specification (Spec):** explains in natural language the behaviour in terms of phenomena that are monitorable and controllable in the system.
- **Has:** defines the object attributes and their meanings. The attributes of an object has a range of value, a multiplicity (by default is a one-to-one multiplicity) and can be rigid if its value does not change during in the entire system behaviours.
- **Domain invariant (DomInvar):** lists the known domain properties about the object as invariant holding in any object state.
- **Domain initialisation (DomInit):** gives the initial value of the attributes and associations for any instance of the object. This property can be written in a natural language or in a formal language.
- **Issue:** uses to remind a question about the object. This property can be written in a natural language or in a formal language.

Figure 5-15 depicts the object 'Customer', its attributes and the characteristics.

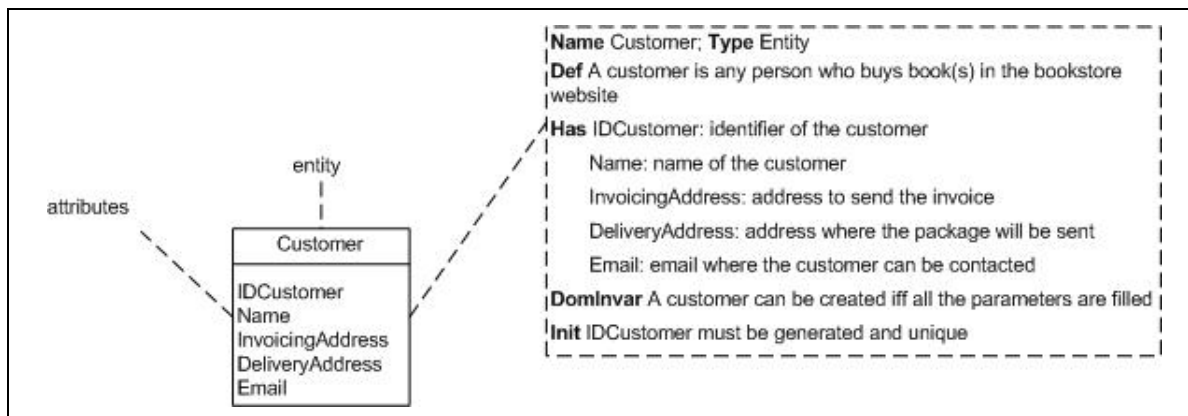


Figure 5-15 An object and its features

There are 2 particular associations between objects: the specialisation and the aggregation. Specialisation occurs when objects can be specialised into other objects that have common features with their parent and their own features. Conversely, if we notice that many objects share the same features, we should generalise it. An aggregation association involves that an object is composed of other objects.

This model follows the UML 2.0 syntax of the class diagram representation. It uses concepts as: entities, attributes, binary association, ternary association, n-ary association, specialisation, composition, aggregation and OR-associations.

An example of object diagram of the running example can be seen in figure 9-6.

5.6 Behaviour Model

A behaviour model allows seeing the required behaviours of system agents in terms of temporal sequences of state transitions for the variables that the agents monitor and control. These transitions appear when operations are executed or when external events happen. The global behaviour of the system is obtained as a parallel composition of agent behaviours.

The behaviour model is composed of scenarios (specific instance behaviour) and state machines (class behaviour). It can either describe the system-as-is –in this case it will be use for analysing a running system– or the system-to-be to define the behaviour of the future system.

A **scenario** is a temporal sequence of interaction events executed by different agent instances. It shows the interactions with the different objects and the chaining between them. It can be positive or negative. A positive scenario shows how to satisfy a desired goal, a negative one explains a possible way of satisfying an implicit obstacle of the goal. A scenario can be split up in sub-scenarios or episodes.

An example of scenario of the running example can be seen in figure 9-7.

State machines are complement to the scenarios: they make state information explicit, show the behaviour of any agent or entity instances (not a specific one) and all possible sequences of state transition.

An example of state machine of the running example can be seen in figure 9-8.

Part II

Contribution

Chapter 6 The KAOS meta-model

After having explained the different semantic constructs of the KAOS language, we are able to draw its meta-model. It will show the different semantic constructs as classes and the relationships that exist between them. One of the usages of a meta-model is to do the semiotic analysis. This analysis consists of verifying the existence of two-ways mapping between semantic constructs and the visual notation of the language or the existence of anomalies (e.g., symbol excess, symbol redundancy, symbol overload and symbol deficit).

The meta-model presented in figure 6-1 is represented as an UML class diagram and it is based on the meta-model described in [Lamsweerde, 2009]. We have completed it with information such as the multiplicity of the associations, the type of generalisation/specialisation such as complete - incomplete and overlapping - disjoint. This information is important for the user and helps him/her to understand the different possibilities offered by the language. To increase the readability of the text, terms that can be found in the meta-model are written in *italic*.

As the system model expressed in the KAOS language is made up of 5 views, the meta-model is decomposed in an aggregation of 5 fragments. These ones model the meta-classes and the meta-relationships used to represent these 5 views: goal model, agent model, operation model, object model, and behaviour model. These views are strongly intertwined and complete each other. These 5 different fragments can be easily discriminable by their different coloured background in the general meta-model of figure 6-1. The use of these colours is summarised in the table 6-1.

Table 6-1 Colours used to differentiate the different models in the meta-model

Model name	Colour
Goal model	Green
Agent model	Orange
Operation model	Violet
Object model	Blue
Behaviour model	Yellow

The overall meta-model provides a common framework within which all views of a system model can be structurally defined and inter-related.

6.1 Goal Model

In the *GoalModel* meta-concept, the *Goal* meta-class is obviously the central element. It has many meta-attributes: *Name* and *Specification* that are mandatory; *Category*, *Priority*, *Source* that are optional. This is a concrete meta-class, which means that it can be instantiated. Goal can be specialised into 2 different types: *SoftGoal* and *BehaviouralGoal*. The latest can be divided into 3 other types: *Achieve* goal, *Maintain* and *Avoid* goal. The *SoftGoal* meta-class has one meta-attribute: *FitCriterion* which is optional, and the *BehaviouralGoal* meta-class that also has one meta-attribute: *FormalSpec* which is also optional.

Table 6-2 Attributes of the goal meta-class and its subclasses

Attributes	Description
Name	unique in the whole system model
Specification	explains in natural language the behaviour in terms of phenomena that are monitorable and controllable in the system
Category	indicates the taxonomic category of the goal
Priority	allows the user to prioritise the goal for comparison with competing goals
Source	gives the origin(s) of the goal
FitCriterion	annotates a soft goal, allows an evaluation of the alternative options against it
FormalSpec	uses to annotate a behavioural goal to formalize its informal specification

When goals are maximally refined, they become *LeafGoal*, which is an abstract meta-class (that cannot be instantiated). This abstract meta-class is specialised into 2 concrete meta-classes: *Requirement* and *Expectation*.

Each specialisation of the *Goal* meta-concept inherits the meta-attributes and meta-relationships of its parent.

As a goal may be OR-refined, the meta-concept of *Refinement* is introduced together with the *OR-ref* meta-relationship between *Goal* and *Refinement*. Refinements can also consist of multiple conjoined goals. To represent this link, we introduce the *AND-ref* meta-relationship. This meta-relationship is represented by a UML OR-association because the meta-class *DomDescript* (domain description) may be involved in the refinement as well. The meta-class *DomDescript* contains properties of the system that cannot be changed.

The multiplicity of this relationship expresses that a goal may be OR-refined into 0 up to an infinite number of refinements, whereas a refinement refines one and only one goal. Regarding AND-refinement, a refinement must be AND-refined into one goal at least while possibly involving 0 to an infinite number of domain description instances.

The *refinement* meta-class has some meta-attributes: *Name* which is mandatory, *Status*, *SysRef* and *Tactic* that are optional.

Table 6-3 Attributes of the refinement meta-class

Attributes	Description
Name	gives a unique identifier to a refinement to avoid confusion
Status	indicates if the refinement is complete or not
SysRef	indicates which alternative is chosen for a version of the system
Tactic	explains how a refinement was found

Goals can be in conflict with each other if these goals can not be achieved together. They may also be obstructed by *Obstacles* or by domain descriptions. In this case, a ternary relationship called *Divergence* occurs. It means that under some *BoundaryCondition* these goals become divergent in the considered domain *DomDescription*. The multiplicity of the relationship *ObstructedBy* determines the type of divergence: when an obstacle obstruction involves a single goal, then the multiplicity 1..* attached to the role *ObstructedBy* is reduced to 1; in case of conflicting goals, this multiplicity is reduced to 2..*, as a boundary condition for conflict that involves at least 2 divergent goals.

Some goals can resolve the problem caused by some boundary conditions. In this case, a meta-relationship called *Resolution* is defined between them.

The *domain description* meta-class has a dashed border because it is a duplicate element from the object model. It has been done for the sake of diagram readability. It has one optional meta-attribute: *FormalSpec* that contains a formal specification. Its specialisation is described in the object model.

Like goals, obstacles can be AND-refined or OR-refined into sub-obstacles and domain description. This is represented by the *OR-Ref* and *AND-Ref* meta-relationships. Obstacle refinements can be complete or not, which is expressed by the attribute: *Status*.

6.2 Agent Model

In the *AgentModel* meta-concept, the *Agent* meta-class is obviously the central element. It has 2 mandatory meta-attributes: *Name* and *Specification* and an optional one: *Load*. This is an abstract meta-class that involves that it cannot be instantiated. This meta-class is specialised in 2 other meta-classes *SoftwareToBeAgent* and *EnvironmentAgent* which are concrete meta-classes. These classes indicate the category of the agent. As any element of the meta-model, agents can be refined. This is indicated by the aggregation link on the element itself. The meta-model proposed in [Lamsweerde, 2009] does not make a distinction between existing and new software agent but it could be done easily by adding a meta-attribute in the meta-class *SoftwareToBe*. We will not add it because it will not be used in the rest of the work. The same reflexion can be done for malevolent agent.

Table 6-4 Attributes of the agent meta-class

Attributes	Definition
Name	unique in the whole system model
Specification	explains in natural language what the agent has to do
Load	work load supported by this agent

There are *Responsibility* meta-relationships firstly between *SoftwareToBeAgent* and *Requirement*, and secondly between *EnvironmentAgent* and *Expectation*. Agents can also express their wishes for some goals, this is shown by the meta-relationship *Wish*.

As explained in section 5.3, agents can monitor and control the attributes of objects or object associations. These agent capabilities are described by 2 meta-relationships: *Monitoring* and *Control* that link the meta-class *Agent* to the meta-class *Association* and/or *Attribute*. The latest plays the *stateVar* role in these meta-relationships. To avoid overloading the class diagram, *InstanceVariable* meta-attributes attached to the *Responsibility*, *Monitoring* and *Control* meta-relationships are not shown.

As there are many possibilities to assign a leaf goal to an agent, we have created the meta-class *Assignment* and the *OR-Ass* meta-relationship between the meta-class *LeafGoal* and the meta-class *Assignment*. The one-to-many multiplicity expresses that there could be multiple assignments for a same leaf goal. The optional attribute *SysRef* indicates which alternative is taken in which version of the system.

The meta-relationship *Dependency* is a 3 part meta-relationship between the dependee (which is an agent), the dependant (which is another agent) and a goal.

6.3 Operation Model

In the *OperationModel* meta-concept, the *Operation* meta-class is obviously the central element. It has 4 mandatory meta-attributes: *Name*, *Specification*, *DomPre* and *DomPost* and an optional one: *Category*.

Table 6-5 Attributes of the operation meta-class

Attribute	Definition
Name	unique in the whole system model
Specification	explains in natural language which goal the operation has to fulfil
DomPre	domain pre-conditions
DomPost	domain post-conditions
Category	indicates the category of the operation (software operation or environment operation)

An operation can be performed by only one agent who is expressed by the meta-relationship link *Performance* between *Operation* and *Agent* meta-classes and the multiplicity one-to-one on the agent side.

Operations are done to fulfil at least one leaf goal which is shown by the meta-relationship *Operationalisation* and the one-to-many multiplicity on the operation side. On the other hand, a single leaf goal can be fulfilled thanks to many operations. This is expressed by the one-to-many multiplicities on the leaf goal side. If necessary, the meta-relationship *Operationalisation* can carry the required conditions for goal satisfaction as meta-attributes: *ReqPre*, *ReqTrig* and *ReqPost* which are all optional.

The operation signature can be found thanks to the instantiations of the *Input* and *Output* meta-relationships. These meta-relationships allow doing the link between the operation model and the object model. To avoid overloading the class diagram, *InstanceVariable* meta-attributes attached to the *Performance* and *Input/Output* meta-relationships are not drawn.

For the clarity of the meta-model, the fact that operations can be activated when they receive an internal event does not appear in the meta-model.

6.4 Object Model

In the *ObjectModel* meta-concept, the *Object* meta-class is obviously the central element. It has 2 mandatory meta-attributes: *Name* and *Specification* and an optional one: *InstanceOf*. This meta-attribute is a boolean, when it is set to true it means that the object can be initialised for the corresponding instance.

Table 6-6 Attributes of the object meta-class

Attribute	Definition
Name	unique in the whole system model
Specification	explains in natural language the behaviour in terms of phenomena that are monitorable and controllable in the system
InstanceOf	indicates if the object can be instantiated or not (boolean)

The meta-class *Object* is an abstract class that has to be specialised to be instantiated. It can be specialised into *Association*, *Entity*, *Event* or *Agent*. An Association is defined by the meta-relationship link *Link* between at least 2 objects. This association has 3 mandatory meta-attributes: *Role*, *Multiplicity* and *Position*.

As explained in the section 5.5, associations can be reflexive if they link many times the same object at different roles. The arity of an association is defined by the multiplicity of the relationship between the association and the number of objects involved. And finally, there are 2 main types of association: the application specification (meta-class *ApplicationSpecification*) and built-in associations called specialisation and aggregation (represented by the meta-classes: *Specialisation* and *Aggregation*).

Every object is characterised by attributes, domain description and domain initialisation. The meta-class *DomDescription* allows giving details about the domain thanks to its mandatory meta-attributes: *Name* and *Specification* and the optional meta-attribute *FormalSpec*. The domain description cannot be instantiated directly and have to be specialised into the meta-classes: *DomInvar* and *DomHyp*. The attributes of an object will be defined in the meta-class *Attribute*, an attribute can have its values within a range; in this case, such values can be multiple and time varying or not, as indicated by the *ValuesIn* meta-relationship. All specialisations of the meta-class *Object* inherit these characteristics.

Table 6-7 Attributes of the domain description meta-class

Attributes	Description
Name	unique in the whole system model
Specification	explains in natural language the domain description
FormalSpec	uses to annotate a domain description to formalize its informal specification

Objects concern goals, which are expressed by the meta-relationship *Concern* between the meta-classes *Goal* and *Object*. The relation multiplicities express that an object must be referred by at least one goal and a goal must concern at least one object.

6.5 Behaviour Model

The *BehaviourModel* meta-concept describes the abstraction used for modelling the system behaviours through scenarios and state machines. It contains 2 main parts: the first one is dedicated to instance behaviours (the right side of the model) while the second one is dedicated to class behaviours (the left side of the model).

Concerning the instance behaviours, the main object is the meta-class *Scenario* that illustrates some goal(s) and can be decomposed into zero-to-many sub-scenarios with the meta-relationship *Episode*. The meta-relationship *History* expresses that a scenario as a historical sequence of one up to an arbitrary number of timeline slices (meta-class *TimelineSlice*), each being a parallel composition of one or more interactions (meta-class *Interaction*) at the same point in time. An interaction is defined by one source agent instance, one-to-many target agent instance and one interaction event instance.

Regarding the class behaviours, the main object is the meta-class *AgentSM* that details the behaviour of any agent instance. A state machine is use to fulfil a goal which is shown by the meta-relationship *ClassCoverage* between the meta-classes: *Goal* and *AgentSM*. The latest is an aggregation of state machines (meta-class *StateMachine*), there is one state machine per state variable that the agent controls. These variables can be object associations or object attributes. Each state machine is defined by one-to-many state (meta-class *State*) which is expressed by the meta-relationship *Path*. A state can be decomposed into sequential or concurrent sub-states. States are used as input/output (meta-relationships *Input/Output*) for the meta-class *Transition*. A transition can be labelled by zero-to-one *Event* (which has to be specialised into the meta-classes: *InternalEvent* or *ExternalEvent*), zero-to-one *Guard* and zero-to-many *Operation*.

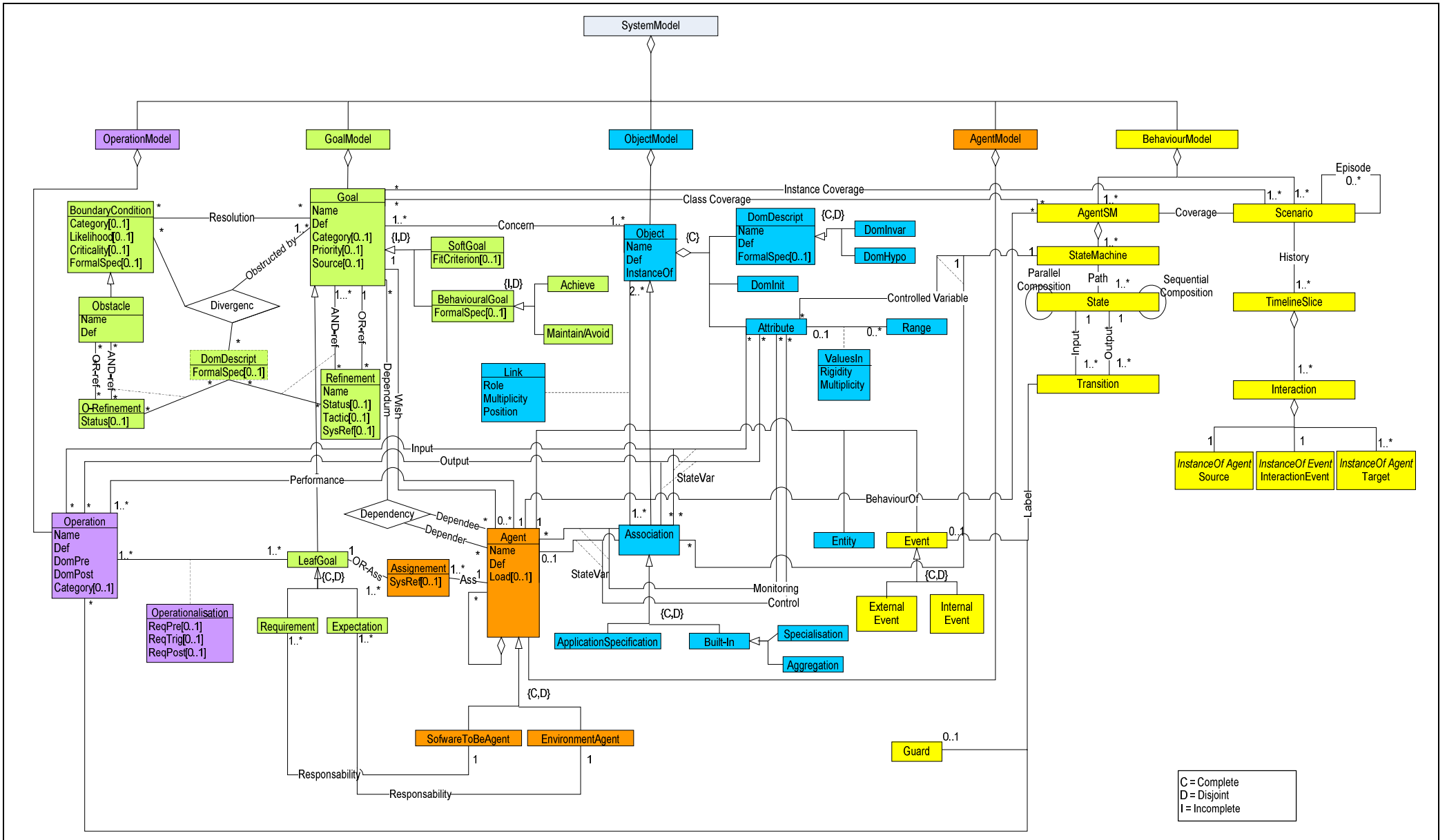


Figure 6-1 Meta-model of KAOS

Chapter 7 Applying the Physics of Notations to

KAOS

Chapter 4 explained the principles of the Physics of Notations and Chapter 5 described the KAOS language and its visual notation. In this chapter, each principle described in the Physics of Notation theory is applied to the KAOS visual notation.

Which order will we follow?

We will not study the principle in the order given [Moody, 2009], but we will group them by theme. Firstly, we will check if the KAOS visual notation corresponds to its meta-model (principle of semiotic clarity) and its cognitive effectiveness for the different users that will have to work with it (principle of cognitive fit). Secondly, we will take care of principles that are necessary to have a good diagram: principle of perceptual discriminability, principle of semantic transparency and finally the principle of visual expressiveness. Thirdly, we will study the principles that could improve the visual notation: principle of dual coding and principle of graphic economy. And finally, we will focus on the complexity management and navigation in large diagram with the principle of manageable complexity and the principle of cognitive integration.

How will we proceed to study a principle?

For each principle, we evaluate the situation. Then if we estimate that it is not completely respected we will suggest general improvements and/or general recommendations. In Chapter 8 we will provide particular improvements and recommendations depending on the cognitive fit of the studied case.

As seen in section 4.3, there are interactions among principles. Thus if a principle is improved, it can result in a negative effect on other principles. If this happens, we will discuss the advantages and the disadvantages of each proposition to find the best one.

In this chapter, we focus on the visual notation of the goal model. We have chosen it because in goal-oriented language the most important target is to represent goals. This model has also been chosen for its reusability during comparison with other goal-oriented languages.

7.1 Principle of Semiotic Clarity

7.1.1 Analysis results

Firstly we analyse the principle of the semiotic clarity which verifies the matching between the semantic constructs of KAOS and their visual translation.

To do this analysis, we need the KAOS meta-model (see figure 6-1) and the symbols used in KAOS (aka. KAOS vocabulary). This vocabulary is presented in figure 7-1. Symbols can be sorted in 2 groups: 2D figures (on the left) and 1D elements (on the right).

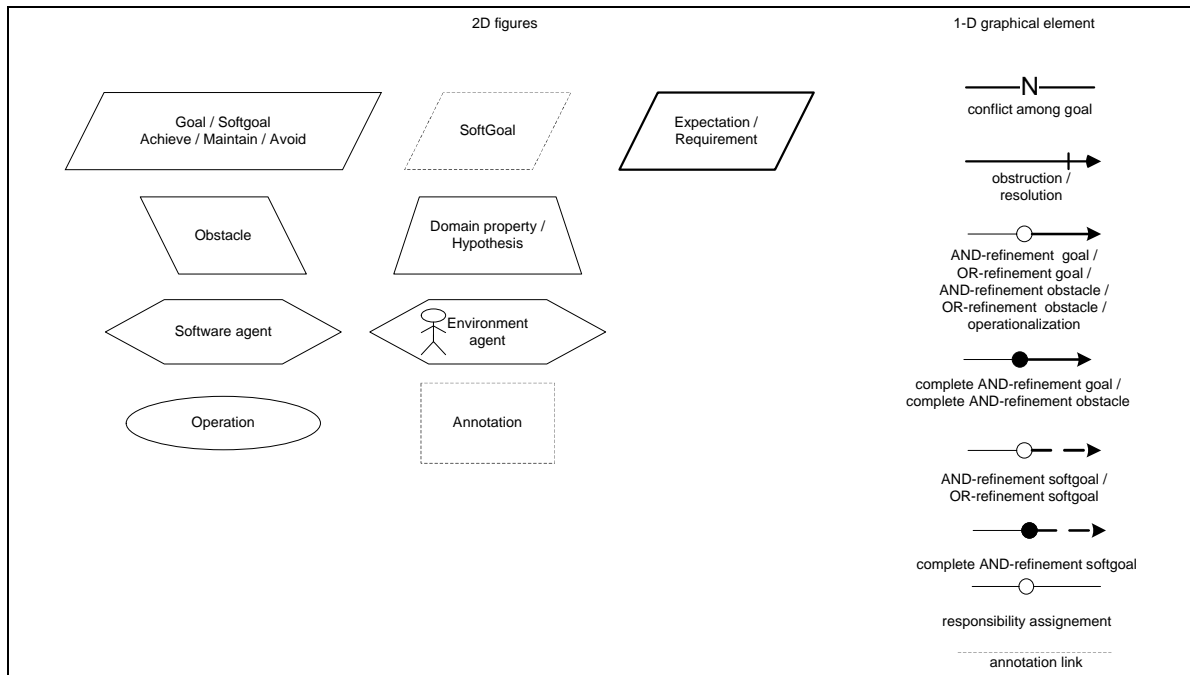


Figure 7-1 The KAOS visual vocabulary

There are 30 semantic constructs in KAOS (they are detailed in the annex1), distributed in 13 elements types and 17 relationship types. This set defines the **semantic complexity** of the notation. Visually, there are 17 distinct graphical constructions, distributed in 9 area types and 8 line types. This defines the **graphic complexity**.

Differences between the number of graphical constructions and the number of semantic constructs are due to semiotic anomalies. In [Moody, et al., 2010], the relationship between the number of constructs, the number of symbols and violations of semiotic clarity is defined by the following equation:

$$n(\text{symbol}) = n(\text{construct}) + n(\text{symbol redundancy}) - n(\text{symbol overload}) + n(\text{symbol excess}) - n(\text{symbol deficit})$$

In table 7-1, the value corresponding to the variables of the equation can be seen. The details of the operations to find these numbers can be consulted in annex 1.

Table 7-1 Semiotic clarity analysis of the KAOS visual notation

Constructs = 30		
Symbols = 17		
Anomaly type	Cases	% (number of cases/number of constructs)
symbol redundancy	1 (soft goal)	3%
symbol overload	13	43%
symbol excess	2 (annotation)	6%
symbol deficit	3 (all complete OR-refinement type)	10%

Table 7-1 shows that there are anomalies in the KAOS visual notation: symbol redundancy (3%), symbol overload (43%), symbol excess (6%) and symbol deficit (10%).

7.1.2 Recommendations to improve the semiotic clarity

As said in section 4.2.1, to improve the semiotic clarity, we should remove all semiotic anomalies.

Symbol redundancy

The only semantic construct that causes symbol redundancy is the concept of 'soft goal', that can be represented either by a right-oriented parallelogram with a continuous line as border or by a right-oriented parallelogram with a dashed border. Moreover, in [Moody, 2009], the dashed border is also used to represent stable elements in an analysis. Then to avoid any confusion, it is better not using anymore the right-oriented parallelogram with dashed border to represent soft goals.

Symbol excess

Symbol excess is due to graphical symbols that do not represent any semantic constructs. It is the case with the symbol that represents the annotation and the annotation link.

Symbol overload (homograph)

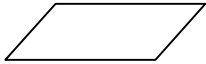

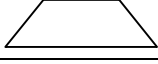
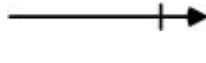
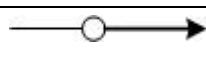


The number of symbol overload is given by the number of semantic constructs of a graphical symbol minus one (a symbol should represent only one semantic construct).

Homographs are equivalent to homonyms in visual notations. According to [Goodman, 1968], it is the worst type of anomaly as it leads to ambiguity and the potential of misinterpretation. When this anomaly occurs one or many time in a visual notation, the graphical language is said to be "ontologically unclear". We should complete the notation to make it more effective.

In other words, the KAOS visual notation violates the principle of monosemy which means that each semantic construct has to be represented by its own graphical construction.

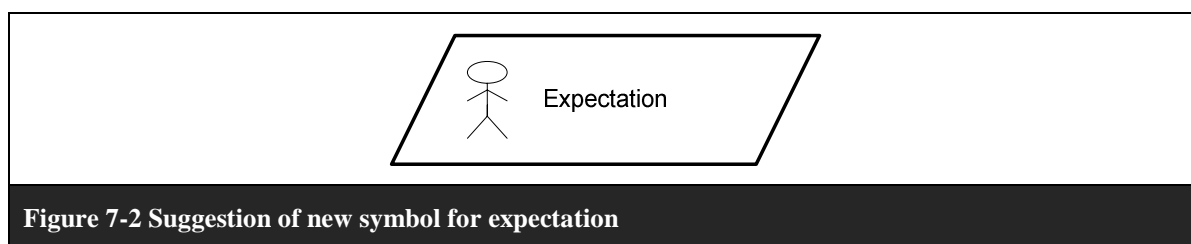
Table 7-2 details the overloaded symbols in the KAOS visual notation. For each overloaded symbol, we will study the possibility to find other symbols.

Table 7-2 Symbols overload analysis of relationships

Symbols	Semantic relationships	Symbol overload
	<ul style="list-style-type: none"> • Goal • Soft goal • Achieve goal • Maintain goal • Avoid goal 	4
	<ul style="list-style-type: none"> • Expectation • Requirement 	1
	<ul style="list-style-type: none"> • Domain property • Hypothesis 	1
	<ul style="list-style-type: none"> • Obstruction • Resolution 	1
	<ul style="list-style-type: none"> • AND-refinement goal • OR-refinement goal • AND-refinement obstacle • OR-refinement obstacle • Operationalisation 	4
	<ul style="list-style-type: none"> • Complete AND-refinement goal • Complete AND-refinement obstacle 	1
	<ul style="list-style-type: none"> • AND-refinement soft goal • OR-refinement soft goal 	1
7	20	13

The symbol that represents a goal can also represent soft goal, achieve goal, maintain goal and avoid goal. If we look at the KAOS meta-model, we can see that all of these semantic constructs are specialisation of the meta-class goal. To differentiate them, a textual differentiation is done in [Lamsweerde, 2009]: the name of the goal is preceded by squared parenthesis ([]) to identify the type of goal and avoid confusion.

