Quality Starts with the Definition of Goals

Robert Darimont, Emmanuelle Delor, André Rifaut
CEDITI, Avenue Georges Lemaître, 21, B-6041 Charleroi -- Belgium
{Robert.Darimont, Emmanuelle.Delor, Andre.Rifaut}@cediti.be

1. Introduction

A weekly journal for private sector executives and directors recently featured an article entitled: "No baker would survive with only 16% satisfied customers. Half of the failures in IT projects are due to inadequate communication". This article referred to a statistic published in 1995 by the Standish Group [STG 95] which reveals a very harsh reality from a sample of more than 8000 IT projects carried out by some 350 American firms: only 16 % of projects can be described as complete successes. They eventually succeed while keeping to the anticipated costs and functions. Almost one in three projects never succeed and the remainder (over half) show a half-measure result with partially installed functions, a cost approaching 200% compared to the initial budget, and with a significant delay.

When the causes of these failures are analysed, it emerges that almost half of them are due to **inadequate analysis of requirements.** The survey goes further and breaks down that inadequacy as follows: lack of information for the user represents 13% of the causes of failure, inadequacy of the specifications 12%, their changing nature 11%, unrealistic expectations 6%, and confused objectives 5%. Discovering a requirement during the course of development leads to cost overruns and delays, which are even larger when this discovery is made late in the day.

This paper presents the principles of the Kaos method, the functions of the Grail tool, the process followed during analyses, and feedback from experience from carrying out requirements analyses, from drawing up IT master plans, strategic analyses and IT specifications.

2. The Kaos method

Most of the software engineering methodologies currently in use are dedicated to the specification of a solution and do not really address the description of the problem, that is, specifying the goals to be met, describing the application domain, deciding who will perform which actions to satisfy the goals, etc.; in one word, to perform a requirement analysis.

A requirements analysis in Kaos [DAR 93, DAR 96, VLA 98{a,b,c}] is broken down into three phases:

- gathering of the information to be used as a guide for the goals to be achieved
- modelling
- drawing up of a report (specifications, synthesis, IT master plan, etc.)

The aim of the modelling phase is to identify all the relevant concepts by completing an information framework having a pre-established structure (metamodel). This framework enables the analyst to classify the concepts that he/she identifies and to relate them to each other.

The framework pre-defines different types of concepts and different types of relationships between concepts. Those concepts types and relationships are explained in the following sections. A single example will be used throughout the paper to illustrate the approach. This example is described first.

2.1 Case study

> *We have been called by the authorities of a large city to help select the best configuration for a new crossroads. The authorities expect a report that motivates the choices we advocate and that anticipates the requirements document for the actual construction of the crossroads.*

Let us observe that we are asked to analyse a problem, not *necessarily* a software problem. If a software component is needed to implement the solution, it will be introduced later on. This yields two key ideas of the Kaos method :

Key idea. A requirement analysis should not only study the software system we have in mind but also the part of its environment with which the system will interact.

Key idea. Before attempting to describe a **solution** in a requirements document, it is necessary to describe and analyse the **problem** which we are faced with. In fact, we strongly believe that a good requirements document specifies the problem to be solved and minimal technical constraints on the solution to be built, no less and no more.

2.2 Goal model

The first task of the analysis will be to address the following questions: 'who are the stakeholders implied in this problem?' and 'what do they want?'. Answers to these questions will be gathered from interviews, written documents, observation of an existing system, and knowledge about the problem domain. The more an analyst is acquainted with a specific domain, the easier it will be for her to find good abstractions and to elicit the needed properties.

Case study. The following table shows some of the stakeholders and some of their wishes for the road crossing example.

Table 1: Stakeholders and their wishes

Stakeholder	Stakehoder's wishes
Car driver	no crashes with other road-users free traffic flow road to follow is clearly known
Pedestrian	safe road crossing short waiting time
Public authorities	construction costs minimised running costs minimised no casualties
Biker	not to be run over by cars free traffic flow

In Kaos, stakeholder's wishes are named **goals**. Goals can be shared by different stakeholders and goals typically need the cooperation of different stakeholders to be satisfied.

For instance the goal *Free traffic flow* is shared by car drivers and bikers. It needs the cooperation of all road users; the way the crossroads is configured and managed will probably contribute to satisfy this goal.

Goals can often be defined in terms of system states to reach (or leave) or system states to maintain (or avoid). A goal can also require the optimisation of some parameter(s).

