Towards an Environment to Support Requirements Formalisation

JAELSON FREIRE BRELAZ DE CASTRO
CHRISTIAN JEAN GAUTREAU
MARCO ANTONIO TORANZO CÉSPEDES

Departamento de Informática, UFPE
Caixa Postal 7851, Recife
CEP 50732-970 Brazil
{jbc, cg, matc}@di.ufpe.br

Abstract

A major issue in requirements engineering is how to bridge the gap between ill-defined application situations and production of precise, formal specification of functional requirements. Although various formal languages have been proposed in the literature, practitioners still have great difficulties specifying requirements in their chosen formal language. One way of overcoming this barrier is to integrate formal techniques with existing methodologies (CORE, OMT, etc) and providing tool support for building formal specifications incrementally. In this paper we present The Multiview Tool, which addresses the problem of capturing requirements in the context of formal specification. Multiview enables multiple stakeholders to express their viewpoints based on a number of interrelated models, written in a variety of different notations. A detailed description of the project of the Multiview is provided. We start with the System Interface Context Diagram and a general overview of the tool Domain. A critical part of this domain is elaborated in terms of an Information Model. We also show one of the metamodels supported by the tool (Object Class representation schema). Several Interface Prototypes are also presented, together with an Object Message Model for a specific scenario. Statecharts are used to represent the behaviour of editors supported by the tool.

KEY WORDS: Requirements, Method, Tool, Formalisation.

1 Introduction

Requirements analysis is the first technical activity in software development. It is concerned with eliciting, clarifying and documenting the desired functionality of a particular system.

A number of formal languages for capturing requirements have been proposed in the literature [25]. A formal language is a mathematically based language for describing a system. Using formal languages people can systematically specify, develop, and verify a system. The capability of expression has greatly improved. Early work (ex. Z,VDM, Larch) could only address data and operation of the system described in terms of first order assertions (conditional equations, pre/prost-conditions and invariants. Recent developments have enabled us to reason about agent responsibilities [17], [14], goal satisfaction [26], [15], the triggering events and causalities [7], [12].

Unfortunately, few people understand exactly what formal languages and methods are or how they are applied [4]. Therefore, one of the most challenging issues in requirements engineering is how to proceed from informal, fuzzy, individual statements of requirements to a formal specification that is agreed upon by all stakeholders [5].

In this paper we address this problem of requirements elicitation and formalisation. In particular we advocate the use of multiple perspectives, i.e ViewPoints, to present prescriptive methods for facilitating this activity [8, 18].

We present the project of a tool called *Multiview* aimed at supporting the use of multiple representation schemes used by the VSCS Method (Very Structured Common Sense) which guides and organises the activity by which a formal requirement specification in structured modal action logic [17] is obtained from an informal application concept.

The remainder of the paper is organised as follows: Section 2 introduces structured modal action logic. Section 3 provides an overview of the method VSCS. It is a configuration of following ViewPoints templates: Object (structure), Text (informal description), Event life-cycle (interaction), Statechart (dynamic behaviour), Causal (deontics) and Structured MAL (formal specification). In Section 4 we discuss the process by which the formal specification is obtained from The Statechart ViewPoint. Section 5 describes an overview of the Multiview tool. Section 6 concludes the paper.

2 Structured Modal Action Logic

Our aim is to be able to formalise requirements in Modal Action Logic (MAL). In this section we review some syntactical features of structured MAL and show how it can be used to describe the behaviour of objects of the system. An interested reader can find a full description of the logic in [23, 17].

Objects can interact by sharing attributes or by sharing actions. An *attribute* is part of the state of the system—a predicate or function which varies not only with its arguments but also with the state of the object to which it belongs. In general, objects should be as self-contained as possible; the actions of an object should only update attributes of that object, and vice versa.

In structured MAL, a specification has a set of declarations and a set of axioms. The modal operator [ ] is used to capture the effect of the occurrences of actions. For example, a car can be considered as the aggregation of several smaller objects. The atomic agents which might make up the specification include:

• **Brake**, with the actions 'depress brake', 'release brake' and the attribute 'brake-pos' (to denote the brake position).
• **Accelerator**, with the actions 'depress accelerator', 'release accelerator' and the attribute 'acc-pos' (to denote the accelerator position).
• **Transmission**, with actions 'push F', 'push R', 'push N' and the attribute 'gear-pos' (to denote the gear positon).
• **Ignition**, with the actions 'turn key to start', 'release key', 'turn key off' and the attribute 'ig-st' (to denote the ignition status).

