CREWS Report Series 98-15

From Early to Late Formal Requirements: a Process-Control Case Study

Eric Dubois[1], Eric Yu[2] and Michaël Petit[1]

[1] University of Namur, Belgium, {edu,mpe}@info.fundp.ac.be
[2] University of Toronto, Ontario, Canada, eric@cs.toronto.edu

Published in the Proceedings of IWSSD98, April 1998 Isobe, Japan.

From Early to Late Formal Requirements:
a Process-Control Case Study[*][†]

Eric Dubois[1], Eric Yu[2] and Michaël Petit[1]
[1] University of Namur, Belgium, {edu,mpe}@info.fundp.ac.be
[2] University of Toronto, Ontario, Canada, eric@cs.toronto.edu

Abstract

In this paper, we consider three distinct and connected modelling activities at the Requirements Engineering (RE) level. Within the context of reactive systems, we suggest how these three activities can be supported by the use of appropriate formal languages, namely Kaos, Albert II and Timed Automata. The i framework is used for linking the various formal models and for providing a "high level" model in terms of which organizational issues are captured. A small process control example is used to illustrate the proposed approach.*

1 Introduction

For a long time, requirements analysis has been considered a key activity in any Software Engineering (SE) process. Recently, we have witnessed the emergence of Requirements Engineering (RE) as a distinct process within the SE process. This results from some distinguishing features of RE, e.g., (i) the focus on real-world problems rather than on the implementation of its software-based solution and (ii) the variety of involved stakeholders ranging from domain experts and end-users to software engineers.

Like the SE process, the RE process also needs to be characterized in terms of its various activities and associated inputs and outputs. Tentative characterizations of the RE process have been proposed either 'in-the-large' (see, e.g., the 'magic cube' RE process proposed in [16]) or 'in-the-small' (see, e.g., the basic elicitation, modelling, verification and validation activities described in [13]). In this paper, we would like to focus on aspects related to the modelling of the behaviour of the required software. Comparable to the so-called 'transformational' view proposed in some SE process [2], we claim that it is possible to make a distinction between different modelling activities with different kinds of inputs/outputs at the RE level. This fact can be easily established if we consider the variety of notations that have been proposed for RE. For example, there are notations (such as 'use-cases' in UML [17]) which focus on the behaviour of the software seen from the point of view of its environment, and other ones (such as SART [9]) which focus on the modelling of the software internals and consider the environment as a black box capable of issuing and receiving messages or dealing with events.

In this paper, we will consider three distinct and connected modelling activities:

1. the modelling of the *goals* (purposes or objectives) associated with the introduction of some software within an organization;

2. the modelling of the *software requirements*, i.e. the role played by the software in solving the goals pursued within the organization;

3. the modelling of the *software internals*, i.e. the functional behaviour of the software and the protocols used for exchanging input/output messages with its environment.

To illustrate these three types of activities, we will use a process-control case study featuring some real-time constraints. The choice of this specific application domain will influence the process roughly sketched above:

- in most cases, problems arising in this application domain do not lead in the introduction of a single piece of software but of a composite system [7] made up of humans, devices, hardware and software components. We therefore prefer to use the word *system* instead of *software* and, thereby, make the distinction between *system goals*, *system requirements* and *system internals*.

[†]This work is partially funded by the ESPRIT LTR project CREWS *Cooperative Requirements Engineering With Scenarios*

- the development of such applications requires consistent *specifications* that have precise semantics and which enable formal reasoning. That is why, in this paper, we will use formal specification languages rather than natural languages or semi-formal 'box and arrows' notations.

The case study that we will use is inspired by the *Coalmine* example introduced in [18]. This case study is about the following:

> There are several dangerous factors in a coalmine, two of them are the level of water percolating in the mine and the presence of methane. In any case, workers can only work safely in the mine if the levels of water and of methane are below critical values.

Formal specification languages that we will use are: *Kaos* [5] for reasoning about the system's goals, Albert II [6] for specifying the system's requirements and *Timed Automata* for modelling the system's internals.