The Kaos method invites the analyst to collect and relate goals with each other: some goals contribute to other ones; others can be conflicting with each other. Abstraction can be introduced to highlight common goals. To help the analyst relate goals with each other, he/she is prompted to ask the following questions for each goal that has been identified:

- the '**why**' question: why is this goal needed ? What is the upper-level goal that motivates this goal?
- the '**how**' question: how can this goal be satisfied? What are the lower-level goals that will contribute to the satisfaction of this goal?

In Kaos, a goal **refinement** is a relationship between goals that allows the analyst to connect a goal with its contributing subgoals.

Case study. Figure 1 shows a Kaos goal diagram related to the case study. Rectangles stand for goals. Circles stand for refinements of upper goals into a set of lower subgoals. Certain goals have been introduced to put some structure in the graph as for instance *Costs minimised* or *Adequate crossroads*. A goal can contribute to more than one parent goal; for instance, the goal *Road to follow clearly identified* contributes to the *Free traffic flow* as the driver will not hesitate about the direction to follow if road signs are clear; it also contributes to the goal *Safe road crossing* as a driver hesitating about his/her way or looking for road signs increases the risk of a crash.

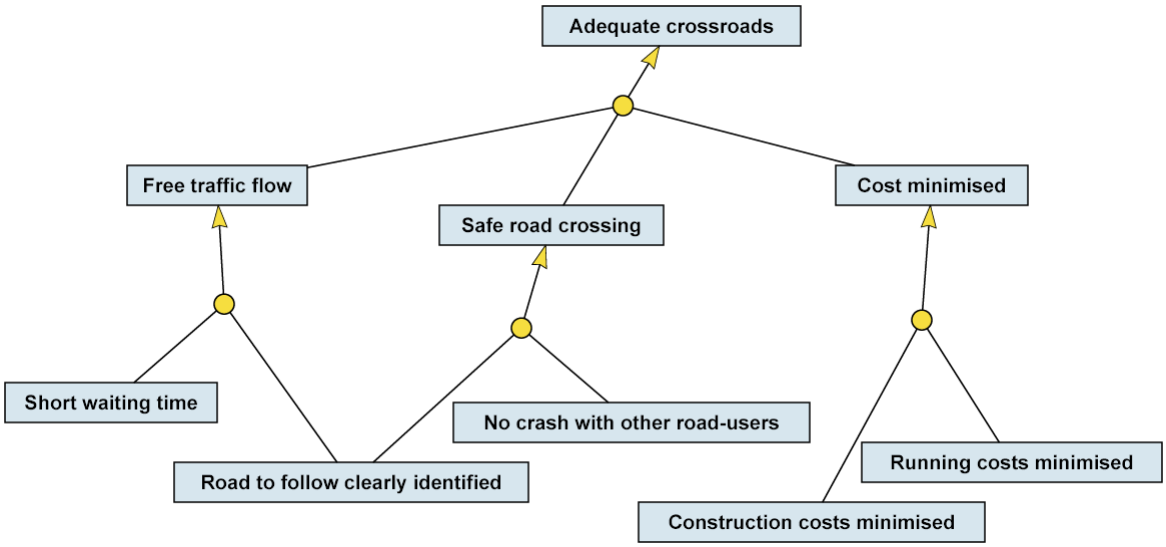

FIGURE 1 : High-level goals

Alternatives describe different ways to achieve goals. They can be motivated by different policies or design decisions as for instance the technology used or the quality of the solution they propose (reliability, performance, ...). Alternatives are not necessarily exclusive, that is, one can decide to implement a system with several of them included, for instance, to improve system reliability. Alternatives do not define partitions as different alternatives may share common subgoals.

Case study. Figure 2 shows some alternatives refining the goal *No crash with other road-users*. To avoid a crash, that is, two crossing road-users at the same place at the same time, one can choose to never have two crossing road-users at the same place by separating the lanes they follow (as in interchanges). One can also choose to allow crossing road-users to be at the same place but never at the same time by regulating the traffic flow. Both alternatives share a common subgoal: highway code must be respected by the road-users. Traffic regulation, in turn, can be achieved either by forcing road-users to merge their trajectories inside the road crossing (roundabout) or by alternating traffic in each crossing direction. A way to implement the latter is to require traffic lights.