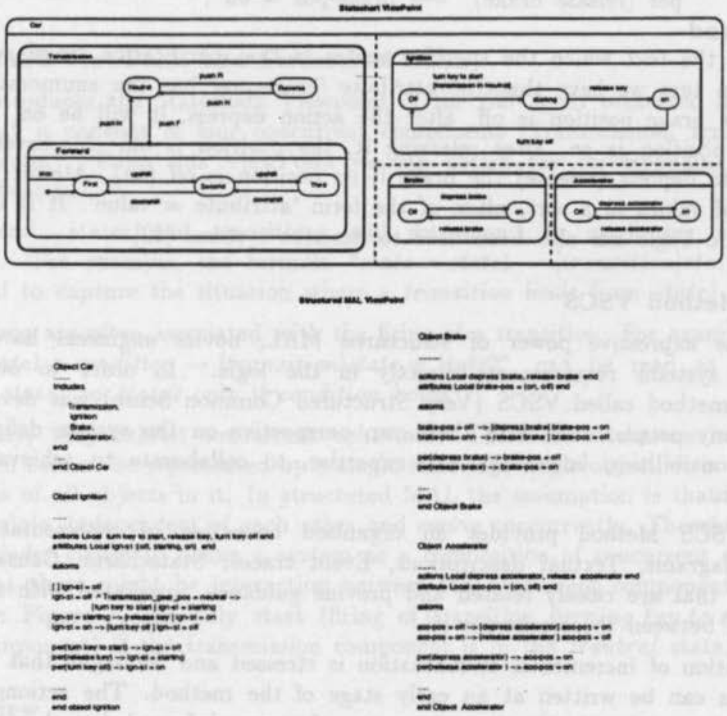


Figure 1: Concurrency and Structured MAL ViewPoint

A specification is essentially a set of agent or object descriptions. One of these will be a description of the object corresponding to the system being specified, and the others will be descriptions of sub-components of that system (see Figure 1).

Brakes can be depressed or released. So **Brake** looks like this:

```
agent Brake
attributes
    brake-pos : (on, off) end
actions
    depress brake, release brake end
axioms
    brake-pos = off  ⟶ [depress brake] brake-pos = on ;
    brake-pos = on   ⟶ [release brake] brake-pos = off ;
    per (depress brake)  ⟶ brake-pos = off ;
    per (release brake)  ⟶ brake-pos = on ;
end
```

This is the text which the specifier writes in the specification language. From the specification text we have that the attribute 'brake-pos' has the enumerated sort (on, off). If the brake position is off, after the action depress, it will be on. Similarly, if the brake position is on, after releasing it, the position is off. Moreover, it is only permitted to depress (release) the brake if its position is off (on). Atomic formulas are equalities of values in a sort, often of the form 'attribute = value'. It is worth noting that Deontic Logic has also been used to describe systems [22]

3 The Method VSCS

Despite the expressive power of structured MAL, novice engineers have difficulties expressing systems requirements directly in the logic. In order to overcome this barrier, a method called VSCS (Very Structured Common Sense) was developed[8]. It enables many people, each with their own perspective on the system defined by their skills, responsibilities, knowledge and expertise, to collaborate to achieve the formal specification.

The VSCS Method provides an organised collection of representation schemes (Object Diagrams, Textual descriptions, Event traces, Statecharts, Causal Diagrams, MAL, etc) that are closely related and provide guidance, integrated with a work plan, for moving between these schemes [6].

The notion of incremental formalisation is stressed and we argue that most formal declarations can be written at an early stage of the method. The actions, attributes, functions, etc. produced at earlier stages are, of course, informal, but they are produced in a sufficiently constrained way that the transformation into the formal language of structured modal action logic is possible.

The Object ViewPoint is the first phase of the method. Its role is to describe the structure of objects in the system - their identity, their relationships to other objects, their attributes, and their actions. In section 5.4 we present the Object class metamodel elements.

The Event life-cycle ViewPoint shows interactions and information exchanges. One or more typical dialogues (scenarios) between the user and system are prepared.

The Statechart ViewPoint captures control, that aspect of a system that describes the sequences of actions that occur in response to external stimuli. The graphical

notation is that of statechart [16].

The purpose of the Causal ViewPoint is to capture the deontic aspects of actions in specification, i.e., when actions are permitted and obliged to occur. We have adapted statechart notation to capture some obligation structures found in structured MAL.

VSCS is not just a sequence of steps, each of which produces graphical or informal English documentation. The aim is to develop a formal specification in terms of structured MAL. The full description of the method can be found in [7]. Due to space limitations, we present only an overview of the dynamic behaviour (Statechart ViewPoint) and show how to obtain the corresponding formal counterpart in structured MAL.

4 Statechart ViewPoint and Structured MAL

Figure 1 introduces the Statechart Viewpoint of the *Car* agent presented in Section 2. Observe that it consists of four concurrent components (Transmission, Ignition, Brake and Accelerator). From this ViewPoint we can obtain the corresponding Structured MAL specification.

In general, states and transitions will correspond to attributes and actions, respectively. For example, the formula "$state = state1 \rightarrow [transition]state = state2$", can be used to capture the situation where a *transition* leads from *state*1 to *state*2.

Conditions are often associated with the firing of a transition. For example, formula "$state = state1 \wedge condition \rightarrow [transition]state = state2$", can be read as "*transition* leads from *state*1 to *state*2 only if *condition* holds".

Statecharts may exhibit concurrent behaviour. There are situations in which the entire system cannot be represented by a single state in a single object; it is the product of the states of all objects in it. In structured MAL the assumption is that components are, in principle, independent of each other and evolve concurrently. Therefore, we shall use the *includes* clause to define a system as a composition of concurrent components. Observe that there might be interaction between the concurrent components. The *car* described in Figure 1, can only start (firing of transition *turning key to start* in the *ignition* component) if the transmission component is in the *Neutral* state.

5 Multiview

In this section we present a general overview of the Multiview tool design, aimed at supporting the use of multiple representation schemes used by the VSCS Method. This tool will be integrated with the Software Engineering Environment being developed by the FORMLAB Project, which is a collaborative work among the following Brazilian institutions: Universidade Federal de Pernambuco (UFPE), Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Universidade Estadual de Maringá (UEM) and Equitel.

In a previous report [9, 10], the Multiview analysis phase was presented in terms of Use Cases, Domains Map and Information Model among other models. In this report we concentrate on the design phase. In the following subsections we shall describe

the System Interface Context Diagram, Domains, Design Information Model, Interface Prototyping, Object Messages Models, Object Class Metamodel and the dynamic behaviour (in terms of statechart) of the Diagram Editor Class. The design phase is presented in terms of well known notations [21, 3, 24, 11].

