Towards a Goal-oriented Security Framework for Elaborating Formal Cooperative System Requirements

Eric Dubois, Suchun Wu, and Jean-Marc Zeippen[*]

Institut d'Informatique - Facultés Universitaires de Namur
Rue Grandgagnage, 21, B-5000 Namur (BELGIUM)
{edu, swu, jmz}@info.fundp.ac.be

Abstract

In this paper, we report an experimental use of a goal-oriented framework in the context of the 'securisation' of information systems. Functional requirements for such systems are written in the **AlbertII** *language and goals are expressed in terms of the Chung's model. Moreover, on the basis of a small case study, we suggest how a completely formalized framework may support the whole approach of writing and proving requirements specifications.*

1 Introduction

Since 1990, we have been involved in the design of a fully formal requirements specification language (in the line of languages like RML [GBM86], MAL [PFAB86], ERAE [DHR91], TROLL [Jun93] or LCM [FW93]) for the purpose of capturing functional descriptions inherent to complex, safety critical, distributed, and composite (heterogeneous) systems. The name of the resulting language is **Albert** (which is an acronym for 'Agent-oriented Language for Building and Eliciting Real-Time requirements') and the use of it is reported in recent publications [DDDM94] [DP94] [DDZ95] dealing with CIM (Computer Integrated Manufucaturing) applications and advanced telecommunications systems. In this paper, we focus on the second version of **Albert** for which a specific real-time temporal semantics based on RT-OSL [BS93] is proposed in [DuB95]. The **AlbertII** language is briefly introduced in Sect. 2 by using a small case study in the context of medical information systems.
At the Requirements Engineering (RE) level, one needs not only an adequate requirements specification language but also effective methods for supporting its use when writing requirements documents for complex systems. Recent advances have been reported including:

- approaches based on a progressive elaboration of simple system descriptions (where some simplification assumptions are first made on the systems to be specified) towards more complex ones [Fea89];
- the elaboration of scenarios [PTA94] permitting the critique and validation of the formal specification;
- the use of goal-oriented frameworks where global objectives over the whole composite system are first defined before being refined in terms of different contractual responsibilities that can be attached to the different actors (agents) of the system [Dub89] [vLDM95].

[*] Part of this work is supported by the European Community under Project 8319 (MODELAGE) of the ESPRIT program.

In this paper, we propose to adopt a goal-oriented approach in the context of the 'securisation' of an existing information system. In particular, we have used and instantiated the goal-oriented model proposed by [MCN92] to this end. Some results about its use are described in Sect. 3.
Finally, as the result of having applied a goal-oriented approach, we need at some point to be able to check that the final functional requirements are specified to satisfy the identified goals. In our particular context, we want to be able to prove that some formalized goals are met by the **AlbertII** functional specification. This aspect is covered in Sect. 4 where we suggest some extensions to the **AlbertII** language for the purpose of expressing goals formally.

2 About the AlbertII Language

Without providing an extensive presentation of the **AlbertII** language (see [DDDM94] or [DuB95] for such a presentation), we are illustrating in this section the basic features of the language through a small example. This is further used for discussing the security issues in the following sections. The informal description of the system is described below.

> A Medical Information System (MIS) within an hospital is set up for recording the patients' health data. The MIS is used by Physicians who may update the recorded data. For particular reasons, a specific rule says that the recoded data, which have not been changed for more than 10 years, are removed from the system.

Basically, **AlbertII** is based on a variant of *real-time temporal logic*. It introduces mainly three extensions:

1. the introduction of *actions*. Actions are associated with changes that may alter states in histories. Using actions in **AlbertII** makes possible to overcome the well-known *frame* problem [BMR92], a typical problem resulting from the use of a declarative specification language;

2. the introduction of *agents*. An agent, which can be seen as a specialization of the object concept [Sho90], is characterized through (i) its *internal state* recording the knowledge of the agent with respect to its environment, (ii) its *perception* of what is happening in its environment and (iii) its *responsibility* with respect to actions having some effects on its state or on states of other agents.