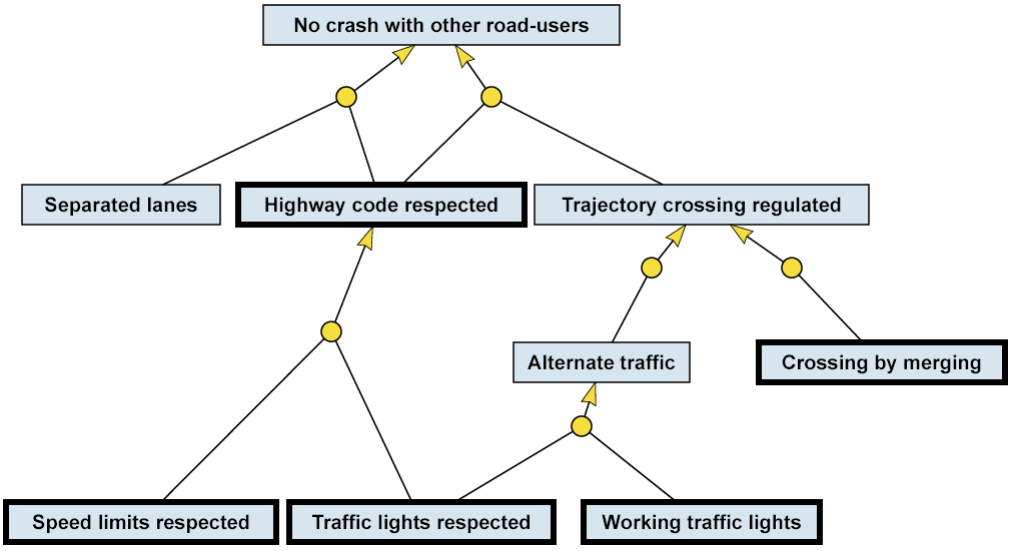

FIGURE 2 Alternatives

Conflicts. Goals expressed by different stakeholders may be conflicting. In Kaos, a conflict stands between two goals if there can be system states in which both goals are logically contradictory. It is important to identify conflicting goals as early as possible in the life cycle to minimise the risk of system rejection by some category of stakeholders.

Case study. The goal *Separated lanes* in Figure 2 may be conflicting with the goal *Construction costs minimised* in Figure 1as the construction of an interchange is one of the most expansive designs to organise a road crossing. The goal *Speed limits respected* in Figure 2 may be conflicting with the goal *Free traffic flow* as they could down the traffic flow. Conflicts are graphically represented in a goal graph by connecting the conflicting goals with a labelled link (see Figure 3).

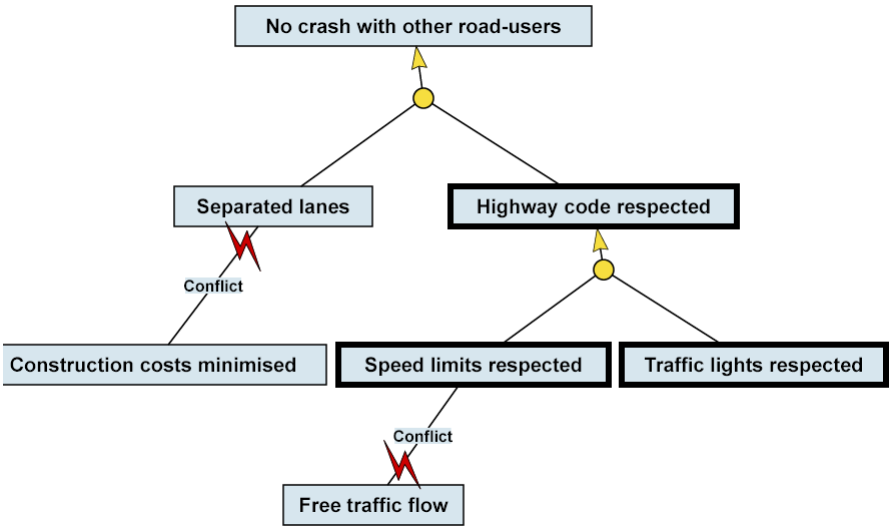

FIGURE 3. Conflits

2.3 Responsibility model

Key idea. In Kaos, a **requirement** is a goal, the satisfaction of which can be put under the responsibility of an agent.

An **agent** in Kaos is a processor (human or software) capable of behave to achieve assigned goals.

Case study. Requirements have already been shown in Figure 2. There are represented as goal rectangles appearing with a thick border. For instance, *Highway code respected* and *Working traffic lights* are examples of requirements. The figure also shows that requirements, like goals, can be refined. For instance the requirement *Highway code respected* is refined into two requirements: *Speed limits respected* and *Traffic lights respected*.

A Kaos responsibility model typically contains several diagrams. Each diagram focuses on the responsibilities of an agent in the system. The agent is drawn at the centre of the diagram and all the requirements for this agent are displayed around it.

Case study. Figure 4 shows requirements for the traffic light controller agent. Agents are represented by hexagons. Responsibility relationships between an agent and a requirement are also represented graphically.