5.1 Multiview System Interface Context

The System Interface is defined in terms of the set of operations (messages) that the Multiview can receive and the events that it can send. The system communicates with active agents (actors) in the global environment. Agents request system operations that can change the system state and produce output events.

The Fusion Method is a second-generation object-oriented analysis and design method. Fusion was designed to integrate and extend features from the Rumbaugh/OMT. Booch, and CRC methods [1]. In figure 2 we use the Fusion notation [11] to present the System Interface Context Diagram. It was constructed from the Use Cases [3] developed during the analysis phase[10] and the process model. For simplification, due to the great number of external events available on this very interactive system, we show only the triggering events.

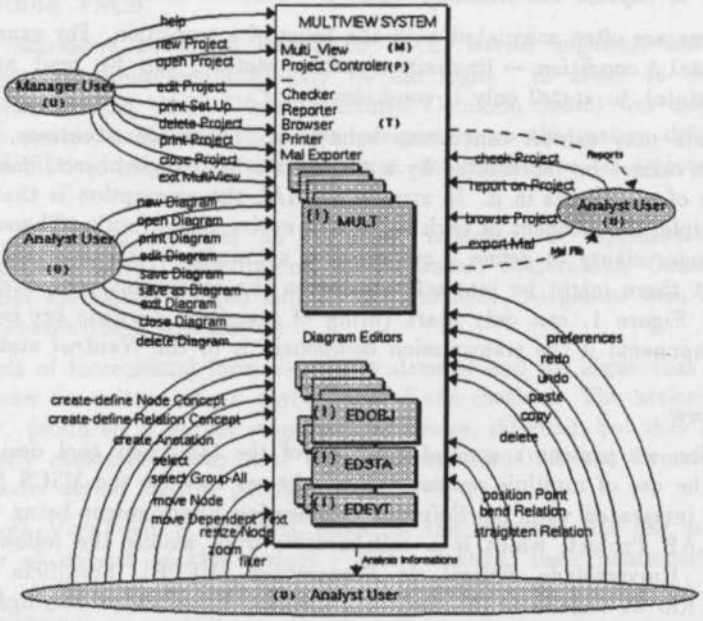


Figure 2: System Interface Context Diagram

5.2 Multiview Domains

Multiview is a quite complex system. It can be structured in terms of *domains*[24], which are a set of related objects. Each domain can be separately developed by a specialist and some of them may be reused in other applications. In Figure 3, filled rectangles represent domains, while arrows indicate client/server relationships.

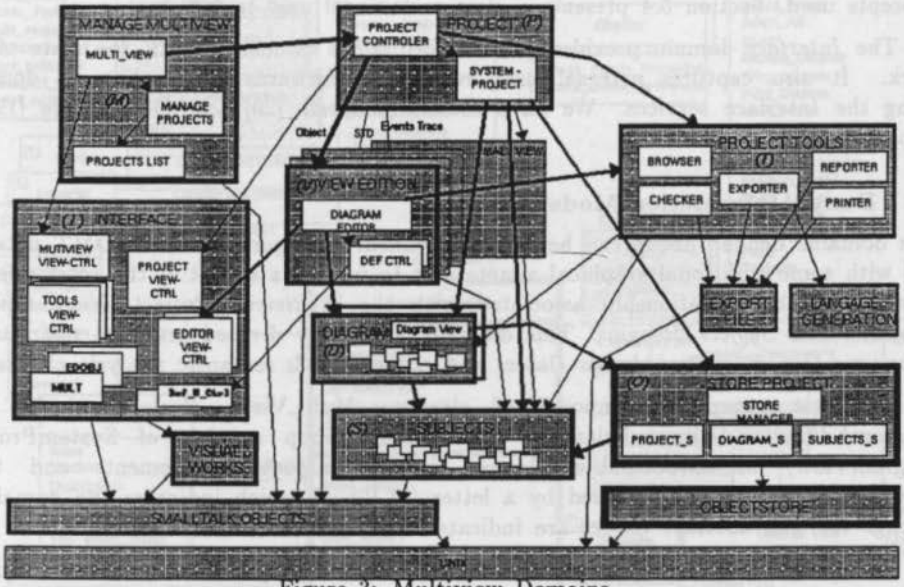


Figure 3: Multiview Domains

The top-level domain of the system is *Manage Multiview*. It is responsible for system initialisation and management of projects. This domain relies on services provided by two other domains: *Interface* and *Project*.

The *Project* domain is responsible for managing current project information and the activation of specific editors to create/modify a diagram (in the *View Edition* domain). It can also initiate tools associated with the project (from *Project Tools* domain) and relies on services from the *Interface* domain for its own communication. It uses the *Store Project* domain to retrieve/save the project and the data dictionary (*Project_S* and *Subjects_S* sub-domains).

The *View Edition* domain is responsible for the construction of each specification ViewPoint. It uses services from the *Diagram* domain to structure the representations and record associated information. It also can activate tools in the *Project Tools* domain.

The *Diagram* domain contains the structure of the graph of the current diagram as well as the visual representation of each diagram's elements. It uses services from *Store*

Project to retrieve/save diagram information. For each representation, an associated element dictionary is maintained (in *Subjects* domain).

Method such as VSCS, can be represented by connected ViewPoints metamodels, that hold the structure, properties and relations of the concepts generated from each different supported viewpoint (object, statecharts, etc). These metamodels are necessary for the implementation of a CASE tool. The *Subject* domain contains the metamodel defining the ViewPoints supported by the tool. It provides semantics of the modeling concepts used. Section 5.4 presents a class metamodel used in VSCS.