Following our proposed specification process results in the production of three distinct products written in three different specification languages, each of them having its own merits for the supported activity. This means that it is often not obvious how fragments of one product are related to fragments of another. We propose that such relations should be established at another level: the 'why' level. Our three products describe the "what's", but as it has been show at the SE level, it is equally important to capture the "why's" behind them, i.e., the rationales underlying their elaboration. At the RE level, such rationales should be characterised in terms of the *organizational* environment: the actors involved in the organization, their responsibilities, the choices they make and the social dependencies existing among them. To capture them, we will use the *i** framework [22]. All along the proposed RE process, we will show how the *i** model complements each product specification and how the models evolve from one product specification to the next one.

The rest of this paper will follow the structure of the ideas presented above. In Section 2, we will describe the *system goals* level using *Kaos* and introduce the *i** framework. In Section 3, emphasis will be put on *system requirements* and we will show that the Albert II language is well suited for modelling them. Then, in section 4, we will focus on the specification of *system internals*. Finally, the paper will conclude by outlining some ongoing efforts relating to a possible infrastructure for linking the different descriptions and for supporting change management.

2 System's goals

We will first characterize this activity by the contents of its "what's" and "why's" parts.

- **the "what's"**: the output of this activity is a clear picture of the *goals* to be met by the system to be introduced. Goals will be usually structured in terms of an 'and/or' hierarchy where higher goals are refined and decomposed in terms of finer goals. These goals will be characterized in terms of expected properties that should hold in the environment (or *problem domain*) when the future system is introduced. This requires the identification of the key components in the environment.

- **the "why's"**: the identification of pertinent *goals* results from an understanding of the actual organization, i.e., the actors in place, their responsibilities and their existing dependencies.

The "why's" part is studied through the production of a first *i** model. Within the context of the *Coalmine* case study, the resulting *i** model is depicted in Fig. 1.

The environment consists of the mining company, miners, and shareholders. (The coal mine itself is not represented explicitly in the *i** model since it is not intentional.) The *i** model represents intentional dependencies among actors. By depending on each other, actors are able to achieve goals that might not be achievable otherwise. However, actors also become vulnerable because of their dependencies on other actors [20].

The company depends on shareholders for capital to operate the mine, while shareholders in turn depend on the company to be profitable. Four types of dependencies – resource, task, goal, and softgoal – are used to differentiate the kinds of autonomy that the actors have in their dependency relationships. Intentional actors can be further differentiated into *agents, roles*, and *positions*. Agents have physical embodiment, while roles are abstract. An agent usually occupies a position which is a bundling together of several roles.

In our example, a mine worker (a person) occupies the miner position, which has a role "Do Mining". In an organization, roles need to be defined and packaged into positions. Agents with the right qualifications are sought to fill the positions. There can be intentional relationships among agents, roles, and positions. [22].

The miner position, potentially covering several roles, depends on the company for wages (a resource-dependency). The company depends on the "Do Mining" role of the Miner to extract coal according to some procedures (a task-dependency). The mine worker (human person) depends on the company to maintain safe working conditions (a softgoal-dependency) in the coal mine.

The above features of *i** focus on the external relationships between actors. To model and reason about how an actor achieves its goals, *i** provides a model for describing