3. the identification of *typical patterns of constraints*. The use of a logical and formal language may be compared to the use of an assembly programming language where a set of basic constructs are available but there is a lack of support for the analyst in writing complex and consistent statements. To overcome this problem, a number of typical patterns of formulas have been predefined in **AlbertII** (in the past, a similar need for typical patterns was identified in RML [GBM86]).

Using the language involves two activities: (i) writing *declarations* introducing the vocabulary of the considered application and (ii) expressing *constraints*, i.e. logical statements which identify possible behaviours of the composite system and exclude unwanted ones. A graphical syntax (with a textual counterpart) is used to introduce *declarations* and to express some static properties. The expression of the other constraints is purely textual.

2.1 Declarations

Agents are grouped into societies. Societies themselves can be grouped together to form larger societies. In fact, a specification consists in a hierarchy of agents (a tree-like structure) based on two combination mechanisms: *aggregation* and *classification*. Fig.1

shows the structure of the society corresponding to the Medical System (*MS*): the *MS* society is an aggregate of one *MIS* (Medical Information System) and several *Physician*s.

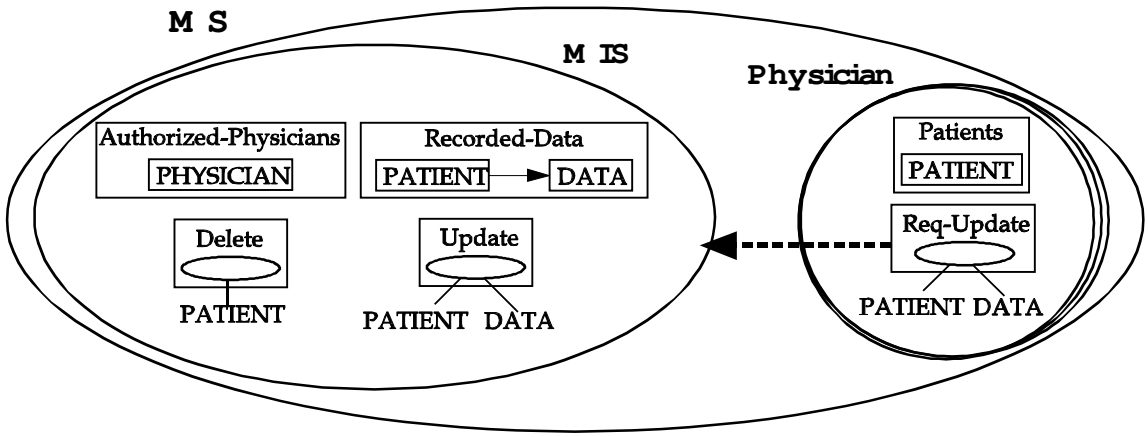

Figure 1 - Declaration of the *MS* Society

The declaration part of an agent consists in the description of its *state structure* and the list of *actions* its history can be made of.
Agents are considered as *specialized* objects; therefore, our modelling of a state structure is largely inspired by recent results in O-O conceptual modelling (see, e.g. OBLOG [SSE89] and O* [Bru91]).
Agents include a key mechanism that allows the identification of the different instances. A type is automatically associated to each class of agent. This corresponds to the type of agents identifiers within that class. For instance, each *Physician* agent has an identifier of type *PHYSICIAN*.
The state is defined by its components which can be *individuals* or *populations*. Usually populations are *sets* of individuals but they can also be structured in *sequences* or *tables*. Elements of components are typed using (i) predefined elementary data types (like, *STRING, BOOLEAN*, *INTEGER*...), (ii) user-defined elementary types (for which no structure is given), (iii) user-defined constructed types built using predefined type constructors like, e.g. cartesian product, sequence, union, enumerated type, etc., or (iv) types corresponding to agent identifiers. In Fig. 1, one can see that the *MIS* agent has two state components: *Recorded-Data* is a table containing, for each patient, some data (*PATIENT* is a user-defined type); *Authorized-Physician*s is a set of physicians. Each *Physician* agent has one state component *Patient*s, viz. the set of controlled *Patient*s.
Actions which may happen in the life of an agent are also declared graphically. *Update, Delete, and Req-Update* are actions which may happen in the history of the *MIS* (the last one being under the responsibility of another agent, i.e. *Physician*). Actions can have arguments (this is the case for the *Delete* action which has one argument of type *PATIENT* indicating the *Patient* to be removed).
The diagram also includes graphical notations used to express static visibility relationships linking the agent to the outside (*importation* and *exportation* mechanisms). Boxes without arrow denote the information (state components or actions) which is not visible from the outside while boxes with arrow denote the information which is exported to the outside (for instance, the *Req-Update* action of *Physician*).