The symbol that represents an expectation can also represent a requirement. As for the previous case, a textual differentiation could be done. But as expectations will be under the responsibility of an environment agent, it is better to add a symbol that represents a user inside the figure. The new symbol is drawn in figure 7-2. Doing this, it should help the user to remind that this kind of goal will have to be assigned to a human agent



The symbol that represents a domain property can also represent a hypothesis. The difference between these 2 semantic constructs is subtle: a domain property is a descriptive statement about the environment, expected to hold invariably regardless of how the system behaves, while a hypothesis is a descriptive statement satisfied by the environment and subject to change. In fact, there are specialisations of the semantic construct 'domain description' and as the semantic gap between them is subtle, we propose not to change them.

The symbol that represents an obstruction relationship can also represent a resolution relationship. According to us, the name of the relationship depends on the direction of the arrow. If the arrow starts from the obstacle to the goal, we call it an 'obstruction' and if it starts from the goal to the obstacle, we call it a 'resolution'. The reader cannot confuse the meaning because a goal cannot be an obstruction to an obstacle and an obstacle cannot be a resolution a goal.

The symbol that represents an AND-refinement goal relationship can also represent an OR-refinement goal, an AND-refinement obstacle, an OR-refinement obstacle and an operationalisation. The fact that a same symbol represents different kinds of refinements (goal or obstacle) does not cause any trouble. But we can suggest adding a horizontal line that links the different OR-refinements of a goal or an obstacle. It will be clearer for the reader. Figure 7-3 illustrates this improvement: on the left side (part A) the figure depicts an AND-refinement between goals and, on the right side (part B), there is an OR-refinement.

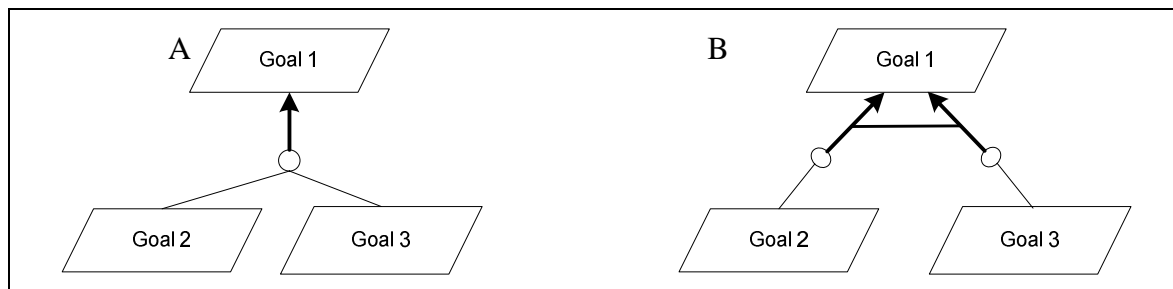


Figure 7-3 Suggestion to improve the differentiation between AND-refinement and OR-refinement

Regarding the fact that the symbol can also represent operationalisation, we find it very confusing and we suggest introducing a new symbol to express this semantic construct. But we will not do any suggestion because operation relationship link belongs to operation model.

The symbol that represents an AND-refinement soft goal relationship can also represent an OR-refinement soft goal relationship. The difference with the symbol that represents refinement consists of a dashed bold line between the circle and the parent goal instead of a continuous line. This difference was probably introduced because soft goals could also be represented by right-oriented parallelogram with dashed border. We suggest to completely removing this visual representation as we have suggested removing the specific one for soft goals.

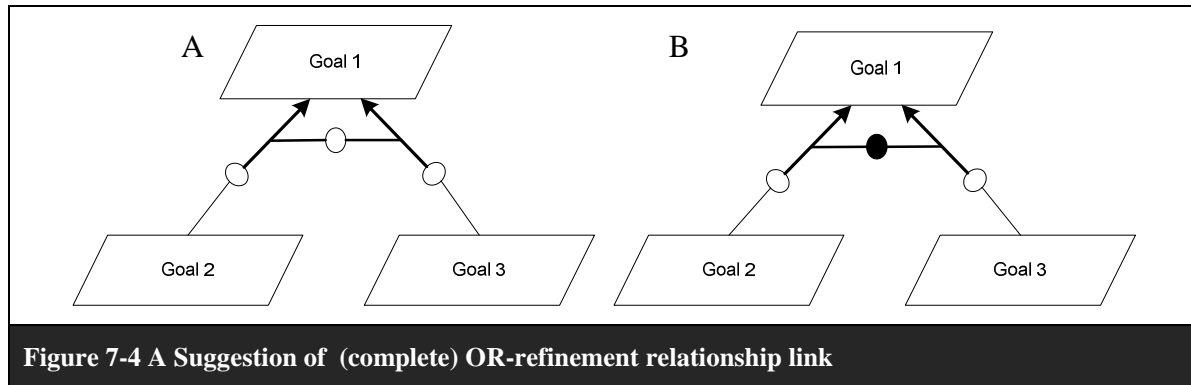
But before giving a different visual symbol for each semantic construct, we have to keep in mind that symbol overload is a common way of dealing with excessive graphic complexity. A way to resolve this problem would be to use visual variables (instead of text differentiation) to distinguish between different semantic constructs [Moody, et al., 2010].

Symbol excess

There are 2 cases of symbol excess in the KAOS visual notation. It concerns the shape that represents annotation and the dashed line that links the annotation to the element that it details. These graphical symbols do not correspond to any semantic construct and it is not possible to adapt the meta-model to include them. This anomaly cannot be corrected.

Symbol deficit

There are 3 cases of symbol deficit in KAOS and all of them concern the OR-complete refinement. This concept is defined in the meta-model and in practice it can also occur if we have a choice between a determined number of possibilities. To solve this deficit, on the horizontal line (suggested in the symbol overload to identify clearly OR-refinement), we could add a circle –like in the AND-refinement– that will be filled in black if the refinement is complete. Figure 7-4 implements this suggestion: on the part A, there is an OR-refinement between goals and part B depicts a complete OR-refinement between goals.



7.2 Principle of Cognitive Fit

After studying the principle of semiotic clarity, we have to check that the visual notation corresponds to the user needs. Thanks to the principle of the cognitive fit, this will be verified.

7.2.1 Analysis results

The cognitive fit theory involves that a language should be adapted to the form of representation it will use, to the task characteristics (e.g., sketching on whiteboards or using computer-based drawing tools) and to the problem solver skills (experts or novices) [Moody, 2009].

The KAOS language has only one set of symbols for all types of tasks, audience and problem solver skills.

7.2.2 Recommendations to improve cognitive fit

KAOS is a language created for many usages, such as, to create a dialog with future users of the system, to discuss within software analyst team members, to explain the work to the developers. It would be definitively interesting to improve its cognitive fit. We would support to provide KAOS with at least 4 dialects: one for experts and another one for the novices and 2 others that could be used depending on the context.

Experts vs. novices

The dialect for experts will follow the semantic constructs and the figures explained in [Lamsweerde, 2009] while the other one for novices should follow the pieces of advice given by Moody: “This means that notations designed for novices should use clearly distinguishable symbol (perceptual discriminability), mnemonic convention (perceptual immediacy), clarifying text (dual coding), and simplified visual vocabularies (graphic economy) [Moody, 2009]”.

The application of this principle is discussed in section 8.2

Detailed study vs. in a meeting

A third dialect could be created for meetings with stakeholders of the future system. The goal of this dialect is to be easy and quick to sketch by hand on paper sheet, because –while one meeting participant draws, the others have to wait. If it takes too much time they will have the feeling to waste their time. Finally this dialect should be easily understood by non-technical person.

By hand vs. CASE tools

Finally, the last dialect that we suggest, could be used in requirements engineering software that uses KAOS as modelling language (e.g., Objectiver [Objectiver, 2007]). This dialect could support more easily colours because, when diagrams are sketched on sheets of paper, it is not always easy to have colour pens or pencil at hand). Tools can also provide more complex symbols that could not be drawn easily by hand.

In Chapter 8, we will give recommendations targeting different groups of users and for different situations. Then we have to take their specificities into account and use the appropriate dialect.

7.3 Principle of Perceptual Discriminability

From this section, we start to study the 3 principles that are necessary to have a good diagram. The first one is perceptual discriminability. According to [Moody, 2006], this principle could be divided into 2 parts: the absolute perceptual discriminability and the relative perceptual discriminability. As absolute perceptual discriminability is not a part of [Moody, 2009] but explained in [Moody, 2006], then we will not study it in this section. However, it will be taken into account to write the recommendations presented in Chapter 8.

After studying the principle of perceptual discriminability, we will study the principle of the principle of semantic transparency and then the principle of visual expressiveness.

7.3.1 Analysis results

The relative perceptual discriminability is based on many several points: the visual distance, the primacy of shape, the natural perceptual pop out of the symbols, the textual differentiation and the visual-semantic congruence.

Visual distance

In [Moody, 2009], Moody defines the perceptual discriminability as "the ease and accuracy with which symbols can be differentiated from each other". After he explains that "discriminability is determined by the visual distance between symbols, which is measured by the number of visual variables on which they differ and the size of these differences (number of perceptible steps)."

Differences can be counted thanks to the visual variables (horizontal and vertical position, shape, colour, orientation, texture, value and size). The more differences and the bigger they are, the easier it will be to distinguish between symbols. And if the differences are too subtle, interpretation errors can occur. The shapes used in the KAOS syntax (see figure 7-5) uses only 2 visual variables: the shape and the texture.

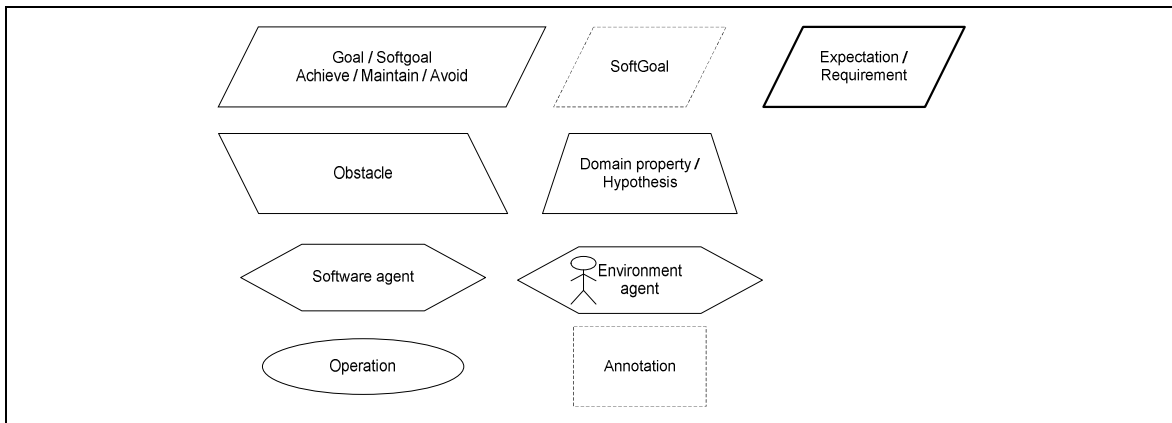


Figure 7-5 Use of shapes and textures in the KAOS notation

Among the shapes used in the notation, quadrilaterals are often encountered and in particular the parallelogram. The only visual difference between a goal and an obstacle is subtle and concern the visual variable "shape". Each of these semantic constructs is represented by a parallelogram but they can be differentiated by the orientation of this parallelogram. Parallelograms used to describe goals are right-oriented and those used for obstacle are left-oriented. This difference is very small for novices whereas the semantics of these constructs are totally different.

KAOS shapes also use texture to differentiate them. The difference between the shape that represents a goal and the shape that represents an expectation or a requirement is a bold border. This kind of differentiation is not very cognitively effective because the user has to look at the diagram very carefully.

Primacy of shape

To increase cognitive effectiveness, each shape has to represent a different concept. This is not the case for several shapes that are overloaded. In particular, the right-oriented parallelogram that can represent many semantic constructs: goal, soft goal, achieve goal, maintain goal and avoid goal. To distinguish between the different concepts, [Lamsweerde, 2009] suggests using the textual differentiation that allows dealing with excessive graphic complexity.

Perceptual pop out

As most of the visual elements used in KAOS have a unique value for at least one visual variable and they do not use combination value, they pop out from diagram without effort.

Nevertheless, some elements do not follow this rule and consequently they do not 'pop out' directly. These elements are represented in figure 7-6.

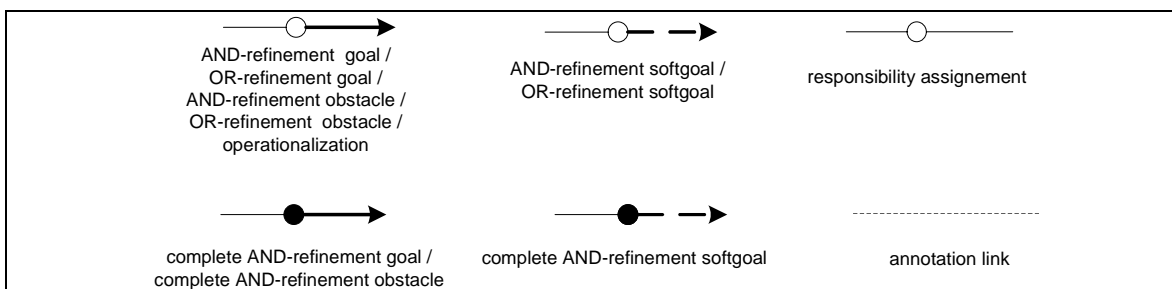


Figure 7-6 Elements that do not visually pop out

These elements use combinations of values of shape (small circle) and texture (normal, dashed or bold) to create different relationships. The reader needs more time to analyse them and understand their meaning.

The difference between an AND-refinement and an OR-refinement is easy to understand at first sight if the number of goals is limited (the number of figures that can be understood at first sight by the reader is limited by working-memory) but when goals are too many, the graphic complexity is increased and recommendations for this principle are described in section 7.8.

Figure 7-7 illustrates an example of an AND-refinement and an OR-refinement. The goal 'EasyToUseSystem' is OR-refined 2 times. Each of these OR-refinements contains 2 AND-refinements. To see which goal belongs to which refinement, the reader has to analyse the diagram carefully.

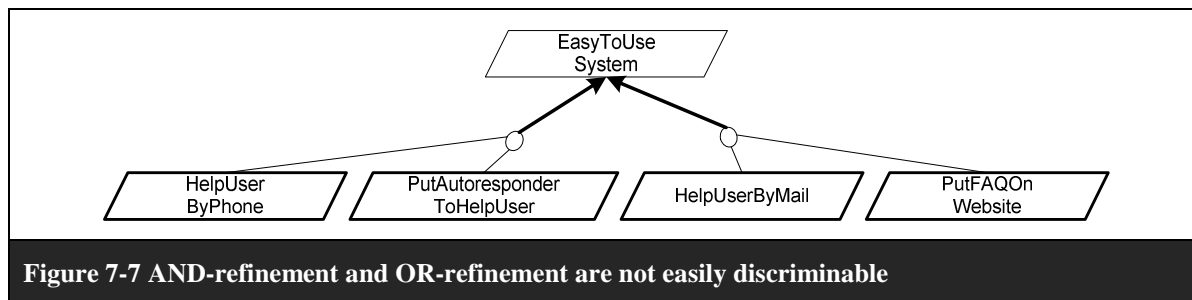


Figure 7-7 AND-refinement and OR-refinement are not easily discriminable

Textual differentiation

Software engineering sometimes relies on text to distinguish between symbols. The KAOS visual notation uses this technique to allow the differentiation between goal, soft goal, achieve goal, maintain goal and avoid goal.

Visual-semantic congruence

In general, the visual distance between symbols should be congruent to the semantic distance between the constructs they represent: constructs which are very different in meaning (large semantic distance) should have very different symbols (large visual distance), while constructs that are similar in meaning should have similar symbols.

In KAOS, goal, soft goal, achieve goal, maintain goal and avoid goal have a semantic meaning very close and there are both represented by a right-oriented parallelogram (see section 7.1). In this case, visual congruence is respected and this contributes to the cognitive effectiveness of the notation.

Shapes that represent goals and leaf goals (expectation or requirement) are the same, and they can only be distinguished by a bold border for leaf goal. Once again, the visual congruence is respected because leaf goals are a specialisation of goals.

Finally, goals and obstacles are semantic constructs that have opposite meanings and their visual notations are also different: the goals are represented by a right-oriented parallelogram while obstacles are represented by a left-oriented parallelogram. In this case, even if the visual-semantic (non)congruence is respected this choice will be discussed in section 7.4 about semantic transparency.

7.3.2 Recommendations to improve the perceptual discriminability

Redundant coding

Redundant coding is one of the techniques used to increase the visual distance between symbols. According to [UsabilityFirst, 2011], the redundant coding consists of "representing information in more than one way so that users have more than one opportunity to perceive and understand it, to reinforce the information, to make the information more accessible (because one representation may not work for a certain technology or user), or to suit user preferences". One of the ways to improve the perceptual discriminability, using the redundant coding, is to add a coloured background to the symbols used in the KAOS visual notation.

This improvement has already been done in Objectiver, a modelling CASE tool for KAOS. This tool uses redundant coding to differentiate the goals, domains and the obstacles. Goal shapes are light blue, domains are purple and obstacles are red. The fact that each symbol is filled with a distinct colour helps the reader to distinguish more easily the different types of semantic constructs.

Recommendations for primacy of shape anomaly

Even if a right-oriented parallelogram can represent many semantic constructs, we will not suggest using a shape by semantic constructs because it is conflicting with the principle of graphic economy. Moreover, as seen in section 7.1.2, the semantic meanings of the different concepts are very similar.

Recommendations for visual distance

To differentiate obstacles from goals on an easier way, the shape that represents obstacle (left-oriented parallelogram) could be replaced by a triangle like in figure 7-8. The choice of this shape has 2 advantages: it looks like the 'careful' road sign and it is not quadrilateral shapes that are already mostly used in the KAOS visual notation.

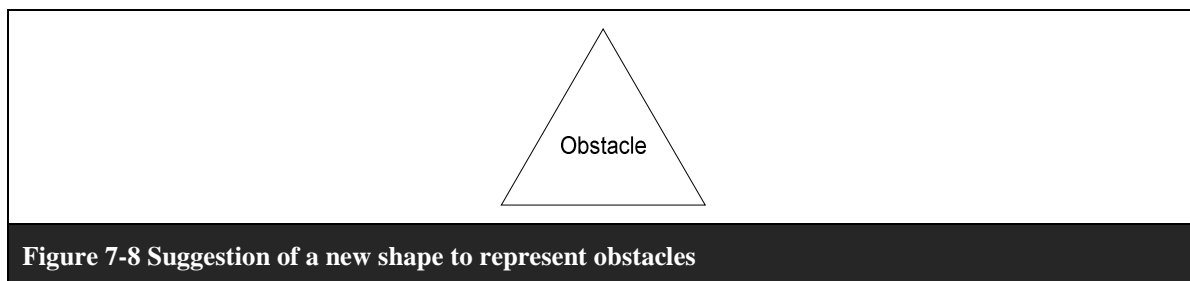


Figure 7-8 Suggestion of a new shape to represent obstacles

Another suggestion would have been to represent it by the 'stop' road: an octagon. The disadvantage of this shape is to be too close as the hexagon used to represent agents.

Recommendations for perceptual pop out

As explained in section 7.1.2, to avoid symbol redundancy, the shape that represents soft goal (a right-oriented parallelogram with a dashed border) should not be used anymore. Removing this shape involves to remove also specific relationship visual notations linked to this shape (see figure 7-1, the 2 soft goal refinement links). Then finally, there are only 3 relationship links that do not pop out.

If it is true that the way of refinement, complete refinement and performance pop out is not completely cognitively effective, it is not either the worst. As the number of combinations is very limited and very distinguishable, we should continue with these graphical representations. If we try to improve this anomaly, the principle of graphic economy will be less respected and in this case the disadvantage is greater than the advantage.

Recommendations for textual differentiation

Even, if the KAOS visual notation uses textual differentiation and that is cognitively ineffective, we will not change it to avoid increasing the graphic complexity. In the case of the different goal types, as they have very close semantic meanings, textual differentiation seems to be a good compromise.

Recommendation for semantic-visual congruence

This aspect of the KAOS visual notation is respected. We propose no changes.

7.4 Principle of Semantic Transparency

7.4.1 Analysis results

Semantic transparency involves the use of graphical representations whose appearances suggest their meanings [Moody, 2009]. We generally use symbols or geometrical shapes as mnemonics to help the reader of the diagram. Indeed, some geometrical shapes are used as road sign and can be recognised even if they are covered by snow (e.g., the octagon which means 'stop' or the reverse triangle which means 'leave priority').

This principle is complementary to the principle of perceptual discriminability.

As seen in section 4.2.3, semantic transparency is not a binary variable; it has an unlimited number of values. This variable is subjective because it can vary from a user to another depending on his/her background or culture. According to his/her culture, the user will be able to associate an idea to a symbol.

KAOS uses mostly geometrical shapes: parallelogram, trapeze, hexagon, rectangle and oval. These figures are semantically opaque. Users have to study them to understand and remember their meaning. Without explanation, it is impossible to guess their meaning.

The KAOS visual notation contains only one symbol that is not semantically opaque: a sticky man that represents a user.

7.4.2 Recommendations to improve the semantic transparency

The principle of semantic transparency is almost not respected in KAOS; it is one of its main weaknesses. Many improvements can be done to improve this principle. The more the visual notation is semantically transparent, the easier for the novice users to guess the meaning of the semantic constructs represented by the symbols. To reach this goal, we suggest using symbols that are semantically transparent or that represent something that the reader knows. It will help him/her to find the meaning of the different KAOS concepts.

The only part of this principle which is respected is the semantic transparency of refinement relationships. The direction of the arrow between a parent and its children help the user to understand that they are linked and that children (sub-goals) are derived for its parent (goal). We have to add that the positioning of children compared with its parent position - usually the children are drawn below its parent- also help the reader to understand the relationship link between them.

Below we try to find a drawing or a picture that could help the user to guess the different concepts of KAOS. As most of these semantic constructs are represented by abstract shapes, we suggest transforming them into symbols that have a meaning for the reader or at least that could help him/her to remember the semantic construct linked to the symbol. To do it, we will try to associate the meaning of each semantic construct to objects of the real life that the users are supposed to know. The different suggestions are presented in figure 7-9.

Goals

Goals are objectives that the future system should fulfil, they are targets to reach. This could be symbolised by a dart target (we could also use a football goal but it is more difficult to draw by hand).

Obstacles

Obstacles are elements of the system that will impede to reach goals. These can be part of the existing system or events that will occur. To establish the future system, designers will have to "jump over" these obstacles. This meaning can lead us to think about horse jumping.

Domain properties

If we read the definition of the 'domain' in a dictionary, it is defined as "a territory held in possession of someone". Here the domain can be considered as we have to build the new software: existing hardware, software and environment. Considering the first definition, a domain can be schematised by a house.

Software agents

As software is an abstract concept and as no representation, we will represent it by a computer as software runs on computer.

Environment agents

An agent is a human that has to complete some tasks. We will simply represent it by a user (this is the only icon already uses in the KAOS visual notation).

Operations

Operations are realised by agents and it means that a process has to be executed. Running gears could be a good figure to represent a process that has to be done.

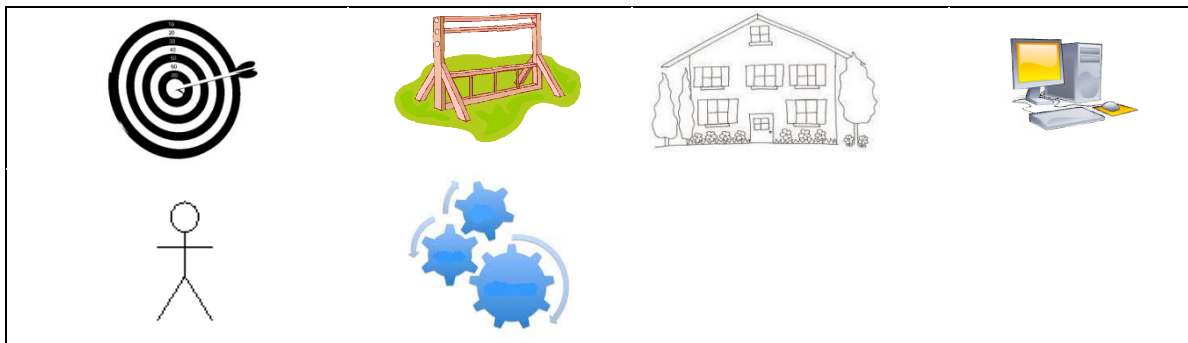


Figure 7-9 Different icons to represent the different concepts of KAOS.

From left to right and from top to bottom: goal, obstacle, domain property, software agent, environment agent, operation.

These symbols are not easy to draw quickly; we will derive abstract draws that are easy and quick to draw from the initial symbols (figure 7-10). To do it, we will identify the main lines that characterise the picture and used them to do simpler symbols.

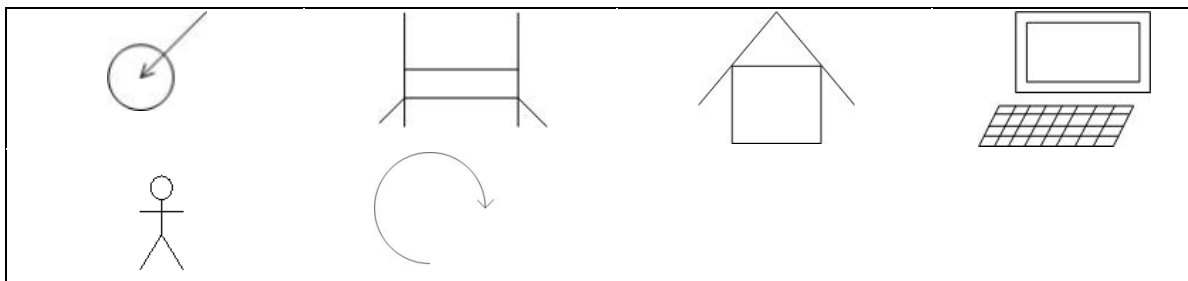


Figure 7-10 Simplified symbols to represent the different KAOS concepts

The semantic transparency can also concern relationship links. In [Lamsweerde, 2009], there is no piece of advice about the position of a goal and its sub-goals even if he always draws the sub-goals under the refined-goal. This practice is good because it shows clearly the relationship link between the different goals, then it should be added in the description of the visual notation of KAOS. This remark is also relevant for the obstruction/resolution relationship link: the obstacle of a goal should be placed under the goal that it will obstruct.

7.5 Principle of Visual Expressiveness

7.5.1 Analysis results

Visual expressiveness is defined by Moody in [Moody, 2009] as "the number of different visual variables used in a visual notation and the range of values used of each. It measures the utilisation of the graphic design space". This definition leads us to the notions of the primary notation and second notation. The first notation contains the visual variables that carry information and the second notation refers to free variables.

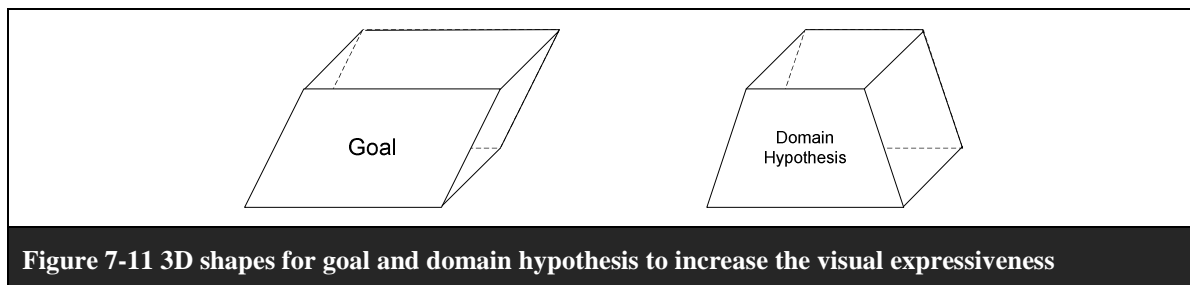
In most cases, the use of variables is clear. However, it is difficult to classify the use of bold border. We will associate it to the **texture** visual variable because it can have different **value** (medium grey bold, dark grey bold, black bold). In other words, bold border can not be associated to the value visual variable because this variable is used to define the brightness of a colour (e.g., light blue or dark blue).

In this analysis, we consider that goals (right-oriented parallelogram) and obstacles (left-oriented parallelogram) are 2 distinct symbols because one is not a variation of the other. In particular, the orientation visual variable is not used in this case because the symbol that represents an obstacle cannot be found from the goal symbol if it turns around a determined point.

The **level of visual expressiveness** of the KAOS visual notations is 2 on a scale of 1 to 8 because it relies on 2 visual variables: shapes and textures. Currently, KAOS uses at least 3 levels of texture: normal, dashed and bold line and a limited range of shapes (quadrilaterals, hexagons and oval). Their combinations allow distinguishing the different semantic constructs of the KAOS language.