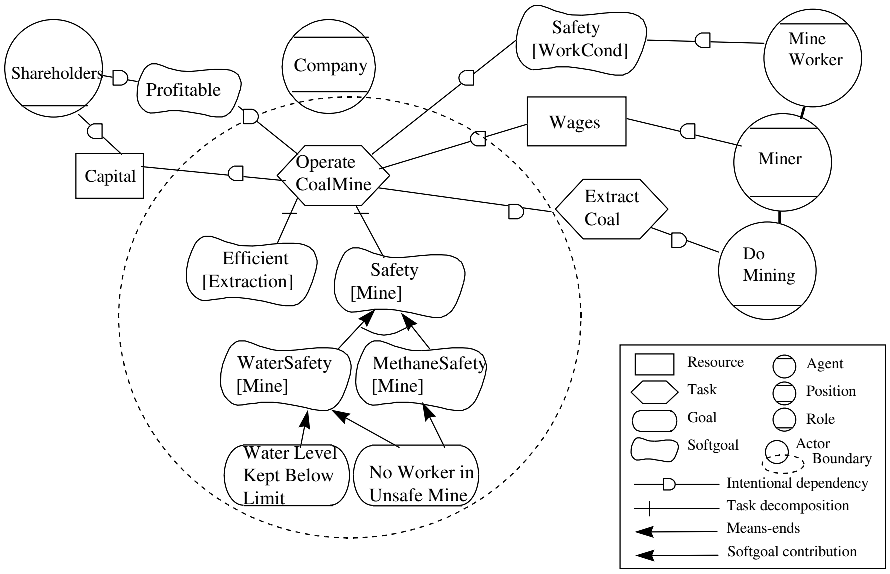

Figure 1: the initial *i** model.

an actor's rationales. We briefly illustrate this in the example. The main task of the mining company is to operate the coal mine. This task can be decomposed in different ways into subgoals and subtasks (not shown). The choice among different ways of operating a coal mine is guided by softgoals – for example, that the extraction process be efficient, in order to be profitable and to meet payroll; and that the mine be safe, in order to provide safe working conditions for the worker. Safety in the mine can be achieved in terms of safety in the water level, and safety in the methane level.

These softgoals can in turn be more precisely characterized in terms of two goals (*Water Level Kept Below Limit* and *No Worker in Unsafe Mine*). Such goals should be defined explicitly. This could be done with natural language but, in this case, we prefer to use a more formal framework, that is the *Kaos* [5] language. At this stage of the RE process, we do not need to use all the concepts proposed in *Kaos* but only a subset dealing with this central concept of *goal*.

According to *Kaos*, a goal is a *non-operational* objective that motivates the introduction of a new system. Coming back to our case study, we can provide the formalization for the two identified goals (see Fig. 2).

The description of goals is only possible through the identification of the *objects* (according to the *Kaos* terminology) belonging to the application domain (environment of the system): for example, the level of water is a so-called

Goal 1

System Goal *Avoid* [WorkerinUnsafeMine]
Instance-of SatisfactionGoal
FormalDef (forall m: MineWorker)
Inside(m) $\Longrightarrow$
((LevMet(Mine)$\leq$ LevMaxMeth) and
(LevWat(Mine)$\leq$ Lev1Wat))
InformalDef miners cannot be in the mine when the level of methane or the level of water is exceeding the limit.

Goal 2

System Goal *Maintain* [WaterLevelKeptBelowLimit]
Instance-of SatisfactionGoal
FormalDef LevWat(Mine)$<$ LevelMaxWat
InformalDef the level of water should not exceed a certain level.

Figure 2: *Kaos* goals expression.

Goal 1

System Goal *Achieve* [NoWorkerinUnsafeMine]
Instance-of SatisfactionGoal
FormalDef (forall m: MineWorker) Inside(m) and
(((LevMet(Mine)> LevMaxMeth) or
(LevWat(Mine)> LevMaxWat)))
$\Longrightarrow$ sometimes$_{<5min}$ not Inside(m)
InformalDef if the level of methane or (and) the level of water is exceeding the limit, then the workers are leaving the mine within the next 5 minutes.

Goal 2

System Goal *Maximize* [WaterLevelKept-BelowLimit]
Instance-of SatisfactionGoal
FormalDef LevWat(Mine)< LevMaxWat
InformalDef in normal situations (no overfloading), the level of water should not exceed a certain level.

Figure 3: *Kaos* goals expression (revisited version).

entity (passive object) while mine workers are *agents* (active objects) as in *i** .

Kaos is based on a formal temporal logic which supports reasoning on the specifications. For example, in our case study, one may wonder about the respective values of *LevelMaxWat* and *Level1Wat*. If *Level1Wat* is greater than *LevelMaxWat*, then we have to revise the first goal in order to simplify it (since the second guarantees that the *Level1Wat* will never be achieved).

Formal reasoning in *Kaos* is also at the basis of goal ('and/or') reductions. This aspect is not illustrated here (see [19] for more details). Revisions of goals can also occur in some *de-idealization* process. This is needed because initial goals may be over-optimistic. This is the case in our example: on the one hand, it is difficult, due to sudden changes of the methane level, to guarantee that nobody is in the mine when the level of methane is too high and, on the other hand, there are some situations where the mine can be over-flooded in a few seconds and thereby, the level of water really gets beyond any control. The revised version of our goals is the one shown on Fig.3.