2.2 Constraints

Constraints are used for pruning the (usually) infinite set of possible lives (histories) of an agent resulting from the graphical declaration. Unlike usual design specification languages, the **AlbertII** semantics is not operational. A life must be extensively considered before it can be classified as possible or not, i.e. adding new states and changes at the end of a possible life does not necessarily result in a possible life. Fig.2 introduces the specification associated with the behaviour of the *MIS* agent and refers to the graphical declaration

introduced in Fig.1. Formal constraints are expressed in terms of the different *patterns* available in the language. Only the ones that are illustrated in the example (see Fig. 2) are explained here.

MIS

LOCAL CONSTRAINTS

STATE BEHAVIOUR

The data associated with a patient cannot remain the same for more than 10 years.

$$In\text{-}dom(Recorded\text{-}Data, p) \wedge d = Recorded\text{-}Data[p] \Rightarrow \neg \Box_{\geq 10y}\ In\text{-}dom(Recorded\text{-}Data, p) \wedge d = Recorded\text{-}Data[p]$$

EFFECTS OF ACTIONS

The *Update* action results in a modification of the data associated with a patient if this patient already exists in the *Recorded-Data* table.

Update(p, d) with In-dom(Recorded-Data, p): Recorded-Data[p] := d

The *Update* action results in the creation of a new entry in the *Recorded-Data* table when the patient is a new one (i.e. not yet recorded in the table).

Update(p, d) with ¬In-dom(Recorded-Data, p): Recorded-Data :=Add(Recorded-Data, p, d)

The *Delete* action results in the removal of the data associated with a patient in the *Recorded-Data* table.

Delete(p): Recorded-Data := Remove(Recorded-Data, p)

ACTION COMPOSITION

The *Updating* process is initiated by a request for update issued by a Physician followed by the actual update action performed by the *MIS*.

$$Updating \leftrightarrow {}^{*}Physician.Req\text{-}Update(p, d); Update(p, d)$$

CAPABILITY

The *Delete* action cannot be performed for a patient who does not exist in the *Recorded-Data* table.

$$F\ (\ Delete(p)\ /\ \neg In\text{-}dom(Recorded\text{-}Data, p)\ ) \qquad \textbf{(1)}$$

COOPERATION CONSTRAINTS

ACTION PERCEPTION

The *MIS* is a reliable agent which takes into account all the requests issued by physicians.

$$XK\ (\ \text{-}.Req\text{-}Update(p, d)\ /\ TRUE\ ) \qquad \textbf{(2)}$$

Figure 2 - Constraints of the *MIS* Agent

- It should be noted that there are *local constraints* which are related to the internal behaviour of the agent, as well as *cooperation constraints* which specify how the agent interacts with its environment (the latter offers a dynamic dimension to importation and exportation relationships expressed in the declaration part of the specification).

- Under the 'State Behaviour' heading are described two kinds of constraints which must hold in an admissible history of an agent: static constraints are constraints which are true in all states (usually referred as *invariants*); dynamic constraints are constraints on the evolution of the state. They are expressed using real-time temporal connectives.

- Under the heading 'Effects of Actions', the effects of the different actions happening are described. The effect of an action is expressed in terms of a property characterizing the state which follows the end of an action occurrence.

- Under the 'Action Composition' heading are described the different *processes* under the responsibility of the agent. Processes describe how the different actions affect each other. In particular, how an agent has to react to an action issued by another agent.

- Constraints under the heading 'Capability' describe the responsibility of the agent with respect to the occurrences of its own actions. The default rule is that all actions are permitted whatever the situation but specific constraints can be added to specify circumstances under which obligations and preventions are associated with actions occurrences.