7.5.2 Recommendations to improve the visual expressiveness

The first recommendation concerns the shapes actually used in the visual notation: they are mostly quadrilateral (parallelogram and trapezium). But empirical studies have shown [Bar, 2006] that curved shapes, iconic and 3D shapes are preferred by users. There is only one curved shape used in KAOS visual notation: the oval. Iconic and 3D shapes are not used at all. Nevertheless they are more cognitive effective than 2D abstract shapes [Bar, et al., 2006] [Irani, et al., 2003]. Quadrilateral 2D shapes could be easily transformed into 3D shapes as represented in figure 7-11.



As described in the analysis, KAOS uses only a small subset of the visual variables and a small subset of their variations. It involves that KAOS uses only a fraction of the design space. These visual variables are actually not used: vertical and horizontal position, size, colour, position and value. Using them would increase the visual expressiveness.

Colour is not used in the original KAOS visual notation in [Lamsweerde, 2009] but it is used as secondary notation in the software 'Objectiver' [Objectiver, 2007] to reinforce the differences between the different KAOS elements.

Using colour is also suggested in the principle of perceptual discriminability (section 7.3) to create a redundant coding and increase the cognitive effectiveness of the users.

More over, some studies [Mackinlay, 1986] and [Winn, 1993] have shown that "colour is one of the most cognitively effective visual variables because the human visual system is highly sensitive to variations in colour and can distinguish between them quickly and accurately". And others [Lohse, 1993] and [Treisman, 1982] have demonstrated that different colours are detected 3 times faster than shapes. However, colour has to be used carefully, otherwise it can undermine communication.

Following these observations, it appears clearly that colours should be added to the KAOS visual notation. To do it in the more effective and cognitive way, they have to be chosen to be as discriminable and as mnemonic as possible. In figure 7-12, we suggest to use:

- Green for goals because they are positive idea that we have to reach.
- Red for obstacles because they are negative elements or events that could prevent us to reach goals.

- Orange for domain properties and hypothesis because we have to take them into account and keep them in mind (e.g., the flashing orange traffic light used to delimitate work area).
- Blue for agent like the uniform of the police agent.
- Grey for operations because they involve executing processes like in manufacture.

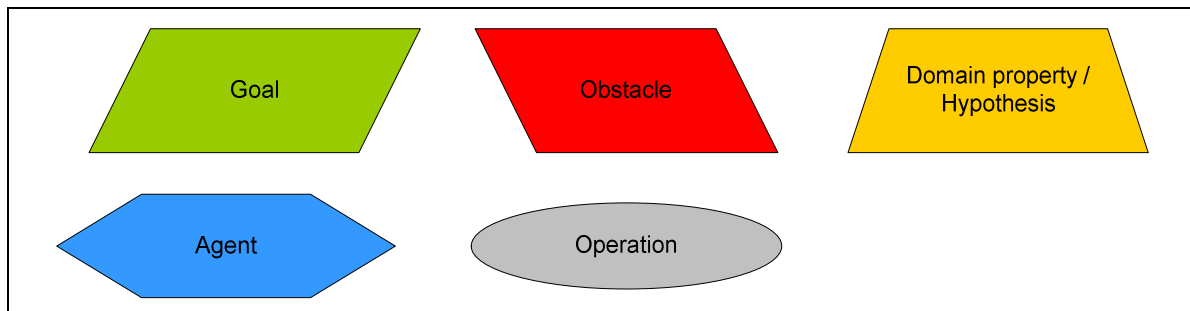


Figure 7-12 Add colour to shapes to increase the visual expressiveness

As second piece of advice, we may suggest to use the value visual variable to show to the reader which goals are the more important and which obstacles are the more dangerous. The darker is a goal or an obstacle in a refinement, the more important (or dangerous) it is. Figure 7-13 shows that the goal `EasyToUseSystem` is more important than `CheapSystem` and `EfficientSystem` because its colour is darker.

Another advantage of using colours to show the priority is to highlight the more important goals in the whole goal model.

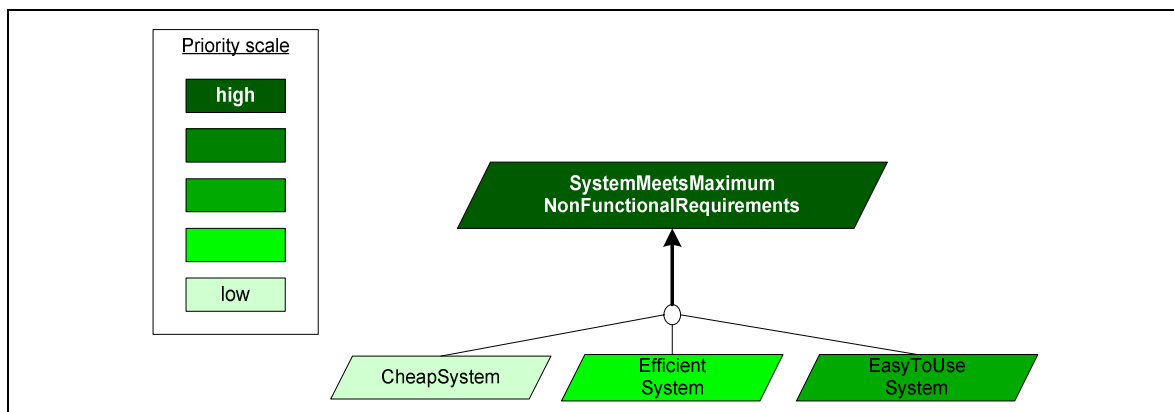


Figure 7-13 The intensity of the colour suggests the priority of the goal

And finally, the use of planar variables (especially the vertical position) should be officially prescribed in the KAOS visual notation. Indeed, drawers usually put the sub-goals under the goal they refined (the same remarks can also be done for an obstacle and its refinement). It is a good and intuitive practice but it should be good to formalize it in the visual notation.

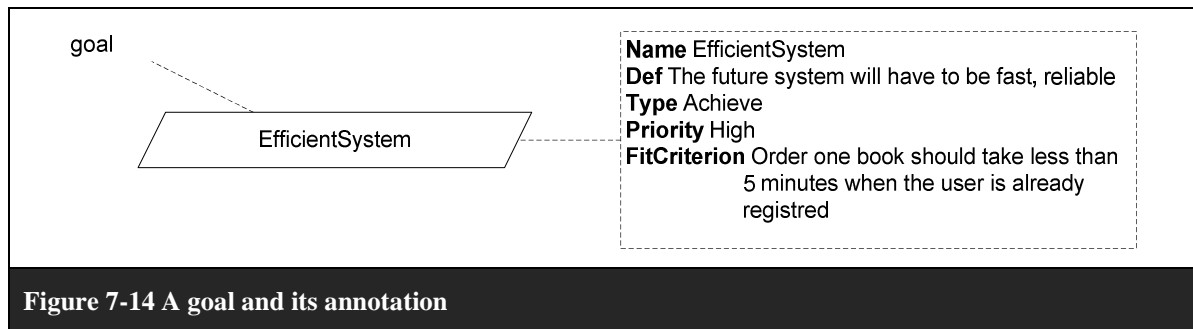
7.6 Principle of Dual Coding

7.6.1 Analysis results

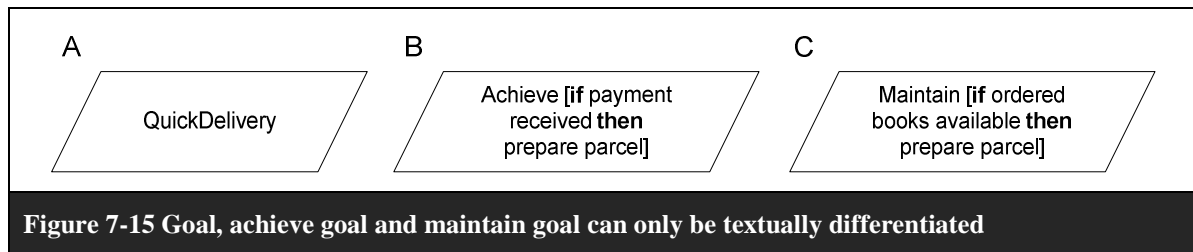
Dual coding is used to reinforce cognitive effectiveness by encoding information in 2 complementary ways: by drawing and by text. Text is used to clarify and refine the meaning of the diagram.

This technique is used in the KAOS visual notation through textual annotations. They improve the understanding of the diagram and are used as a form of dual coding to reinforce and clarify the meaning of the different part of a diagram.

Figure 7-14 shows a goal and its annotations. The annotation supplements the symbol and gives more information to the reader such as: the definition of the goal, its type, the priority and the fit criterion.



KAOS uses the technique of dual coding to differentiate different concepts. To do it, text is added to the shape to differentiate the different semantic constructs. For example, these subtypes of goals: maintain, avoid, achieved can only be differentiated by the text inside the shape. Figure 7-15 represents an example of a goal (A), an achieve goal (B) and a maintain goal (C). But, according to Moody [Moody, 2009] it is not a good way to differentiate them.

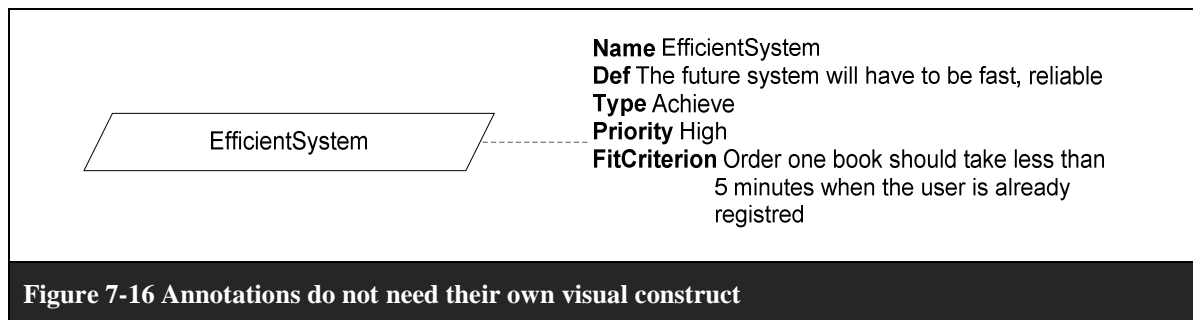


7.6.2 Recommendations to improve dual coding

Annotations

Dual coding is created by adding annotations to the visual symbols to give more information to the user. But this technique has the disadvantage that annotations can be very large and it will overload the graphic. As seen in figure 7-13, the annotation for a goal contains: a name and a specification and optionally a category, a source, a priority, a fit criterion and a formal specification.

According to the principle of spatial contiguity [Mayer, et al., 2003] including annotations on the diagram itself is more effective than writing them separately. More over, it is better not to use the shape that contains annotations because it will introduce symbol excess in the semiotic clarity principle. A simple block of text linked to the element that it annotates will help not to overload the diagram and avoid any misinterpretation. This recommendation is illustrated in figure 7-16.



Figures distinguishable by text

Figures that can be distinguished only by text should be avoided, but text can be used as a form of dual coding to reinforce and clarify the meaning. However it is a cognitively inefficient way to differentiate between symbols as text processing relies on slower, sequential cognitive processes [Moody, 2009]. Following these considerations, it involves that on one hand, it should be more cognitively efficient if each concept has its own figure.

But, on another hand, the principle of visual expressiveness advises using similar shapes for similar constructs (visual-semantic congruence).

As there are advantages and disadvantages to use the textual differentiation, it probably means that we have to find a good balanced between the different principles.

Labelling of elements

Annotations are very useful, but in KAOS, the modeller can put until 7 pieces of information for a single element. If all elements of a diagram are fully annotated, the diagram will be highly overloaded and the essential information will be disseminated across the diagram. The modeller has to fill the information shortly but in an accurate way.

In this section we can also approach the question of the elements labelling. We recommend standardising them for an easier understanding for the reader. Developers of Objectiver have already written some pieces of advice: “for a goal: a word followed by verb in its passive form (example "Service requested" instead of "Request service"). This is to avoid confusion between goals and operations (agent behaviours). Operation labels are a verb followed by words to describe an action” [Objectiver, 2007].

Labelling relationships

In KAOS, relationships are differentiated by different graphical constructs and not by labels. We could add some key-words to reinforce the information and improve the cognitive effectiveness. These key-words could be:

- AND-REF, OR-REF for and-refinement and or-refinement.
- COMPLETE for complete and-refinement or complete or-refinement.
- PERFORM for performance for the relationship between an agent and its operations.

These improvements can be seen in figure 7-17:

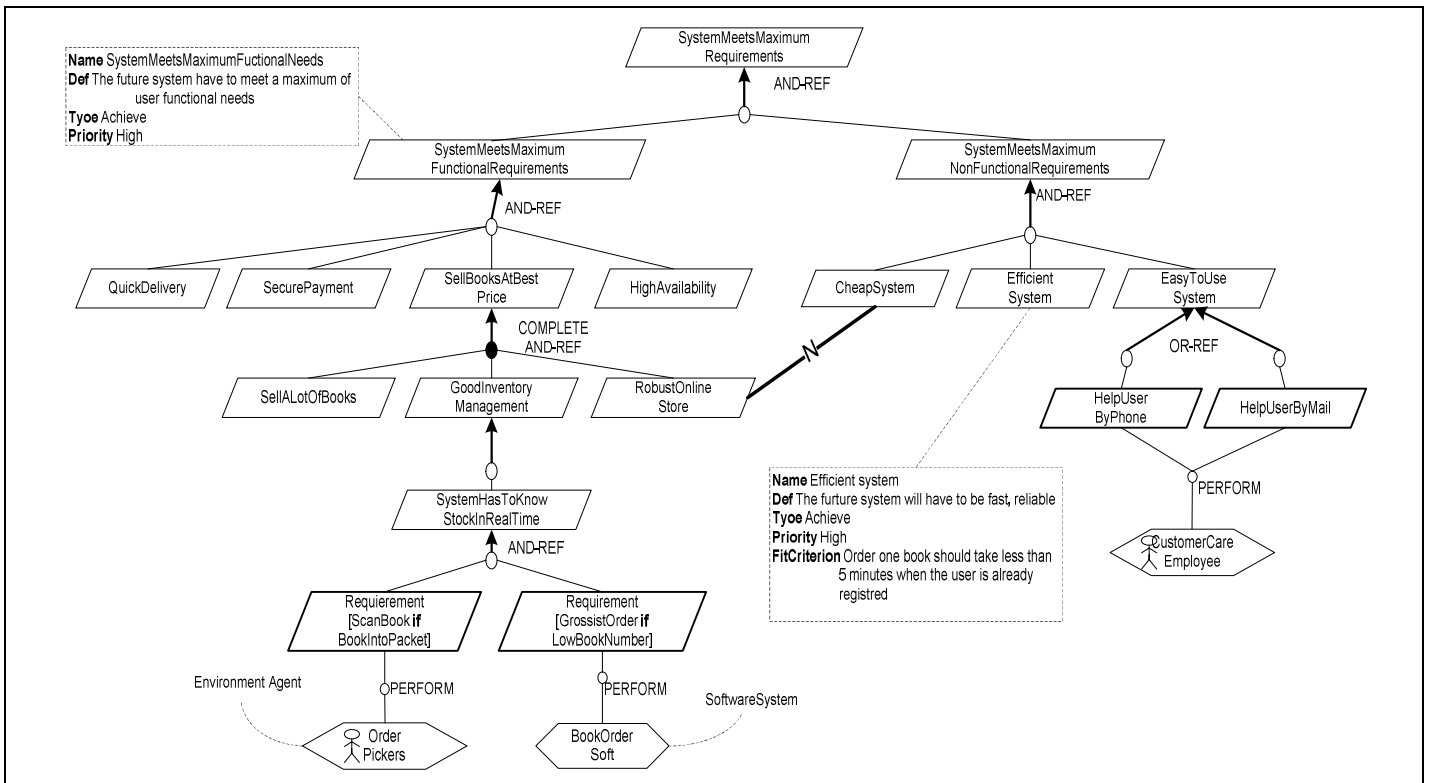


Figure 7-17 Dual coding: each link is labelled

7.7 Principle of Graphic Economy

7.7.1 Analysis results

Graphic complexity is defined by the number of graphical symbols in a notation: the size of its visual vocabulary. The KAOS visual notation contains 9 geometric shapes and 8 lines types, and then its graphic complexity is equal to 17. This number is too high and will be a problem for any visual notation, especially for those used by novices. Indeed, Miller [Miller, 1956] defines the human ability to discriminate between perceptually distinct alternatives is around 6 categories.

Reducing the graphic complexity or maintaining it at the same level requires efforts because notations tend to increase inexorably. It is due to the improvements of the semantic expressiveness which will increase the number of semantic constructs. These new semantic constructs will lead to the introduction of new graphical symbols. Moreover, if modellers think to improve the semantic expressiveness they do not think to reduce the graphic complexity by removing some symbols. But to a certain point, adding new symbols will reduce the cognitive effectiveness.

In [Moody, 2009], 3 strategies are described for dealing with excessive graphic complexity and improve graphic economy:

- Reduce semantic complexity
- Introduce symbol deficit
- Increase visual expressiveness

7.7.2 Recommendations to improve graphic economy

The first one: "reduce semantic complexity" consists of simplifying the KAOS meta-model due to the fact that graphic complexity is mainly determined by the number of semantic constructs. This technique is easily applied to the KAOS visual notation because the meta-model is divided into 5 fragments that will be used to represent different models: goal model, agent model, operation model, object model and behaviour model. Doing this, the number of graphical symbols by model will decrease considerably.

The second and the third techniques are: introducing symbol deficit and increase visual expressiveness

Introducing symbol deficit

Graphic complexity can be reduced directly by introducing symbol deficit (see discussion about semiotic clarity in section 7.1). It involves not showing some semantic constructs in graphical form and avoiding showing too much information. In the KAOS visual notation, there are 4 graphical constructs that could disappear without losing too much information: leaf goal, soft goal and its specific refinement relationship link and finally environment agent.

Leaf goals such as requirement and expectation do not really need their own symbols because at a moment or another they could be refined once more. In this case they will not be leaf goals anymore. Then we could use the same graphical symbols as this one for goal.

As soft goals can be represented by 2 symbols (which is an anomaly called symbol redundancy), one of them could be ignored. Textual differentiation will be used to discriminate this semantic construct from the other ones. The same reasoning can be done for the specific soft goal refinement relationship links.

The annotation shape can also be removed from the set of shapes as explained in section 7.6.2.

Finally, software agent and environment agent could use the same symbol which could help novices.

Figure 7-18 depicts the new KAOS visual alphabet. Introducing symbol deficit will allow starting with a visual alphabet of 17 graphical constructs and arrive to 11 graphical constructs which decreases the graphic complexity of more than 30%.

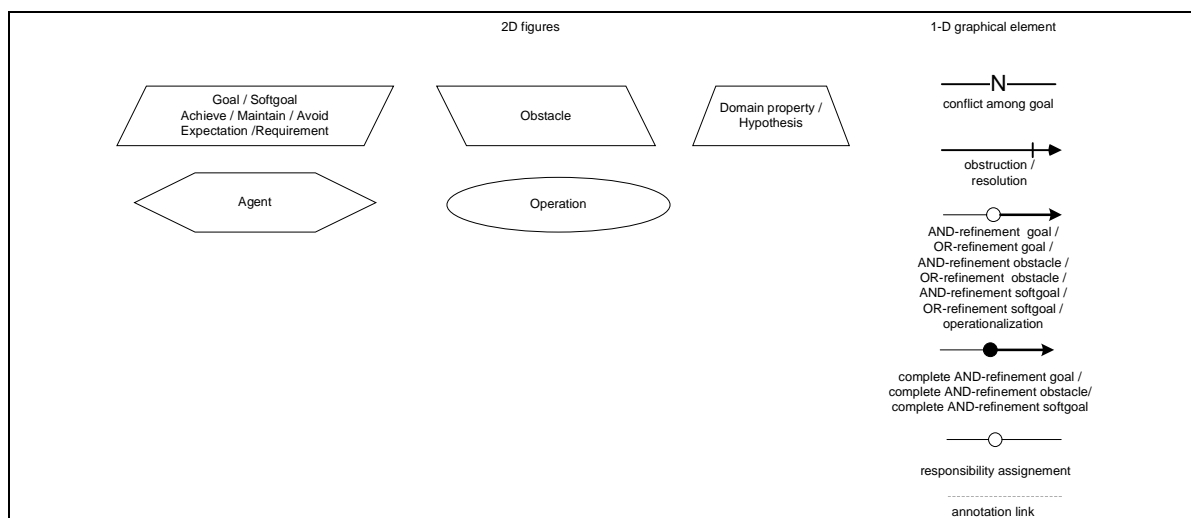


Figure 7-18 Using symbol deficit produces a smaller KAOS visual alphabet

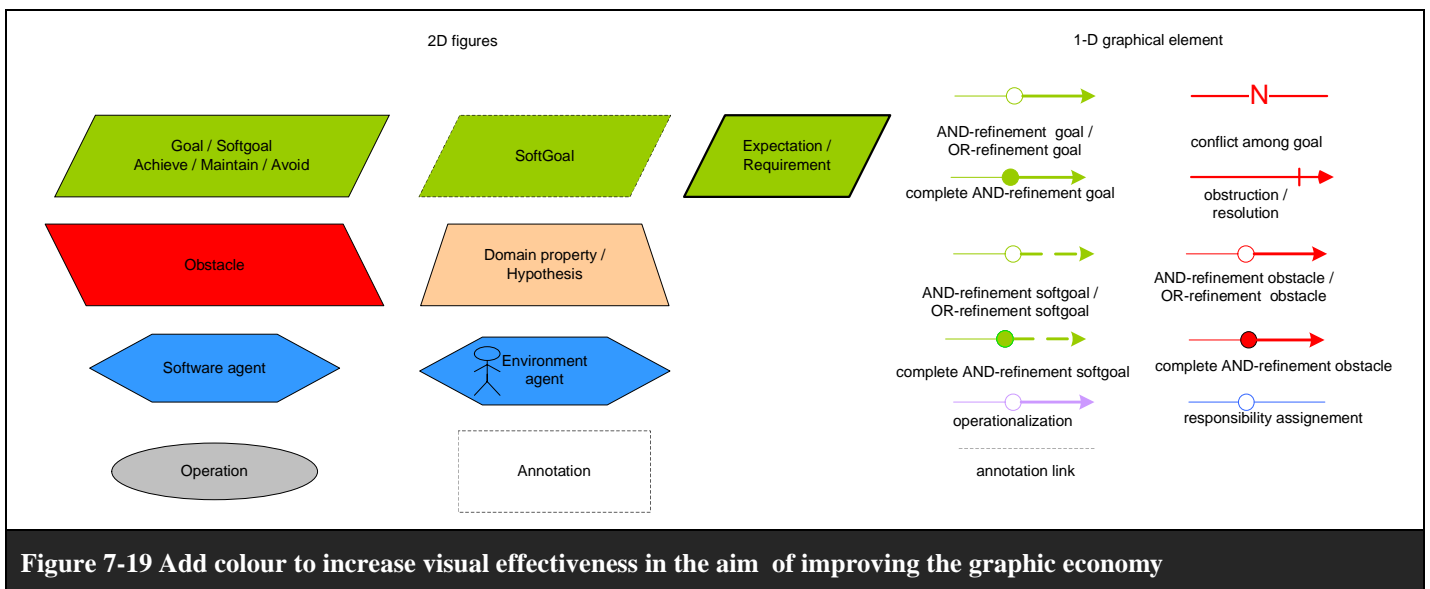
Using so much symbol deficit goes against the principle of visual expressiveness but in some cases (depending on the cognitive fit), this could be interesting.

Increase visual effectiveness

Increase visual effectiveness is a method that suggests not to reduce the number of symbols but to increase the human discrimination ability between symbols. Miller in [Miller, 1956] explained that the human ability to differentiate between stimuli can be expanded by increasing the number of perceptual dimensions on which stimuli differ. The 7 symbol limitation is applied only when a single visual variable is used in a diagram but when multiple visual variables are used to differentiate between symbols, it can increase human discrimination ability in an almost additive manner [Moody, 2009].

This technique produces a new symbol set that increases visual expressiveness by an order of magnitude (from 1 to 3), in this case graphic complexity should be manageable [Moody, 2009].

Applying this technique to the KAOS visual notation could consist of adding colour (a visual variable that is not used) to each symbol. Figure 7-19 details the new visual alphabet. On this picture we can observe that there are no more than 6 figures (plus or minus 2) with the same colour.



7.8 Principle of Manageable Complexity

The 2 last principles: manageable complexity and cognitive integration allow users to manage systems that represent a large amount of information.

7.8.1 Analysis results

The principle of manageable complexity focuses on the complexity of the diagrams. If the amount of information on a diagram is large, the reader (especially the novices) will have difficulties to understand it because he/she has perceptual and cognitive limits. To manage this complexity, Moody suggests 2 methods: the modularisation and the hierarchical structuring.

Modularisation is used to decrease the number of elements in a diagram by distributing them into smaller diagrams. This method respects the human cognitive limit which is limited by working memory capacity that can understand 7 plus or minus 2 elements at the same time [Miller, 1956]. This method is not used actually in the KAOS visual syntax.

Hierarchical structuring consists of grouping diagram information by level of abstraction, it helps to organise diagram into a coherent structure. As the KAOS language contains 5 different models that group information by abstraction (goal model, agent model, operation model, behaviour model and object model), then we can say the hierarchical structuring is used.

The KAOS language does not contain primary or even secondary notations that encourage the usage of duplicate visual representations to reduce line crossing. This is a good practice. Indeed, this method is not recommended because it attacks the symptom of the problem more than the cause. In other words, it means that it will reduce the number of elements of a complex diagram but without changing the way of producing a large amount of elements on the diagram. The second reason why duplicate visual representations should be avoided is that it decreases one of the primary cognitive advantages of diagrams: location indexing. It implies that the same piece of information can be found at many places, which is counterproductive. It was one of the cognitive advantages of the diagram over text: the information about a concept is at a single location [Cheng, 2004] [Larkin, et al., 1987].

7.8.2 Recommendations to improve the manageable complexity

As seen previously, the modularisation method is not used by KAOS. It can be applied at least to the goal model, because elements that are inside (goals and obstacles) are defined recursively (AND-refinement or OR-refinement). It is not the case in other models. Syntactic rules and graphical conventions have to be defined to automate the process.

As main syntactic rule, the drawer should modularise refinements. Then, the sub-diagram represents the parent goal or obstacle and its children. Thanks to this rule, new sub-diagrams do not overlap and allow keeping local indexing property. But we can have many exception cases like:

- If a parent has more than 8 children in its AND-refinement (then we will have more than nine elements in the diagram), the modeller will leave them together but we advise him to review his/her diagram and separate them below 2 or more parents to have different AND-refinements.
- If a parent produces more than 8 children with its different OR-refinements, we will suggest to do separate diagrams with the different refinements (but we will need to implement cognitive integration mechanism to do the link between them).
- If a parent and its children produce a really small diagram (less than 5 elements), we will suggest to respect the main rule. It will be useful for the evolution of the diagram.

For instance, the diagram representing the goal model of the running example is quite large and it contains more than 20 symbols, which violates the cognitive limit defined by Miller [Miller, 1956]. In figure 7-20, the goal model of the running example is modularised according to the previous rules.

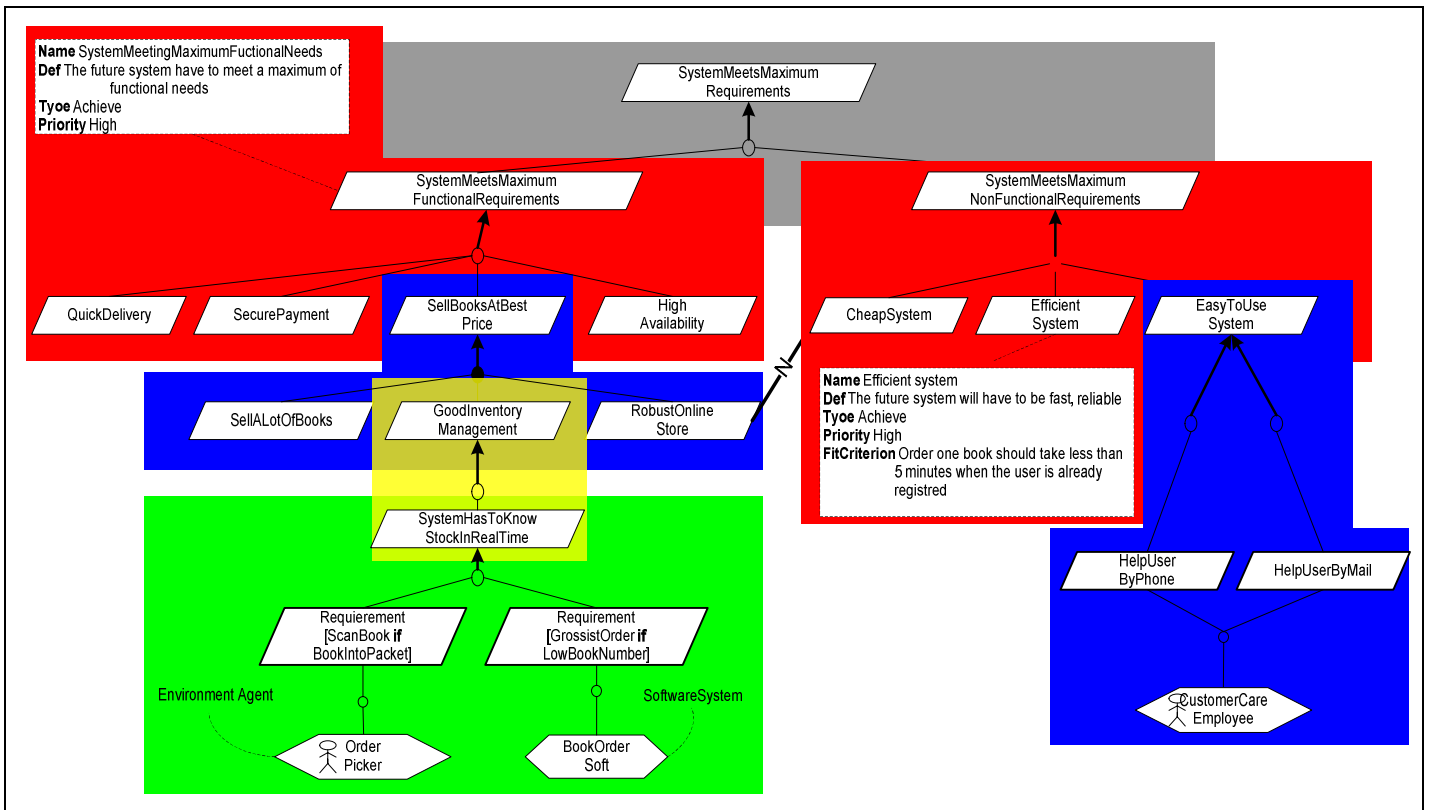


Figure 7-20 Goal model of the running example, suggestion of modularisation

Usage of modularisation can also be applied to agent model when agents are refined.