Note that the last goal is a *Maximize* goal denoting that the goal has to be met in normal situations but that there can be 'abnormal' situations where it cannot be reached[1].

[1] giving a formal meaning to a *Maximize* goal would require to use deontic logic where it is possible to distinguish among normal and abnormal situations

3 System's requirements

The output of the previous activity is the set of goals that should be met by the future system to be introduced. The precise definition of the role of this system is the result of this *system's requirements* definition activity.

- **the "what's"**: The role of the system from the environment (external) point of view is defined by (i) characterizing the properties of the *objects/agents* which should influence the system and (ii) specifying what is the control brought by the system on the environment.
- **the "why's"**: The requirements engineer should understand the rationale behind the introduction of a system with a certain behaviour instead of another system with some other behaviour. At the organizational level, new dependencies are established which are inherent to the introduction of the new system.

At the *system requirements* level, we need to *operationalize* the system's goals defined earlier in terms of constraints related to the role of this system as well as the role of the environment surrounding it. Obviously, several solutions can be imagined for the system and, therefore, different roles can be designed. At that level, the objective is to model these different possibilities and their consequences on the possible organizational dependencies.

Having identified and analyzed organizational goals using the complementary techniques of *i** and *Kaos*, we now proceed to introduce a system in such a way as to meet those goals. It is realized that water safety can partly be achieved by having a system that regulates pumps to keep water level below a safety limit. However, methane level, as well as water level (since pump systems are not perfect) can still exceed safety limits. The system cannot by itself bring about a safe condition. Instead, it can warn workers of unsafe conditions so that workers can exit the mine in time.

Figure 4 shows that the company depends on the system to warn workers of danger, and on workers to exit the mine when so warned. The workers, in their "Observe Safety" role, depend on the system for the warning signals. This is modeled as a resource dependency ("Danger Warning") which leaves open the way this information will be communicated. The system, being a "logical" entity at this stage, is modeled as a position, consisting of the two roles "Pumping" and "Warning".

For brevity, we have focused on the safety aspect in this example. In reality, one usually needs to make tradeoffs among multiple competing goals such as safety and productivity. Using *i** , one would explore the space of possible alternative solutions (e.g., more powerful pumps, more accurate and reliable sensors, or better danger prediction

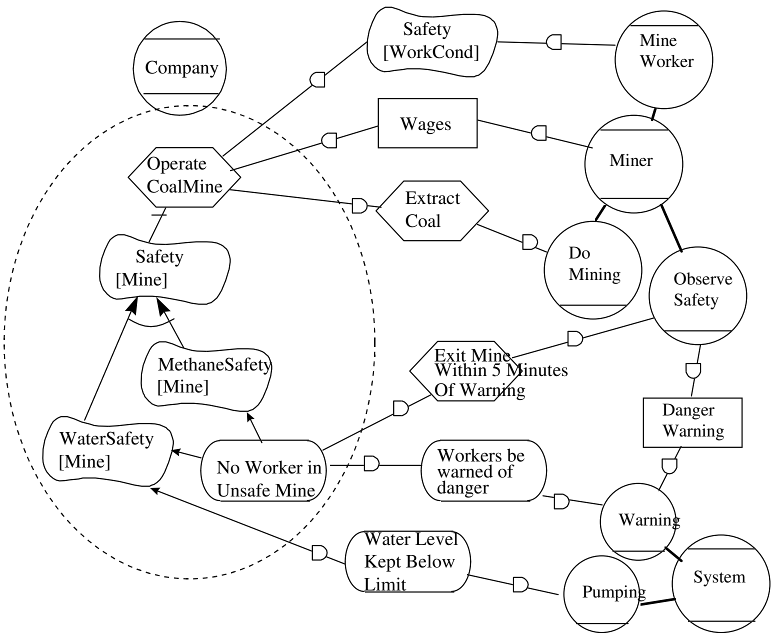

Figure 4: the *i** model (second version).

algorithms) guided by the safety and productivity softgoals and their refinements [4].