The *Interface* domain provides visual informations associated with the state of the work. It also captures external user events and forwards information to domains using the Interface services. We have chosen Smalltalk [20] and ObjectStore [19] as implementation tools.

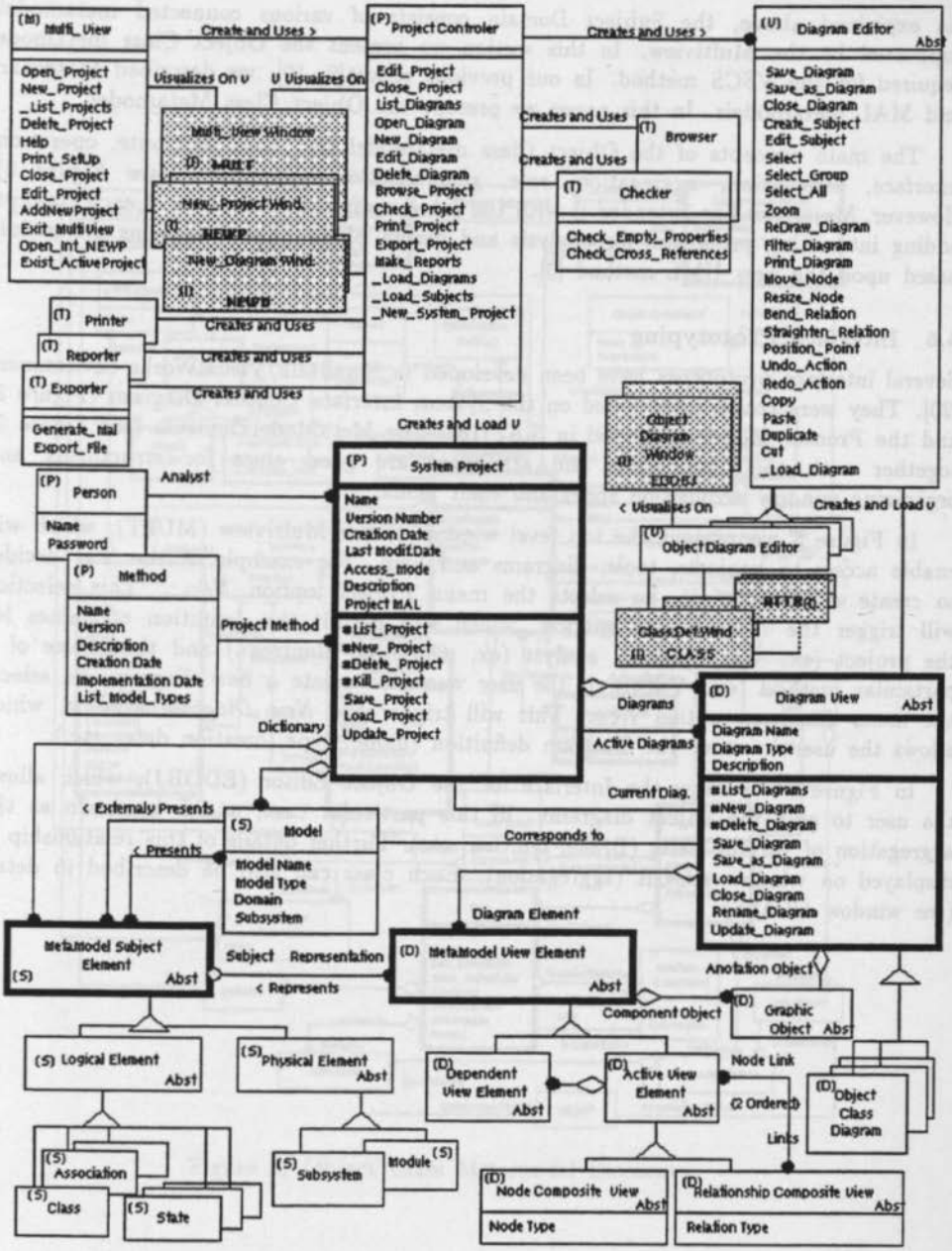
5.3 Design Information Model

The domains defined above can be further detailed. In Figure 4 we use OMT notation [21] with some additional graphical annotations to present a subset of the most relevant classes and their relationship associated with the *Multiview, Project, View Edition, Diagram, and Subject* domains. This diagram is logically divided into two main areas: communication & control design classes and conceptual & semantic analysis classes.

The first group is composed of classes: Multi_View, ProjectControler and DiagramEditor and its subclasses. The second group consists of SystemProject, DiagramView, MetaModelSubjectElement and MetaModelViewElements and their subclasses. Each class is indexed by a letter (M,P,I,...) which indicates the domain it belongs to. The abstract classes are indicated with the abreviation "abst".

The Multi_View class is responsible for providing all high level operations required for project management and selection of a specific project. For example, these services are used for accessing the Multiview (see interface MULT, Figure 6) and the creation of a new project (interface NEWP, Figure 6). The ProjectControler class is responsible for managing the current project and activating the tools associated with it. The rest of the classes are derived directly from analysis. A user is an authorised person who can work with the Multiview tool. He uses a specific method (such as VSCS) for requirements elicitation and formalisation. He creates a *System Project* consisting of various viewpoints (*Diagram View*) supported by the method.

