Integrating Semi-formal and Formal Requirements*

Roel Wieringa[1], Eric Dubois[2], Sander Huyts[3]

[1] Faculty of Mathematics and Computer Science, *Vrije Universiteit*, De Boelelaan 1081a, 1081HV Amsterdam, the Netherlands. Email: roelw@cs.vu.nl.
[2] Facultés Universitaires Notre-Dame de la Paix, Institut d' Informatique, Rue Grandgagnage 21, B-5000 Namur, Belgium. Email: edu@info.fundp.ac.be.
[3] Work performed at Belgacom Research, Development and Engineering Bd. E.Jacqmain 177 - 12T83 B1030 Brussels, Belgium. Now at Cambridge technology Partners, Apollolaan 15, 1077 AB Amsterdam, the Netherlands. Email: sander.huyts@ctp.com.

Abstract. In this paper, we report on the integration of informal, semi-formal and formal requirements specification techniques. We present a framework for requirements specification called TRADE, within which several well-known semiformal specification techniques are placed. TRADE is based on an analysis of structured and object-oriented requirements specification methods. In this paper, we combine TRADE with the logic-based specification language Albert II and show that this leads to a coherent formal and semiformal requirements specification. We illustrate our approach with examples taken from a large distributed telecommunication application case study performed in the context of the Esprit project 2RARE.

1 Introduction

The field of requirements specification techniques has reached a state where it is not desirable to introduce yet another technique without indicating how this technique can be combined with other techniques in a coherent specification of software requirements. Each software product can be described from many different perspectives and no technique can be used for the specification of software requirements from all these perspectives. Consequently, the utility of a requirements specification technique depends for a considerable part on the way in which its relationship to other specification techniques is defined, used for the specification of requirements from other perspectives.

In particular, it is important to define ways to integrate semiformal and formal specification techniques. By **semiformal techniques** we mean diagram techniques and tabular techniques that present information in structured form. By **formal techniques** we mean mathematics, logic or algebra, in which the

* Supported by Esprit Project 2RARE, contractnr. 20424.

syntax, semantics and manipulation rules for the specification language are explicitly defined. By **informal techniques** we mean unrestricted natural language. The majority of specifications in practice will use a mix of these three techniques. For example, an informal requirements specification may be illustrated by a number of semiformal diagrams and formal techniques may be used for critical or complex parts of the requirements, where the consequence of errors in the requirements will be severe.

In the Esprit project 2RARE (Two Real Application of Requirements Engineering), requirements for three complex distributed critical applications are specified using a variety of specification techniques [1], including the formal specification language Albert II [3], the object-oriented specification language Oblog [18] and some semiformal, diagram-based techniques. In order to combine these specification techniques, we need a general framework that allows us to state which part of required system properties we specify by which technique, and what the relationships between the different parts of the specification are. In this paper, we present such a framework and show how it can be used to combine Albert II with a number of well-known semiformal requirements specification techniques. This result is not only relevant for Albert II but for any formal specification technique that should be combined with semiformal techniques. The framework, called TRADE (Toolkit for Requirements And Design Engineering), is based upon an analysis of structured and object-oriented requirements specification methods [22, 21, 23]. We use one of the systems specified at Belgacom, a VoD system, as a running example [24]. A VoD system consists of a set top box located in the home of a customer, connected to the television set of the customer. The set top box offers roughly the functionality of a video player. It is connected through a telephone network to a video server located at a service provider. Through the set top box, the consumer can request videos to be shown on his or her television.

We start in section 2 with the methodological framework of TRADE. In section 3, we present some of the semiformal tools in TRADE, using the Belgacom VoD application as an example. In section 4 we present Albert II specifications of parts of this system and show how these are connected to the semiformal specification.

Section 5 winds up the paper with a discussion of general results gathered in the 2RARE project and and some topics for further work.

2 The Methodological Framework of TRADE

In this section, we briefly sketch the major principles of TRADE. Because all elements of TRADE are borrowed (with minor adaptations) from the literature, sources and related approaches will be acknowledged as we go.