At the "what's" level, we need to give a precise specification of the system. At this stage, several authors have argued for a characterization of the system in terms of its environment without revealing the system internals. Jackson and Zave advocate making a clear distinction between *indicative requirements* related to the behaviour of the environment and *optative requirements* associated with the system [12]. The Albert II language will be used for expressing these requirements. Besides supporting the distinction made above, this language is also characterized by:

- its *naturalness*, i.e., the possibilities offered by the language to map informal statements provided by customers straightforwardly onto formal statements expressed in the language. The objective is to avoid the introduction of extra elements (over-specifications) in the formal specification which do not have a counterpart in customers' statements. This naturalness property is guaranteed by the possibility to write requirements by adopting an *operational* and/or a *declarative* style of specification.

- the existence of different *templates* associated with specific categories of requirements and which provide methodological guidelines to the analyst in the elicitation and the structuring of the requirements specifications.

The interested reader can find more information about this real-time distributed RE language and its applications in [6].

A specification in Albert II is made up of (i) a graphical part where the vocabulary is *declared* and (ii) a textual part where the logical formulae *constraining* the admissible behaviours are stated. Figures 5 and 6 illustrate the use of Albert II within the context of our case study. For the sake of brevity, only the requirements inherent to the management of the methane danger are provided. Moreover, the textual specification is presented only for two of these agents.

In Fig. 5, we can see three basic agents, associated with active components having a time-varying behaviour[2] involved in the specification (note that *Mine-Worker* is a class) as well as their associated states (depicted with rectangles) and events/actions (depicted with ovals). *WatChge* and *MethChge* are instantaneous events associated with modifications of the level of water and of methane. There are also export links among agents, those denote how an agent can be potentially affected by the occurrence of an action and/or the value of a state associated with another

[2] agents in Albert II correspond to roles in *i** . However as it can be seen in the graphical declaration, a "CoalMine" agent has been introduced which has no counterpart in the *i** model. This is because this is not an intentional agent involved in social dependencies.

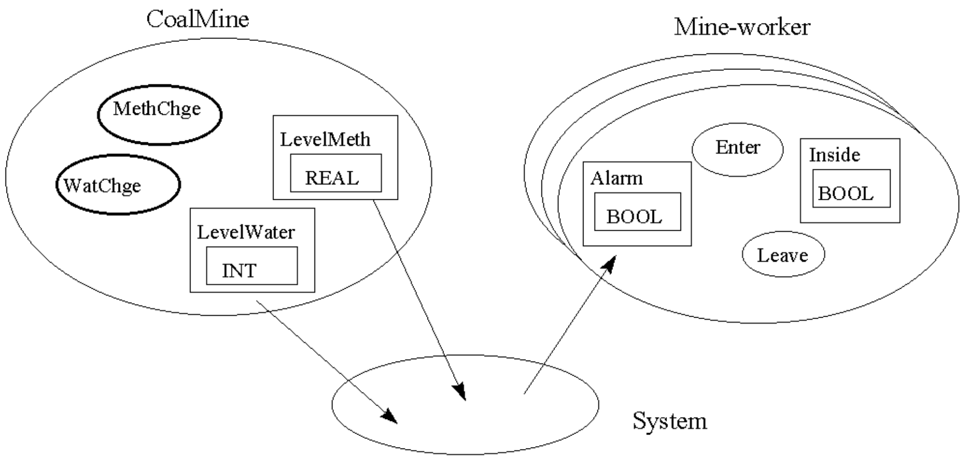

Figure 5: AlbertII graphical declarations.

agent. In our example, we express that *System* controls the *Alarm* state component while it is affected by the values of water and methane levels.

We hope that from the informal comments given in Fig. 6, the reader will be able to get a sense of the expressiveness of AlbertII. The internal behaviour of an agent is characterized in terms of 'operational' and 'declarative' constraints. The constraints associated with the *Mine-Workers* correspond to 'indicative' requirements while those associated with the *System* are 'optative' requirements. It is important to note that they are only expressed in terms of states and actions belonging to the environment. 'Cooperation constraints' are also essential in order to express in what situation we expect an agent to be sensitive to external happenings (states values and actions) as well as under what conditions an agent will have some influence on its environment. For example, one of the *State Perception* clauses associated with the *System* indicates that the monitoring of the methane level is only guaranteed by the *System* if the mine is not over-flooded (*LevWater* < *LevelMaxWat*). At the level of a *Mine-Worker*, we suppose that we have 'reliable' persons who always perceive the status of the *Alarm*. If this could not be the case, the global goals identified in the previous section could not be guaranteed and would need to be revised.