The diagram construction is associated with two types of information: graphical representation (*Metamodel View Element*) and conceptual definition (*Metamodel Subject Element*). The two are related. The first one (graphics) represents concepts defined within the dictionary composed of *Metamodel Subject Elements*. All concepts used by a method (class, association, state, etc) are subtypes of the *Logical Element*. Note that a dependency relationship (simple aggregation) can exist between elements of a diagram (*Dependent View Elements and Active View Element*). Moreover, a diagram can be considered as a graph. Some elements (*Active View Elements*) have node behaviour (*Node Composite View*) and others have arc behaviour (*Relationship Composite View*).



5.4 Refinement of Subject Domain

As explained above, the Subject Domain consists of various connected metamodels supported by the Multiview. In this section we present the Object Class metamodel required for the VSCS method. In our previous report[9, 10], we described Statechart and MAL metamodels. In this paper we present the Object Class Metamodel.

The main concepts of the Object Class metamodel are: class, attribute, operation, interface, association, aggregation, role, generalisation and object (see Figure 5). However, Multiview has extended it with the more complete description of each concept, adding interesting properties for analysis and design phase. The extensions are mainly based upon the new UML method [3].

5.5 Interface Prototyping

Several interface prototypes have been developed in Smalltalk/VisualWorks environment [20]. They were constructed based on the System Interface Context Diagram (Figure 2) and the Process Model developed in S.A. [13]. The MetaModelElements (see Figure 5) together with their properties and attributes are good clues for structuring and organizing window acquisition zones and their fields.

In Figure 6 we present the top level window of the Multiview (MULT), which will enable access to projects, tools, diagrams and help. For example, if the user decides to create a new project, he selects the menu *Project*, option *New* . This selection will trigger the *New Project* window, which will permit the definition of names for the project (ex. Car System), analyst (ex. Christian Gautreau) and the choice of a particular method (ex. VSCS). If the user wants to create a new diagram, he selects the menu *Diagram*, option *New*. This will trigger the *New Diagram* window, which allows the user to enter the Diagram definition (name, type, creation date, etc.).

In Figure 7 we show the Interface for the Object Editor (EDOBJ), which allows the user to edit the object diagrams. In this particular case, a *Car* is shown as the aggregation of various parts (Brake, Ignition, etc). Further details of this relationship is displayed on window AGGR (aggregation). Each class can also be described in detail (see window CLASS).