Contrary to the modularisation, the hierarchical structuring is included in the KAOS language due to its meta-model that is composed of 5 fragments. Each fragment represents a different model: goal model, agent model, object model, operation model and behaviour model. Then if modellers want to apply the technique of the hierarchical structuring, they have to be disciplined and respect the separation between the different models. It is not the case in figure 7-20, where agents are represented in the goal model. These shapes should be placed in the agent model.

7.9 Principle of Cognitive Integration

7.9.1 Analysis results

Cognitive integration is closely linked with the principle of management complexity. As explained in section 7.8, complexity management consists of reducing the size of the diagrams by arranging its elements in different sub-diagrams (or modules). Doing this, readers will need extra cognitive abilities to mentally integrate information from different diagrams and keep track of where he is [Siau, 2004]. There are 2 methods to improve the cognitive integration: the conceptual integration and the perceptual integration.

One of the biggest advantages of the KAOS visual notation is that semantic constructs are always represented by the same visual symbols through the goal model, operation model and agent model (i.e., those who do not use the UML visual syntax). For example, agents are always represented by a hexagon in the 3 cited models. This is very important for the cognitive integration of the reader.

KAOS uses only the hierarchical structuring method to manage the complexity but it has no method to deal with the cognitive integration of the different diagrams.

7.9.2 Recommendations to improve the cognitive integration

Mechanisms to improve cognitive integration have to be included in the KAOS language as it uses at least one mechanism to manage the complexity. Cognitive integration mechanisms become essential if we decide to use also the modularisation (as it is highly recommended).

Conceptual integration

Conceptual integration enables the reader to integrate information distributed across different diagrams into a coherent mental representation. If we decide to include modularisation as system to manage complexity, the mechanism should be applied 2 times. The first time to manage hierarchical decomposition diagram and the second time to manage modularisation sub-diagrams.

The first map represents the different models that are produced to study the system-to-be (figure 7-21).

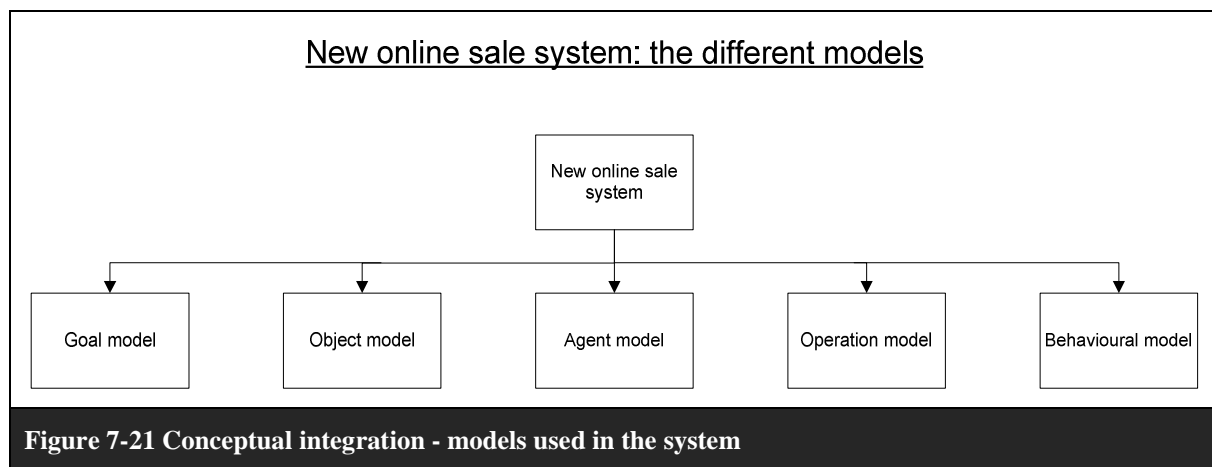


Figure 7-21 Conceptual integration - models used in the system

Then, for each model, we have to apply the mechanism to create general map that will help the user to locate each sub-diagrams (produced by the modularisation method) relative to the others. In the case of the goal model, this map will look like figure 7-20 but it will have some improvements:

- (i) Remove all annotations to simplify the drawing.
- (ii) Use different coloured/dashed line for each module.
- (iii) The module numbers are composed as following: x.y where x represents the level of the module and y the number of the module in its level (from left to right).
- (iv) The number of each module has to be written clearly inside each set that represents a module.

Another advantage of this summarised diagram is the visibility of the links between the sub-diagrams.

We also suggest to add a legend at the bottom of this map –or if there is not enough space– on another page. This legend contains the different numbers written on the map and gives a title for each of them. The titles appear on the top of each module. We can also notice the conflict link between the sub-diagrams 3.1 and 2.2. Figure 7-22 contains these improvements.

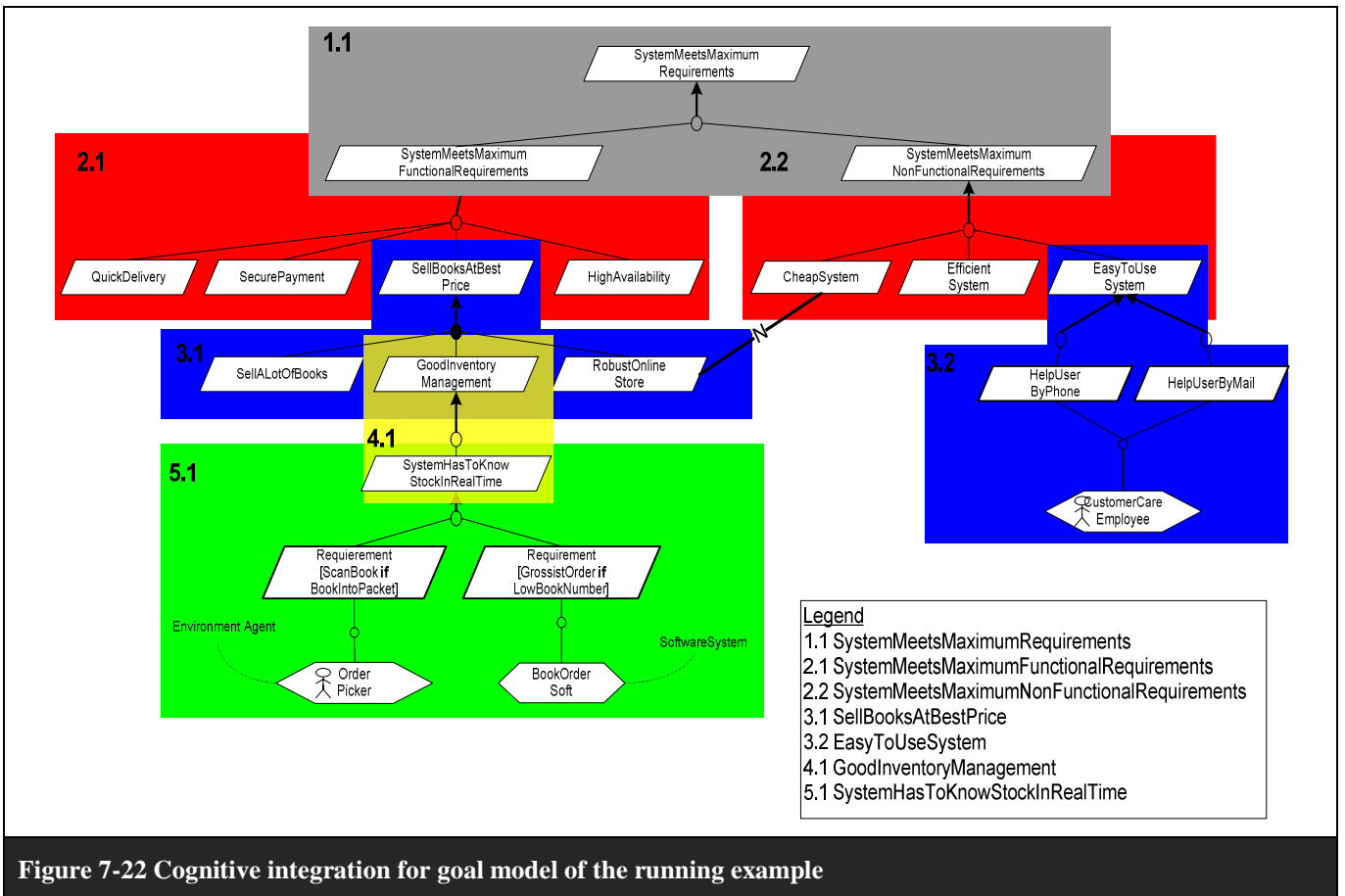


Figure 7-22 Cognitive integration for goal model of the running example

This kind of map has to be done for each modularised model.

Perceptual integration

Perceptual integration provides perceptual cues to assist navigation and transitions between sub-diagrams. It allows the reader to know where he is in the main diagram, and to know to which sub-diagrams he can go. To do this, we provide the following recommendations:

- (i) To facilitate navigation between the different modules, they have to be clearly labelled. At the top of each module, we suggest to add a title. This title corresponds to this one defined in the legend of the conceptual (e.g. 3.1 SellBooksAtBestPrice for the module 3.1).
- (ii) Inside a module, the elements (goals, obstacles or operations) that are themselves modularised should contain a fork sign to indicate that it is modularised and can be seen on another sub-diagram.
- (iii) Around the sub-diagram, modellers should add a frame that contains the number of the sub-diagrams surrounding the current one. It allows the navigation between the different sub-diagrams of the model.
- (iv) Outside of the frame described in point (iii), we suggest to put on the left a miniaturisation of the conceptual map at model level and on the right the conceptual map at system level. In these map, the place where the current module is situated is highlighted

Figure 7-23 represents the implementation of these recommendations for the sub-diagram 3.1 of the goal model represented at figure 7-22. The fork sign informs the user that the goal GoodInventoryManagement is also modularised and the arrow around the graph indicates to the users which sub-diagrams precede, follow or are sibling of this sub-diagram. In the conceptual map at system level, we can see that we are in the goal model and in the conceptual map at model level, we can see that we are in the module 3.1.

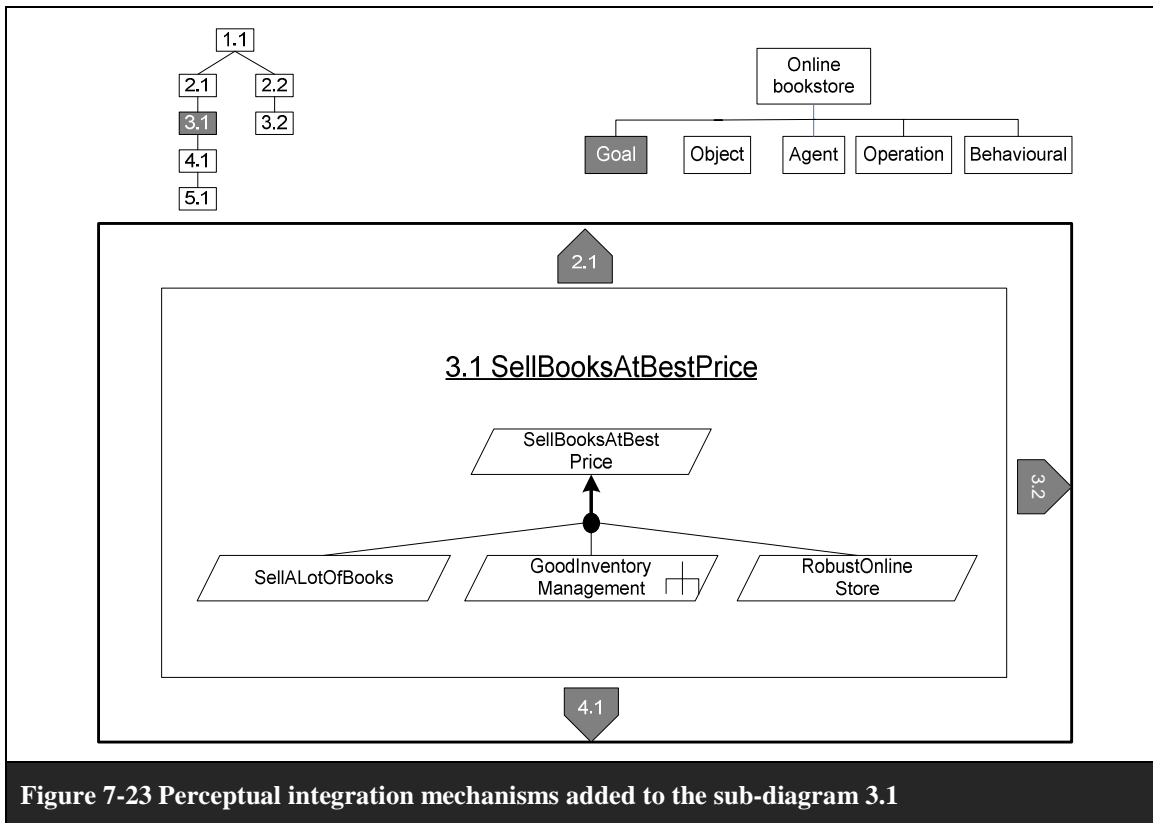


Figure 7-23 Perceptual integration mechanisms added to the sub-diagram 3.1

7.10 Summary

The table 7-3 summarises the 9 principles of the physics of notation, if needed, principles are divided into criteria that are evaluated individually.

Table 7-3 Recommendations grouped by principle of the Physics of Notation

Principle	Criteria	Evaluation	Recommendation
Semiotic clarity	symbol redundancy	➤ 1 redundant symbol	➤ no real utility, the redundant symbol can be removed
	symbol overload	➤ 43% of the symbols are overloaded	<ul style="list-style-type: none"> ➤ when semantic constructs are very close, textual differentiation is sufficient ➤ expectation and requirement can be easily differentiated ➤ suggestion to improve the differentiation between AND and OR refinement ➤ look for a good balance between symbol overloaded and graphic complexity
	symbol excess	➤ 2 symbol excess (annotation and its link)	➤ remove the shape inside which the annotations are placed but keep the link
	symbol deficit	➤ there is no symbol to express complete OR-refinement relationship	➤ add a symbol for complete OR-refinement
Cognitive fit		➤ one language for all usages	➤ foresee to create different language for different usages (experts vs. novices, detailed study vs. in a meeting, by hand vs. drawing software)
Perceptual discriminability	visual distance	➤ shape and texture are the only 2 visual variables that are used	➤ add colour to increase visual distance
	redundant coding	➤ not used	➤ use colour in the background of the shapes
	primacy of shape	➤ a shape can represent different concepts	➤ look for a good balance between primacy of shape and graphic complexity
	perceptual pop out	➤ few elements does not pop out (mostly relationship link)	➤ no specific recommendation as the problem is limited
	textual differentiation	➤ frequently used	➤ look for a good balance between textual differentiation and graphic complexity
	visual-semantic congruence	➤ shapes respect the visual-semantic congruence	➤ no specific recommendation
Semantic transparency		➤ shapes are mostly abstract, conversely they are semantically opaque	➤ use symbols that help the user to remember the semantic meaning
Visual expressiveness		➤ equals to 2 on a scale that varies from 1 to 8	<ul style="list-style-type: none"> ➤ use 3D shapes ➤ use colour in an effective way and to show priority of goal and the dangerousness of obstacles
Dual coding		➤ used only for annotations	<ul style="list-style-type: none"> ➤ do not put annotations in a specific shape ➤ add name to the relationship links ➤ standardise the label of element

Graphic economy		<ul style="list-style-type: none"> ➤ manageable due to fact that the meta-model can be divided into 5 models 	<ul style="list-style-type: none"> ➤ use symbol deficit ➤ increase visual effectiveness
Manageable complexity		<ul style="list-style-type: none"> ➤ the 5 models allows to do hierarchical structuring ➤ no primary or secondary notation that advices duplicating elements 	<ul style="list-style-type: none"> ➤ introduce modularisation technique
Cognitive integration		<ul style="list-style-type: none"> ➤ a same element represents always the same concepts in the different model ➤ conceptual integration and perceptual integration are not implemented 	<ul style="list-style-type: none"> ➤ introduce conceptual integration mechanism ➤ introduce perceptual integration mechanism

Chapter 8 Recommendations

In this chapter, we will formulate recommendations for 3 different usages. They are based on those previously presented and discussed in Chapter 7. Moreover, we have to keep in mind that there are interactions among principles: when a principle is improved, sometimes another one is decreased. The set of these recommendations should lead to 3 coherent versions of KAOS with a good balance between principles.

As seen in the cognitive fit principle section, the syntax of the language should fit to 3 parameters: the users, the media and the task characteristics. We have chosen 3 situations where the visual notation could be improved to facilitate its use.

The first group of recommendations is for language engineers to improve the visual notation for the novices. As these persons are not used to work with graphical representations, they have to be adapted to be cognitively effective.

The second group of recommendations is for meeting participants that can be IT experts as well as business stakeholders. During the meeting, diagrams have to be drawn quickly (otherwise participants will have the feeling to waste their time) and generally in meeting rooms we dispose only of sheets of paper and some colour pens.

The last group of recommendations is dedicated to developers who will create software that implements the KAOS visual notation. The future software will produce visual representations that could be read by different kinds of public (experts as well as novices).

We will start to give general recommendations that are applicable in all situations, and then we will define specific ones depending on the group to which we want to give pieces of advice.

8.1 General recommendations

We will start this chapter with general recommendations that are applicable in all situations. The table 8-1 summarises these recommendations.

1. Improve the semiotic clarity

Improve the semiotic clarity by removing the redundant symbol for soft goal (the right-oriented parallelogram with dashed border) to avoid any confusion. To remove the symbol deficit anomaly, we have to add the relationship link presented at figure 7-3 to represent the OR-refinement relationship link.

2. Use annotations appropriately

Use annotations only when it is appropriate to avoid overloading diagrams. The text will be associated with a dashed line to the element that it clarifies (e.g., figure 7-16). Details and proprieties of semantic constructs will be placed in a separated document.

3. Standardise the label of elements

If possible, the label of a goal should be a word (that represents the subject) followed by a verb in its passive form. Conversely, the label of operations should be composed of a verb followed by words to describe the action to avoid any confusion.

4. Use vertical position to reinforce the refinement links

Use vertical position to reinforce the refinement links. The object (e.g., goal, obstacle and agent) that will be refined should be put on the top of its sub-objects. Using this rule is a good practice and helps to increase the semantic transparency.

Table 8-1 Summary of the general recommendation

Principle	Recommendation
Semiotic clarity	<ul style="list-style-type: none">➤ Remove redundant symbol for soft goal➤ Add a symbol for OR-refinement
Dual coding	<ul style="list-style-type: none">➤ Use annotations only when appropriate➤ Standardise the label of element
Semantic transparency	<ul style="list-style-type: none">➤ Use vertical position to reinforce the semantic of refinement links

8.2 Recommendations for language engineers to improve visual notation for novices

In this section we will add a set of recommendations to the general ones to improve the original KAOS visual. The target audience of these recommendations is language engineers. Its goal is to improve the visual notation in case it is used by novices. Novices can be stakeholders of the project that do not belong to the software development team, such as future users of the system or managers of the company. Consequently, they are not familiar with diagrams and technical visual notations. However, if these tools are effectively used they could become powerful.

As seen in Chapter 7, the KAOS actual visual notation suffers from lacks. The following recommendations should increase its cognitive effectiveness. They are summarised in table 8-2.

1. Increase absolute discriminability

Following the principle of discriminability, the size, the contrast and the proximity of elements should follow some recommendations.

Elements should have minimal size, as it is explained in [Moody, 2006], to be easily readable. According to [Moody, 2006], if it is possible all elements of a same type should have the same size to discriminate them easier.

To discriminate elements from the background, elements should have clearly different surface properties compared to the background. As the main property of the background is colour, we suggest colouring the background of the shapes to clearly distinguish them. With this recommendation, we suggest the use of colours but without imposing colour values. Pieces of advice about these values are given in the next recommendation. But, as a general rule, the default colour should be contrasted enough with the background.

And finally, elements can not be put at random on the diagram; it should be a minimal distance between them.

2. Increase relative discriminability

Like in the previous paragraph, our suggestion aims at increasing the discriminability. The first one concerns the concepts of goals and obstacles. As their figures are very similar (a right-oriented parallelogram for goals and a left-oriented parallelogram for obstacle), we have to increase the visual distance between them. To achieve this, we will use dual coding and suggest the use of traffic light colours to denote the different semantic meaning.

As diagrams are usually drawn on white background, we suggest the following combinations (proposed in section 7.5.2): goals in green, obstacles in red, domain properties and hypothesis in orange, agents in blue and operations in grey.

We have chosen these colours, mainly for their meaning:

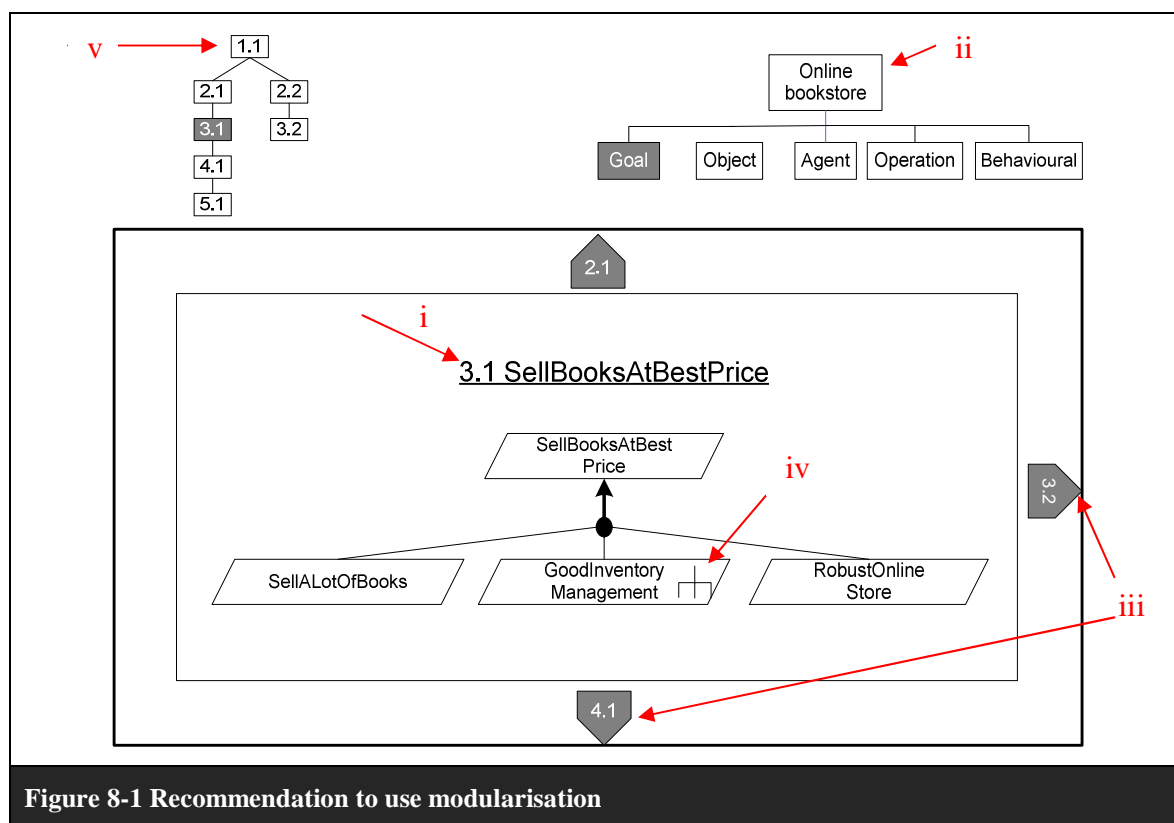
- green for goals because they are positive idea that we have to reach,
- red for obstacles because they are negative elements or events that could prevent us to reach goals,
- orange for domain properties and hypothesis because we have to take them into account and keep them in mind,
- blue for agent (as the uniform of policeman),
- grey for operations (as manufacture).

3. Use of modularisation

As goal model diagrams can be very large (e.g., the goal model of the running example, figure 9-1), diagrams should be modularised to contain 7 elements plus or minus 2 [Miller, 1956]. Each module of a diagram will be on a separate sheet. For the clarity of diagrams and to help the user navigate between the different pages, each one should contain:

- i. a title with the number (composed as following: x.y where x represents the level of the module and y the number of the module in its level –from left to right) and the title of the module.
- ii. in the right top corner, the user should find a diagram with the model used in the system. The model that contains the current module should be highlighted.
- iii. around the diagram, it should be information about the modules surrounding the concerned module.
- iv. elements that are themselves modularised should be marked by a 'fork'.
- v. a summary of the current model –this summary will contain only the modules of the model– with the current module highlighted; it allows the user to know the context of the diagram.

An example of this recommendation is illustrated in figure 8-1.



4. Improve cognitive integration at system level

As suggested in [Moody, 2006], improving the cognitive integration is going in pair with the navigation map/diagram that helps the user to know where he is. Modellers have to draw a general map that shows all developed KAOS models in the project. This map is used to guide the user. Models that have been analysed can be highlighted as represented on figure 8-2.

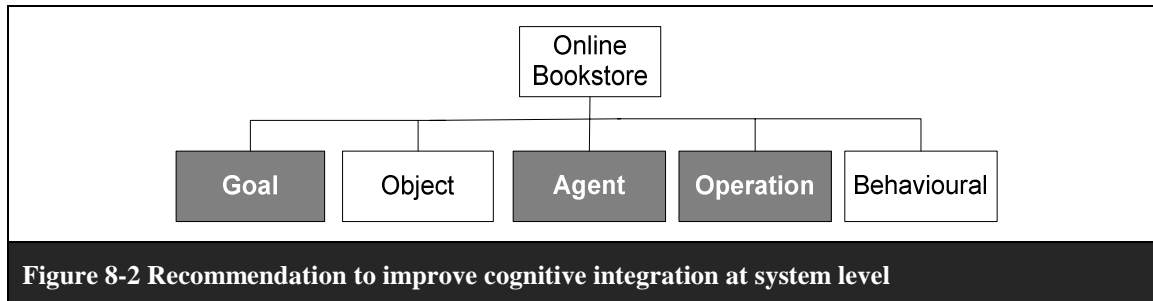


Figure 8-2 Recommendation to improve cognitive integration at system level

5. Improve cognitive integration at model level

To improve the cognitive integration at model level, we recommend creating a general map that will help the user to locate each sub-diagrams (produced by the modularisation method) relative the others. To avoid overloading the map, we suggest to:

- remove all annotations to simplify the draw
- use different coloured/dashed line for each module
- the module numbers are composed as following : x.y where x represents the level of the module and y the number of the module in its level (from left to right)
- the number of each module has to be written clearly inside each set that represents a module.

Another advantage of this summarised diagram is the visibility of the links between the sub-diagrams.

Finally, we suggest adding a legend at the bottom of this map –or if there is not enough space– on another page, this legend will take the different numbers written on the map and give a title for each of them. These titles will have to appear on the top of each module. An example of cognitive integration map at model level can be seen at figure 7-20.

6. Improve the cognitive integration by using a legend

Diagrams should contain a legend with the figures and relationship links used in diagram on the sheet.

7. Improve the semantic transparency

To help the novice user, using symbols instead of abstract figures is more powerful and increase the cognitive effectiveness of the latest. However, transforming the abstract figures commonly used in the KAOS visual notation into symbols will destabilise experimented users. Then to avoid this problem, we suggest miniaturising these symbols and putting them inside the abstract figures.

We could use the symbol represented in figure 7-9.

Figure 8-3 depicts an example of abstract shapes containing symbols to improve their semantic transparency.