Using the underlying formal framework based on a specific real-time temporal action based logic, it should be possible to prove that the goals expressed in *Kaos* are met by the AlbertII specification.

4 System's internals

The definition of the system's internals, which is the last RE activity, is what is usually considered the main Requirements Analysis activity in the traditional view of the SE process. It aims at providing the functional specification of the system (software).

- **the "what's"**: The architecture of the system is defined in terms of the devices, software, humans and hardware components that interact to make up the system. The functional specification of each of these components as well as the protocols existing between the different agents for exchanging information (within the system as well as with the environment) are provided.
- **the "why's"**: For the design of the architecture, there are several possibilities which have to be evaluated according to the constraints imposed by the customers. For the solution considered, we have to identify new physical agents that need to be introduced in the organization. Dependencies between these new agents and the existing ones have to be carefully identified.

In this specification stage, we identify the components of the system and their inter-relationships. *i** modelling is used to guide the mapping of logical roles to physical component agents, and to relate their requirements back to dependencies from external actors.

Figure 7 shows that the (physical) system consists of sensors for water and methane levels, an alarm and a pump, together with software controllers for the last two. By analyzing the strategic dependencies that external actors (miner, company) have on the components of the system, one can arrive at the requirements on the components – both functional (e.g., conditions under which the alarm needs to be activated) and non-functional (e.g., the performance and reliability of the warning subsystem hardware and software). Some of these requirements relate to the way environment agents interface with the new system. For example, on Fig. 7, we can see that a miner has a new task associated with his/her monitoring of the alarm status.

As we can see, the purpose of this activity is to transform the system requirements into a system solution. There is a strong *mirroring* relationship between the information handled in the problem domain and symbols used for describing the system internals. As it has been indicated by Bubenko [3], information systems manage symbols which are mirroring real information belonging to the problem domain. For control intensive systems, Jackson [11] showed that sensors and actuators are used for connecting system's internal states to the behaviour of real-world entities.

A number of specification languages have been proposed for modelling the behaviour of software components. Most of them are based on automata extended with structuring mechanisms (e.g., Statecharts [8] and SCR [10]) and/or equipped with facilities for dealing with real-time properties [1] [14]. The style of specification used at that level is usually a much more *constructive* (operational) style which reflects the work of the analyst who has elaborated a solution for the problem.

In the case study, rather than to introduce yet another formal specification language, we just provide a (semi-formal) graphical representation associated with a Timed Automaton and we will assume that the reader is familiar with such graphical notations. Fig. 8 shows a fragment of the automaton associated with the behaviour of the *Alarm Controller* in the presence of Methane.

It is possible to check formally that the behaviour of an automaton meets a more global (declarative) property. These techniques could be used for verifying the system's requirements expressed in Albert II.

5 Conclusion

In this paper, we have suggested that the modelling of requirements has to be done at different levels of abstraction (ranging from the early phase to the late phase of RE) and with different formal requirements languages (*Kaos*, Albert II, *timed automata*). We have also shown how the *i** model is considered as providing a "high-level" model that is used from the early phase through the late phase, and for linking the various formal models. In this view, the formal models need to be narrow-spectrum, whereas the strategic modelling of *i** is relatively broad-spectrum.

As it is now recognized at the SE level, it is difficult to develop a 'wide spectrum' language which can support the different SE activities (specification, design and coding). We think that the same reasons can be advocated at the RE level and we have tried to illustrate how *Kaos*, Albert II and *Timed Automata* languages have their own merits for supporting each of the activities. However, in

Agent: System

DECLARATIVE CONSTRAINTS

STATE BEHAVIOUR

$(LevMeth > MaxLevMeth) \Longrightarrow \mathsf{WithinF}_{5min}\ Alarm{=}\mathsf{TRUE}$

> When the level of methane is exceeding the limit the alarm has to be set within the next 5 minutes.