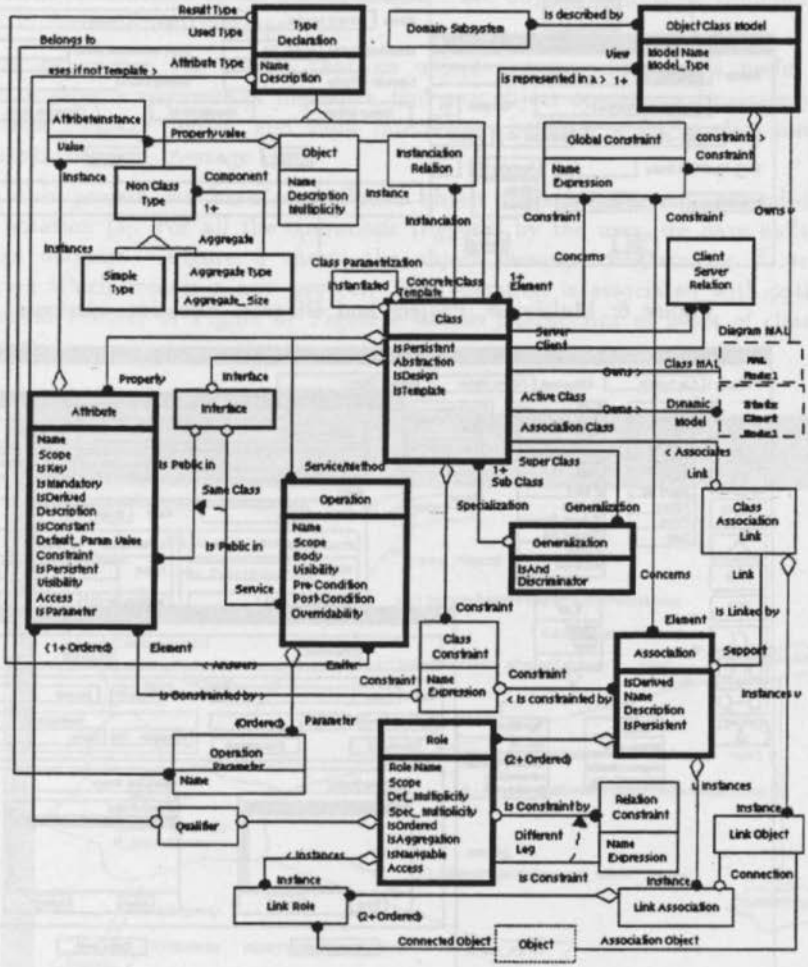


Figure 5: Object Class Metamodel Elements

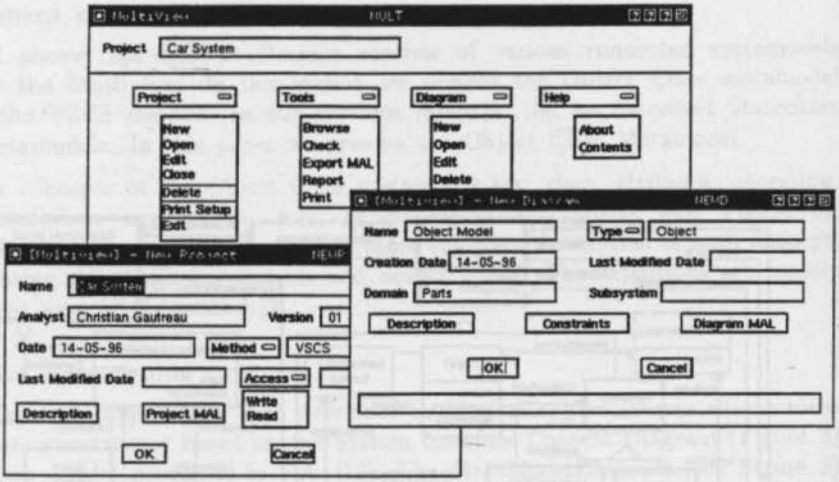


Figure 6: Multiview, Project and Diagram Creation windows

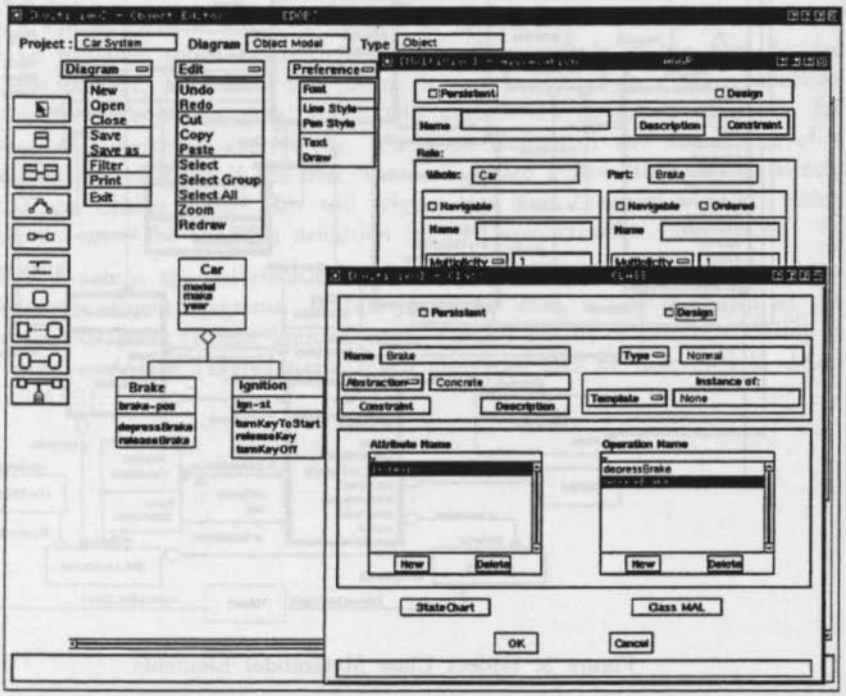


Figure 7: Object Editor, Class and Aggregation Defintion

5.6 Object Message Models

We used object diagram to show the existence of objects previously defined by the Class Models (Figure 4 and 5), their responsibilities, and the use of their dynamic and static relationships in the logical design of Multiview. An object diagram is a scenario diagram that shows the sequence of messages which implement an operation or a transaction [2]. It shows the objects and links that exist just before the operation begins as well as the objects and links created or destroyed during the operation. The two essential elements of an object diagram are objects and their communication via static or dynamic relationships.

Our experience has shown that an object message diagram is useful in logical design to show a sequence of messages, find new object operations (messages), structure and refine message bodies, and show interactions between a set of class instances and synchronizations of message calls.

In this project we have used Booch object message diagram, extended with the UML notation [3]. For all the operations triggered by the user, we have built an object message diagram. Figure 8 shows the object message diagram for *"1:new_Project"* operation which creates a new project. This operation is associated with option *new* of the button Project of Figure 6. Figure 8 details interactions of a set of class instances of Figure 4.

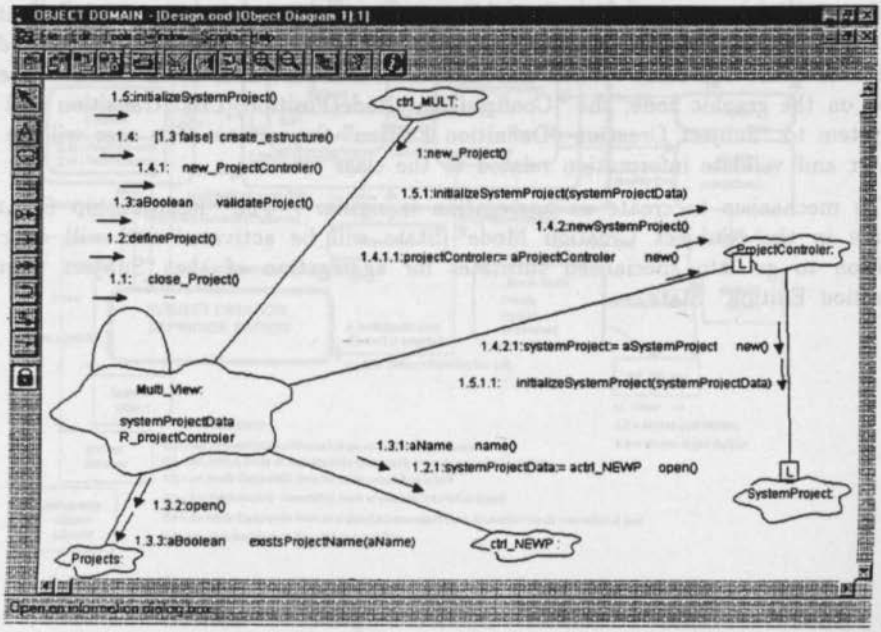


Figure 8: Object Diagram Message for new_Project() Operation

The new_Project operation is triggered from object *ctrl_MULT*, controller of window MULT presented in Figure 6. Operation new_Project is composed of operations indicated by numbers 1.1, 1.2,... 1.5. For example, "1.2.1: systemProjectData:= actrl_NEWP open()" is a refinement of *"1.2:defineProject"*.