2.1 System Environments

The central methodological principle in TRADE is the **principle of environment modeling**: in order to model the requirements on a product, we must

model the desired situation in the environment of the product. The reason for this is that any product exists in order to provide a service to its environment. This service may be to make certain behavior in the environment possible or to enforce certain behavior in the environment. There are many environments of the product that may be relevant during a particular development process: examples are the physical, social, financial and normative environments. Four environments are singled out in TRADE as being important in every software product development.

- The **problem environment** is the place and time of the world in which the problem exists to which the product must provide a solution.
- The **solution environment** is the place and time of the world in which the product exists and contributes to a solution of the identified problems. Synonyms used in TRADE are *operating environment* and *usage environment*. The elements of the solution environment needed to understand the behavior of the product are called **external entities**.
- The **implementation platform** is the technical infrastructure on top of which the product is to be built. It includes the operating system, database management systems, software libraries and legacy components on top of which the desired software product is to be built. It does not include components that have to be built specially to satisfy required product properties.
- The **subject domain** of a software product is the part of the world that is represented in the software product. Synonyms are *subject environment* and *Universe of Discourse* (UoD). The elements of the subject domain represented by the product are called **subjects**. In TRADE we use the hypothesis of **subject-orientation**, which says that the state of a software product always contains a representation of the state of the subject domain.

It is important to distinguish a specification of properties of the environment of the product and the *product specification* itself. The former expresses how the environment will behave in presence of the product [14] while the latter focusses on the description of the behavior of the product. Similar distinctions are made in the Esprit project Nature [16]. The concept of external entity is well-known from structured analysis [9]. The importance of subject-orientation for modeling data-intensive system requirements was pointed out by Chen [6]. Jackson [13] showed that subject-orientation is useful for a significant number of control-intensive systems as well. We should note that the operating environment and subject domain a software system may overlap.

2.2 System Properties

A second methodological principle in TRADE is the logical independence of the decomposition of a system into subsystems and the refinement of system functions into subfunctions. This principle originates from general systems theory and is used elsewhere as the basis for a framework for system development methods [22]. Harel and Pnueli [12] represent this by what they call the **magic**

square, which sets off a behavior refinement dimension against a system decomposition dimension. In general, a development process advances by reducing uncertainty about the required decomposition and required behavior of the product simultaneously. In this process, we may meander in any way through the square and we may use other decomposition principles besides the principle of functional decomposition.

At any point in the development process, we have reached a level of certainty about two sets of desired system properties: requirements on external system behavior (the horizontal dimension of the magic square) and constraints on system decomposition (the vertical dimension). Properties of external behavior can be classified as either properties of observable states or properties of observable state transitions. This leads us to a classification of three kinds of system properties: properties of *states*, properties *state evolution* and properties of *decomposition.* Each subsystem in turn may be characterized in the same way by specifying its states, state changes and internal decomposition.

We now apply this classification of system properties to the product as well as to its environments.

2.3 Environment Properties

Subject domain properties. The subject domain can be modeled by representing its decomposition into subjects and their connections. If there are many subjects, as in the case of data-intensive systems, we usually do this by identifying subject *types* and connection *types* and we use one of the many variants of the entity-relationship notation to represent this. If there are a few subjects, as in many control-intensive systems, we can do this at the instance level by drawing a graph whose nodes represent subjects and whose edges represent connections.

Operating environment specification. The operating environment can be modeled by a graph in which one node represents the product and in which the other nodes represent external entities. Figure 1 shows such a graph, discussed in more detail later. The edges represent connections through which the nodes communicate. We then get an **extended context diagram** as defined by Jackson [14], which is a generalization of the convention used in structured analysis.

Implementation platform properties In a software requirements specification, the implemtation platform plays a role in as far as this imposes *constraints* on the physical decomposition of the product. Constraints may also follow from the properties of the external entities. In a distributed system, an important constraint is that the system is implemented on a distributed collection of physical nodes. This network is embedded in the decomposition of the operating environment into external entities. For example, the VoD system must be implemented in a network consisting of a collection of set top boxes, the telephone network, a gateway and a video server.