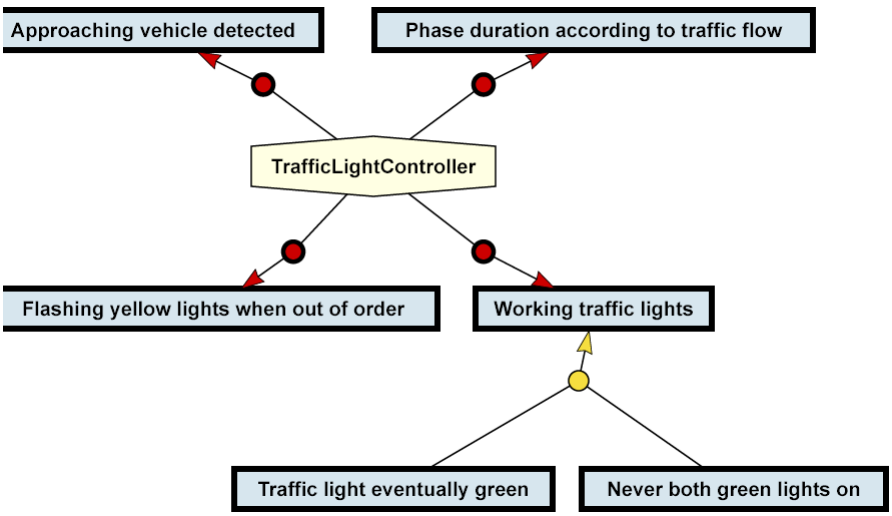

FIGURE 4: Responsibility relationships

During analysis, different candidate agents can be assigned to fulfill a requirement. Selection among agents can result from a cost, reliability or performance comparison. Even if, in the final requirements document, only one agent is responsible for a requirement, it is important to keep track of the rationale behind the choice of a particular agent for satisfying a requirement as this choice can be revisited later on for next releases of the system.

Key idea. A Kaos model is elaborated non only to produce a particular result at a particular instant in time (as for instance a requirements document), but also to accompany the system documentation during its lifetime.

2.4 Object model

Goals are defined by means of properties relating system states with each other. In Kaos, these system states consist of object states. Object states aggregate attribute name-value pairs defining the underlying objects. The Kaos object model allows the analyst to define Kaos objects almost as a UML analyst defines classes in a class diagram with the following differences:

- A distinction between four kinds of objects is made: entities, relationships, agents, and events. Agents are active objects capable of actions to achieve goals. **Entities** are passive objects whose states are modified by actions performed by agents. **Events** are instantaneous objects that cause and stop actions. Agents, entities and events are autonomous, that is, their definitions do not rely on the definition of other objects. **Relationships**, on the other hand, are objects dependent on other linked objects.
- Relationships are first-class objects. They can be binary, ternary or n-ary. Attributes can be attached to them; they can be specialised. Minimum and maximum cardinalities can be specified on links. A cardinality on a link indicates a number of times a same instance of an object can be linked through the relationship.

Case study. Figure 5 shows an object model in which the agents for the crossroads have been put together. The agent *Roaduser* has been introduced as a common abstraction for *Car drivers, Bikers,* and *Pedestrians.* This agent will be used in the model when the indication of a specific agent category is not relevant. Entities are represented by parallelograms and relationships by flattened hexagons (compare with agents). Links between a relationship and its linked objects are shown by arrows linking the relationship to the linked objects. For instance, *Crossroads configuration* is a ternary relationship between two entities *Crossroads* and *Lane*, and an agent: *Traffic Light Controller*.