5.7 Diagram Editor Behaviour

Statechart diagrams can describe the temporal evolution of an object of a given class in response to interactions with other objects inside or outside the system [3]. These diagrams should be used mainly for objects that receive external events from actors outside of the system (in our case, the analyst user). We follow the suggestion of [24] and use a prefix in front of each trigger event to denote the event sender (for example U: for user).

The Diagram Editor is an abstract class which defines the common properties of all Multiview graphical editors (see Figure 4). In Figure 9 its dynamic behaviour is described in terms of a statechart.

Consider, for example, the case of the Object Editor. The editor starts executing the initialization state and moves to the state "No selected element". If the user activates the *class creation* icon, on the window left border by clicking the mouse left button[LB] (Subject_Icon(Type=Class))[LB] event), the editor will go to the "Subject Creation Mode" state. As the chosen concept is class, it goes to "Node Creation" substate in which it will execute an operation to wait for a location event, to find the position to place the class. When this information is provided, by clicking mouse left button on the graphic zone, the "ConfirmCreateNode(Position)[LB]" transition will lead the system to "Subject Creation /Definition Edition" Superstate. The user will be able to enter and validate information related to the class creation.

The mechanism to create an aggregation is similar . The "Relationship Creation" substate in the "Subject Creation Mode" State will be activated and will offer the transition to go into specialized substates for aggregation of the "Subject Creation /Definition Edition" State.

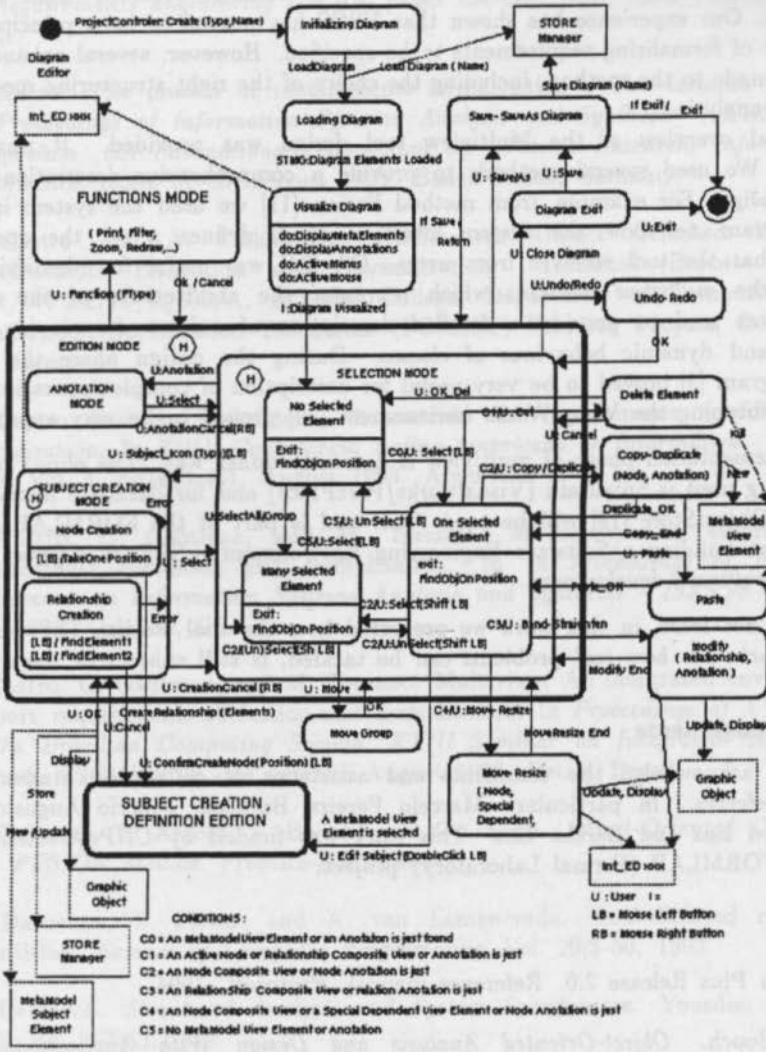


Figure 9: Diagram Editor Statechart

6 Conclusion

In this paper we addressed the problem requirements capture in the context of formal specification, and provided a general overview of the Multiview tool design.

A prescriptive method called VSCS, aimed at guiding and organising the activity by which a structured modal action logic specification is obtained from informal requirements. Our experience has shown that VSCS has proved to be a principled and effective way of formalising requirements to be specified. However, several enhancements can still be made to the method, including the choice of the right structuring mechanism and deontic analysis.

A general overview of the Multiview tool design was provided. It was based on [9, 10]. We used several methods to provide a comprehensive description of the Multiview design. For example, from method Fusion [11] we used the system interface context diagram, to show the system interface which defines a set the operations (messages) that the tool receives from users. OA [24] was useful for identifying and structuring the *multiview domains* which represent the architecture of the system. OMT [21] was used to provide a detailed description of *subject domain* (ex. *class metamodel*) and dynamic behaviour of classes. During the design phase the Object Message Diagram [3] proved to be very useful for description of complex operations. For interface prototyping the VisualWorks environment [20] proved to be very appropriate.

