17

Semantic units and connectors : towards domain knowledge reuse

C. Cauvet, F. Semmak***
(*) Univ. de PARIS I/IAE, 162, Rue St Charles 75015 Paris, France
(**) Univ. de PARIS I/CRI, 17, Rue de Tolbiac 75013 Paris, France
email : {alecsi, semmak}@masi.ibp.fr

Abstract

This paper presents a reuse-based approach in the context of conceptual modelling. A precondition for reusability is the existence of reusable components; our belief is that domain engineering facilitates the identification and capture of generic structures for families of applications. The domain modelling approach we propose in this paper is based upon two knowledge basic building blocks : semantic units and semantic connectors. A semantic unit captures a pattern dealing with domain object behaviour considered within a particular context. Semantic connectors prescribe the ways in which semantic units can be connected together. Distinguishing between semantic units and connectors allows to build complex generic models much richer than classic libraries of components. Furthermore, semantic units and semantic connectors are associated with mechanisms increasing genericity and the ability to capture common properties as well as the discriminant ones for a set of applications of the same domain. These mechanisms are powerful for increasing both the applicability of a generic model and supporting an efficient component retrieval process.

Keywords

Reusability, domain analysis, generic model, reusable component.

1 INTRODUCTION

System engineering is a process consisting, on the one hand, in apprehending the functions and behaviour of a system in order to extract valid specification and, on the other hand, to use this specification to design and develop software. This process relies on the fundamental principle of conceptualisation aiming at differentiating conceptual specification from system implementation.

As far as system engineering is concerned, the maturity of engineering methods and the power of the current tools allow to consider new orientation in research emphasising optimisation of products and processes engineering (Castano, 1994) (Krueger, 1992). Reuse is viewed as a new approach aimed at improving both product quality and process performance. This approach has been first used during implementation where programmers have repositories of software components and design architecture (McIlroy, 1976) (Booch, 1987) (Deutsch, 1989). Conversely, the reuse principles are barely used during conceptual

A. Sutcliffe et al. (eds.), *Domain Knowledge for Interactive System Design*

modelling (Hall, 1992) (Bellinzona, 1993). This paper presents a reuse-based approach in the context of conceptual modelling.

Practically, conceptual design focuses in representing a system at a high level of abstraction. Therefore, this activity is centered on the definition of a conceptual solution. Nowadays, many authors agree to introduce a new dichotomy leading to separate a requirements specification from a conceptual solution specification. For instance, Jackson (Jackson, 1993) dissociates "problem frame" from "solution", and Jarke (Jarke, 1993), in a framework, dissociates "system world" from "domain world". Having a library of components where problem descriptions are associated with conceptual solutions leads to reformulate the conceptual design process as a problem solving process. The engineer makes decisions to reuse a solution associated with a problem he has to solve.

Furthermore, conceptual design nowadays essentially consists in finding out new solutions. Nevertheless, many systems share similar properties and can be viewed as instances of application families. Identifying abstractions from a set of applications belonging to the same family brings to reusing generic structures (O'Connor, 1994) (Johnson, 1991) (Wirfs-Brock, 1990). In conceptual design, the process then becomes an activity where an engineer reuses components, i.e. specialises and integrates the predefined solutions thanks to the existence of generic conceptual model libraries.

A precondition for reusability in conceptual modelling is the existence of reusable components. There is a lack of systematic approaches for producing reusable information. As in (Arango, 1989) (Arango, 1991) (Prieto-Diaz, 1990) (Simos, 1990) (Mili, 1995), our belief is that **domain engineering** will facilitate the identification and capture of generic structures for families of applications.

In this paper, we propose a domain engineering approach for defining domain-specific knowledge components. Similar approaches appear in (Reubenstein, 1991) (Arango, 1985) (Wartik, 1992). The approach provides concepts to formalise domain knowledge and mechanisms to organise this knowledge and make it reusable. Domain reusable components called **semantic units and semantic connectors** are organised in a generic model based on the inheritance and genericity principles (Figure 1).