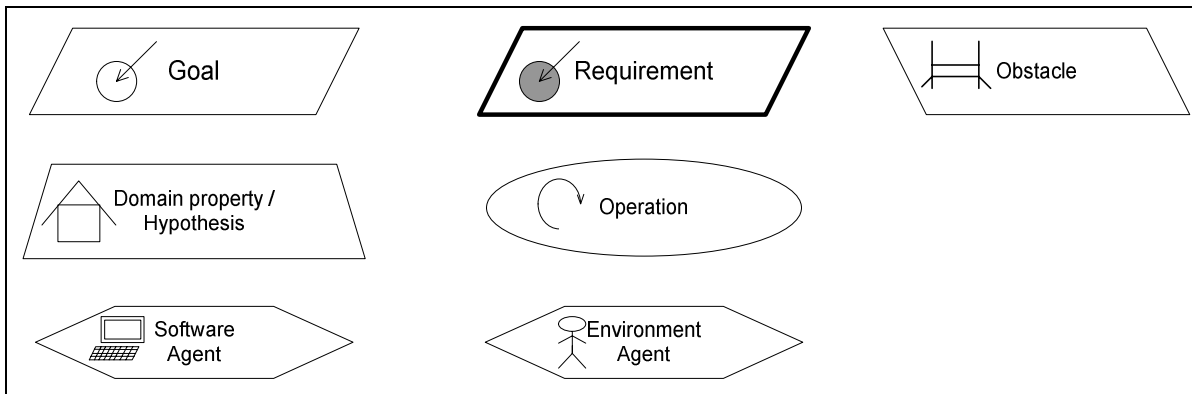


Figure 8-3 Improve semantic transparency by adding symbols inside the shapes

The choice of the dart target to represent goal has another advantage: it reinforces the semantic transparency because it can be coloured in black to represent expectation or requirement.

8. Improve the semantic transparency a little bit more

If we add the previous recommendation to the recommendation number 2 (increase relative discriminability), figures of the KAOS language will have a visual distance sufficient to be clearly distinguished and symbols become semantically transparent without changing the initial shape. Figure 8-4 summarises the 2 recommendations.

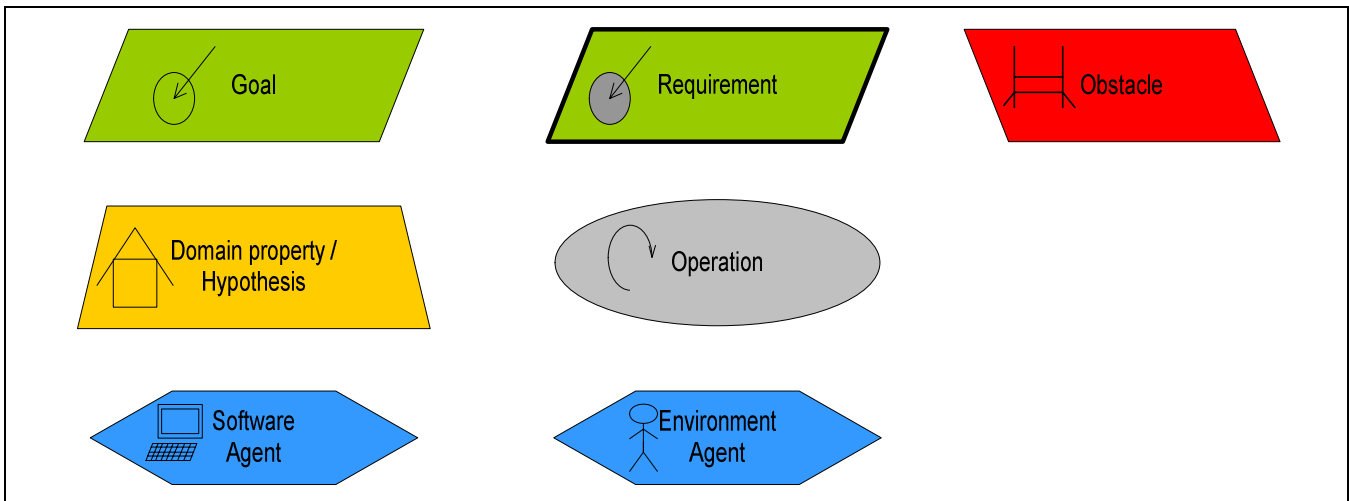


Figure 8-4 Combine colour and symbol to improve semantic transparency

9. Improve the visual expressiveness

Improve the visual expressiveness by using one more visual variable: value. The latest can be used to show the priority of a goal or an obstacle. The darker is a goal, the most it is important to reach it. The darker is an obstacle, the more dangerous it is. An example can be seen on figure 7-12.

Table 8-2 Summary of the recommendations for language engineers

Principle	Recommendation
Increase absolute discriminability	<ul style="list-style-type: none">➤ Elements should have the same size➤ Elements have to be separated by a minimal size➤ Use coloured background for elements to differentiate them for the background of the diagram
Increase relative discriminability	<ul style="list-style-type: none">➤ Add specific colours background to elements
Manageable complexity	<ul style="list-style-type: none">➤ Use modularisation
Cognitive integration	<ul style="list-style-type: none">➤ Draw a map at system level (conceptual integration)➤ Draw a map at model level (conceptual integration)➤ Add navigation cues to each module (perceptual integration)➤ Add symbol legend
Semantic transparency	<ul style="list-style-type: none">➤ Add symbols into shapes➤ Add specific colours to improve semantic transparency (traffic lights)
Visual expressiveness	<ul style="list-style-type: none">➤ Use the 'value' visual variable to show the priority of goals or the dangerousness of obstacles

8.3 Recommendations for meeting users

In this section, we add recommendations to the general recommendations for people who are working during a meeting. These people are not familiar to work with goal-oriented languages. They belong to the stakeholder group and can be end-users, sellers, managers. During a meeting, one of them can start to draw a diagram to explain his/her idea's to other participants. Thanks to this diagram, every participant should be able to understand the problem, give his/her opinion and if he/she agrees with the proposition, validate the solution.

As this diagram is done in a meeting, it has to be drawn quickly to avoid that people who are not drawing have the feeling that they waste their time. As media, meeting participants will generally use a sheet of paper (A4 or A3), a pen and sometimes colour pens (we advice having some fluorescent pen which are always useful).

The set of recommendations is summarised in table 8-3.

1. Increase absolute discriminability

As drawings are done on papers, it is not easy to add a background to the figure to distinguish between them. Anyway, we recommend to the modellers not to put elements too close from each other for 2 reasons. The first one is to follow the discriminability principle and the pieces of advice given in [Moody, 2006]. The second is to foresee the future development of the diagram. Indeed when a modeller starts to draw a diagram in a meeting, he will never know which parts will be developed.

2. Increase semantic transparency

There are 2 ways to increase semantic transparency: using colours and/or adding a symbol.

If the meeting participants have some colour pens, they can use them to draw the border of the shapes to increase the semantic transparency. They could use green for goals and red/orange for obstacles. These colours are chosen in reference with traffic light as explained in section 7.5.2. Figure 8-5 illustrates this recommendation.

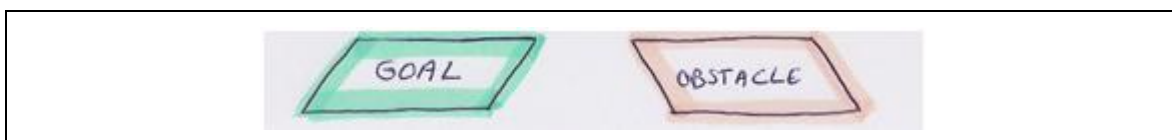


Figure 8-5 Recommendation for meeting users - Increase Semantic transparency by using colours

The second method consists of adding a symbol in the shape to increase its semantic transparency. As symbols suggested in figure 7-9 could be a little bit long and difficult to draw, we suggest to use the sign '+' for goals or '-' for obstacles inside the figures to help stakeholders to remind the meaning of the figures. Figure 8-6 illustrates this recommendation.



Figure 8-6 Recommendation for meeting users - Increase Semantic transparency by using + and - signs

3. Improve the semantic transparency for priority

The semantic transparency of the attribute priority can be improved by using 1, 2 or 3 times the symbol '+' or '-' added in the goal and obstacle figures. The more there are signs, the more a goal is important or an obstacle is dangerous. In figure 8-7, the goal 1 is more important than goal 2 and goal 3.

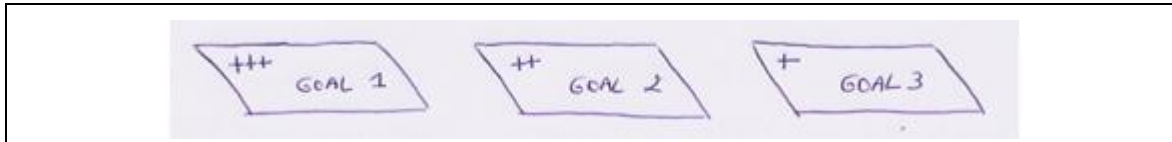


Figure 8-7 Recommendation for meeting users - Increase Semantic transparency by adding 1 or many times the sign that represents goals

4. Improve graphic economy

As diagrams have to be understood by novices, the graphic complexity should be as small as possible. To do it, we recommend using the technique of the symbol deficit. As discussed in the section 7.7.2, the symbol that represents requirement and expectation is not needed (a right-oriented parallelogram with a bold border) and this one which represent environment agent can use the same as the software agent.

Table 8-3 Summary of the recommendations for meeting users

Principle	Recommendation
Increase absolute discriminability	➤ Elements have to be separate by a minimal size
Semantic transparency	<ul style="list-style-type: none"> ➤ Add symbols into shapes ('+' for goal and '-' for obstacles) ➤ Add specific colours to improve semantic transparency ➤ Use from 1 to 3 symbols that represent goal or obstacle to show the priority and the dangerousness
Graphic economy	➤ Use symbol deficit

8.4 Recommendations for software developers

This third set of recommendations is written for the requirements engineering software developers who would like to create a tool suitable to draw diagrams in KAOS language.

As the aim of these recommendations is to build a new software (or improve an existing one), the media that will be used is a screen, interactions between the users and the software can be done via a mouse and the users know the KAOS visual notation at different levels (from novice to expert).

1. Increase absolute discriminability

This recommendation is the same than the first one for the language engineers.

Firstly the software should be able to adapt the size of the shapes to give them the same size and there should be a minimal distance between them. This distance could be fixed in the diagram settings to avoid not putting elements too close from each other.

Secondly, users should have the possibility to put a coloured background to the shape.

And finally, elements can not be put at random on the diagram. Following the piece of advice given in [Boucher, 2008], users would like a mechanism that automatically reorganise elements. The idea is to select some or all elements and reorganise them on basis of a reorganisation algorithm which respect the minimal distance. This mechanism could also involved a direct interpretation (e.g., if one node has a tree layout, the position of sub-elements suggests that they are leaves of the root) because the human mind will group elements together according to their position or the structure they have.

2. Increase relative discriminability

This recommendation is the same than the recommendation number 2 for the language engineers.

To remind, it suggests the use of traffic light colours to improve the relative discriminability.

3. Usage on modularisation on screen

As goal model diagrams can be very large and as seen in section 7.8.2, this kind of diagram can be modularised, then the software should offer the possibility to see –or not– some parts of the diagram. This can be done thanks to a button to hide or unhide some parts of the refinement trees of the diagram. With this functionality, the software user will be able to see the part of diagram on which he wants to work.

The button could represent the sign '+' to show the user that he can see more details clicking on the button and when the tree is expanded at its maxim, the button will contain the sign '-' to show to the user that he can hide this part of the diagram.

Figure 8-8 illustrates an example of this recommendation. On the left of the figure, the figure that represents goal `SystemHasToKnowStockInRealTime` contains a '+' sign. It means that this goal is modularised and details can be seen if you click on the '+'. The result is on the right part of the figure.

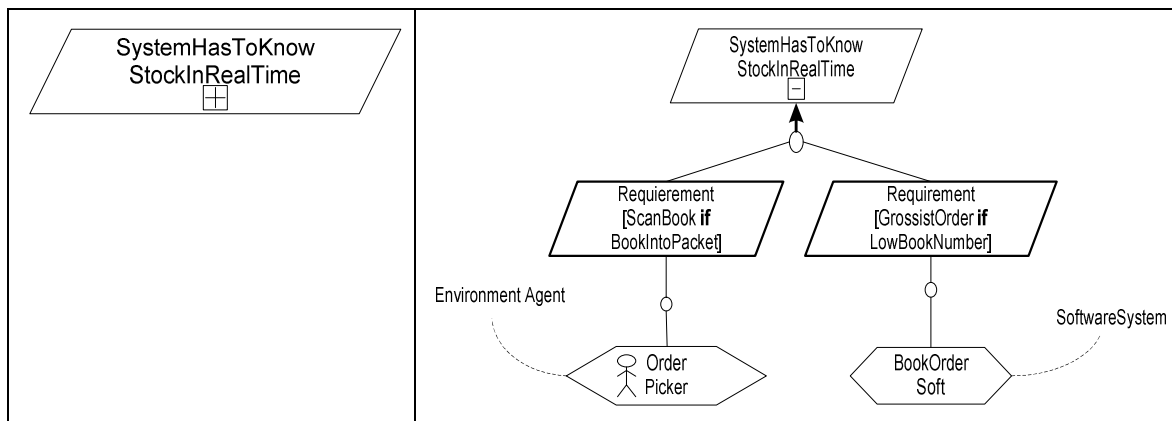


Figure 8-8 Recommendation to use modularisation

4. Use of modularisation when printing

If the user wants to print one of his/her models (that will be certainly bigger than an A4 sheet), the software needs to be smart enough to suggest him/her a modularisation of the model in a print preview mode. The user will be free to accept it or to refuse it and in this case, he will be able to create himself/herself the different modules. Each module will be printed on a different page.

The algorithm that will suggest this modularisation will be based on the fact that a module should not contain more than a root node and its 6 leaves. But it will have to foresee that sometimes there are exceptions as seen in section 7.8.2.

When printing, the software should offer an option to add navigation cue as described in the recommendation number 3 for the language engineers. The main points of this recommendation are:

- i. a title with the number and the title of the module,
- ii. in the right top corner, place general map of the project ,
- iii. around the diagram, place information about the modules surrounding the concerned module,
- iv. elements that are themselves modularised should be marked by a 'fork',
- v. a summary of the current model –this summary will contain only the modules of the model– with the current module highlighted.

5. Improve cognitive integration

As suggested in [Moody, 2006], improving cognitive integration, is going in pair with navigation map/diagram that will help the user to know where he is. The software tool should help the user to create a whole project that will use all KAOS models.

Figure 8-9 represents a screenshot of the future software that will implement the KAOS visual notation. It is composed of 5 windows: the project window, the concept window, the global view, the annotation window and the current view window.

In the top left corner, there is the 'Project window' (i). It provides a general map to guide the user. This general map contains the models used (or already described) in the project. When clicking on a model, the other windows will be adapted to the chosen model.

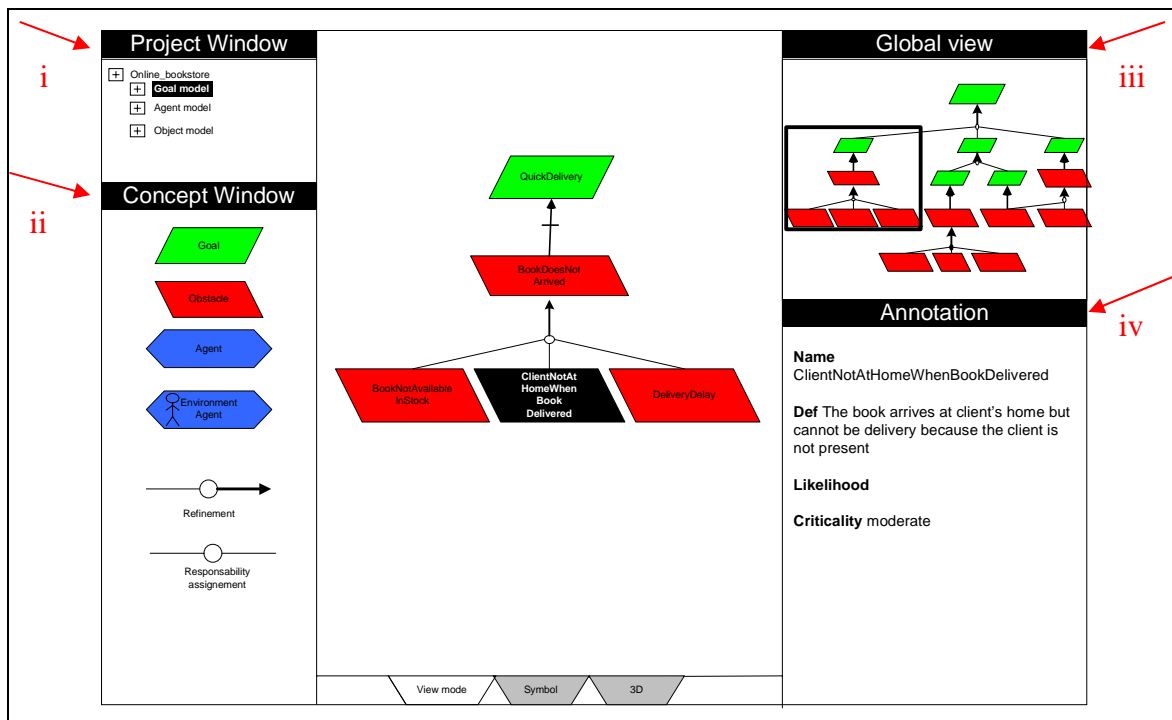


Figure 8-9 Screenshot of the software that will implement KAOS

The concept window (ii) will contain the shapes that are available for the selected model. It will help the user to know the possibilities he/she has to draw his/her diagram. In figure 8-9, the goal model is chosen, then the concept window contains goal, obstacle, software agent, environment agent and some relationship links. The user can add them to the current diagram by doing a drag-and-drop.

To help the user to navigate inside a specific model, there is a frame called "global view" (iii), it contains a whole miniaturised model. In this frame, there will be a rectangle that can be minimised, maximised and that can move. Objects that appear inside the rectangle will appear in the current view window in real size.

Annotations should not be on the diagram. In the software tool, element annotations should be put and organised in the annotation window (iv). This window summarises the characteristics of a selected object (e.g., in figure 8-9, these are the characteristics of an obstacle). The user will be free to make appear or disappear the annotation window. But if the user wants to add information on the diagram, it should have this possibility. He should do it only when the annotation has a real added value. This annotation should not be in a specific figure but linked directly to the concerned element with a dashed line.

In figure 8-9, the obstacle *ClientNotAtHomeWhenBookDelivered* is highlighted (it is in black while the other obstacles are in red) and we can see in the annotation window, the different characteristics of this element.

5. Improve semantic transparency

To help the novice user, using symbols instead of abstract figures is more powerful [Moody, 2009] and increase the cognitive efficiency of the latest. The future tool could offer a view of the diagram with symbols instead of abstract figures. Users could switch from the usual KAOS visual representation to the symbol view and conversely. Symbols described in figure 7-9 can be reused but the tool should also provide the possibility to change the different figures because as explained in section 7.4.2, symbols could have different meaning in the mind of the stakeholders depending on their cultural background.

Figure 8-10 shows the symbol tab (i) of the goal model. All abstract shapes have been turned into symbols that are friendlier for novice users such as end-users.

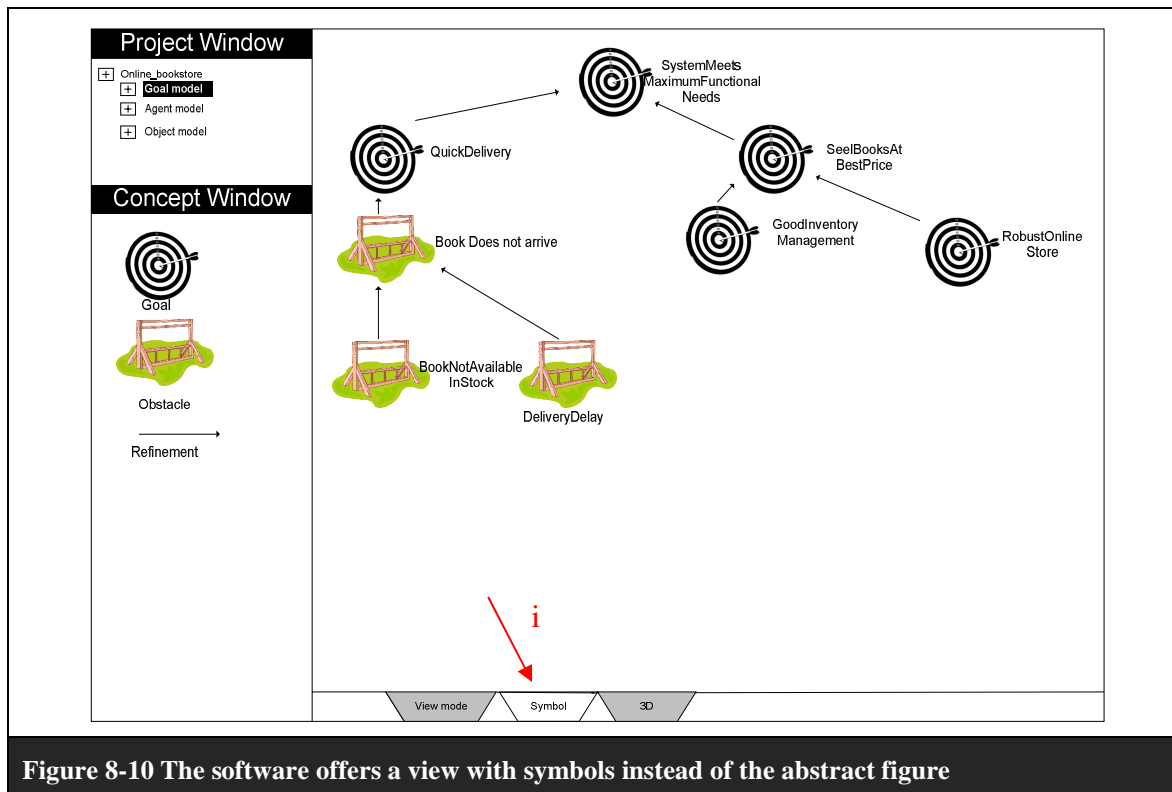


Figure 8-10 The software offers a view with symbols instead of the abstract figure

6. Improve the visual expressiveness by using 3D-shapes

As discussed in section 7.5.2, 3D shapes are preferred by users. The software should offer a third tab to view the diagrams in 3D as shown on figure 8-11.

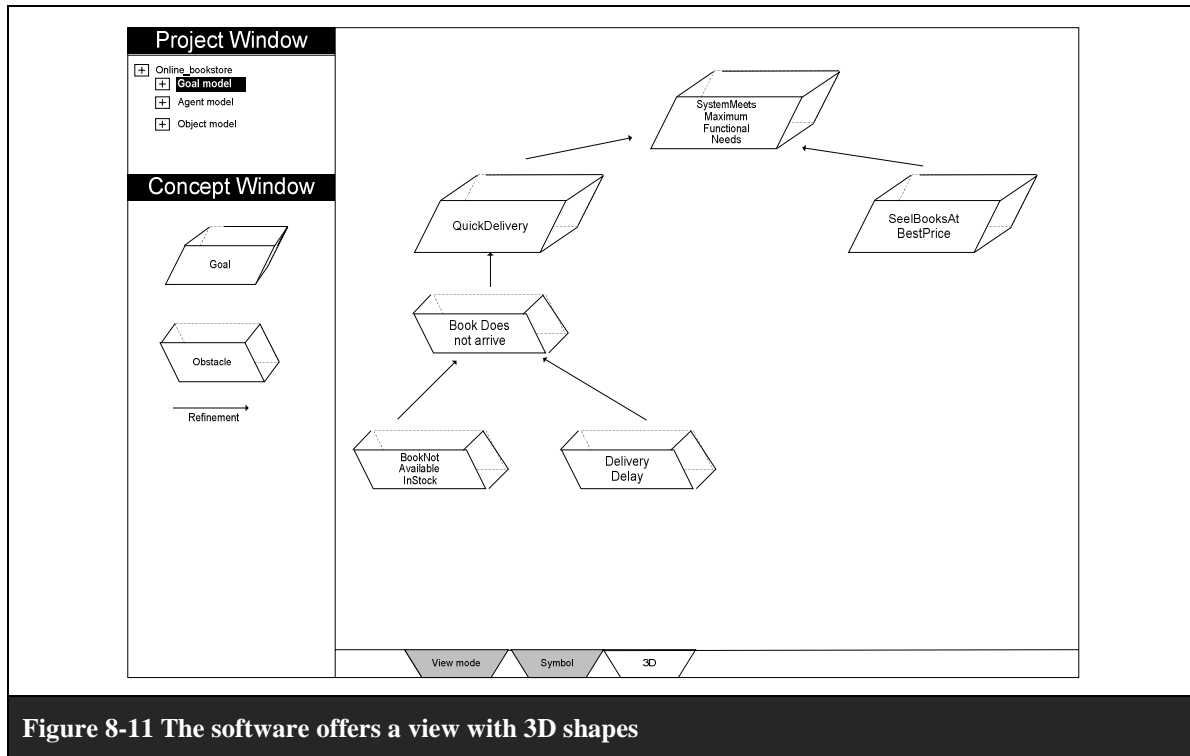


Figure 8-11 The software offers a view with 3D shapes

7. Improve the visual expressiveness

Improve the visual expressiveness by using one more visual variable: the value. The latest can be used to show the priority of a goal or an obstacle. The darker is a goal, the most it is important to reach it. The darker is an obstacle, the more dangerous it is.

8. Improve perceptual discriminability by using a legend

Among the printing options, the user should have the possibility to add a legend below each module. This legend should contain symbols and relationships used in the module.

9. Provide help to users

The software should contain at least a manual tool and a tutorial to explain to the user the different abilities of the software. These tools will explain him/her the advantages of the options and the effects they could produce on the future reader. They will also show how to use the product in a more efficient way. The tutorial should be a video that explains in few minutes the main functionalities of the tool while the user manual will describe the tool in details.

Table 8-4 Summary of the recommendations for software developers

Principle	Recommendation
Increase absolute discriminability	<ul style="list-style-type: none"> ➤ Elements should have the same size and have to be separate by a minimal size (algorithm to reorganise elements) ➤ Use coloured background for element to differentiate them from the background of the diagram
Manageable complexity	<ul style="list-style-type: none"> ➤ Provide modularisation algorithm to print easily ➤ Use technique to show/hide elements of modules on the diagrams
Cognitive integration	<ul style="list-style-type: none"> ➤ Draw a map of the models that are developed (via project window) ➤ Add navigational information on screen (via global view) ➤ Add navigational information when printing ➤ Suggest to add a legend when printing
Increase relative discriminability	<ul style="list-style-type: none"> ➤ Add coloured background to elements
Semantic transparency	<ul style="list-style-type: none"> ➤ Add symbols into shapes ➤ Add specific colours to improve semantic transparency (traffic lights) ➤ Add a view to see only symbol
Visual expressiveness	<ul style="list-style-type: none"> ➤ Use the 'value' visual variable to show the priority of goal of obstacle

Part III

Illustration

Chapter 9 An illustrative example

In this chapter we describe a running example that is used to illustrate suggestions and recommendations we proposed in Chapter 5. This example is based on an online bookshop.

9.1 Context Description

9.1.1 Online discount bookstore

The bookstore "Oh my book" would like to increase its sales and its CEO and the shareholders decide to sell books via internet. This online bookstore has the goal to sell best sellers at the lowest price of the market. There is no legacy software, so designers are given a free hand to build a new system from scratch.

9.1.2 What are the different activities?

Selling books on the Internet requires structuring and synchronising several activities. The main activities are: create a sales platform, manage online sales, manage book stock, manage customer care department and organise communications with outside world. They are the core activities necessary for starting online sales. However, an evolution phase is already scheduled to check if any improvements on the current system are needed.

Create and maintain a sales platform (website)

To sell books online, our system has to contain a sales platform that will have to offer: quick response time (the user will not have to wait during long time delay before seeing the result of his/her action), and a high availability (it should be accessible nearly all the time).

The website will contain a catalogue of books. Users will have many possibilities to consult it: by author, by ISBN number, by language or by category.

If a user wants to buy one or many books, first he has to identify himself/herself (he/she can already be known by the system or not). Then, he/she will choose the book(s) to buy and the quantity and finally he/she will have to pay it/them.

The platform will have to be highly secure as well for personal data as for the payment. If possible, there will be many possibilities to pay online (not only by credit card).

When the order is complete, the customer will have the possibility to track his/her order and to consult the state of his/her order. Some of the steps could be: done order, paid order, packed order, sent order and delivered order.

Manage online sales

When a customer has ordered one or many books and has paid his/her order, order pickers have to prepare the package as well as possible. A package is well done if it is quickly prepared and does not contain any mistakes: the exact quantity of ordered books in the package. Then the customer address will have to be pasted on the top of the package (the right address on the right package).

Manage book stock

Book stock has to be up-to-date for many reasons. Firstly, it is more efficient to prepare packages as quickly as possible (because if a customer bought a book that has the status 'in stock' on the website but in fact it was not in stock, he/she will have to wait before receiving his/her package). The second advantage is that book stock managers know when they will have to re-order books at the wholesaler and it will avoid having some books 'out of stock'. And finally it is one of the keys to have the best prices of the market.