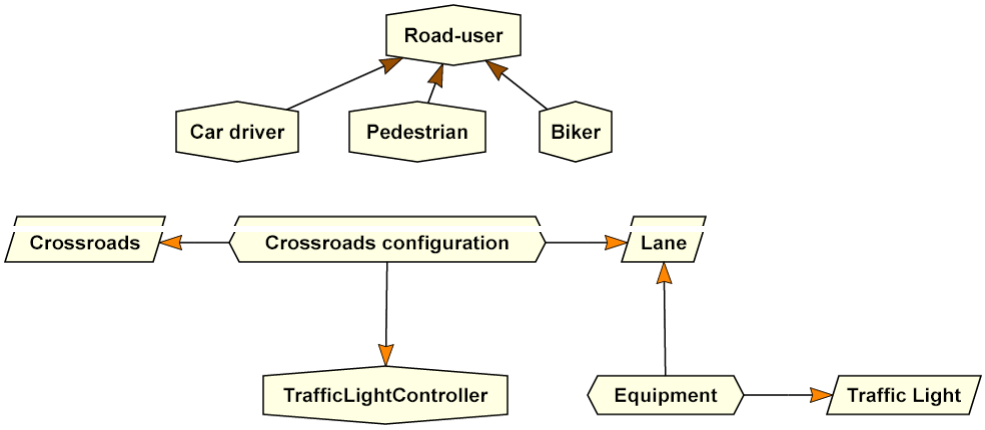

FIGURE 5. Object model

2.5 Operational model

Besides defining goals, objects and requirements under the responsibility of agents, Kaos allows the analyst to describe the behaviours that agents have to adopt in order to satisfy the requirements assigned to them. In Kaos, behaviours are named **actions** and represent state transitions. They are described in terms of preconditions (minimal conditions on the system state which must be met before executing the action), trigger conditions (conditions on the system state making execution of the action mandatory), and postconditions (conditions on the states resulting from performing the action if the precondition holds). Each kind of condition is split into two categories:

- domain conditions: conditions that hold independently of specific requirements. They describe a "standard" behaviour according to predefined agent capabilities.
- strengthened conditions: additional conditions that are introduced to strengthen agents' standard behaviours so that they will be able to achieve preset requirements.

When an action allows an agent to fullfill a requirement, an **operationalisation** relationship is created between the requirement and the action.

Key idea. A clear distinction between domain and strengthened conditions favors traceability from goals to operational required behaviours. Impact and risk analyses will also be easier to produce when future releases of the system put certain requirements and behaviours in question.

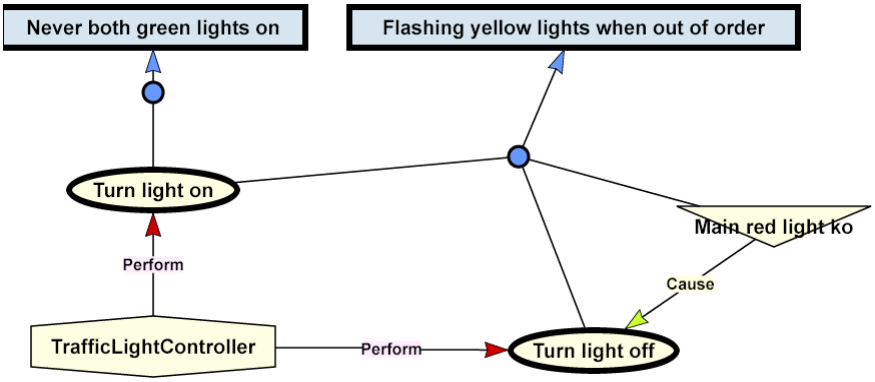

FIGURE 6. Operational model

Case study. Figure 6 shows the action *Turn light on* performed by the agent *Traffic Light Controller*. The action operationalises the constraint *Never Both Green Lights on.* Details on the specification of the action show that the domain precondition for the action is 'the light must be turned off' and that the domain post-condition for the action is 'the light is switched on'. A strengthened precondition will prescribe that if the action is applied on a green light, the green lights for the other directions must be off. The figure also shows how the constraint *Flashing yellow lights when out of order* is operationalised by two actions (*Turn light on* and *Turn light off*) and one event that triggers the *Turn light off* action (*Main red light ko)*.

3. The Grail tool

A prototype tool has been developed to support Kaos analyses [DAR 97]. It consists in a graphical editor to represent the concepts graphically, a form editor to describe each concept textually, a report generator extracting information from the model and presenting it according to a predefined template, a database for querying about concepts and their "neighbourhoods", and a hypertext generator for browsing through the model.

To address the increasing interest in the methodology, a commercial version of the tool, Grail, is currently being developed. Version 1 of the tool will contain the following components:

- a graphical editor to represent the concepts and their relationships,
- a text editor allowing the analyst to record interview summaries or to associate descriptive texts to diagrams,
- an attribute editor to specify predefined attribute values or user-defined attribute-value pairs
- an explorer to retrieve diagrams, text documents, and concepts by names, types or occurrences
- a hypertext documentation generator that allows one to inspect the entire model with a Internet browser tool in a very user-friendly way.

The Grail tool is a ***meta*** case tool: the Kaos methodology is not hard-coded in the tool. Evolution or specific customisations of the tool will be easier to implement. Specifications are saved in XML format, including diagrams (in the SVG format).

The tool will be available on all major platforms since it is implemented in Java. It has been designed to be easily integrated with other CASE tools. Integration can be achieved in three ways:

- by exchanging data in XML format;
- by interfacing with an event-based API;
- by querying the tool with OQL (an object-oriented query language standard similar to SQL).