- Finally, the constraints under the 'Action Perception' heading specify the perception that an agent may guarantee with respect to actions issued to it by other agents.

As it can be seen, the basic objective underlying the design of **AlbertII** has been to allow the analyst to write her/his specification in a natural way, i.e. without being constrained by the different kinds of constructs available in the language. For example, using a language based on the ECA (Event-Condition-Action) style would have restricted the expressiveness and naturalness of the language.
As an illustration of this property of the **AlbertII** language, one may consider the specific statement written under the 'State Behaviour' heading in Fig. 2. This statement says that 'data associated with the patient cannot remain the same for more than 10 years'. What are the consequences of this constraint when looking at the rest of the specification?
First, under the 'Effect of Actions' heading, we learn that modifications of data can only result from their *Update* and *Delete* actions. Second, from the 'Action Composition' heading, we learn that the action *Update* can only occur in response to the *Req-Update* action issued from the outside. So, we can conclude that in order to ensure the required state invariant described above, the *MIS* should trigger the *Delete* action on data which have not be modified during 10 years. Finally, we must point out that no textual constraint has been associated with *Physician*. In other words (see Fig. 1), this means that they can issue update requests (*Req-Update*) whenever they want.

3 A Security Requirements Framework

In the previous section, using the **AlbertII** language, we have expressed the functional requirements prescribing the functional behaviour that the *MS* must ensure. The system that we have described was not intended to be a secure system. Let us now consider its 'securisation'.
To fulfill this objective, we have not only to modify the initial functional specification of the system but also to achieve a good understanding of the rationale behind these modifications. In other words, through the **AlbertII** specification, if we can express the 'what's' of the system, we have also to trace the 'why's' behind it. Some information related to these 'why's' is related to the goals according to which the corresponding functional behaviour of the system has to be designed. In our particular example, these goals correspond to the security properties expected from the intended system.
In Fig. 3, we pictorially represent a *goal-tree* that we have established when handling the *MS* case study. This representation is based on the model proposed in [MCN92], i.e. it

consists of a set of *goals* and *links*. For the purpose of this paper, we only define one type of links: *decomposition link*. This type of links is used to model the interrelation between subgoals of their parent goal, i.e. multiple links can contribute to a node either on a joint effect (AND) or on disjointed effects (OR). As described above, the goal-tree can be considered as an extension of the conventional AND/OR goal-tree used for problem solving in [Nil71].
Goals are represented as sorts in a predicate form. For example, the goal, 'The MS should be secure', can be represented as *Secure[MS]*. In order to refine this goal into one or more offsprings, we may apply some generic design knowledge to the domain. This knowledge is called *method*. For instance, a generic decomposition for the *MS* can be expressed as follows by applying a decomposition method from the standpoint of information system security[1]:

Secure[MS] $\rightarrow$
(PhySec[MS],CompSec[MS],PersSec[MS],CommSec[MS],DataSec[MS]) **(3)**

i.e. the goal for securing the *MS* is composed by the goals: physical, personnel, computer, communication and data security measures for the *MS* [Lan93].