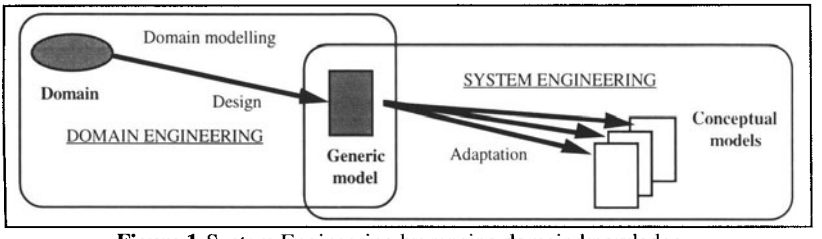

Figure 1 System Engineering by reusing domain knowledge

Distinguishing between semantic units and connectors allows to build complex generic models. As other authors (Garlan, 1993) (Allen, 1994), we claim that both semantic units and connectors express domain knowledge; the connectors have to be made explicit in the generic models for the sake of reuse. In most existing approaches, among which the object-oriented approach, links between components are defined only implicitly; they are typically encoded in the components as the services that can be called by other components. By including domain connectors in a generic model, we achieve a structure represented as a graph of semantic units and connectors applicable to the design of different systems.

The paper is structured as follows. The next section introduces the principles of our approach. Section 3 details the concepts of semantic units and connectors we propose for domain analysis. Constructing and organising generic models is dealt with section 4. We end with some concluding remarks and draw lines for future work.

2 OUR APPROACH TO DOMAIN ENGINEERING

Our goal is first to develop a domain analysis approach for capturing domain components and second to define mechanisms to organize and make these components reusable within a generic model. A generic model is related to an application domain. It aims at being reusable for the construction of specific systems belonging to the same application domain.
The proposed approach is based upon two main concepts, namely the **semantic unit** and the **semantic connector**. Semantic units are domain components and connectors describe relationships between these components. Both express domain knowledge and are reusable.

The semantic units and connectors both encapsulate domain knowledge in the descriptor part and system knowledge in the realisation part (see figure 2). The descriptor represents a domain problem - i.e. a requirement - whereas the realisation provides a solution to this problem. The descriptor is expressed using the concepts of the domain ; the realisation is a generation procedure translating the descriptor into a conceptual solution by using traditional formalisms such as OMT, OOA, etc..

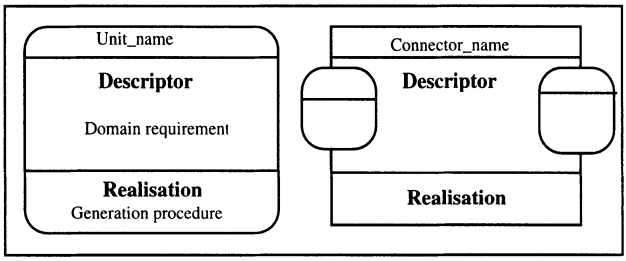

Figure 2 Semantic unit and connector

As shown in figure 3, we consider the domain engineering process as composed of two steps, namely domain analysis and generic model engineering.

- **Domain analysis** concerns domain knowledge acquisition and modelling. This activity is based on modelling principles using adequate concepts for domain knowledge representation. The resulting domain model is a set of semantic units descriptors and semantic connector descriptors.
- **Generic model engineering** is centered on the organisation of the set of semantic units and connectors. This activity is based, on the one hand, on principles and concepts borrowed from the object-oriented approach (e.g. inheritance and genericity) with the aim of building a generic architecture, on the other hand, on the use of modelling formalisms such as OMT (Rumbaugh, 1991), OOA (Coad, 1990), etc. to carry out the conceptual solution directly reusable when building a particular system. The generic model is composed of a set of semantic units and connectors organised in hierarchical networks using inheritance and composition links.

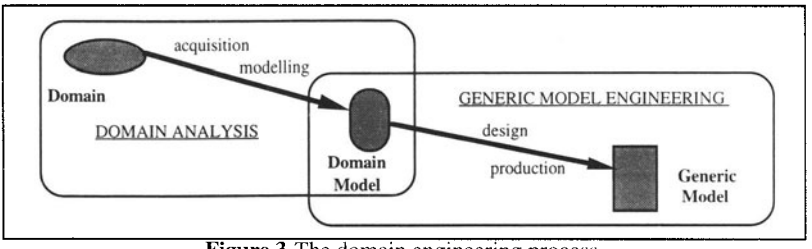

Figure 3 The domain engineering process

In the two following sections, we focus respectively on domain analysis and generic model engineering principles. Because generic model engineering principles are based on well-known object oriented concepts, we limit the second part.