2.4 Product Specification

The models of the subject environment and operating environment have a simple relationship with a model of the desired properties of the product along each of the two dimensions of the magic square. Briefly, the model of the operating environment can be used to specify desired system functions, and the model of the subject domain can be used to represent a conceptual system decomposition.

- Because each interaction between an external entity and the product is a desired product function, the context diagram of the operating environment gives us a model of desired system behavior. This is usually given by a list of desired **product functions**, where each function is a useful piece of external product behavior. Product transactions are atomic product functions. The list of desired product functions, possibly documented by a function refinement tree and data flow diagrams to specify the effect of a function, is usually called the **function view** of the system. A specification of temporal properties of these functions is usually called the **behavior view** of the system.
- According to the principle of subject-orientation, the state of a software product contains a representation of the subject domain. This means that the software state between different product transactions contains a set of **surrogates** that represents the set of currently existing subjects. Following JSD [13], we use the subject domain model as a guideline to find a conceptual decomposition of the software product into surrogates, which we call the **conceptual model** of the product. The conceptual model may indicate other relevant components of the product, that have no counterpart in the subject domain, such as function objects in a JSD model of the product, or control– and interface objects in an Objectory/OOSE model of the product [15, 17]. In data-intensive systems, a conceptual model of the surrogates in the software product is often called the **data view** of the product, because it defines the meaning of the surrogates (data) in terms of the subject domain.

There is a relationship between the functions of the product and its conceptual decomposition: Every function must be realized by the collaboration of its conceptual components. This can be represented by the well-known **traceability tables** of systems engineering, which set off the product functions against product components. An entry in this table shows the role that the component plays in the realization of this function [8, page 192], [20]. Such a table shows in basic tabular format what the collaboration diagrams of object-oriented analysis show in more detail in graph format.[4]

Note that the conceptual model mentioned above is independent of the **physical decomposition** of the system into processors and processor connections

[4] The term collaboration diagram comes from Wirfs-Brock [25] . Booch calls them object diagrams [4, page 208] and in Fusion they are called object interaction graphs [7, page 63]. In the UML they are called collaboration diagrams [5].

or pre-existing software components in the implementation environment. The components of the conceptual decomposition must be allocated to the components of the physical decomposition. This can be represented by a second kind of traceability table, in which conceptual components are linked to physical components.

In the next two sections, we illustrate the use of semiformal techniques to specify the operating environment and subject domain and show how the product requirements can be specified formally using Albert in a way that yields a coherent formal-semiformal specification.

3 Semiformal Specification Techniques

In this section, we present some of the semiformal specification techniques in TRADE, using the VoD system as an example. All names introduced in TRADE diagrams can be documented informally in a specification dictionary. This is not illustrated here. The reader is warned that the TRADE techniques contain no surprises. An effort has been made to stay in the middle of the road. Simplicity and familiarity have been preferred above complexity and novelty.

3.1 The Operating Environment

The operating environment of the VoD system consists of end-users and several service providers. It can be represented in TRADE by a **communication diagram**, which is an undirected graph whose nodes represent components and whose edges represent communication connections between components. A communication diagram of an operating environment is also called a context diagram (fig. 1). The context diagram can be used to keep track of the transactions of the product with its environment. For example, all leaves of a fully grown function refinement tree of the product (treated next) should correspond to interactions between the product and external entities in the context diagram.

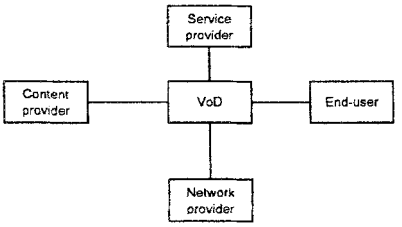

Fig. 1. The operating environment of the VoD system.

3.2 Desired Product Functions

The context diagram can be used to find a list of required product functions. For the VoD system, it is convenient to group the functions according to the external entity for whom these functions are performed. For presentation purposes, it is convenient to organize the functions into a **function refinement tree.** (Due to lack of space, we do not give an example here.) Such a tree says nothing about any decomposition of the VoD system. It is merely a convenient way of organizing the functions that the system must have for various external entities. It corresponds to the horizontal dimension of the magic square, not to the vertical dimension.