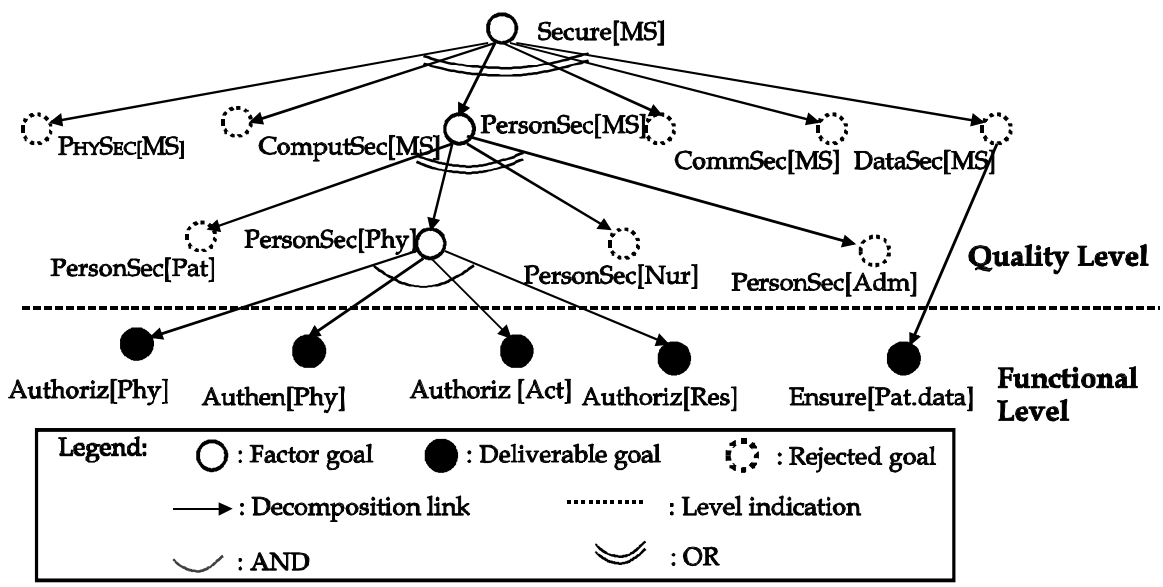

Figure 3 - Security Requirements Goal-Tree of the *MS*

From Fig.3, it can be read that the goal-tree is organized in terms of two abstraction levels, viz. the *quality level* (the upper level of the tree) and the *functional level* (the lower level of the tree). These two levels reflect the progressive goals refinement that is performed by the analyst during the requirements engineering process.
At the quality level, the goals are referred as *soft-goals* [Yu95] related to the usually so-called *quality* (or *non-functional*) requirements. In general cases, they often cannot be well defined and formulated. This requires that the analysts, starting from the abstract goals, progressively identify more concrete goals belonging to the functional level, i.e. corresponding to goals that should be precisely defined in terms of quantifiable conditions on the system's world. Regarding to their descriptions, quality-level goals will be described in an informal way (using the natural language) while we propose a formal way of defining the functional-level goals (by proposing some extensions to the **`AlbertII`** language).
In other words, this means that, as the goal analysis process goes forward, goals can be understood and decomposed by their finer decomposition goals which, in turn, can be

[1] Information system security is as the broader view, incorporating system analysis, and design method, manual information, managerial issues, and both social and ethical problems [Bas94].

- either further refined in terms of finer functional goals (this issue is not illustrated in Fig. 3)
- or operationalized [DVF93] in terms of constraints (functional requirements) enforced by agents available in the composite system. Such goals, called *contractual goals*, are represented by black circle in Fig. 3.

In Fig. 3, among the identified security goals - see also (3) -, we choose the goal *PersonSec[MS]* to be further developed. This means that we choose to rely on personnel security measures rather than on others (which are marked as the rejected goals represented by the dashed circles). In a *MS*, it typically involves *Physician*s (Phy), *Patient*s (Pat), Nurses (Nur), and Administrators (Adm). So, the goal *PersonSec[MS]* is naturally instantiated by its offspring shown in the figure. Among the new generated goals, we still need to make decisions to choose them. Let us assume that we choose the goal *PersonSec[Phy]*, i.e. personnel security of the physicians of the *MS*. In order to achieve this goal, four deliverable goals are identified:

Authoriz[Phy]: a physician who accesses to the system should be the one authorised;
Authen[Phy]: the identity of a physician must be authenticated;
Authoriz[Act]: a physician should perform the actions that are allowed;
Authoriz[Res]: a physician should use the system's resources that are allowed.

From the set of so-called functional goals, it is expected that they can be precisely (formally) defined as it will be shown in the next section.

4 Operationalizing Functional Goals

As it was mentioned in Sect. 2, the **AlbertII** language has been designed for capturing functional requirements inherent to composite systems. As such, it does not offer facilities for expressing overall goals over these systems. As a specific contribution in this paper, we propose some extensions to the **AlbertII** language towards this end. For doing so, we suggest to see how the operationalization process described above can be fully supported by an adequate formal framework.
At the end of the previous section, we have isolated a number of terminal functional goals. Among these, we propose to focus on *Authoriz[Act].* The goal can be rephrased as follows:

> The MS should guarantee that only the authorized physicians can perform the action for changing the patients' medical data.