2 .1 The domain analysis principles

Domain analysis is based upon designing the descriptors of semantic units as well as connectors. Two principles characterise this activity : the separation principle and the discrimination principle. These principles are considered as essential for reuse-based domain engineering. Furthermore, they allow to evaluate differences between domain modelling and system modelling.

The separation principle

The domain is perceived as a set of problems for which either known solutions exist (i.e. are implemented in existing systems) or solutions to be defined (they will be implemented in future systems). As this strategy for domain modelling is problem-oriented, domain analysis consists in eliciting and acquiring requirements and then in formalising them independently from a solution.

A domain model describes the system requirements in terms of goals and domain rules. The separation principle advocates a clear distinction between domain modelling and system modelling. In other words, a domain model expresses requirements in terms of goals and rules of the domain whereas a system model describes the architecture and behavioural rules of a particular system in term of objects and control structures.

For example, in a hotel reservation system, a requirement could be to normalise room prices. A solution could be to classify rooms by category and for each room category to determine a minimum and a maximum price. The corresponding conceptual specification, using for instance OMT, could be composed of two object classes : "Room-Category" and "Room" and an aggregation relationship between them. Other solutions to this requirement could be considered (e.g. to calculate a room price according to the category of the hotel and the surface of the room or according to the cost price, etc.). The benefit of separating problems from solutions is to increase the potential for reuse. Indeed, when a problem has been identified, one of its solutions may be reused. For instance, the principle of price normalisation can be as well applied to hotel rooms as to car sells.

The discrimination principle

The main difficulty in domain analysis is to find out similar and discriminate properties between applications of a domain. A domain model does not only have to represent what is common between the different systems of a domain, but also to represent the differences bet-

ween systems. Classical approaches for reuse emphasise the expression of similarities between systems. We claim that being able to express differences with accuracy is a major issue.

As an illustration to this principle, let us consider two conceptual specifications describing two different libraries A and B. In library A, subscribers have to pay a fee and the payment has to be managed by the system according to specific rules whereas in library B, the system is only asked to record subscribers; there is no fee management. We consider that these two possibilities in subscriber management should be visible in the generic model to those intending to reuse knowledge in the library management domain.

2.2 Principles of generic model engineering

Domain analysis is the process of acquisition and modelling of domain knowledge. We believe that building generic models is made easier when founded on the two principles previously introduced.

The aim of generic model engineering is to build components (named semantic units and connectors) that will effectively be reused when modelling specific systems of a domain. This activity precisely consists in organising components and in realising them.

Quality criteria guiding choices to organise components are reusability, flexibility and extendibility. The object-oriented approach is nowadays broadly recognised in the achievement of such properties. We have adapted it to the semantic units and connectors context. Furthermore, the realisation of reusable components consists in defining a transformation procedure of descriptor (domain requirements) in to a solution described in any formalism. In this paper, we focus our study of generic model engineering on the organisation of the generic model. We consider that the realisation of the components is not the most original part of our work.

3 THE DOMAIN ANALYSIS CONCEPTS

The definition of domain analysis concepts is guided, on the one hand, by a specific vision of the domain and, on the other hand, by the concern of being able to produce reusable domain components. At the top level, we propose two main concepts, namely the semantic unit and semantic connector. In regard to domain analysis, these concepts impose to view a domain as a set of "problem chunks" and a set of relationships between these chunks. This section details and illustrates those concepts while section 4 presents them according to reusability.

3.1 Domain view

The separation principle leads to consider a domain as a set of requirements for which solutions are provided (Wartik, 1992). In our approach, a domain requirement is associated with a set of semantic units and connectors. The requirement specification and its solution are respectively described in descriptors and realisations. The major difficulty lies in the specification of the requirement and therefore in the definition of unit and connector descriptors. Our approach of domain modelling is based on two assumptions :

- domain requirements are matters of interest which can be expressed through constraints and behavioural rules on domain objects. This assumption suggests to represent a domain model with a set of domain objects, constraints and rules. These constraints and these rules are in fact mainly concerned with business. For instance, a subscriber in a library cannot

borrow more than three books and becomes excluded if he does not bring back his books on time,

- furthermore, in the domain, there exist flows which involve and affect the behaviour of domain objects. These flows express domain laws and are achieved through the evolution of domain objects. This second assumption suggests to represent the domain flows and their consequences on domain objects. Thus, in a library, lending a copy of a book is seen as a flow from the library to the subscriber.