3.3 The Subject Environment

The subject environment of the VoD system is the part of the world about which data is stored and manipulated. Figure 2 shows a **class-relationship diagram** (CRD) of some of the data relevant for the VoD system. It stays close to well-known conventions but contains some adaptations motivated by ease of use as well as formalizability. A rectangle represents an object class or a link class. A link class is also called a relationship. A relationship rectangle is connected to its components by dashed arrows. Each rectangle in a CRD must contain a class name, followed optionally by a list of attributes of instances, followed optionally by a list of transactions that may change the state of an instance. Note that objects and links may be used to represent *subjects* in the subject domain or their *surrogates* in the system.

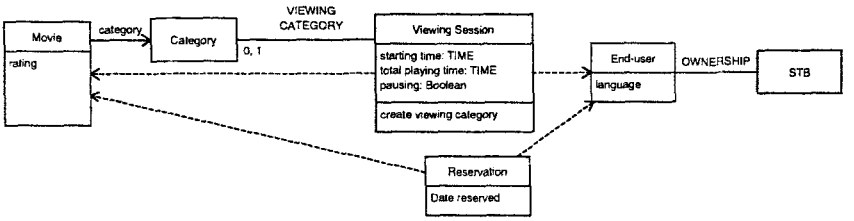

Fig. 2. CRD of part of the subject domain of the VoD system. Create Viewing Category is an action that creates a link between the viewing session and a movie category.

Any relationship can be represented by a box, but binary relationships can alternatively be represented by a line or arrow. A many-many relationship can be represented by an undirected line and a many-one relationship can be represented by an arrow. In the line and arrow representations, local attributes or transactions of relationship instances cannot be shown (but they may be present).

A relationship can itself be related to other elements. An example is the VIEWING_CATEGORY relationship, that is created when a user selects a cate-

gory during a viewing session. Relationships may be annotated, as usual, with cardinality properties and with the role names of the components.

The difference between a CRD and a communication diagram is that a communication diagram represents possible communications between class instances, whereas a CRD represents the way class instances are identified. An object class box in a CRD says that the class instances are identified independently. A relationship box (or line or arrow) says that the relationship instances are identified by tuples of component identifiers.

3.4 System Decomposition and Implementation Environment

The VoD system implementation will be distributed over different processors, network and devices. This physical decomposition is given in advance of development. It is an important constraint that can be represented by an extension of the context diagram that shows the major hardware components on which the VoD software must run (fig. 3). Just like the context diagram, this is a communication diagram. The connections represent possible communications.

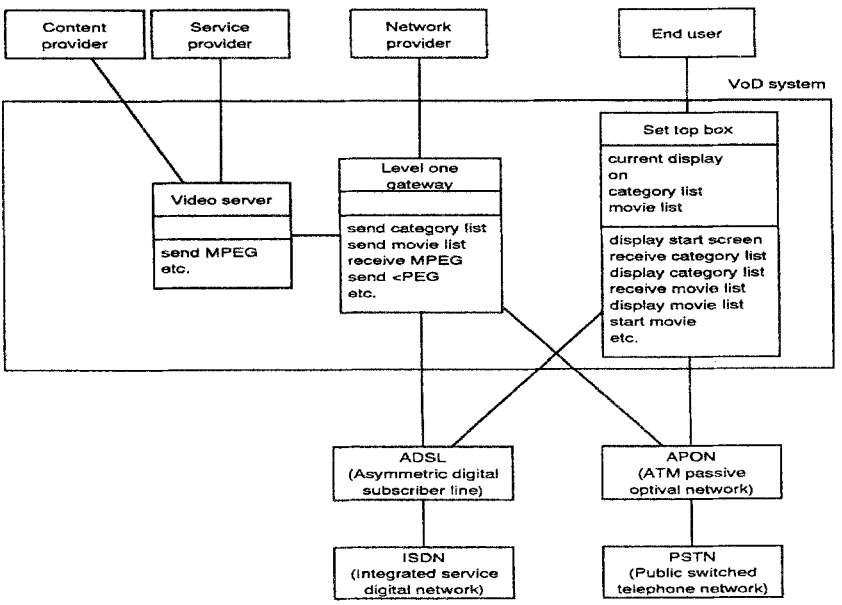

Fig. 3. Simplified physical decomposition of the VoD system in its operating environment and implementation environment.