OPERATIONAL CONSTRAINTS

EFFECTS OF ACTIONS

Set : [ ]
  Alarm:=TRUE
Reset : [ ]
  Alarm:=FALSE

COOPERATION CONSTRAINTS

STATE INFORMATION

$\mathcal{K}$ (*Alarm.MineWorker* / TRUE)

> The MineWorker is informed of the status of the alarm at any moment.

STATE PERCEPTION

$\mathcal{K}$ (*Mine.LevMeth* / *LevWater*< *LevelMaxWat*)

> The system is sensible to the mine's level of methane only when the water does not exceed the limit.

$\mathcal{K}$ (*Mine.LevWater* / TRUE)

Agent: MineWorker

OPERATIONAL CONSTRAINTS

EFFECTS OF ACTIONS

Enter : [ ]
  Inside:=TRUE
Leave : [ ]
  Inside:=FALSE

TRIGGERINGS

$Alarm{=}true \wedge Inside\ /\ 0 \rightsquigarrow Leave$

> The MineWorker should leave when he is in the mine and the alarm is set.

COOPERATION CONSTRAINTS

STATE PERCEPTION

$\mathcal{K}$ (*System.Alarm* / TRUE)

> The mineWorker is always aware of the status of the alarm.

Figure 6: Albert II constraints on the *MineWorker* and *System* agents.

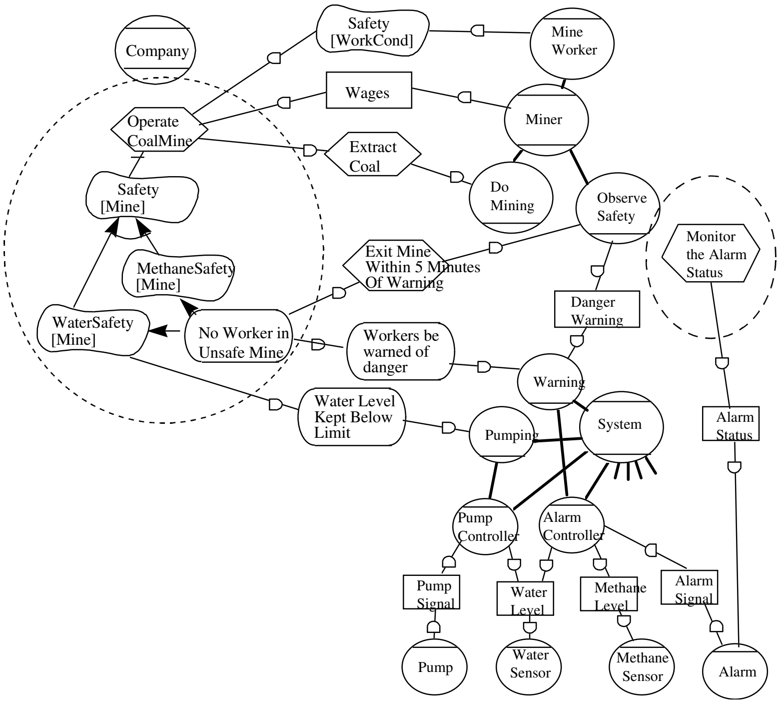

Figure 7: the *i** model (final version).

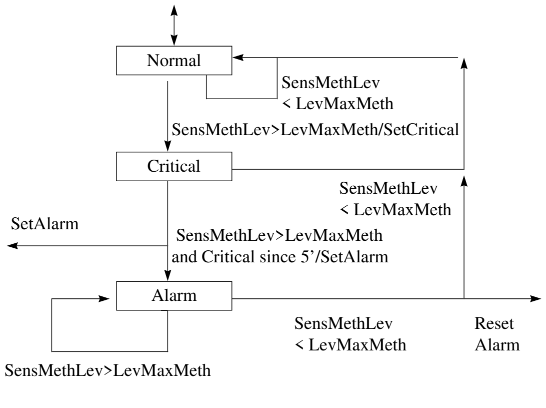

Figure 8: The automaton associated with the alarm controller (partial).

practice, it may happen that some languages originally designed for supporting one activity can also be used for supporting another activity. For example, at the system's requirements stage, it may happen that customers express their requirements only in a state/transition constructive style. In such a case, an automaton-based formalism will be preferred to Albert II since its declarativeness property is not used.