According to these two assumptions, the domain view emphasises the behavioural aspects across, domain objects and their evolution rules, and across domain flows and laws imposed on the domain activity. This vision of a domain is motivated by the fact that the actors usually express requirements in terms of behaviour. We also believe that the domain object behavioural rules and the domain activity laws are the main discriminators between systems of the same domain. They provide a powerful means to differentiate systems within a generic model.

The domain knowledge is expressed within the domain model by semantic units and connectors. The former allow to capture rules performed on domain objects while the latter allow to capture domain flows and laws imposed on the domain activity.

The following figure presents the meta-model describing the concepts and relationships used for the definition of semantic unit and connector descriptors.

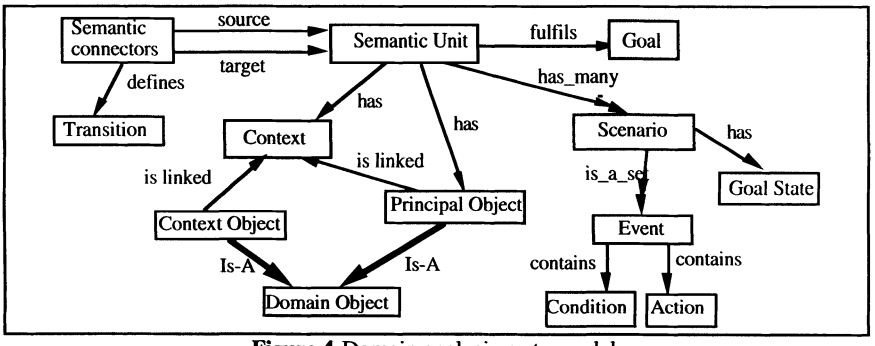

Figure 4 Domain analysis meta-model

Section 3.2 presents the different concepts describing a semantic unit whereas section 3.3 presents the concepts describing semantic connectors.

3.2 The semantic units

The concept of semantic unit assumes that any domain requirement can be modelled as a 4-tuple : <goal, context, principal_object, scenario>. A Semantic Unit (SU) captures a domain requirement through a *goal* supported by the evolution of a domain object named *principal object*. The evolution of a domain object is represented by a set of *scenarios*. The inter-dependencies between objects always lead to consider the principal_object with regard to another object. This object, along with its dependency with the principal object constitutes what is called the *context* of the semantic unit. Let us consider an example of a SU in a library system associated with the requirement related to the supervision of book return. The goal of "controlling delays on book return" is supported by keeping track of the evolution of the domain object "Subscriber" who plays a principal object role in this semantic unit. The object

"Subscriber" is considered with regard to the object "Book" borrowed by the subscriber. The context of the SU is composed of the object "Book" and a relationship "borrow". The scenario is a state graph which describes the possible evolution of the subscriber through the states "valid, reminded, penalised".

Domain objects and contexts

A domain object is an abstract representation of a phenomenon and its evolution is a concern for the domain. For instance, a customer is represented as a domain object whose domain actors focus on the frequency of his orders and his ability to pay on time. As a second example, a book in a library is also represented as a domain object being the subject of various concerns : its availability, the number of times it is borrowed, etc..

In the requirements, the domain objects support goals by applying constraints and evolution rules. The complementarity of domain objects leads to consider the evolution of an object with regard to another object and is established through a binary relationship between those two objects. A SU focuses on the evolution of one domain object, called the principal object of this SU, in a certain context i.e. with regard to a relationship with another object.

Figure 5 provides three examples of semantic units allowing to capture the evolution of the domain object "Subscriber" (principal object for the SU) in three different contexts:

- the first SU allows to consider the subscriber as being a full member of the library. The existence of this SU is related to the goal of precisely keeping track of subscribers who have to pay an annual fee.
- the second SU considers the subscriber in a different context where a subscriber is able to borrow books. The existence of such a SU results from the need to control the validity of subscribers towards the loan system.
- the third SU concerns the subscriber in its relationship with books that he has borrowed. The goal of this SU consists in checking the book return deadlines.