The ASDL and APON external entities are part of the implementation environment. They exist in advance of development and will continue to exist during and after development. It is a constraint that the set top box and level one gateway must communicate through these networks. They are connected to two

additional external entities, ISDN and PSTN. These are shown because the VoD system implementation should *not* cause any interference with these entities. This is a requirement of non-feature interaction.

The communication diagram of fig. 3 can be used to identify the actor classes of an Albert specification of VoD requirements. Later, we give an example specification of the set-top-box (STB) component as an Albert agent.

3.5 Allocation and Flowdown

Both the required functions of the VoD system and the conceptual objects manipulated by the system must be allocated to physical components of the system. Figure 4 shows part of a table that allocates and flows down the functions of the VoD system to functions of the physical components of the system. This flowdown can be further illustrated by showing message sequence charts of the communication between the physical components of the system. (Due to lack of space, this is not shown here.) Note that the ADSL and APON networks are not shown in the flowdown because these components act as channels through which messages are passed undisturbed.

The data specified by the CRD is stored at several places in the system as shown in the allocation table of fig. 5. This is explained by the fact that when a set top box is switched on, data about movies and movie categories is downloaded from the video server to the set top box. Billing data is maintained by the level one gateway.

4 Albert Specification

Albert II (called "Albert" for short) is a formal specification language based upon real-time temporal logic [3, 2]. It has been validated in the specification of non-trivial systems like Computer Integrated Manufacturing [10] and telecommunications systems [11]. Albert organizes its specification around the agents identified in the operating environment, where an **agent** is an active entity that can perform or suffer *actions* that change or maintain the *state* of knowledge about the external world and/or the states of other agents. Actions are performed by agents to discharge contractual obligations expressed in terms of *local constraints*, applicable to the agents itself, and *cooperation constraints*, that apply to the interaction between agents. A specification in Albert is made up of (i) a graphical component in terms of which is *declared* the vocabulary of the application to be considered and of (ii) a textual component in terms of which the specification of the admissible behaviors of agents is *constrained* through logical formulas. Herafter, we illustrate the application of the language on the specification of the *Select-category* service identified in the hierarchy presented in fig. 4.

VOD system	boot	select service	select category	select movie	play	pause
Consumer	Power on STB	Select	Select	Select	Play	Pause
VOD Set Top Box	boot display welcome load service menu	select service load application	select category	select movie request movie display MPEG stream display VCR logo monitor VCR com. play	display MPEG stream display VCR logo monitor VCR com. play	freeze current screen display VCR logo monitor VCR com. pause
VOD Level One gateway	download service menu	activate control ch. activate MPEG ch. transport application		transport request accounting transport MPEG	transport request transport MPEG	transport request
VOD video Server		access control allocate resources download application		control & coord. movie control load movie play normal speed	control & coord. processVCR request play normal speed end detection signal MPEG	control & coord. processVCR request suspend playback

Fig. 4. Allocation and flowdown of some of the functions of the VoD system to functions of its components.

	CATEGORY	MOVIE	VIEWING	RESERVATION	USER	STB	ADDRESS	BILL
VIDEO SERVER	x	x		x	x			
LEVEL 1 GATEWAY			x		x	x	x	x
SET TOP BOX	x	x						

Fig. 5. Allocation of conceptual components of the VoD system to its physical components.

4.1 Graphical Declarations

Figure 6 contains part of the graphical declaration of the VoD system according to the Albert conventions. Each agent is represented by an oval and multiplicity is indicated by shadowing an oval. Note that this declaration is derived from the context diagram of fig. 1. Figure 6 also declares the internal structure of the VoD agent. It declares the state structure and the actions that may happen during the lifetime of an agent and which may change the state of the agent. State components are represented by rectangles and actions are represented by ovals. State components are typed and actions can have typed arguments. Types may vary from simple data types to complex data types (recursively built using the usual data type constructors). The information provided in fig. 6 is informally rephrased in the first part (*Declarations*) of fig. 7, playing the role of data dictionary. However, from graphical conventions used in fig. 6, we also know that *Movies* and *Display* are tables respectively indexed on MOVIE and ENDUSER (the type associated the identity of the End-User agent) while *List-cat* corresponds to a set of CATEGORY and is derived (see below) from the *Movies* component.