But as long as different languages are used, then a key issue is the development of an integrated framework to support and guide the interplay of the RE activities at the various levels, and to support traceability and change management. As a first step in this direction, we can report on the on-going work performed by the authors around the coupling of the *i** and Albert II languages [21]. Separate tools exist for the two languages but both rely on the use of a Telos based repository [15] in which descriptions are stored and organized according to the meta-model associated with each language. Traceability links can be established at the level of these meta-models and impact analysis can be performed on the basis of these links.

Acknowledgments

This work was partially supported by the Esprit projects CREWS (21.903) and CoopIS (ISC-CAN-080 CIS) and by the Information Technology Research Centre of Ontario and the Natural Sciences and Engineering Research Council of Canada. The authors wish to thank Mike Bissener for his contribution to this reported work.

References

[1] M. Archer and C. Heitmeyer. Mechanical verification of timed automata: A case study. In *Real-Time Applications Symposium*, 1996.

[2] R. Balzer. Tranformational implementation: An example. In *IEEE Trans. on Software Engineering*, January 1981.

[3] Janis A. Bubenko. On concepts and strategies for requirements and information analysis. In *Information modeling*, pages 125–169. Chartwell-Bratt, 1983.

[4] K. L. Chung, B. Nixon, J. Mylopoulos, and E. Yu. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, to appear.

[5] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993.

[6] P. du Bois, E. Dubois, and J-M. Zeippen. On the use of a formal requirements engineering language: The generalized railroad crossing problem. In *Third IEEE International Symposium on Requirements Engineering*. IEEE CS Press, January 1997.

[7] Martin S. Feather. Language support for the specification and development of composite systems. *ACM Transactions on Programming Languages and Systems*, 9(2):198–234, April 1987.

[8] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: a working environment for the development of complex reactive systems. *IEEE Trans. on Software Engineering*, 16:403–414, April 1990.

[9] D. Hatley and I. Pirbhai. *Strategies for Real-Time Specification*. Dorset House, 1987.

[10] C. Heitmeyer, B. Labaw, and D. Kiskis. Consistency checking of scr-style requirements specifications. In *Second IEEE International Symposium on Requirements Engineering*. IEEE CS Press, March 1995.

[11] M. Jackson. *System Development*. Prentice-Hall, 1983.

[12] M. Jackson. *Software Requirements and Specifications: A lexicon of practice, principles and prejudices*. Addison-Wesley, 1995.

[13] P. Loucopoulos and V. Karakostas. *System Requirements Engineering*. McGraw-Hill Intl Series in Software Engineering, 1995.

[14] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In *Concur'91: 2nd Intl Conf on Concurrency Theory*. LNCS 527, Springer Verlag, 1991.

[15] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about information systems. In *ACM Trans. on Information Systems*. vol. 8(4), 1990.

[16] K. Pohl. *Process Centered Requirements Engineering*. John Wiley, 1996.

[17] Rational. *Unified Modeling Language: Notation Guide, Version 1.1*. Rational Software Corporation, 2800 San Tomas Expressway, Santa Clara, CA 95051-0951, 1 September 1997. URL http://www.rational.com/uml/1.1/.

[18] B. Selic, G. Gullekson, and P.T. Ward. *Real-Time Object-Oriented Modelling*. John Wiley, 1994.

[19] A. van Lamsweerde, R. Darimont, and P. Massonet. Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt. In *Second IEEE International Symposium on Requirements Engineering*. IEEE CS Press, March 1995.

[20] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. Ph.D. Thesis, Univ. of Toronto, 1994.

[21] E. Yu, P. Du Bois, E. Dubois, and J. Mylopoulos. From organization models to system requirements: a 'cooperating agents' approach. In M. Papazoglou and G. Schlageter, editors, *Cooperative Information Systems*, pages 293–312. Academic Press, 1998.

[22] E. Yu and J. Mylopoulos. Understanding 'why' in software process modelling, analysis, and design. In *IEEE International Conference on Software Engineering, ICSE'94*, 1994.