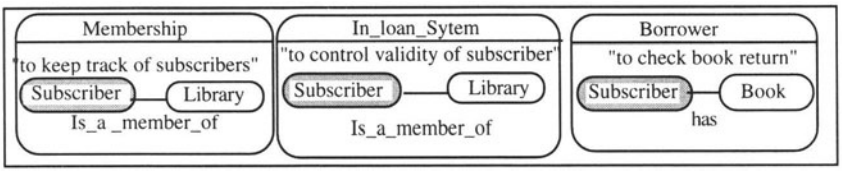

Figure 5 Examples of contexts of semantic units

We could observe in this figure that a same principal object (i.e. "Subscriber") may evolve in different contexts described through different semantic units, these contexts then permit to express evolution's perspectives of this object. The notion of context can be compared to the one of role used by the ORM model (Pernici, 1990) or contract used in (Wirfs-brocks, 1989).

Events and event scenarios

A semantic unit expresses a goal supported by the evolution of a domain object considered within a particular context. The evolution of an object follows the business rules which impose and organise the operational procedure of a domain. In a semantic unit, the evolution of the principal object, considered in a particular context, is described by a set of event scenarios. These scenarios express, at the operational level, the procedure to be followed in order to reach the goal of the semantic unit.

A scenario is a set of ordered events, which describes how the principal object evolves in its context. **Events** are facts which involve and affect the evolution of domain objects. For

example, "a payment of a fee fathers the creation of a new subscriber in the library" and "sending a reminder to a subscriber occurs when he does not return the book on time" are events. The occurrence of an event responds to some constraints expressed in the **occurrence condition** part of the event. For example, an occurrence condition could be "a subscriber must have signed an agreement form before being able to borrow a book from the library". Finally, **the action** triggered by an event conveys the state change of an object at the operational level.

Each scenario is associated with a goal state and all goal states of all scenarios for a given semantic unit are the operational expression of the goal of this semantic unit. For example, the semantic unit having the goal "to check the book return deadlines" includes four scenarios which we now detail in turn.

* The first scenario is composed of the events <loan, return> (loan and return express events).
* The second one, <loan, remind, return> describes an operational procedure for reminding a subscriber who has not yet returned the book a week after the legal delay.
* The third one, <loan, remind, remind_2, return> follows a procedure allowing to remind the subscriber for a second time if he has not yet returned the book after the first reminder has been sent.
* The fourth one, < loan, remind, remind_2, unusual_end> expresses a procedure which considers that the book will never be returned.

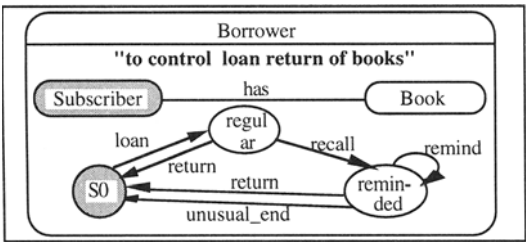

Figure 6 Examples of scenarios of a semantic unit

"So" is a particular state representing the non existence of the object in this context. This is useful if there is a need to represent all events as a state transition between a source state and a target state. In this example, "So" is the goal state of the four scenarios presented above.

3. 3 The semantic connector

Similarly to semantic units, semantic connectors have a specification and are reusable components of the domain. They play an essential role for domain modelling, because they convey the general laws of the activity of a domain. A semantic connector is defined as a couple <flow, transition>. It expresses **a flow** between two semantic units called : semantic unit source and semantic unit target. A flow involves and affects the evolution of domain objects through **a transition.**

As an illustration of the concept of semantic connector, we consider two semantic units: the first one considers the book as a resource for the loan system and the second one deals with books borrowed by subscribers. A semantic connector between these two semantic units allows to define that the books of a library could be transferred from a context - where they behave as a resource for the loan system - into a new context where they behave as a resource allocated to a subscriber.

The domain flows

From the point of view of the domain knowledge, a domain flow implements the business rules of the domain. Its follow-up is a matter of concern. For instance, in a library, a loan request will be managed until its achievement and as a second example, a book loan will be considered from the time the book is borrowed until it is returned.