In addition, the graphical notation also expresses visibility relationships linking agents to the outside. Lines on fig. 6 show (i) how agents make information visible to other agents (e.g., the *Movies* component is made visible by the *Content Provider* to the *VoD*) and (ii) how external agents may influence the agent's behaviour through exportation of information (the *VoD* is influenced by the

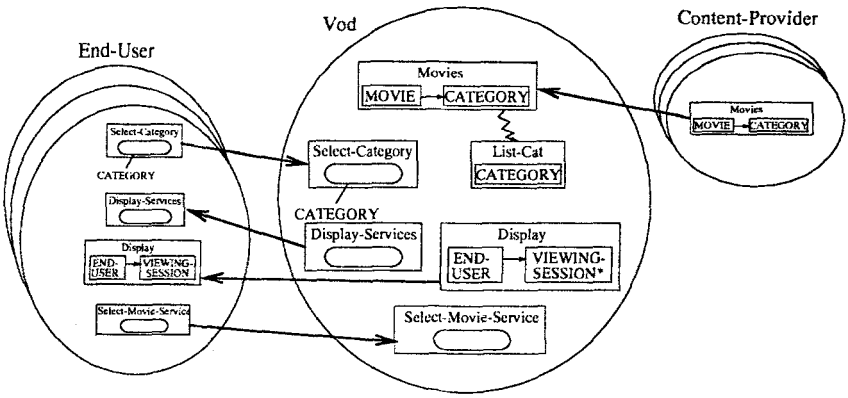

Fig. 6. The graphical declaration of the VoD system.

Select-Category action of the *EndUser* agent).

Finally, it is important to note that, in fig. 6, all the information characterising the *VoD* system is shared by the *VoD* with its environment. This typically results from our *operating environment* perspective focussing on the role of the product in terms of the external entities.

4.2 Classification of Properties

Besides graphical declarations, textual constraints are used for pruning the (usually) infinite set of possible lives associated with the agents of a system. These possible lives will respect different constraints classified in terms of **derived and local** constraints on the internal behavior of the agent and **cooperation** constraints on the interaction of agents within the society. To guide the requirements engineer in the elicitation and structuring of requirements, the constraints are further classified into categories, for each with a characteristic **template** is defined. For example, on fig. 7, we list a fragment of specification of requirements on the VoD, listing the following categories of constraints: **Derived Components, Initial Valuation, State Behaviour, Effects of Actions, Action Composition**,etc. The template for **State behaviour** axioms expresses restrictions on the possible agent's behavior only in terms of the values that can be taken by its state components, while the template for **Action Composition** axioms expresses restrictions only in terms of admissible sequences of actions/events. Details about the different templates of the Albert language fall outside of this paper and are given elsewhere [3].

The informal comments in fig. 7 should allow the reader to judge the expressiveness and naturalness of the language. As pointed out before, informal comments that paraphrase the formal constraints can be used to validate the specification with customers. These comments could correspond to sentences in a natural language specification of requirements, for example.

Vod

DECLARATIONS

STATE COMPONENTS

Movies denotes the information sent by the different Content-Providers about the available movies and the category to which they belong.

List-Cat is a (derived) component synthesizing the list of categories for which movies are proposed.

Display is associated with the state of the viewing session of each End-User. The '*' denotes that this state can be undefined at some moment, this denotes that the VoD is not in use by the End-User.

ACTIONS

Select-movie-service is the action triggered by an End-User when he/she wants to access to a movie service.

Select-Category is the action triggered by an End-User when he/she selects the category for which he/she desires to consult available movies.

Display-Services is the action triggered by the Vod in order to display the list of available services for an End-User.