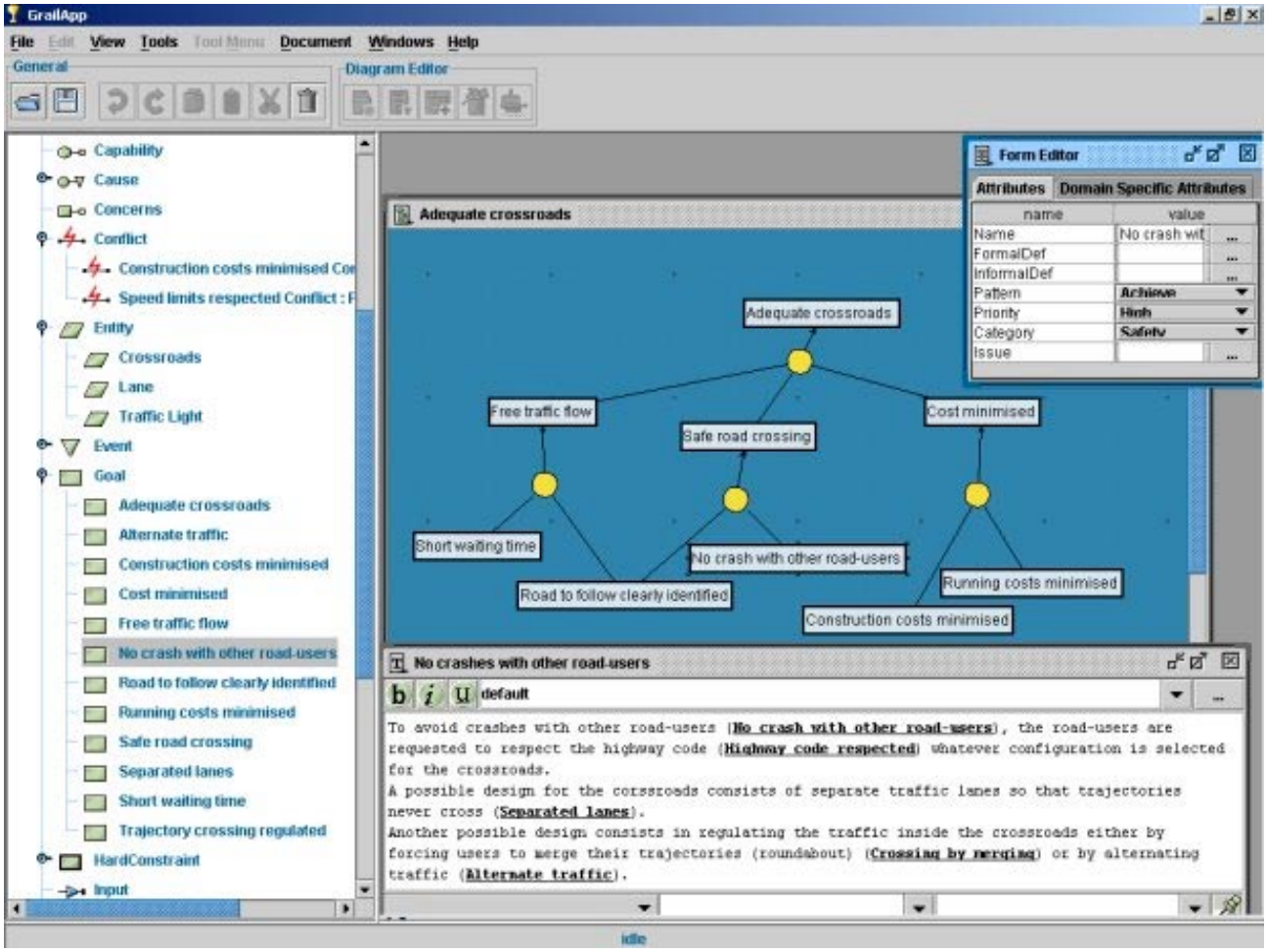

FIGURE 7. Grail

Figure 7 shows a screen shot of the tool. The explorer appears on the left of the screen. The right part shows a diagram edited by the graphical editor, the form editor and a descriptive text edited by the text editor.

Figure 8 shows the same information but in the Web-based generated documentation.