To know the book stock in real time, order pickers have to update it every time a book is placed in a package. If a book is returned by a customer, order pickers have to decide if it can go back –or not– to the stock. If the book can be placed back in the stock, they have to do the necessary to update the number of this book in the database. Stock managers play also an important role in this task: when new books arrived from a wholesaler, they have to count them and add them to the stock.

Manage customer care department

To satisfy our customers at best, we have to offer a customer care service. The latest will have to answer to the customer questions (e.g., about books, delivery delays) or try to solve problems that might occur during an order (e.g., the payment is not done, the package does not contain the ordered books).

Organise communication plan

To have new customers and to constantly increase sales, a team will have to establish a communication plan. This plan consists of a marketing plan for the bookstore website, a marketing plan to put advertisements for other websites on the bookstore website and newsletters to customers.

9.1.3 Who are the different stakeholders?

User

A user is any person who visits the "Oh my book" website. He/she can have discovered it by advertising, by clicking on a link, by a friend or via another website.

Customer

A customer is a user who buys/has bought one or many books on the bookstore website.

Employee

An employee is any person who works for the company 'Oh my book'. He/she can occupy one or many functions in the system: customer care employee, order picker, book stock manager, marketing employee and IT employee.

Customer care employee

A customer care employee tries to help the users and/or customers. He/she will reply to customer questions and help them if they encounter any problem during their orders.

Order picker

An order picker employee is an employee who prepares the packages that contain the customer orders.

Book stock manager

A book stock manager is an employee who checks the book stock. When it is necessary, he orders books to the wholesaler.

Marketing employee

A marketing employee is an employee who works on the communication plan and tries to make publicity for the company and its website.

IT employee

An IT employee is an employee who works in the IT department. He tries to fix any problem that can occur on the website and develops new website functionalities.

Wholesaler

A wholesaler is a person or another company that sells books wholesale. To reach our goal to sell at best price, the wholesaler who sells at the lower price will be chosen.

External website manager

An external website manager is any person who has its own website and would like to put some advertisements about his/she website on the bookshop website.

Company owners

The company owner(s) is the person or a group of persons to whom the company belongs.

Shareholder

A shareholder is any person who has financial interest in the company.

Figure 9-1 represents the different stakeholders of the online bookstore.

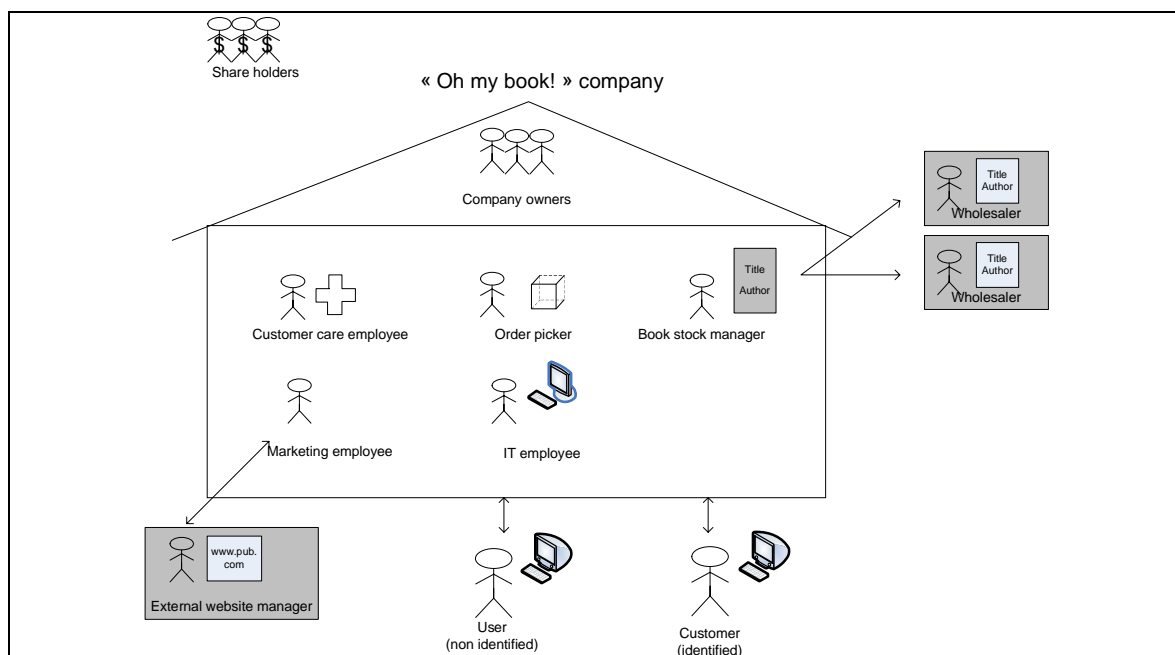


Figure 9-1 The stakeholders of the running example

9.1.4 How does an order happen?

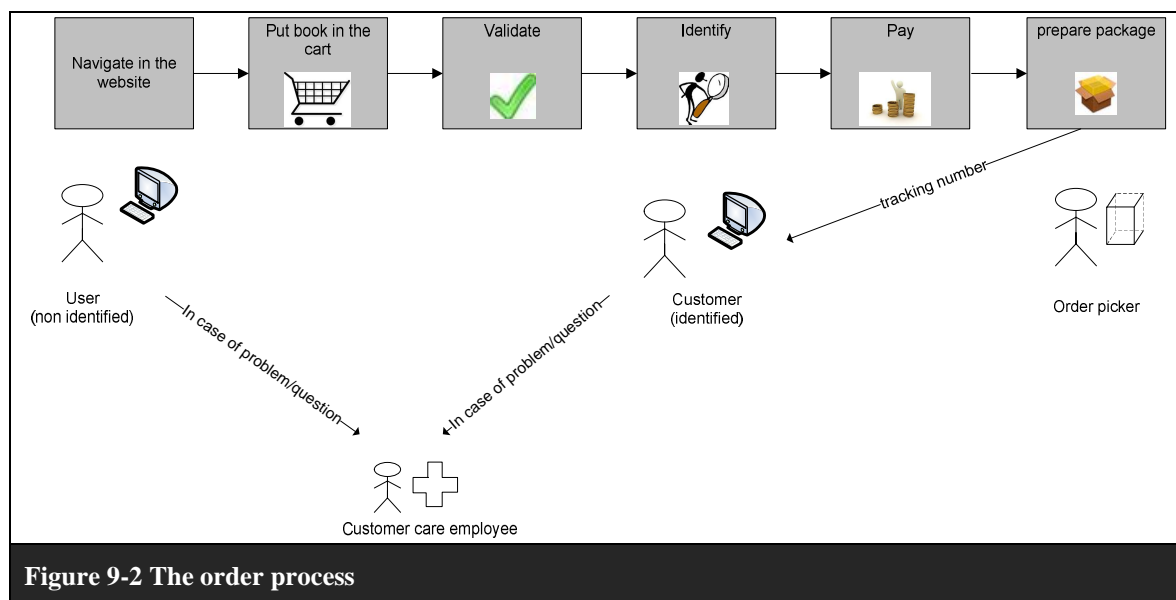
Before doing his/her order, the user navigates through the website to choose the book(s) he/she would like to order. When he/she likes a book, he/she adds it to his/her cart.

At any time, the user can validate his/her cart to order the books that are inside. After validating it, he/she will be invited to identify himself/herself (if he/she is already known as client) or fill in a client form to give his/her name, address, email, After this identification, the customer will be invited to pay his/her order via a payment software.

As soon as the payment is received, order pickers of the company prepare the package with the order. When the package is ready, order picker will give it a tracking number. This number is sent to the customer to allow him/her to follow the order.

If there is any problem during this process, the customer can call the customer care service to ask his/her questions.

Figure 9-2 summarises the process.



9.2 KAOS Analysis

In this section, we will make a (partial) overview of the system described in the section 9.1. In particular we will study the order activity and the other related activities. This overview will be modelled in KAOS (detailed in the Chapter 5).

We start to study the goal model of the system (succinctly because the whole model of this system is huge), then obstacles that could prevent to aim the goal. Agent model and operation model are studied together because they are closely related.

We will not study the behaviour model because its visual notation is quite similar to this one used for the UML use case diagram and UML state diagram. And consequently, we will not give any improvement for the visual notation of this model.

9.2.1 Goal model

Figure 9-3 represents the goal model. The main goal that the system has to encounter is to fulfil a maximum of user requirements. The requirements can be functional or non-functional.

Some of the functional requirements are:

- to have a secured payment system,
- to sell books at best price of the market, what implies a good logistic, to sell a lot of books and a robust online store,
- to offer high availability for the website,
- to deliver quickly customer packages.

Some of the non-functional requirements are:

- to have a system as cheap as possible (note: this requirement is opposite to a robust online store. Generally, robust software is expensive),
- to have an efficient system,
- to have system easy to use. We could provide support by phone or by mail to users.

On this model, there are also some agents:

- order picker (human), a person who prepares the packages,
- customer care employee (human), a person who replies to customer questions,
- bookOrderSoft (software), software that orders automatically books to the wholesaler. This is an existing software bought of an external company,
- payPol (software), a software that allows doing secure payments on different ways (not only by credit cards).

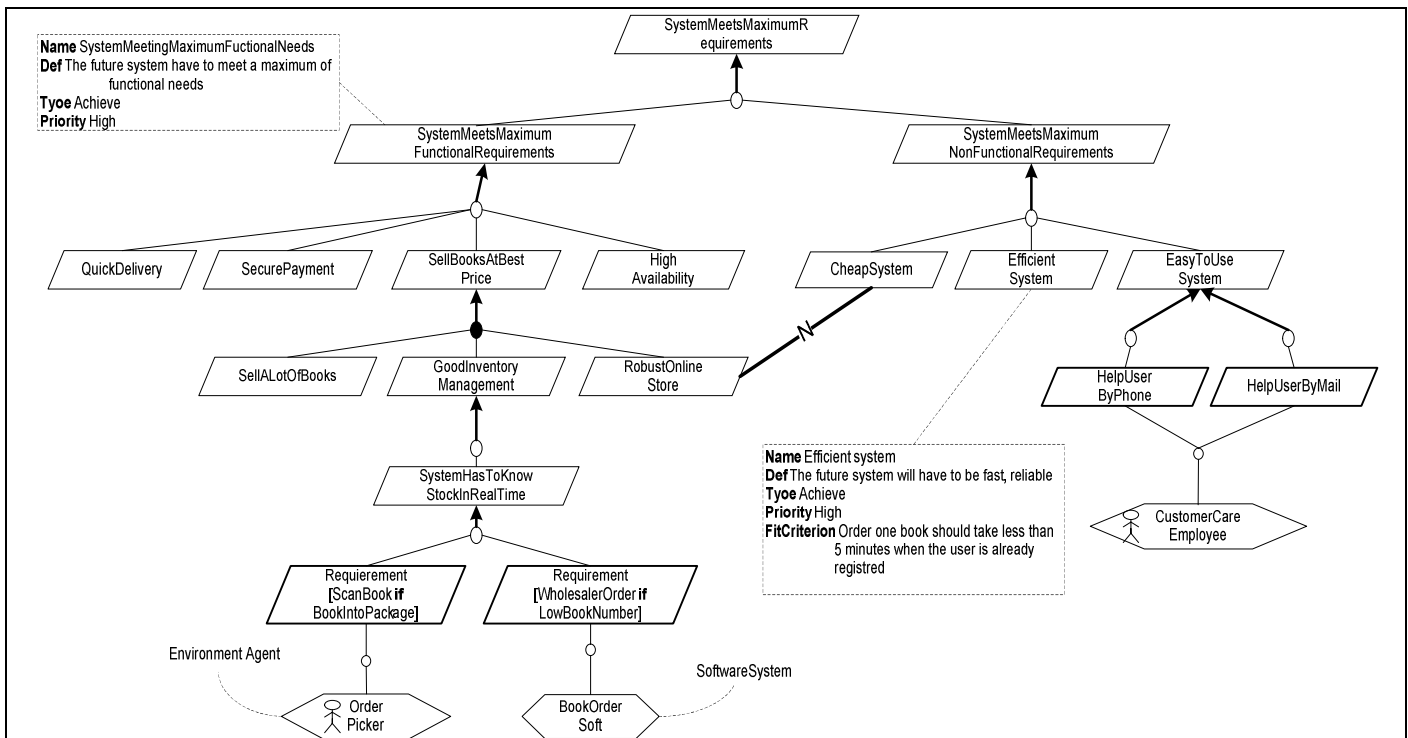


Figure 9-3 Goal model of the system-to-be of the online store 'Oh my book'

Figure 9-4 represents the obstacle model. We have chosen some main goals and we have tried to find the obstacles that will prevent to achieve the latest's. If we cannot by-pass these obstacles, the goals will not be achieved.

For the goal 'QuickDelivery', obstacles could be that: (i) books are not available in stock, (ii) the customer is not at home when books are delivered, and (iii) there could be some delay either in the preparation or the delivery of the package.

Some obstacles can prevent us to sell books at best price. These obstacles can be grouped in 2 types: the management of the book stock is not efficient and the website is not robust.

The book stock cannot be up-to-date if order pickers forget to scan the book before putting it in the package, if books are stolen or if employees do not check that the quantity of books they received by the wholesaler is correct.

The website will not be robust if there are too many technical problems.

Finally, the goal 'HighAvailability' will not be met if the website is frequently unavailable due to frequent maintenance or frequent technical problems.

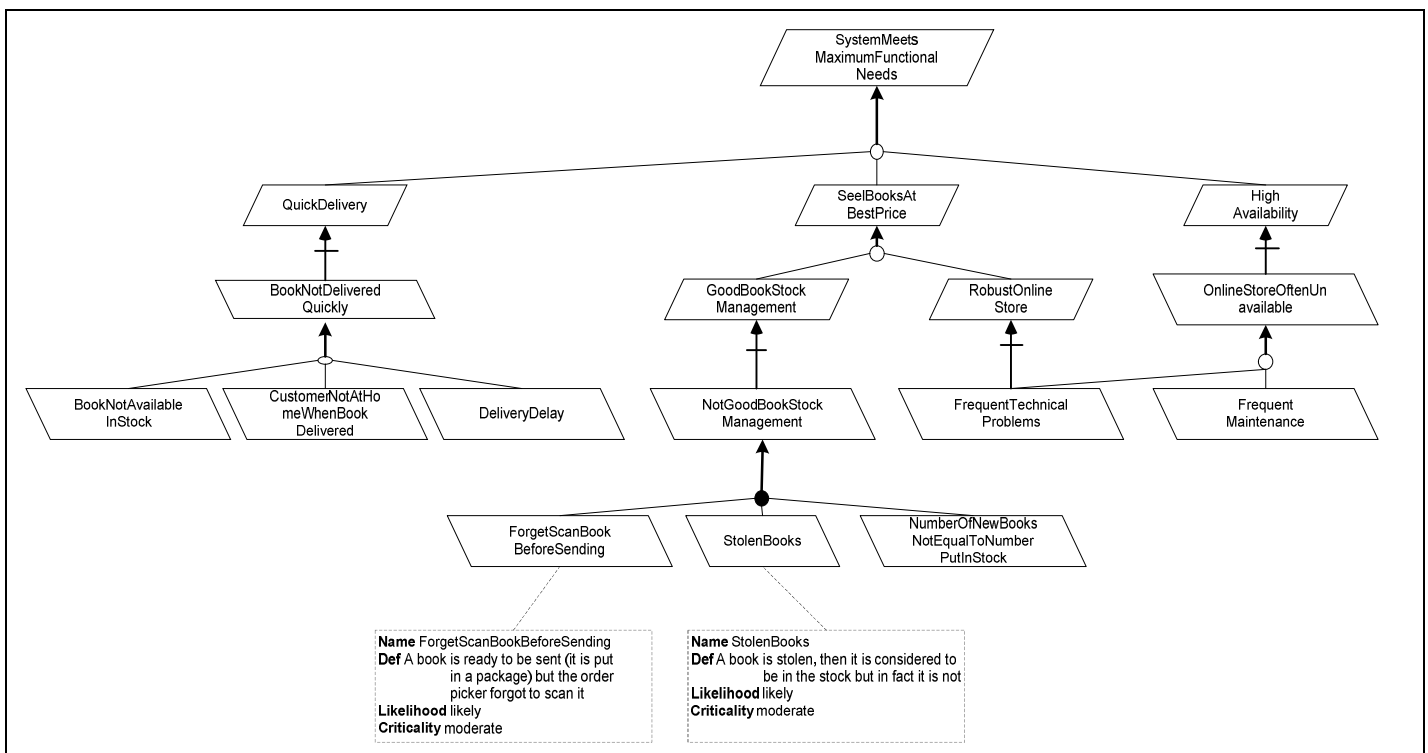


Figure 9-4 Obstacle model of the system-to-be of the online store 'Oh my book'

9.2.2 Agent and operation model

For the clarity of the diagram and the ease of the reader, we will describe only the operation that consists of preparing a package. Obviously, there are a lot of operations in our complete running example such as 'pay an order', 'order books to the wholesaler' and 'reply to customer question'.

Figure 9-5 shows the operation model for this operation. We can see that the agent 'OrderPicker' has to operationalise 'PreparePackage' as soon as a bill is paid (pre requirement). This operation will interact with the object 'Package'. To prepare a package, we have to give the list of ordered books (Bill.Books[]) and the delivery address of the customer (Customer.DeliveryAddress). When the package is ready, the tracking number of the latest is provided (Package.TrackingNb).

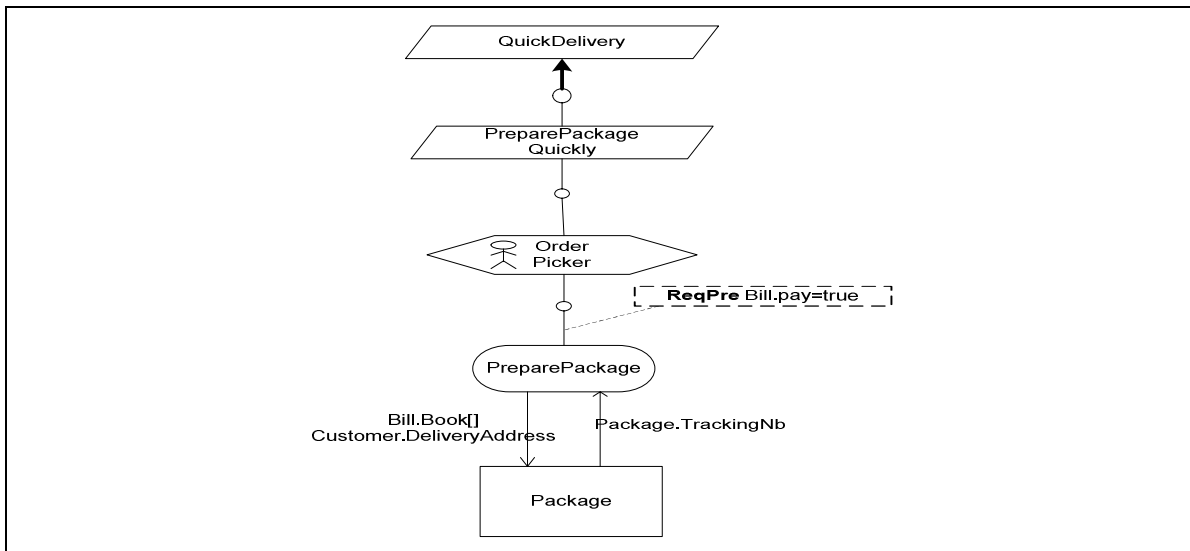


Figure 9-5 Agent and Operation models of the system-to-be of the online store 'Oh my book'

9.2.3 Object model

Figure 9-6 represents the object model of the running example. It contains the different objects used in the bookstore system. There are employees that can be sorted in 2 categories: customer care employees –who reply to customer questions- and order pickers– who prepare the packages according the orders. Every employee has a unique number in the system (IDEmployee), a name and a phone number.

Customers can be identified thanks to their customer number (IDCustomer), their names and addresses are known in the system. We can also contact them by email. Customers can put books in their cart before ordering. When they have made their choice, they can order them. Books are identified by their ISBN number (9 or 12 digits); they have a name and they are written by 1 to 10 authors. The number of books in stock is known in real time thanks to the field "NbCopyInStock".

When customers order one or many books, they receive a bill with the amount to pay. When this bill is paid, an order picker will prepare the package with the books ordered by the customer. This package has a tracking number to be identified.

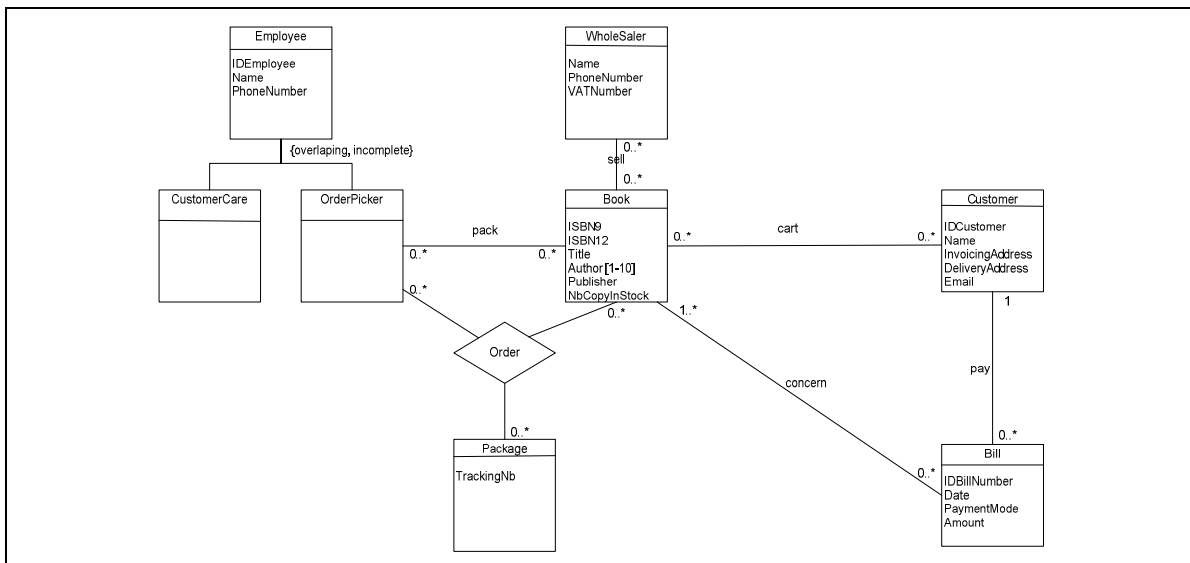


Figure 9-6 Object model of the system-to-be of the online store 'Oh my book'

9.2.4 Behaviour model

As explained in Chapter 5, the behaviour model uses the UML notation of the sequence diagram and the state machine diagram.

Figure 9-7 describes a sequence diagram of the order process. On this diagram, the user wants to order the books he/she has in his/her cart. To do this, the online bookstore asks to validate the order, then to identify himself/herself as customer. The next step is the payment of the order. As soon as the order is paid, a signal is sent to the order picker to prepare the package. When it is done, the order picker sends asynchronously the tracking number to the customer.

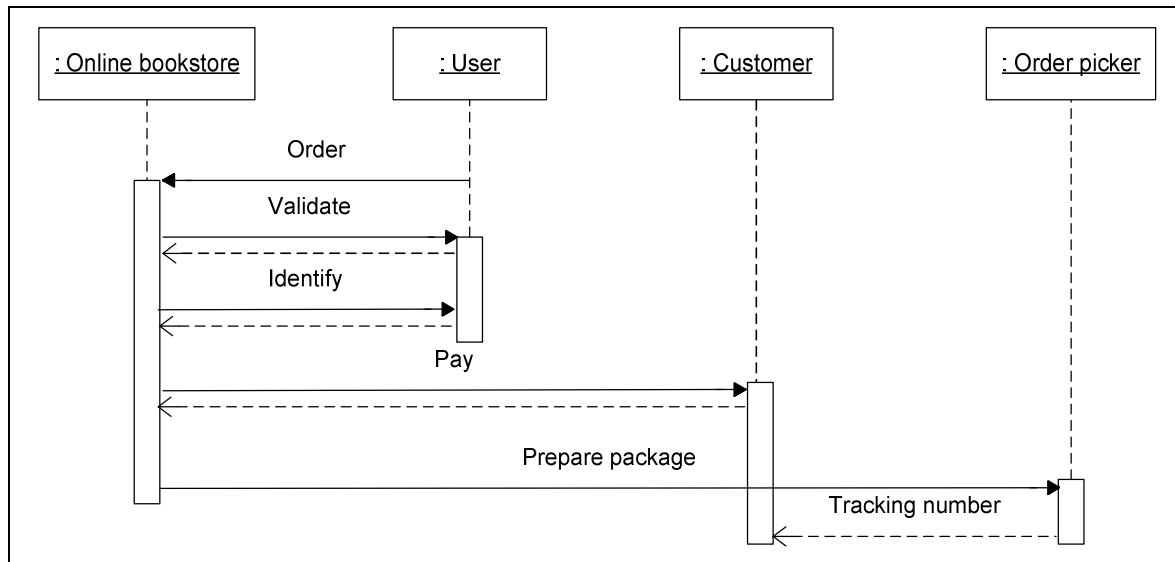


Figure 9-7 Sequence diagram of the order process

The state machine studied in figure 9-8 represents the different states of an order. The latest can be validated when the customer is ready to pay. When it is paid, the order is prepared otherwise it returns to the initial status. When the order is prepared, it is sent and finally received by the customer.

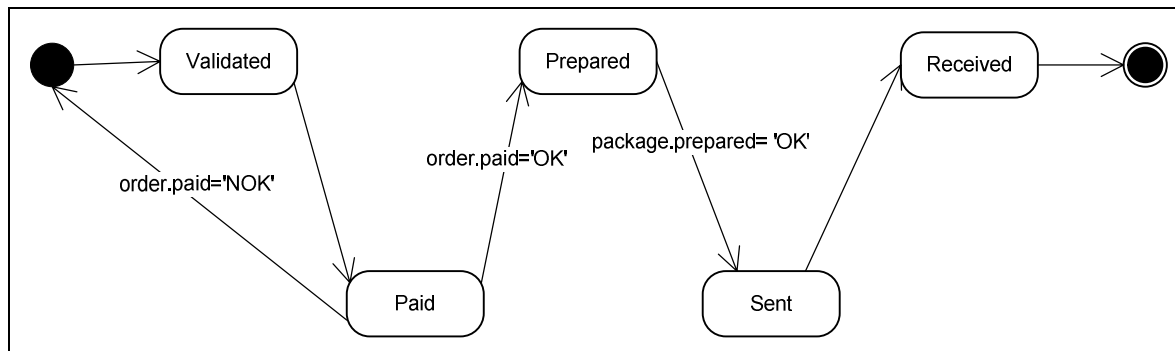


Figure 9-8 State machine diagram of an order

9.3 Modified versions of diagrams

In this section, we will adapt the goal model with the recommendations done for the language engineers in the section 8.2.

The first step consists of creating an integration map with the models that are developed for the running example (figure 9-9). As we will limit the example to the goal model, this is the only one which is highlighted on the figure.

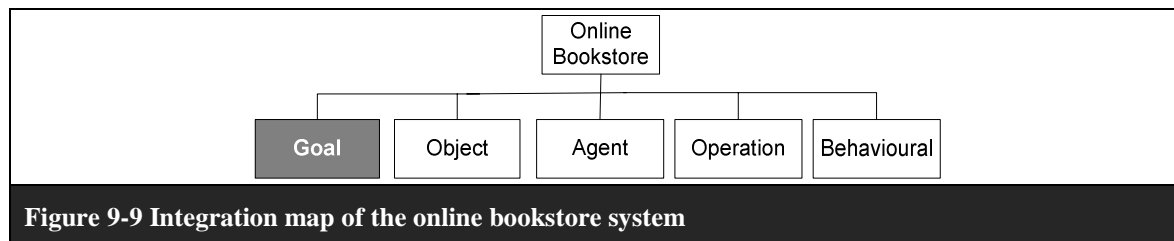


Figure 9-9 Integration map of the online bookstore system

Then we will work on the integration map at goal level (figure 9-10). First, we will standardise the name of the goals.

The recommendation tells the label of a goal should be a word (that represents the subject) followed by a verb in its passive form. When we tried to apply it, we have met a problem because some of the original labels are only a noun (all encountered problems are summarised and discussed in the section 9.4).