This goal can be expressed more rigorously on the top of the existing specification in the following manner:

$$Phys.Req\text{-}Update(p, d) \Rightarrow Mis.In\text{-}dom(Authorized\text{-}Physicians, Phys)$$

This statement can be reformulated as 'when a physician issues a request he/she is automatically identified by the MIS as one belonging to the authorized physicians'. It should be noted that for the purpose of expressing such goals, we had to enrich the **AlbertII** language with additional features which make possible to describe the responsibilities and knowledge of the different agents with respect to the different actions and states happening in the world.
Having expressed the goal, we are now concerned with the application of the operationalization process sketched above. In other words, we have to consider the introduction of modifications to be brought to the initial specification (presented in Fig. 1 and 2) and that will result in a new version in which the terminal goals are met (i.e. can be proved). For instance, one may imagine that the goal *Authoriz[Act]* may be fulfilled under the responsibility of either the *Physician* agent, or the *MIS* agent, or both.

A first choice could have been the assignement of the responsibility for this goal to the *MIS*. As a result, the following modification of the *MIS* specification presented on Fig.2 is introduced (see Fig. 4).

MIS

...

COOPERATION CONSTRAINTS

ACTION PERCEPTION

The *MIS* takes update requests issued by a physician into account if and on if it is able to identify the identity of the physician and to check that it is an authorized one.

XK (phys.Req-Update(p, d) / In-dom(Authorized-Physicians, phys))

Figure 4 - Modification of the Constraints of the *MIS* Agent (First Possibility)

However, this alternative is not a good one because the *MIS* cannot assume this responsibility for being able to perceive the identity of the request-senders. Therefore, we need to consider another alternative responsibility assignement strategy. This leads us to consider that the responsibility should be assumed by both agents (i.e. *Physician* and *MIS*), because in a medical system including multiple *Physician*s, as in an open system, the security confidence should be established on both sides [Gri90]. This means that any security measure (or constraints) implemented for one side may not be satisfactory for the other. For instance, suppose that a security measure (e.g. the strict verification of the Personal Information (PIN) provided by a physician) has set up at the *MIS* side, the *MIS* can still be cheated by a third person if a physician disclosed his PIN to that person.
Adopting the strategy described above results in the assignement of new responsibilities to both *Physician* and *MIS* agents. These responsibilities are related to the modification brought to the *Req-Update* action for which we have decided to add an extra argument (of type *PHYSICIAN*) corresponding to an additional information to be provided about the identity of the physician who requestes the update. In terms of **AlbertII**, the graphical declaration associated with *Physician* (on Fig. 1) has to be modified as shown on Fig. 5.

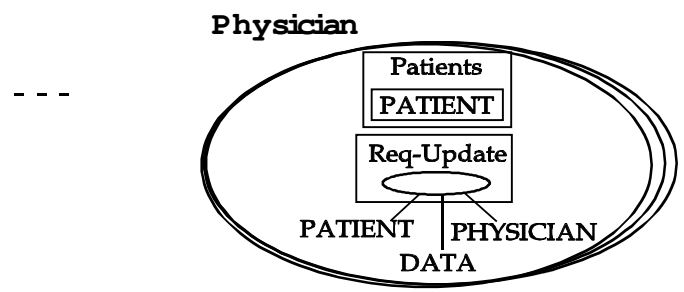

Figure 5 - Modification of the Declaration of the *Physician* Agent (Second Possibility)

In order to fulfill the goal described above, the following modifications (see Fig. 6) have to be brought to the textual description of the different agents.

Physician

LOCAL CONSTRAINTS

CAPABILITY

This deontological constraint says that a physician cannot refer to the identity of another physician when he/she issues a request. Note that *SELF* is a constant refering to the proper identifier of the Physician agent.

F (Req-Update(p, d, phys) / phys ≠ SELF)

MIS

…

COOPERATION CONSTRAINTS

ACTION PERCEPTION

Update requests are only taken into account by the *MIS* when the referred physician is one of those belonging to *Authorized-Physicians.*

XK (-.Req-Update(p, d, phys) / In-dom(Authorized-Physicians, phys))

Figure 6 - Modification of the Constraints of the *MIS* and *Physician* Agents (Second Possibility)

Due to the lack of place, we do not provide the formal proof of the presented goal in terms of the assigned agent responsibilities but the proof is possible thanks to the formal semantics on which **`AlbertII`** is grounded[DuB95].