BASIC CONSTRAINTS

DERIVED COMPONENTS

The list of categories is derived from the information given by the Content-Providers (1).

$(c \in List\text{-}Cat\ Leftrightarrow \exists\ Movies.cp:\ c \in Codom(Movies.cp))$

LOCAL CONSTRAINTS

INITIAL VALUATION

At the starting of the VoD, there is no on-going viewing session (2).

$Display[eu] = \text{UNDEF}$

STATE BEHAVIOUR

The display of the categories list (lc) for a given End-User (eu) remains until the display of services list (sl) or of the movies list (ml) (3).

$Display[eu] = lc\ \mathcal{U}\ (Display[eu] = se \vee Display[eu] = ml)$

The display of the categories list (lc) to a given End-User (eu) does not last for more than 2' (4).

$\neg\ \Box_{>2'}\ \ Display[eu] = lc$

EFFECTS OF ACTIONS

The Select-Movie-Service menu, originated from the End-User (eu), results in the display of the catagory list (lc) for this End-User (5).

$eu.Select\text{-}Movie\text{-}Service:\ Display[eu] := List\text{-}Cat$

CAPABILITY

The display of the services list (sl) should occur when the display of categories list is made for 1 minute (6).

$\mathcal{XO}\ (\ Display\text{-}services\text{-}List(eu)\ /\ \blacksquare_{=1min}\ \ Display[eu] = lc\ )$

ACTION COMPOSITION

The Select-Movie-service action brought by an End-User has to be followed by a Select-Category action made by this End-User or the Display-Services-List (7).

$Category\text{-}Selection \leftrightarrow eu.Select\text{-}Movie\text{-}Service\ ;\ (eu.Select\text{-}Category(c) \oplus Display\text{-}Services\text{-}List)$

ACTION DURATION

The duration of the Category-Selection process should be less than 1' (8).

$0 < |\ Category\text{-}Selection\ | \leq 1'$

COOPERATION CONSTRAINTS

ACTION PERCEPTION

The VoD percieves the Select-Category action brought by an End-User (eu) if the selected category (c) is one of those proposed in the Catgory-List (9).

$\mathcal{XK}\ (\ eu.Select\text{-}Category(c)\ /\ c \in List\text{-}Cat\ )$

STATE PERCEPTION

The VoD always perceives the information of movies brought by Content-Providers (cp) (10).

$\mathcal{XK}\ (\ cp.Movies\ /\ \text{TRUE}\ )$

ACTION INFORMATION

The VoD always informs the end-user of a Display-Services-List action (11).

$\mathcal{XK}\ (\ Display\text{-}Services.eu\ /\ \text{TRUE}\ )$

STATE INFORMATION

The VoD only shows, to the End-User (eu), his/her own viewing session (prodided that this one is on-going) (12).

$\mathcal{XK}\ (\ Display[eu].eu'\ /\ eu' = eu \wedge Display[eu] \neq \text{UNDEF}\ )$

Fig. 7. Albert II specification of some properties of the VoD.

5 Discussion and Conclusions

Our experience with formal techniques in the 2Rare project is that their use allows the discovery of errors in the requirements document. A ratio was established showing that the total amount of time spent in writing the formal specification was marginal (the ratio established it to 10%) with respect of the total of time devoted to the correction of the discovered errors. The extra time required to use formal techniques is on the average the same as the time needed to fix 12 requirements problems. An important conclusion from the project is that the introduction of semiformal or formal techniques for requirements specification must be guided by the maturity level of the software organization. More information on the lessons learned during the project can be found at the URL `http://www.info.fundp.ac.be/ phe/2rare.html`.

To summarize, we have shown that the TRADE framework can be used to write coherent specifications of requirements in semiformal (diagrammatic) and formal languages. This approach was illustrated with some examples from one of the applications studied in the 2RARE project, using Albert II as a formal specification language.

Acknowledgements: Thanks are due to M. Lemoine and J. Foisseau of CERT-ONERA, Toulouse, and P. Dieu and L. Levrouw of Belgacom Research, Brussels, for the fruitful cooperation and stimulating discussions during the 2RARE project.