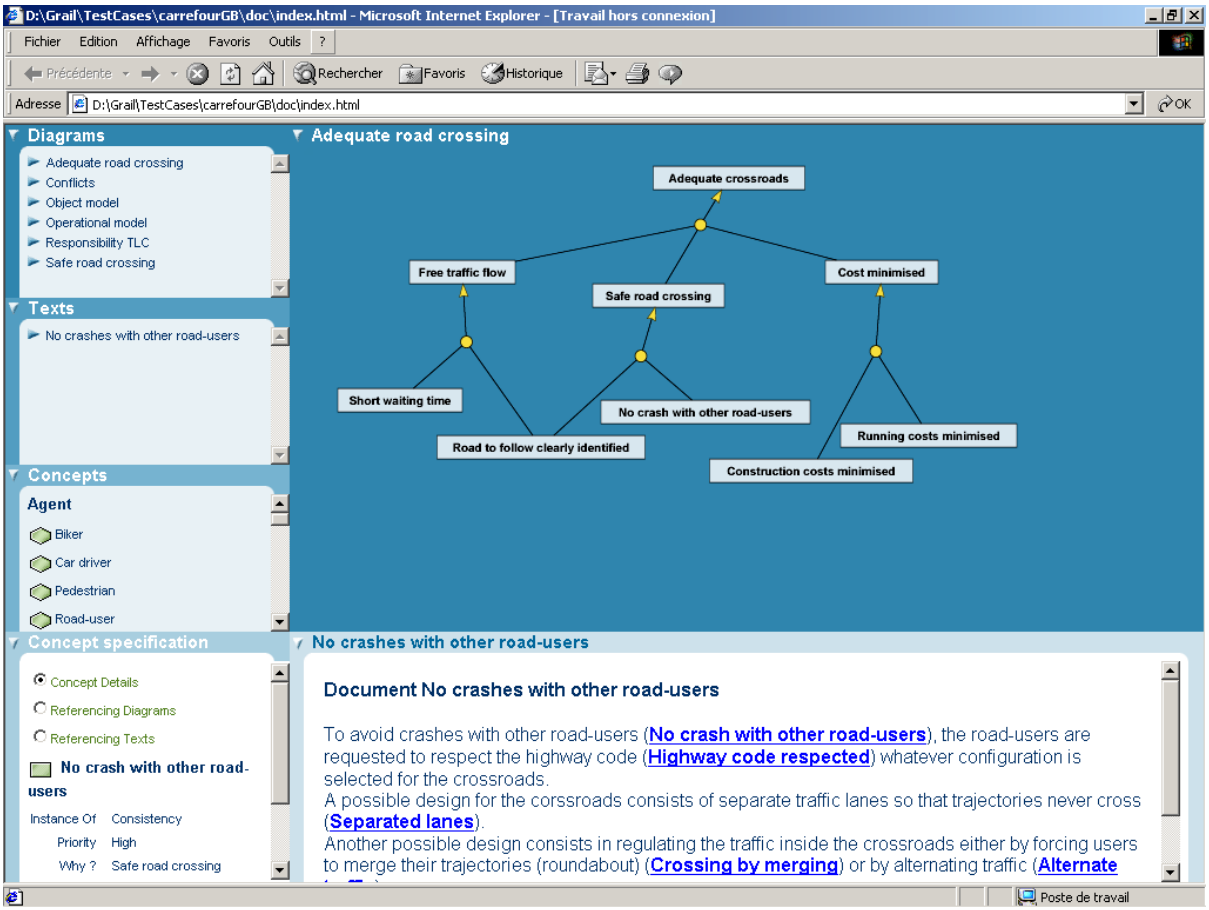

FIGURE 8. Grail Web-based documentation

4. Experience report

Experience acquired from numerous industrial studies performed by CEDITI has shown that the Kaos framework is highly efficient when carrying out requirements analyses, devising IT master plans or producing strategical analyses.

The following table summarises the kinds of projects already realised.

Publishing	Reqs engineering for a complex copyright management system; Reqs document for Media Sales, Distribution & Advertising Management
Aeronautics	Reqs traceability for Air Traffic Control Procedures
Drugs Industry & Distribution	Strategic analysis; Reqs for an e-learning system
Telecommunications	Requirements re-engineering of a cable telephone system
Language Industry	Reqs for Web-based professionnal and on-the-fly translation tools
Hospitals	IT plan, Reqs for standard clinical reporting
Earthmover Factory	Finite scheduling optimisation

Requirements documents (typically 100 to 150 pages long) produced by the method and its associated tools are IEEE 830 standard compliant. About 20% of such documents consist in an exhaustive glossary of terms used in the rest of the document. The glossary is directly derived from the Kaos object model. The requirements are progressively introduced in the text according to a top-down, linearised traversal of the goals graph.