5 Conclusion

In this paper, we have reported on the relationship existing between a framework that can be used for organizing non functional requirements and a formal specification language that can be used for modelling functional requirements. Moreover, we have tried to support the benefit of using a formal framework to ensure the coupling between the two models, through the specification of goals and of responsibilities attached to agents.
All along this paper, we have suggested how this goal-oriented framework could be applied for dealing with security aspects of information systems. For the explanatory purpose, we have only considered a small and trivial case study. A more complex one, concerning security aspects of a message handling system, is also considered by one of the authors [WR94].

References

[Bas94] Richard Baskerville, "Information System Security Design Methods: Implications for Information Systems Development", *ACM Computing Surveys*, Vol. 25, Nr. 4, December 1994

[BCDS93] A.J.C. Blyth, J. Chudge, J.E. Dobson and M.R. Srens, "A Framework for Modelling Evolving Requirements" in *Proc. of the 17th Annual Intl. Computer Software Application Conference - COMPSAC'93*, 1993

[BMR92] A. Borgida, J. Mylopoulos, and R. Reiter, *... and nothing else changes: The Frame Problem in Procedure Specification*, Technical Report DCS-TR-281, Dept. of Computer Science, Rutgers University, 1992

[BS93] Klemens Böhm and Amilcar Sernadas, *Real-Time Object Specification Logic*, Technical Report 7/93, Departamento de Matematica, Instituto Superior Tecnico, Lisbon (Portugal), March 1993

[Bru91] J. Brunet, "*Modelling the World with Semantics Objects*" in *Proc. of the Working Conference on the Object-Oriented Approach in Information Systems*, Québec (Canada) 1991

[Chu93] Lawrence Chung, *Representing and Using Non-Functional Requirements: a Process-Oriented Approach*, PhD Thesis, Department of Computer Science University of Toronto, Toronto, Ontario, Canada, June 1993

[DDDM94] Eric Dubois, Philippe Du Bois, Frédéric Dubru, and Michaèl Petit, "Agent-Oriented Requirements Engineering: A Case Study Using the Albert Language" in A. Verbraeck, H. G. Sol, and P.W.G. Bots (eds.) *Proc. of the Fourth International Working Conference on Dynamic Modelling and Information Systems - DYNMOD-IV*, Delft University Press, Noordwijkerhoud (The Netherlands), September 28-30, 1994

[DDZ95] Eric Dubois, Philippe Du Bois, and Jean-Marc Zeippen, "A Formal Requirements Engineering Method for Real-Time Concurrent, and Distributed Systems" in *Proc. of the Real-Time Systems Conference - RTS '95*, Paris (France), January 11-13, 1995

[DHR91] Eric Dubois, Jacques Hagelstein, and André Rifaut, "A Formal Language for the Requirements Engineering of Computer Systems" in André Thayse (ed.) *From Natural Language Processing to Logic Expert Systems*, Chapter 6, Wiley, 1991

[DP94] Eric Dubois and Michaèl Petit, "Using a Formal Declarative Language for Specifying Requirements Modelled in CIMOSA" in François Vernadat (ed.), *Proc. of the European Workshop on Integrated Manufacturing Systems Engineering - IMSE'94*, pp. 233-241, INRIA Rhône-Alpes, Grenoble (France), 1994

[Dub89] Eric Dubois, "A Logic of Action for Supporting Goal-Oriented Elaborations of Requirements" in *Proc. of the 5th International Workshop on Software Specification and Design - IWSSD'89*, pp. 160-168, IEEE CS Press, Pittsburgh PA, May 19-20, 1989

[Dub91] Eric Dubois, "Use of Deontic Logic in the Requirements Engineering of Composite Systems" in J.J. Meyer and R.T. Wieringa (eds.), *Proc. of the First International Workshop on Deontic Logic in Computer Science - DEON'91*, Amsterdam (The Netherlands), December 11-13, 1991