References

1. 2RARE (2 Real Applications for Requirements Engineering): Project Programme. http://www.info.fundp.ac.be/ ~phe/2rare.html, August 1995. Esprit Contract Number 20424.
2. Klemens Böhm and Amilcar Sernadas. A logic to specify real-time object behaviour. Technical report, Departamento de Matemática, Instituto Superior Técnico, Lisbon (Portugal), 1993.
3. P. du Bois. *The ALBERT II Language.* PhD thesis, Facultés Universitaires Notre-Dame de la Paix, Namur, 1995.
4. G. Booch. *Object-Oriented Design with Applications, Second edition.* Benjamin/Cummings, 1994.
5. G. Booch, I. Jacobson, and J. Rumbaugh. The unified modeling language for object-oriented development. version 0.91 addendum. Technical report, Rational Software Corporation, 2800 San Tomas Expressway, Santa Clara, CA 95051-0951, September 1996. URL http://www.rational.com/ot/uml91.pdf.
6. P.P.-S. Chen. The entity-relationship model – Toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.
7. D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development: The FUSION Method.* Prentice-Hall, 1994.
8. A.M. Davis. *Software Requirements: Objects, Functions, States.* Prentice-Hall, 1993.

9. T. DeMarco. *Structured Analysis and System Specification.* Yourdon Press/Prentice-Hall, 1978.
10. Eric Dubois, Philippe Du Bois, and Michaël Petit. Elicitating and formalising requirements for CIM information systems. In C. Rolland, F. Bodart, and C. Cauvet, editors, *Proc. of the 5th conference on advanced information systems engineering - CAiSE'93*, pages 252–274, Paris (France), June 8-11, 1993. LNCS 685, Springer-Verlag.
11. Eric Dubois, Philippe Du Bois, and Jean-Marc Zeippen. A formal requirements engineering method for real-time, concurrent, and distributed systems. In *Proc. of ICSE-17 Workshop on Formal Methods Applications in Software Engineering*, Seattle WA, April 24-25, 1995.
12. D. Harel and A. Pnueli. On the development of reactive systems. In K. Apt, editor, *Logics and Models of Concurrent Systems*, pages 477–498. Springer, 1985. NATO ASI Series.
13. M. Jackson. *System Development.* Prentice-Hall, 1983.
14. M. Jackson. *Software Requirements and Specifications: A lexicon of practice, principles and prejudices.* Addison-Wesley, 1995.
15. I. Jacobson, M. Christerson, P. Johnsson, and G. Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Prentice-Hall, 1992.
16. M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, and Y. Vassiliou. Theories underlying requirements engineering: an overview of the NATURE approach. In S. Fickas and A. Finkelstein, editors, *International Symposium on Requirements Engineering*, pages 19–31. IEEE Computer Science Press, 1993.
17. Objectory AB. *Objectory: Robustness Analysis*, version 3.6 edition, 1995.
18. Oblog Software. *OBLOG Case V1.2*, 1995.
19. Quality Systems and Software Ltd. *DOORS Unix Reference Manual, Version 2.1*, 1995.
20. R.H. Thayer. Software system engineering. In R.H. Thayer and M. Dorfman, editors, *System and Software Requirements Engineering*, pages 77–116. IEEE Computer Science Press, 1990.
21. R.J. Wieringa. *Requirements Engineering: Semantic, Real-Time, and Object-Oriented Methods.* Course notes., 1995.
22. R.J. Wieringa. *Requirements Engineering: Frameworks for Understanding.* Wiley, 1996.
23. R.J. Wieringa. Advanced object-oriented requirement specification methods. Technical report, faculty of Mathematics and Computer Science, *Vrije Universiteit*, 1997. Tutotial presented at the *International Symposium of Requirements Engineering, 6–10 january 1997, Annapolis, U.S.A.*
24. R.J. Wieringa and S. Huyts. Requirements analysis of the VoD application using the tools in TRADE. Technical report, Faculty of Mathematics and Computer Science, *Vrije Universiteit*, De Boelelaan 1081a, 1081 HV Amsterdam, 1996.
25. R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software.* Prentice-Hall, 1990.