Our experience has shown the following benefits and returns of the Kaos approach:

- requirements documents produced with the method have been unaminously recognised by our customers for presenting remarkable qualities: they are well structured, self-contained, motivated, easily understandable, and calls for tender is managed more easily;

- the Kaos model is a highy effective way to communicate about the system: it provides a common framework helping the different stakeholders to understand each other's viewpoints: managers acknowledge goals, requirements and assignments that underly operational models; technicians acknowledge operational models that are motivated by the goals and that can be easily mapped to UML;
- it turns out that many companies lack vertical traceability between the company strategy and the requirements documents produced on the one hand, and between the requirements document and the solution specification on the other hand. A Kaos model allows one to trace requirements to goals and to trace high-level, coarse-grained behavioral specifications to requirements.

5. Conclusions

In the requirements engineering research community, the Kaos approach is considered as one of the most prominent methods for eliciting, analysing, and formalising requirements (see the References section below). Experiences in various industries have proved the benefits of the approach and have shown how well it improves the quality of requirements documents and, therefore, the quality of solutions based on such documents.

We now feel that the time has come to allow a larger number of companies to benefit from the approach: this explains why CEDITI will henceforth commercialise Grail and provide support for the Kaos approach.

New features that will soon extend the tool's capabilities include integration with existing market leader tools in requirements management and integration with existing UML modeling tools. More advanced features will be progressively incorporated. Among them, the edition of formal assertions and embedded tools to enable features like model checking, simulation, and test generation.

We strongly believe that all these features put together will easily convince professionals that quality definitely starts with the accurate definition of goals.

Acknowledgment. The Kaos approach results from several research projects led by Prof. A. van Lamsweerde (University of Louvain) and funded by the European Union, the Belgian and the Walloon governments. The Grail tool which will soon be commercialised is being built by a strongly motivated team of Java developers: Jean-Luc Roussel, Cédric Nève, Philippe Legrain, Denis Genard, Denis Ballant and Philippe Massonet, with the significant contribution of André Rifaut for providing requirements, feedback, and prototypes for the tool. Kaos analyses that have proved the benefits of the approach have been led by the authors of this paper. Several PhD theses have contributed to explore advanced features eventually deployed in the tool; more are on their way.

6. References

[DAR 93] Dardenne, A., van Lamsweerde, A., Fickas, S., Goal-directed Requirements Acquisition, in Science of Computer Programming, Vol 20, 1993, pp 3-5

[DAR 96] Darimont, R., van Lamsweerde A., Formal Refinement Patterns for Goal-Driven Requirements Elaboration, Proc. FSE-4, ACM 10/96, pp 179-190

[DAR 97] Darimont, R., Delor, E., Massonet, P., van Lamsweerde, A., Grail/Kaos : An Environment for Goal-driven Requirements Engineering, in Proc. ICSE 19, 05/97, pp 612-613

[STG 95] Standish Group, http://www.standishgroup.com/chaos.html

[VLA 98a] van Lamsweerde, A. , Darimont, R., Letier, E., Managing Conflicts in Goal-driven Requirements Engineering, IEEE Transactions on Software Engineering, 11/98

[VLA 98b] van Lamsweerde, A., Willemet, L., Inferring Declarative Requirements Specifications from Operational Scenarios, IEEE Transactions on Software Engineering, 12/98

[VLA 98c] van Lamsweerde, A., Letier, E., Integrating Obstacles in Goal-driven Requirements Engineering, Proc. ICSE'98, IEEE-ACM, 04/98

A full bibliography of publications about the method can be found at URL http://www.info.ucl.ac.be/research/ projects/AVL/ReqEng.html.

7. Bibliographical sketch

Robert Darimont received the MS and PhD degrees in applied sciences (software engineering orientation) from UCL (Université catholique de Louvain), Belgium. He is the manager of the IT consulting and R&D department at CEDITI, the IT transfer center of the University of Louvain. He leads industrial projects using Kaos; and helps companies to adopt up to date and precompetitive software engineering technologies.

Emmanuelle Delor received the MS degree in computer science from the University of Namur, Belgium. She is senior consultant at CEDITI, the IT transfer center of the University of Louvain. She participates to industrial projects using Kaos, and is responsible for the QA program at CEDITI.

André Rifaut received the MS degree in applied sciences (applied mathematics orientation) from UCL (Université catholique de Louvain), Belgium. He is senior consultant at CEDITI, the IT transfer center of the University of Louvain. He participates to industrial projects using Kaos, and is involved in the design of advanced features of Grail, the tool supporting the Kaos methodology.