[DuB95] Philippe Du Bois, *The `AlbertII` Language - On the Design and the Use of a Formal Specification Language for Requirements Analysis*, PhD Thesis, Institute of Computer Science, The University of Namur, Forthcoming

[DVF93] A. Dardenne, A. van Lamsweerde, S. Fickas, "Goal-Directed Requirements Acquisition", *Science of Computer Programming*, Vol.20, 1993

[Fea89] Martin S. Feather, "Constructing Specification by Combining Parallel Elaboration", *IEEE Transactions on Software Engineering*, Vol. 15, Nr. 2, pp. 198-208, 1989

[FW93] Remco B. Feenstra and Roel J. Wieringa, *LCM 3.0: A language for Describing Conceptual Models - Syntax Definition*, Technical report IR-344, Faculteit der Wiskunde en Informatica, Vrije Universiteit Amsterdam, Amsterdam (The Netherlands), December 1993

[GBM86] Sol J. Greenspan, Alexander Borgida, and John Mylopoulos, "A Requirements Modeling Language", *Information Systems*, Vol. 11, Nr. 1, pp. 9-23, 1986

[Gri90] Ruediger Grimm "Security for Open Communications: The DFN Mail Security Project Report", *Computer Network and ISDN System*, Vol. 19, 1990

[JBRSV92] M. Jarke, J. Bubenko, C. Rolland, A. Stucliff, and Y. Vassilac, *Theories Underlying Requirements Engineering: an Overview of NATURE at Genesis*, NATURE Report Series, Informatik V, RWTH-Aachen, Ahornstr, 55, 5510, Germany 1992

[Jun93] Ralf Junglaus, *Modeling of Dynamic Object Systems - A Logic-based Approach*, PhD Thesis, Technische Universität Braunschweig, Braunschweig (Germany), 1993

[Lan93] C.E. Landwehr, "How far Can You Trust a Computer?" in *Proc. of 12th International Conference on Computer Safety, Reliability and Security - SAFECOMP'93*, Invited Paper, Oct. 1993

[MCN92] John Mylopoulos, Lawrence Chung, and Brian Nixon, "Representing and Using Non-functional Requirements: A Process-Oriented Approach", *IEEE Transaction on Software Engineering*, Vol.18, Nr. 6, June 1992

[Nil71] Nils Nilsson, *Problem-Solving Methods in Artificial Intelligence*, New York, McGraw-Hill, 1971

[PFAB86] Colin Potts, Anthony Finkelstein, Michael Aslett, and Judy Booth, *Structured Common Sense: A Requirements Elicitation and Formalization Method for Modal Action Logic*, FOREST Deliverable report 2, GEC Research Laboratories, Great Baddow, Chelmsford (UK), November 1986

[PTA94] Colin Potts, K. Takahashi, and A.I. Anton, "Inquiry-Based Requirements Analysis", *IEEE Software*, March 1994, pp. 21-32

[Sho90] Y. Shoham, *Agent-Oriented Programming*, Technical Report STAN-CS-90-1335, Robotics Laboratory, Computer Science Dept, Stanford University, Stand CA, 1990

[SSE89] A Sernadas, C. Sernadas, and H-D. Ehrich, "Abstract Object types: A Temporal Perspective" in B. Banieqbal, H. Barringer, and A Pnueli, (eds.), *Proc. of the Colloquium on Temporal Logic and Specification*, LNCS 398, Springer-Verlag, 1989

[vLDM95] Axel van Lamsweerde, Robert Darimont, and Philippe Massonet, "Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons learnt" in *Proc. of the IEEE International symposium on Requirements Engineering - RE'95*, IEEE Computer Society Press, pp. 194-203, York (United Kingdom), March 27-29, 1995

[WR94] Suchun Wu and Jean Ramaekers, "Integrating Security into Message Handling Systems" in *Proc. of ICCT'94*, Shangai (China), June 1994

[Yu95] Eric Yu, *Modelling Strategic Relationships Process Reengineering*, PhD Thesis, Dept. of Computer Science, University of Toronto, Ontario Canada, 1995