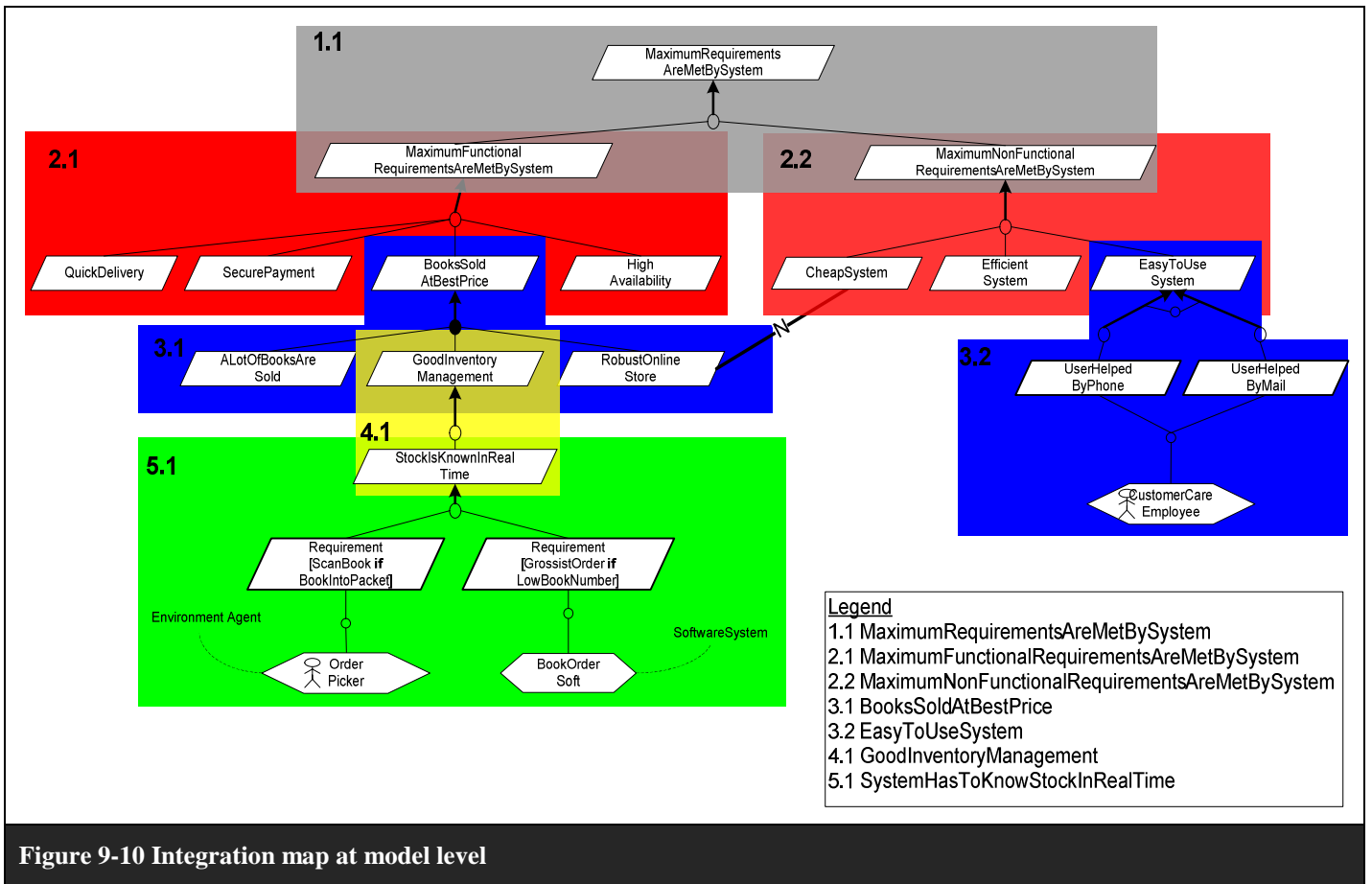
Table 9-1 summarises the translation of the original name of the goal into their standardised name.

Table 9-1 Goal standardised names

Original name	Standardised name
SystemMeetsMaximumRequirements	MaximumRequirementsAreMetBySystem
SystemMeetsMaximumFunctionalRequirements	MaximumFunctionalRequirementsAreMetBySystem
SystemMeetsMaximumNonFunctionalRequirements	MaximumNonFunctionalRequirementsAreMetBySystem
SellBooksAtBestPrice	BooksSoldAtBestPrice
SellALotOfBooks	ALotOfBooksAreSold
HelpUserByPhone	UserHelpedByPhone
HelpUserByMail	UserHelpedByMail
SystemHasToKnowStockInRealTime	StockKnownInRealTime

Then, we have drawn the integration map at model level. According to the general recommendation number 2, the map will contain only appropriate annotation and following recommendation number 4, the vertical position is used to show the semantic of the refinement relationship link.

On this map, we can see the different modules of the goal model. Each module has a number (composed as following: x.y where x represents the level of the module and y the number of the module in its level –from left to right) and a title (generally the name of the parent goal). The map contains also a legend that summarises the number of the modules and their names.



After creating the integration map, we will draw each module with the navigation cues. We will describe the recommendations when they are used.

First, we have followed the recommendation about absolute discriminability and given the same size to each shape. They are also separated by the same distance. The background is coloured in green. The goal symbol and the type to the relationship link have been added to improve the semantic transparency of the figure.

As the diagram represents only one module of goal model, it has a title (the name of the parent goal) and a number (according to the integration map at model level) on the top. In the right top corner, we have placed the general map of the whole system and highlighted the model in which we are working. In the left top corner, we have drawn a map of the different modules of the goal model and highlighted this one in which we are working. Below the diagram, we have added 2 arrows to express that the modules 2.1 and 2.2 are the following sub-diagrams. Goals that are themselves modularised contain the fork sign. Then we have added arrows to give information about the surrounding sub-diagrams situated around the current one. During the elaboration of the diagram, we have encountered a problem. This problem is explained and discussed in section 9.4.

Finally, we have drawn the arrows to indicate the direction of the sub-diagrams according to the integration map. Concretely we have placed them on the bottom left and another one on the bottom right.

Here, all the goals have the same priority -high- then all of them have the same dark green.

And finally, to improve the cognitive integration, we have a legend below the diagram.

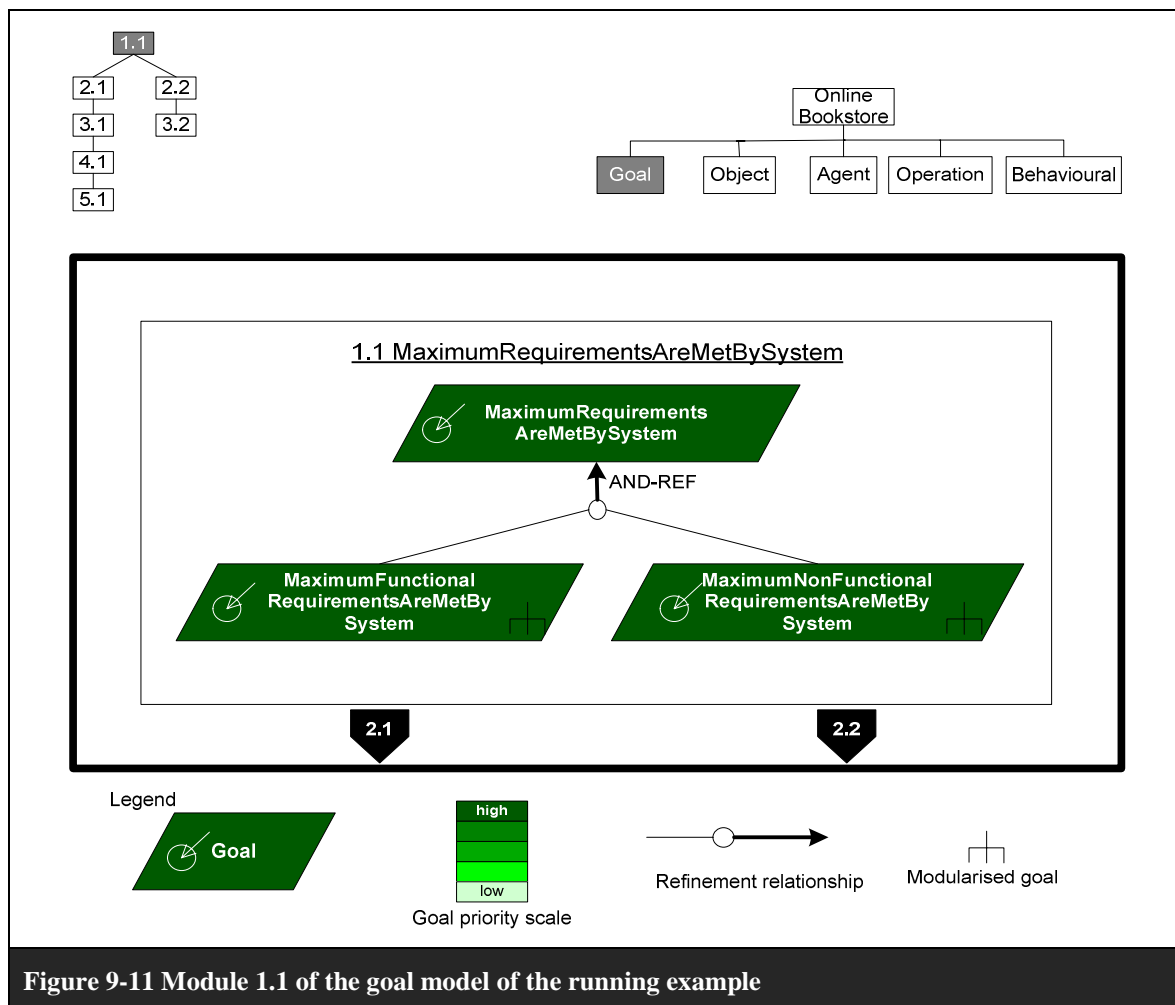
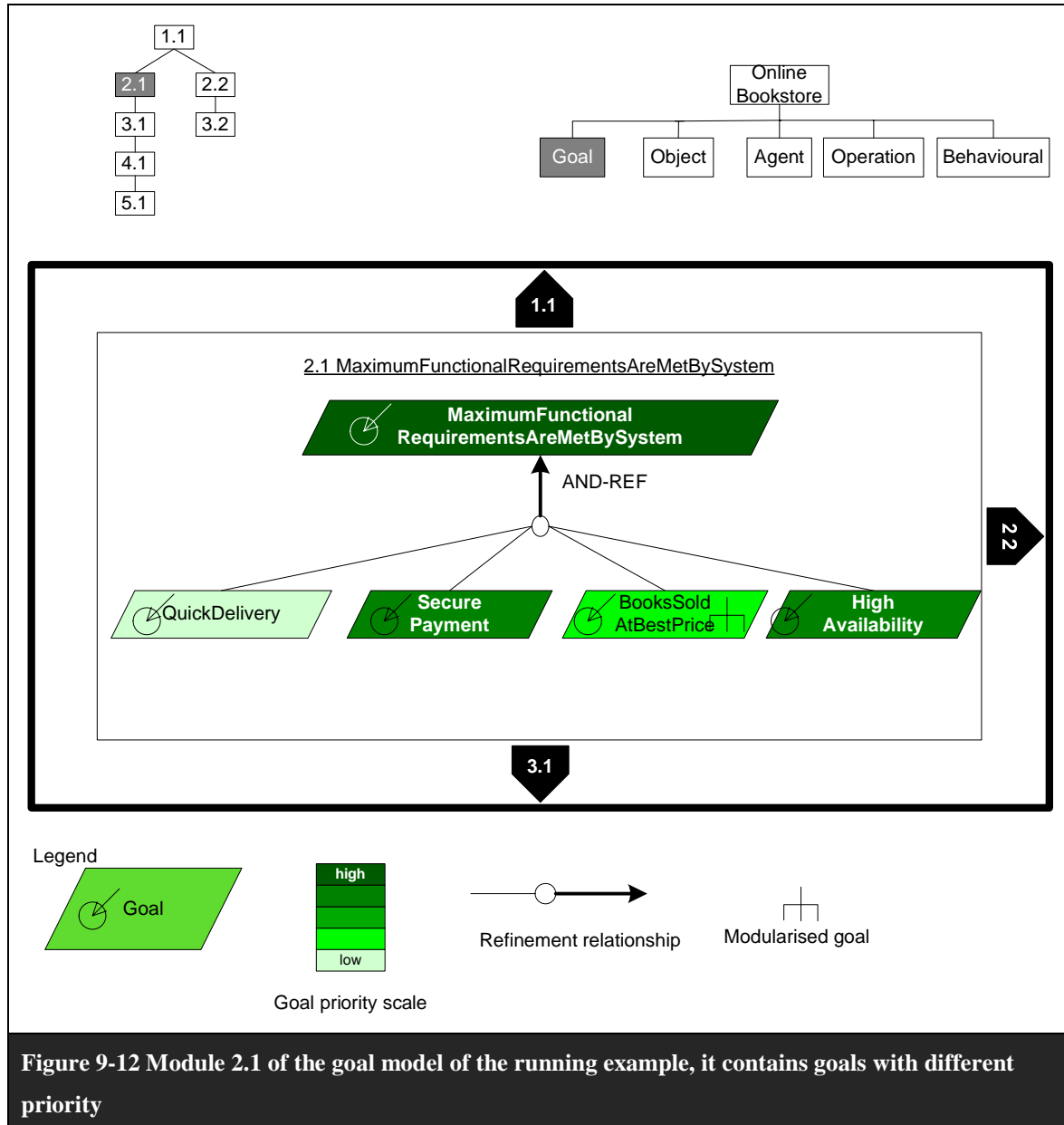


Figure 9-11 Module 1.1 of the goal model of the running example

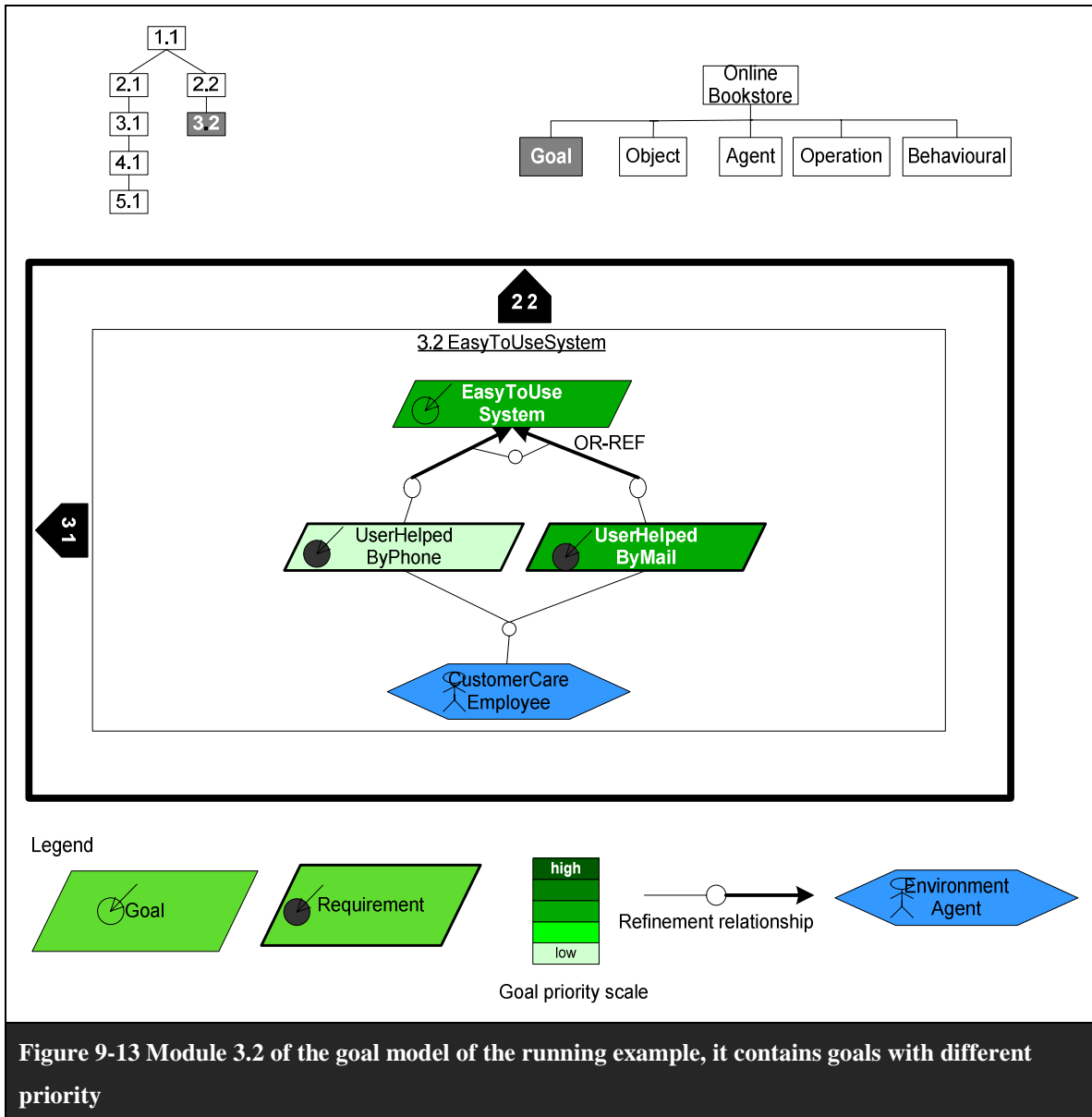
The module 2.1 (figure 9-12) contains also the different recommendations. But in this module, goals have different priorities. To express the priority, we have used the 'value' variable. The darker is goal, the more important it is and the more stakeholders of the project have to fill it.

Giving different colours to a same symbol will cause trouble when the legend will be represented. To bypass this problem, we have added a 'goal priority scale' to remind the user the meaning of the different colours.



Modules 3.1, 4.1 and 5.1 follow the same principles but they do not contain any specificity. They can be seen in the annex 2.

Now, we will study the right part of the cognitive diagram. Module 3.2 will keep our attention (figure 9-13). In this module, there is an OR-refinement. To represent it, we have used the new representation of the relationship link (we have added a line between the different refinements with an adorned circle that allows expressing if the refinement is complete or not). In this module, we can see the figure that represents an environment agent.



9.4 First Evaluation of Recommendations

In this section we will point out some problems that we have met during the elaboration of the diagrams with the new visual representation.

Standardisation of the labels

Goal names should be standardised and defined as a subject and a verb in its passive form, but some goals names are only a noun (e.g., 'SecurePayment' or 'CheapSystem'). To solve this problem we have simply kept the original name.

Navigation cues

The second problem concerns the navigation cues that will help the user to situate the current model to the other. It is done by an arrow that shows the direction of the modules that surround the current module. But modules are not only situated on the top, below, on the right or on the left. Then when we draw these navigation arrows, they have to show to precise direction, in function of the integration map, of the surrounding diagrams.

Add legend

The third problem is encountered when we wanted to add a legend to a module that contains goal with different coloured background (these colours are variations around green to indicate the priority of the goal). To solve it, we have represented a goal priority scale to remind the meaning of the colour to the user

9.5 Limitations

We have only applied the first set of recommendations. It will be difficult to represent the next one that is specific for meeting because it will be mainly used to start a project from scratch or to discuss about a part of the project. When it is done, the software engineers will have to re-draw them with the first set of recommendations.

When modellers will want to use the first set of recommendation, we advice them to do it preferably with an appropriate software that will manage the KAOS specificities. Indeed, it is not easy to manage the integration map that contains all the modules. Adding an adapted legend (that contains only the symbols that are used in the module) at the bottom of each module is time consuming as well as drawing the arrows that allow the navigation between the different modules. Without appropriate software it is also very difficult to manage modifications (that happens frequently during the life cycle of a project).

Part IV

Conclusion

Chapter 10 Conclusion

In this chapter, we retrospect the work we have done and present our majors conclusions. Then we try to self-criticism and finally we open perspective for future work.

10.1 Conclusion

The start point of this thesis is "A picture worth a thousand words" [Miller, 1956]. But in practice diagrams used in requirements engineering may act more as a barrier rather than an aid during communication with stakeholders [Moody, 2006]. However, the goal of requirements engineering languages is to model the needs of the different stakeholders in order to build software. This software matches as much as possible with these needs.

In [Moody, 2009], Moody presents the 9 principles of the Physics of Notations. The goal of these principles is to improve the communication between the software development team and the future users. These principles are based on many disciplines like cartography, cognitive psychology, computer graphics, diagrammatic reasoning and many studies about human behaviour and they are continuously reviewed.

During this work, we have applied these principles on a specific goal-oriented requirements engineering language: KAOS. We have studied the visual notation of this language under the lighting of the principles of the Physics of Notations. We have started studying the visual notation as described in [Lamsweerde, 2009] and analysed how each principle is respected or not. Then, depending on the previous analysis, we have suggested some improvements. They could be done to increase the cognitive effectiveness of the diagrams that will be used to communicate between stakeholders.

Applying the Physics of Notations to the KAOS visual notation has shown that it suffers from lacks. They are mainly situated in the usage of colours, the abstract meaning of the symbols, the management of the complexity and the cognitive integration.

Based on this analysis, we suggested recommendations for the KAOS language engineers to improve communication with novices. This set of recommendations explains how to improve the effectiveness of the diagrams.

After, we elicited a second set of recommendations to improve the KAOS visual notation when it is used during meetings with stakeholders. The aim of this set is to simplify the visual notation to be quickly sketched by hand without losing its cognitive effectiveness.

And finally, we wrote recommendations for software developers who would like to create or enhance tools implementing the KAOS visual notation. These recommendations should help to make a tool that is easy to use and create effective cognitive diagrams. The produced diagrams should be suitable for the different stakeholders of the system.

To illustrate our recommendations, we created a running example and modelled it with the current visual notation of KAOS. Afterwards, we drew the goal model again but by taking into account the recommendations that we formulated for language engineers. With this second version of the diagrams, the information is structured in such a way that the cognitive effectiveness of these diagrams should be improved. This example gives the intuition that, that navigational maps –at system and at model level– help the user to understand the decomposition of a diagram and that legends are very useful for novices.

10.2 Limitations

During our work, we met 3 limitations. In this section, we have made some self-criticism about visual notations, about the author and about our work

10.2.1 Self-criticism about visual notations

Throughout this work, we can understand that visual notations are very important and cannot be designed in few minutes or a couple of hours. Creating a visual notation that is cognitively effective and easy to use is not a trivial work. Each figure, each colour, each line requests a long time of thought. It could be compared to writing an efficient and accurate text: each word and each sentence have to be weighted to be sure on the effect they will produce.

Moreover, many language engineers have conceptualised visual notations as being an issue of "aesthetics" which they think largely irrelevant. They have to keep in mind that diagrams have to be used to convey information clearly and precisely which involves that they have more common point with mathematical notations than art. It is not because a diagram looks good that it communicates effectively [Moody, 2009].

Nowadays, visual notations are created by people who have no training or experience in graphic design. When we want to create a new visual notation we should act as we do for software engineering practices (e.g. web design, user interface designer): we should consult graphic design specialists. It will help avoid that visual notations violate the basic principle of visual notations which will in fine produce cognitive ineffective diagrams.

10.2.2 Self-criticism about the author

As the author was not very familiar with the KAOS language at the beginning of the work and consequently has never modelled a real project with this language, there may be some subtleties that are not known by her. It would have been easier to study the visual language of a well known language.

10.2.3 Self-criticism about the work

This analysis is mainly performed by a single person, the author of this thesis (plus some reviewers who provide advices). Consequently, a part of subjectivity could be found in our understanding and application of the 9 principles. Confronting our points of view with other analysts would have a large added value.

When we wrote our recommendations for software developers, we did not take into account the technical point of view. Maybe some of our recommendations will take too much time to be implemented or they are maybe technically difficult to implement.

Our validation of recommendations is based on only one example. It implies that our suggestions are confronted to only some cases and then we have maybe not tested other ones. Moreover, our running example is not a real-size one. Using such example would have been more accurate but it would have taken more time.

10.3 Future Works

In this section, we suggest some work that could be either to improve the current thesis, either to use it with other documents that are already published.

Compare our work to the real world

Along Chapter 8, we have suggested some improvements for KAOS modellers, recommendations to sketch the KAOS visual notation by hand during a meeting and recommendations to software developers of tools supporting KAOS. These improvements and recommendations are described in details in this work. We have also done a running example to put them into practice.

Until here, our work has been very theoretical. A next step would consist in experimenting our recommendations with users who are used to elaborate models with KAOS or read models that use the KAOS visual notation. One of the extensions of this work could be to ask to some users to put our recommendations into practice on 'real' projects. Then we could ask them their opinion in order to validate our work.

We should also try to find some software developers that could implement our recommendations in a tool. Then, this tool could be tested by users and we could gather their opinions about the diagrams it produces.

After having performed such experimentations, we would be able to refine and, if necessary to revise, the conclusions of our work.

Compare requirements engineering language between them

So far, some of the most well-known goal-oriented modelling languages have been analysed according to the Physics of Notations. In this work, we have studied the KAOS visual notation. Moody *et al.* applied the theory on i^* [Moody, et al., 2010], Tropos was analysed by Boucher [Boucher, 2008] and UML was investigated in [Moody, et al., 2008].

We think it might be valuable to perform a comparison of the visual notations of these goal-oriented modelling languages. It may be possible to elaborate a kind of evaluation grid to score each language according the 9 principles of the Physics of Notations. Such grid could be applied to the current visual notation of these languages before studying it and applied again after new recommendations have been implemented. It should also allow comparisons to select a language that fulfils at best one or several principles of the Physics of Notations.

Glossary

Agent: active object performing operations to achieve goals. Agents can be the software being considered as a whole or parts of it. Agents can also come from the environment of the software being studied; human agents are in the environment.

ArchiMate: an open and independent enterprise architecture modelling language to support the description, analysis and visualisation of architecture within and across business domains in an unambiguous way [ArchiMate, 2010].

CASE: Computer-Aided Software Engineering

Cognitive load theory: this theory defines the number of element that can be loaded in the short-term (working) memory of a person.

Cognitive fit theory: this is a theory that explains how a user will use his experiments to solve new problems.

Cognitive effectiveness: the speed, ease and accuracy with which information can be extracted from representation [Larkin, et al., 1987]. This is the basis design goal to construct visual notations.

Congruence: similarity between objects.

Data flow diagram: graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing (structured design) [DFD, 2010].

Descriptive theory: empirical theory, based on facts or on phenomena...

Diagram: symbolisation of a model with the concrete syntax of a language.

Domain Property: descriptive statement about the environment, expected to hold invariably regardless of how the system behave.

Graphic design space: composed of 8 visual variables to encode graphically the information (see figure 4-2).

Graphical construction: a single or a set of geometrical shapes used to represent a semantic construct.

Graphical representation: synonym of diagram.

Graphic complexity: defined by the number of different symbol types in a notation. It is the size of its visual vocabulary [Nordbotten, et al., 1999].

Goal: an objective that the system should achieve through cooperation of agents in the software-to-be an in the environment [Lamsweerde, 2009].

Hypothesis: a descriptive statement satisfied by the environment and subject to change.

Meta-model: defines the meta-concepts (semantic constructs) of a language and the meta-relationships between them.

Model: abstract representation of a composite system. A KAOS model represents a composite system by means of concepts of different types, mainly, objects, desired or undesired properties (goals, obstacles), and behaviours (operations).

Objectiver: it is a software based on KAOS method described in Axel van Lamsweerde book [Lamsweerde, 2009].

Prescriptive theory: a set of rules that defines a theory precisely (in opposition to a descriptive theory).

Semantic complexity: is defined by the number of semantic constructs represented by the visual notation.

Visual distance: between many symbols is measured by the number of visual variables used on which they differ and the size of these differences.

Visual expressiveness: is the number of different visual variables used in a visual notation and the range of values used for each: this measure the utilisation of the graphic design space [Moody, 2009]

Visual representation: synonym of diagram.

Bibliography

Anton A. Goal Identification and Refinement in the Specification of Software-Based Information Systems [Report] / Georgia Institute of Technology, Atlanta, GA, USA. - 1997.

Anton A. Goal-Based Requirements Analysis [Conference]. - 1996.

Anton A., McCracken W. and Potts C. Goal decomposition and scenario analysis in business process reengineering [Book Section] // Advanced Information Systems Engineering / ed. Wijers Gerard, Brinkkemper Sjaak and Wasserman Tony. - [s.l.] : Springer Berlin / Heidelberg, 1994. - Vol. 811. - 10.1007/3-540-58113-8_164.

ArchiMate Definition of ArchiMate [Online]. - 10 2010. - <http://en.wikipedia.org/wiki/ArchiMat>.

Avison D.E. and Fitzgerald G. Information Systems Development: Methodologies, Techniques and Tools (3rd Edition) [Book]. - [s.l.] : Oxford, United Kingdom: Blackwell Scientific, 2003.

Baldwin C.Y. and Clark K.B. Design Rules Volume 1: The Power of Modularity [Book] / ed. Cambridge. - [s.l.] : MIT Press, 2000.

Bar M. and Neta M. Humans prefer curved visual objects [Journal] // Psychological Science. - 2006. - Vol. 17(8). - pp. 645-648.

Barker R. CASE*Method: Entity Relationship Modelling [Book] / ed. Professional Addison-Wesley. - [s.l.] : Workingham, England, 1990.

Bertin J. Semiology of Graphics: Diagrams, Networks, Maps [Book] / ed. Madison. - [s.l.] : University of Wisconsin Press, 1983.

Bezivin J. and Gerbé O. Towards a Precise Definition of the OMG/MDA Framework [Conference]. - 2001.

Boehm B.W. A Spiral Model of Software Development and Enhancement [Journal] // IEEE Computer. - 1988. - Vols. 21, N°5, May. - pp. 61-7.

Boucher Q. Visually Effective Tropos Models [Report] / FUNDP. - 2008.

Bresciani P. [et al.] Tropos: An Agent-Oriented Software Development Methodology [Journal] // Autonomous Agents and Multi-Agent Systems. - 2004. - Vol. 8(3). - pp. 206-236.

Castro J., Kolp M. and Mylopoulos J. Towards Requirements-Driven Information System Engineering: The Tropos Project [Journal] // Information System Journal. - 2002. - Vol. 27(6).