The implementation phase of multiview is currently under way. The object oriented language being used is Smalltalk (Visualworks/ParcPlace) and for database management system, the ObjectStore [19] will be used. This tool is part of the FORMLAB Project which aims at building a Software Engineering Environment - SEE to support formal methods for software development.

Last but not least, in this work we presented how the tool works. The usefulness of the approach, i.e. how real problems can be tackled, is still subject of research.

7 Acknowledgements

The authors acknowledge the comments and assistance of colleagues, students and anonymous referees. In particular: Marcelo Pereira Barbosa, Marcio Augusto Silva Breno, Eduard Ens and Bertha Ens. This work was funded by CNPq/Protem II as part of the FORMLAB (Formal Laboratory) project.

References

[1] Paradigm Plus Release 2.0. Reference manual. Protosoft, 1994.

[2] Grady Booch. *Object-Oriented Analysis and Design With Applications.* The Benjamin/Cunnings Publishing Company, Inc., Redwood City, CA (USA), 1994.

[3] Grady Booch and James Rumbaugh. *Unified Method For Object Oriented Development - Documentation Set, Version 0.8.* Rational Software Coorporation, 1995.

[4] J. Bowen and M. Hinchey. Seven more myths of formal methods. *IEEE Computer*, pages 34–41, July 1995.

[5] J. Bubenko, C. Rolland, P. Loucopoulos, and V. DeAntonellis. Facilitating "fuzzy to formal" requirements modeling. In *Proceedings of the First Internacional Conference on Requirements Engineering - ICRE*, pages 154–165, April 1994. Colorado Springs. USA.

[6] J. Castro. The process of requirements formalisation : The formlab project. In *In Proceedings of Information Systems Analysis and Synthesis - ISAS'95, Focus Symposium, 5th International Symposium on SYstems Research, Informatics and Cybernetics*, pages 01–05, August 1995. Baden-Baden, Germany.

[7] J. Castro and A. Finkelstein. VSCS: An Object Oriented Method for Requirements Elicitation and Formalisation. Technical Report Deliverable NFR/WP2.2/IC/R/002/A, Imperial College, Department of Computing, Formal Requirement Specification Techniques (FOREST) Research Project, October 1991. 107 pages.

[8] J. Castro and A. Finkelstein. A method for requirements specification and formalisation. In *XVIII Conferência Latino-Americana de Informática - PANEL'93*, pages Vol. 1, pp–453–451, August 1993. Argentina.

[9] J. Castro, C. Gautreau, and M. Toranzo. Multiview: An environment for requirements elicitation and formalisation. In *In Proceedings of International Conference on Information Systems Analysis and Synthesis - ISAS'96 (to appear)*, July 1996. Orlando, USA.

[10] J. Castro, C. Gautreau, and M. Toranzo. Multiview: An integrated environment to support requirements elicitation and formalisation. In *Proceedings of XVI Congress of the Brazilian Computing Society, XXIII Seminar on Integrated Hardware and Software (SEMISH'96 - to appear)*, August 1996. Recife, Brazil.

[11] D. Coleman, P. Arnold, S. Bodoff, and C. Dollin. *Object-Oriented Development: The FUSION Method.* Prentice-Hall, 1994.

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

[13] T. DeMarco. *Structured Analysis and System Specification.* Yourdon Press, New York, NY (USA), 1979.

[14] E Dubois, P. DuBois, and M. Petit. ALBERT: An Agent-Oriented Language for building and eliciting requirements for real-time systems. In J. F. Nunamacker and R. H. Sprague, editors, *Proc. of the 2th HICSS*, volume 4, pages 713–722. IEEE Computer Society Press, January 1994.

[15] M. Feather. Requirements Reconnoitering at the Juncture of Domain and Instance. In *Proceedings of IEEE International Symposium on Requirements Engineering - RE93*, pages 73–77, January 1993.

[16] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(No. 3):231–274, June 1987.

[17] S. Kent, T. Maibaum, and W. Quirk. Formally specifying temporal constraints and error recovery. In *Proceedings of IEEE International Symposium on Requirements Engineering - RE93*, pages 208–215, January 1993.

[18] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationship between multiple views in requirements specification. *IEEE Transaction on Software Engineering*, Vol. 20(No. 10):760–773, 1994.

[19] ObjectStore. Release 4.0. reference manual. ObjectDesign, 1995.

[20] Objectworks. Smalltalk release 4.1. reference manual. ParcPlace Systems, 1994.

[21] J. Rumbaugh, M. Blaha, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall International, Englewood Cliffs, NJ(USA), 1991.

[22] M. D. Ryan. Towards specifying norms. In *First International Workshop on Deontic Logic in Computer Science*, 1991.

[23] M. D. Ryan, J. Fiadeiro, and T. Maibaum. Sharing actions and attributes in modal action logic. In T. Ito and A. Meyer, editors, *Theoretical Aspects of Computer Software*. Springer Verlag, 1991.

[24] S. Shlaer and S. J. Mellor. *Object-Life Cycles: Modeling the World in States*. Yourdon Press, Englewood Cliffs, N.J., 1991.

[25] A. van Lamsweerde, R. Darimont, and P. Massonet. Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering - RE95*, pages 194–203, March 1995.

[26] E. Yu and J. Mylopoulos. Understanding "why" in software process modelling, analysis and design. In *Proceedings of the 16th International Conference on Software Engineering, ICSE'94*, pages 159–168. IEEE CS Press, May 1994. Sorrento (Italy).