Cheng P.C.-H. Why diagrams are (Sometimes) Six Times Easier than Words: Benefits beyond Locational Indexing [Book] / ed. Springer-Verlag. - 2004.

Cheng P.C.-H., Lowe R.K. and Scaife M. Cognitive Science Approaches To Understanding Diagrammatic Representations [Journal] // Artificial Intelligence Review. - 2001. - Vol. 15(1/2). - pp. 79-94.

Chung L. [et al.] Non-Functional Requirements in Software Engineering [Book]. - [s.l.] : Kluwer Academic, 2000.

Citrin W. Strategic Directions in Visual Languages Research [Journal] // ACM Computing Surveys. - 1996. - Vol. 24(4).

Dardenne A., Lamsweerde A. van and Fickas S. Goal-Directed Requirements Acquisition [Journal] // Science of Computer Programming. - 1993. - Vols. 20 (1-2).

Davis A.M. Software Requirements: Objects, Functions and States [Book]. - [s.l.] : Prentice-Hall, 1993.

DeMarco T. Structured Analysis and System Spec [Book]. - [s.l.] : Yourdon Press, 1978.

DFD Definition of Data Flow Diagrams [Online]. - 10 2010. - http://en.wikipedia.org/wiki/Data_flow_diagram.

Dia Dia, a diagram creation program. - 2011.

Fuxman A. [et al.] Model Checking Early Requirements Specifications in Tropos [Conference]. - 2001.

Goodman N. Languages of Art: An approach to a Theory of Symbols [Book] / ed. Co. Bobbs-Merrill. - 1968.

Habra N. Ingénierie des logiciels. - 2009-2010.

Hahn J. and Kim J. Why Are some Diagrams Easier to Work With ? Effects of Diagrammatic Representations on the Cognitive Integration Process of System Analyses and Design. [Journal] // ACM Transaction on Computer-Human Interaction. - 1996. - Vol. 6(3). - pp. 181-213.

Halpin T.A. ORM 2 Graphical Notation (Technical Report ORM2-01) [Book]. - [s.l.] : Neumont University, 2005.

Heymans P. Analyse et modélisation des systèmes d'information. - 2006-2007.

IEEE-STD830 IEEE Recommended Practice for Software Requirements Specifications // IEEE Recommended Practice for Software Requirements Specifications. - 1998.

Irani P. and Ware C. Diagramming Information Structures Using 3D [Journal]. - [s.l.] : ACM Transactions on Computer-Human Interaction, 2003. - Vol. 10(1). - pp. 1-19.

Jackson M. The world and the Machine [Journal] // Proceedings of ICSE'95 - 17th International Conference on Software Engineering. - 1995. - Vols. ACM-Press. - pp. 283-292.

KAOS Definition of Kaos [Online]. - 10 2010. - http://en.wikipedia.org/wiki/KAOS_%28software_development%29.

Kim J., Hahn J. and Hahn H. How Do We Understan a System with (So) Many Diagrams? Cognitive Integration Processes in Diagrammatic Reasoning [Journal] // Information Systems Research. - 2000. - Vol. 11(3). - pp. 248-303.

Kosslyn S.M. Graphics and Human Information Processing [Journal] // Journal of the American Statistical Association. - 1985. - Vol. 80(391). - pp. 499-512.

Kosslyn S.M. Understanding Charts and Graphs [Journal] // Applied Cognitive Psychology. - 1989. - Vol. 3. - pp. 185-226.

Lamsweerde A. Van and Letier E. From Object Orientation to Goal Orientation: A Pradigm Shift for Requirements Engineering [Journal] // Proc. Radical Innovations of Software and Systems Engineering, LCNS. - 2003.

Lamsweerde A. Van Requirements engineering. [Book]. - [s.l.] : Wiley, 2009.

Lapouchnian A. Goal-Oriented Requirements Engineering: An Overview of the Current Research [Report] / University of Toronto. - 2005.

Larkin J.H. and Simon H.A. Why a Diagram is (Sometimes) Worth Ten Thousands Words [Journal] // Cognitive Sciences. - 1987. - Vol. 11(1).

Lohse G.L. A Cognitive Model for Understanding Graphical Perception [Journal] // Human-Computer Interaction. - 1993. - Vol. 8(4). - pp. 353-388.

Lohse G.L. The Role of Working Memoy in Graphical Information Processing [Journal] // Behaviour and Informaiton Technology. - 1997. - Vol. 16(6). - pp. 297-308.

Lynch K The Image of the City [Book]. - [s.l.] : Cambridge, USA: MIT, 1960.

Mackinlay J. Automating the Design of Graphical Presentation of Relational Information [Journal] // ACM Transaction on Graphics. - 1986. - Vol. 5(2). - pp. 110-141.

Mayer R.E. and Moreno R. Nine Ways to Reduce Cognitive Load [Journal] // Educational Psychologist. - 2003. - Vol. 38(1). - pp. 43-52.

Miller G.A. The Magical Number Seven, Plus Or Minus Two: Some Limits On Our Capacity For Processing Information [Journal] // The psychological Review. - 1956. - Vol. 63. - pp. 81-97.

Moody D. L. . - Internal communication.

Moody Daniel L. and J. van Hillegersberg Evaluating the Visual Syntax of UML: An Analysis of the Cognitive Effectiveness of the UML Suite of Diagrams. [Journal] // Proceedings of the 1st International Conference on Software Language Engineering (SLE), Toulouse, France: Springer Lecture Notes in Computer Science. - 2008.

Moody Daniel L. Dealing with "Map Shock": A Systematic Approach for Managing Complexity in Requirements Modelling [Journal] // Requirements Engineering: Foundation for Software Quality. - 2006.

- Moody Daniel L.** The "Physics" of Notations: Towards a Scientific Basis for Constructing Visual Notations in Software Engineering [Journal] // TSE. - 2009.
- Moody Daniel L.** What Makes a Good Diagram ? Improving the Cognitive Effectiveness of Diagrams in IS Development [Journal] // Advances in Information Systems Development. - 2006.
- Moody Daniel L., Heymans P. and Matulevičius R.** Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation [Journal] // Requirements Engineering n^o2. - 2010. - Vol. 15. - pp. 141-175.
- Nordbotten J.C. and Crosby M.E.** The Effect of Graphic Style on Data Model Interpretation [Journal] // Information System Journal. - 1999. - Vol. 9(2). - pp. 139-156.
- Nuseibeh B. and Easterbrook S.** Requirements engineering: a roadmap [Conference]. - [s.l.] : ACM, 2000. - pp. 35-46.
- Objectiver** Objectiver User Manual. - 2007.
- OMG_UML** UML Specification. - 2011.
- Palmer S. and Rock I.** Rethinking Perceptual Organisation: The Role of Uniform Connectedness. [Journal] // Psychonomic Bulletin and Review. - 1994. - Vol. 1(1). - pp. 29-55.
- Patrignani M.** Visualization of Large Graphs [Report] / Università degli Studi di Roma: La Sapienza, Italy. - 2003.
- Peirce C.S.** The Essential Writings (Great Books in Philosophy) [Book] / ed. Moore E.C.. - [s.l.] : Prometheus Books, 1998.
- Petre M.** Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming [Journal] // Communication of the ACM. - 1995. - Vol. 38(6). - pp. 33-44.
- Quinlan P.T.** Visual Feature Integration Theory: Past, Present and Future [Journal] // Psychological bulletin. - 2003. - Vol. 129(5). - pp. 643-63.
- Robertson S. and Roberson J.** Mastering the Requirements Process [Book]. - [s.l.] : ACM Press/Addison-Wesley, 1999.
- Ross D.** Structured Analysis: A Language for Communiting Ideas [Journal] // IEEE Transactions on Sof. - 1977. - Vol. 3(1).
- Rumbaugh J. [et al.]** Object-Oriented Modeling and Design [Book]. - [s.l.] : Prentice Hall, 1991.
- Seidewitz E.** What Models Mean? [Journal] // IEEE software. - 2003.
- Selic B.** UML 2.0: Exploiting Abstraction and Automation [Journal] // SD Times. - 2004. - Vol. v98. - p. p24(2).
- Shaft T.M. and Vessey I.** The Role of Cognitive Fit in the Relationship between Software Comprehension and Modification [Journal] // MIS Quarterly. - 2006. - Vol. 30(1). - pp. 29-55.

Siau K. Informational and Computational Equivalence in Comparing Information Modelling Methods [Journal] // Journal of Database Management. - 2004. - Vol. 15(1). - pp. 73-86.

Simon H.A. Sciences of the Artificial (3rd edition) [Book] / ed. Press MIT. - 1996. - p. 215.

TECLiege Carte des Transports en Commun de la Ville de Liège // Carte des Transports en Commun de la Ville de Liège. - 2010.

Treisman A. and Gelade G.A. A Feature Integration Theory of Attention [Journal] // Cognitive Psychology. - 1980. - Vol. 12. - pp. 97-136.

Treisman A. Perceptual Grouping and Attention in Visual Search for Features and for Objects [Journal] // Journal of Experimental Psychology: Human Perception and Performance. - 1982. - Vol. 8. - pp. 194-214.

UsabilityFirst Definition of redundant coding [Online] // Usability first. - 03 2011. - <http://www.usabilityfirst.com/glossary/redundant-coding/>.

Vessey I. and Galletta D. Cognitive Fit: An Empirical Study of Information Acquisition [Journal] // Information Systems Research. - 1992. - Vol. 2. - pp. 63-84.

Vessey I. Cognitive Fit: A Theory-based Analysis of the Graphs versus Tables Literature [Journal] // Decision Sciences. - 1991. - Vol. 22. - pp. 219-240.

Winn W.D. An Account of How Reader Search for Information in Diagrams [Journal] // Contemporary Educational Psychology. - 1993. - Vol. 18. - pp. 162-185.

Yu E. Towards Modelling and Reasoning for Early-Phase Requirements Engineering [Journal] // 3rd IEEE International Symposium on Requirements Engineering. - 1997. - pp. 226-235.

Annex 1: Analysis of the meta-model concepts

In this annex, we will study some concepts of the KAOS meta-model and their visual representation in KAOS.

The KAOS meta-model is composed of 5 models, but we will focus on the goal model. The goal model is chosen because in goal-oriented language the most important target is to represent goals. This model has also been chosen for its reusability during comparison with other goal-oriented languages. This analysis will take into account semantic constructs of the goal model and also some elements from other models with which they have direct relation.

Goal model contains:

- **Goals**
- As seen on the meta-model diagram, they can be classified into behavioural goals and **Soft goals**. As behavioural goal is an abstract class, it should never be visually represented. But its 2 subclasses have to be taken into account: '**Achieve**' and '**Maintain/Avoid**' goal.
- Goals can be refined in 2 ways: **AND-refinement** or **OR-refinement**. We consider that these 2 types of relationship between goals and refinement are 2 distinct concepts. We can assume this proposition because a refinement is always either an AND-refinement or either an OR-refinement. An AND-goal refinement can be complete or not.
- Goals can be refined into sub-goals or into **domain property** or **hypothesis**.
- As goals can come from stakeholders that have different views and goals (see section 2.5), it is possible to have a **Conflict link** between some of them.
- When goals are completely refined, they become leaf goals. Once again, it is an abstract class that have 2 subclasses: **Expectation** and **Requirement**.
- Leaf goals are under the **responsibility** of agent (abstract class), that can be classified into **system agent** and **software agent**.
- Goals can be **obstructed** by **obstacles**.
- Obstacle can be refined in 2 ways: **AND-refinement** or **OR-refinement**. They follow the same rule as AND or OR obstacle refinements. An AND-obstacle refinement can be complete or not.
- Obstacles come from some boundary conditions on system. We can say that they are a specification of the abstract class 'BoundaryCondition'.
- During the analysis of requirement engineering we have to try to find goal that **resolves** problems described by obstacles.
- **Operations** are done on leaf goals to fulfil them thanks to an operationalisation relationship.

We will not study behavioural and object models because their visual representations follow the standard UML use case diagram and UML state diagram.

Table A1-1 List of semantic constructs and their representation in the goal model

	Concepts (Class of the meta-model)	Element type	Element (E) or Relation (R)	Figure	Comment
1	Goal	meta-class	E	Right-oriented parallelogram	
2	Soft goal	subclass of <i>Goal</i>	E	Right-oriented parallelogram Right-oriented parallelogram with dashed border	Symbol redundancy
3	LeafGoal	abstract class			Cannot be instantiated
4	Requirement	subclass of <i>LeafGoal</i>	E	Right-oriented parallelogram with bold border	
5	Expectation	subclass of <i>LeafGoal</i>	E	Right-oriented parallelogram with bold border	
6	BehaviouralGoal	abstract class			Cannot be instantiated
7	Achieve goal	subclass of <i>BehaviouralGoal</i>	E	Goal with textual differentiation	Achieve[]
8	Maintain goal	subclass of <i>BehaviouralGoal</i>	E	Goal with textual differentiation	Maintain[]
9	Avoid goal	subclass of <i>BehaviouralGoal</i>	E	Goal with textual differentiation	Avoid[]
10	AND-refinement goal	meta-class relationship	<i>goal</i> + R	A line ended by a closed black arrow with a non-coloured circle in the middle	
11	complete AND-refinement goal	meta-class relationship	<i>goal</i> + R	A line ended by a closed black arrow with a black circle in the middle	
12	OR-refinement goal	meta-class relationship	<i>goal</i> + R	A line ended by a closed black arrow with a non-coloured circle in the middle	
13	complete OR-refinement goal	meta-class relationship	<i>goal</i> + R		Symbol deficit
14	AND-refinement soft goal	meta-class relationship	<i>goal</i> + R	A line then a non-coloured circle in the middle followed by bold dashed line ended by a closed black arrow	

15	complete AND-refinement soft goal	meta-class relationship	<i>goal</i> +	R	A line then a black circle in the middle followed by bold dashed line ended by a closed black arrow	
16	OR-refinement soft goal	meta-class relationship	<i>goal</i> +	R	A line then a non-coloured circle in the middle followed by bold dashed line ended by a closed black arrow	
17	complete OR-refinement soft goal	meta-class relationship	<i>goal</i> +	R		symbol deficit
18	Domain Property	class		E	Trapezium (called 'home' shape)	
19	Hypothesis	class		E	Trapezium (called 'home' shape)	
20	conflict link	meta-class + relationship		R	Line with a flash in the middle	
21	Obstacle	subclass <i>BoundaryCondition</i>	of	E	Left-oriented parallelogram	
22	AND-refinement obstacle	meta-class + relationship		R	A line ended by a closed arrow with a non-coloured circle in the middle	
23	complete AND-refinement obstacle	meta-class relationship	<i>goal</i> +	R	A line ended by a closed black arrow with a black circle in the middle	
24	OR-refinement obstacle	meta-class + relationship		R	A line ended by a closed arrow with a non-coloured circle in the middle	
25	complete OR-refinement obstacle	meta-class relationship	<i>goal</i> +	R		symbol deficit
26	obstruction	relationship between <i>Goal</i> and <i>BoundaryCondition</i> (obstruction)		R	A line ended by a closed-arrow with a perpendicular small line	Obstruction: relationship from obstacle to goal
27	resolution	relationship between <i>Goal</i> and <i>Obstacle</i> (resolution)		R	A line ended by a closed-arrow with a perpendicular small line	Resolution: relationship from goal and obstacle
28	Agent	abstract class				
29	SoftwareToBeAgent	meta-class		E	Hexagon	
30	EnvironmentAgent	meta-class		E	Hexagon with a fellow icon	
31	Responsibility	relationship between <i>softwareToBeAgent</i> and <i>Requirement</i> or <i>EnvironmentAgent</i> and <i>Expectation</i>		R	Line with a (white) circle	Responsibility = relationship between requirement and SoftwareToBeAgent or SoftwareEnvironment

32	Operation	meta-class	E	Oval	
33	Operationalisation	relationship between <i>LeafGoal</i> and <i>Operation</i>	R	line ended with a black closed arrow with a white circle in the middle	operationalisation refers to the process of mapping leaf goals, under the responsibility of single agents, to operations ensuring them
34				Rectangle with dashed border	Used to place annotation about a semantic concept (but not itself a semantic concept)
35		Relationship between a graphical construct and its annotation		Dashed line	

Details of the semiotic equation

$n(\text{symbol}) = n(\text{construct}) + n(\text{symbol redundancy}) - n(\text{symbol overload})$ $+ n(\text{symbol excess}) - n(\text{symbol deficit})$

Table A1-2 Details of the semiotic equation

Variable name	Value	Detail
symbol	17	there are 17 symbols in KAOS vocabulary (see figure 6-1)
construct	30	the number of construct in the table above minus the abstract classes (leaf goal, behavioural goal and agent). If we want to take them into account we will have a deficit of 3 symbols
symbol redundancy	1	leaf goals can be represented by 2 symbols
symbol overload	13	each construct that do not have its own symbol (soft goal, achieve, maintain, avoid, requirement, hypothesis, resolution, OR-refinement goal, AND-refinement obstacle, OR-refinement obstacle, operationalisation, complete AND-refinement obstacle, OR-refinement soft goal)
symbol excess	2	
symbol deficit	3	complete OR-refinement goal, complete OR-refinement soft goal, complete-OR refinement obstacle

Annex 2: The modules of the running example

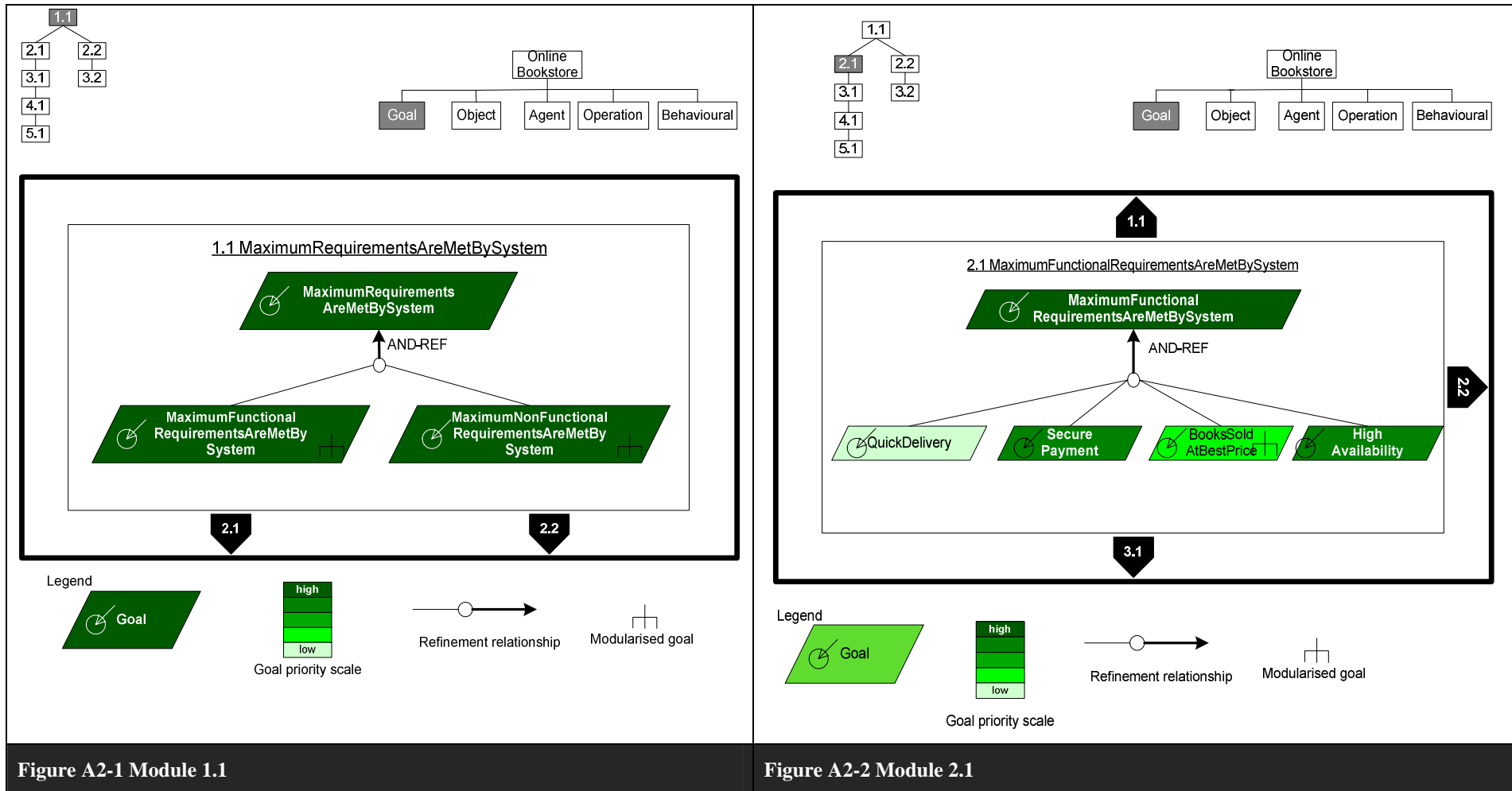


Figure A2-1 Module 1.1

Figure A2-2 Module 2.1

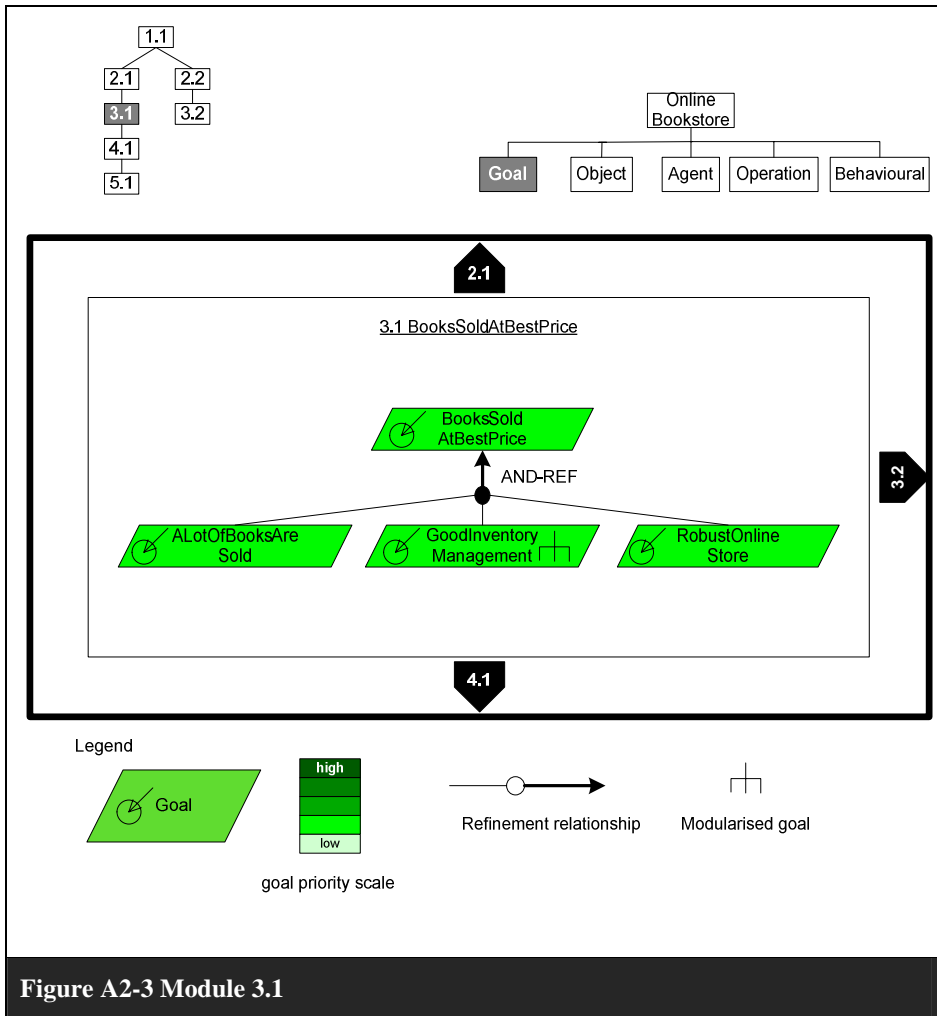


Figure A2-3 Module 3.1

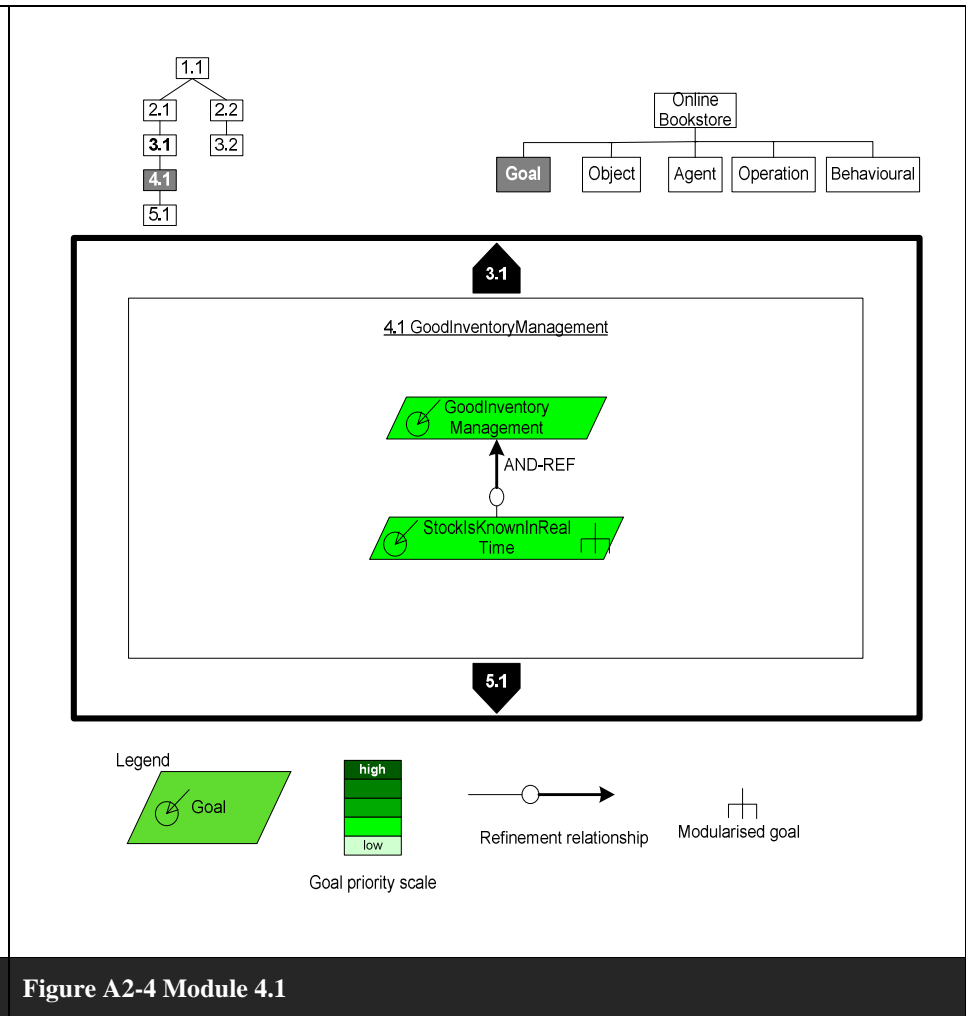
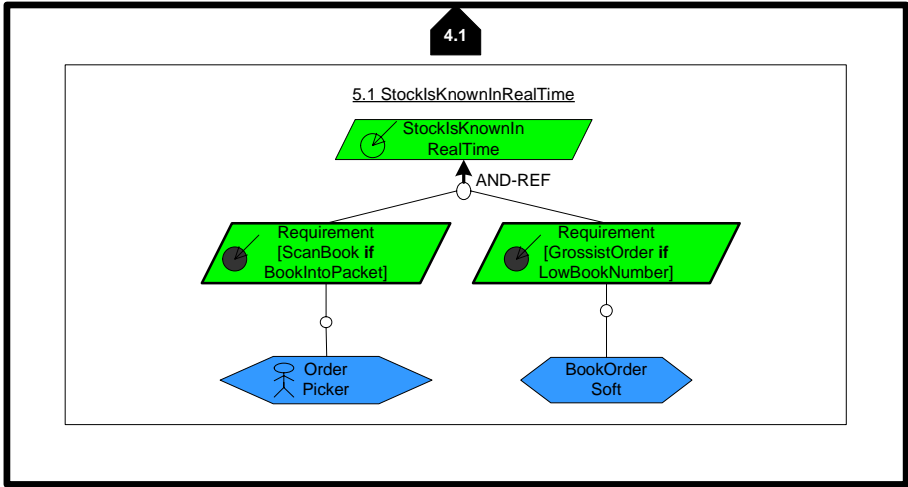
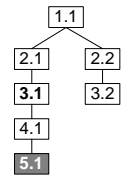


Figure A2-4 Module 4.1



Legend

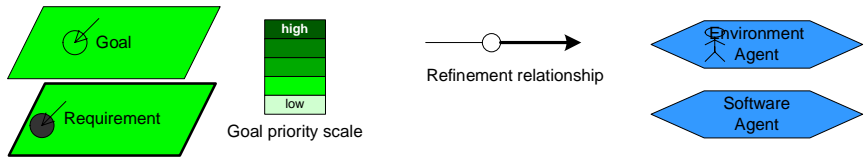


Figure A2-5 module 5.1