In the domain model, the semantic connectors describe flows between two semantics units. We have recognized three types of flows :

- Flow of context change : it expresses an evolution change of the principal object from one context to another one,
- Flow of context dependency : it expresses an interaction between two evolutions, each one being represented by a semantic unit,
- Flow of context use : it expresses a transmission of information between two evolutions.

Figure 7 depicts two semantic connectors :

- the first one describes a change of context in the evolution of a book. The book goes from the context "resource" of the loan system into the context "book_borrowed".
- the second one expresses a dependency between two context evolutions of the same object. A subscriber, reminded for a late fee payment, is suspended from the loan system.

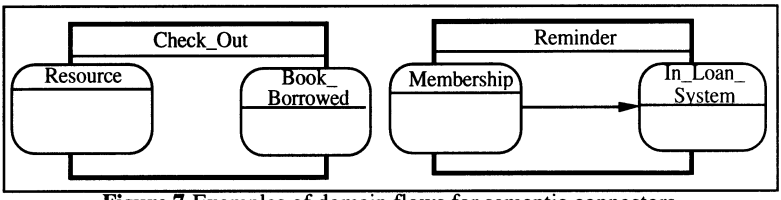

Figure 7 Examples of domain flows for semantic connectors

The transitions

The realisation of a domain flow is a set of elementary transitions rendering the flow at an operational level. An elementary transition is a relationship between a component (a state or an event) of the scenario of the source SU and a component (a state or an event) of the target SU. Figure 8 describes transitions occurring in the two semantic connectors presented in figure 7.

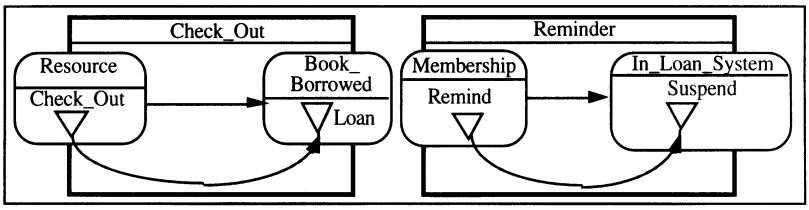

Figure 8 Examples of transitions for semantic connectors

4 SEMANTIC UNITS AND CONNECTORS AS DOMAIN REUSABLE COMPONENTS

In this section, we consider semantic units and connectors as domain reusable components. They are the two basic building blocks to specify a generic model.

First, we emphasise two characteristics of our approach that make semantic units and connectors reusable. Second, we introduce two powerful mechanisms to increase reusability of components. In the following, the term component is used to refer both to semantic units and semantic connectors.

4.1 Semantic components as reuse-oriented basic buildings blocks

We claim that semantic components as previously defined provide basic building blocks well-fitted to a reuse-oriented domain analysis approach. Due to their granularity, semantic units and connectors constitute self-contained components and because connectors are recognized as reusable components, they also allow reuse of large structures.

Granularity of semantic components

According to their definition, unit and connector granularity is based on semantic criteria making them self-contained chunks of domain knowledge. A semantic unit captures a pattern dealing with domain object behaviour considered within a particular context. As a domain-centered concept, the context suggests to describe the behaviour of a domain object according to another domain object. Such an approach leads to specify object behaviour through several semantic units, each one capturing a semantic viewpoint of an object behaviour. In this way, by reusing semantic units we only reuse that viewpoint of interest for a specific application. For example, a reuser can be interested in choosing a semantic unit capturing all subscription management rules of a library, whereas another one needs a semantic unit only including the check-in and check-out of the subscribers.

Semantic connectors prescribe how semantic units can be connected together. They capture an interaction pattern between two semantic units. By reusing connectors, we compose components according to the laws drawn from the domain. In this way, a reuser can be concerned with the connectors "request" and "loan" because the library requires that a subscriber fills a request before loaning a book, whereas another one only needs the "loan" connector because in another library, subscribers directly take the book they want to loan.

Distinction between semantic units and connectors

A generic model is a collection of semantic units and semantic connectors. The first ones could be viewed as components and the second ones as links between components. This separation between components and links makes components very autonomous. Furthermore, by composing semantic units through semantic connectors it is possible to develop and reuse much richer structures than we are able to do with an approach including connectors within components. In particular a generic model makes explicit and reusable the whole generic structure of the specific systems within a same domain.

By including domain connectors within the generic model, we facilitate the composition process by prescribing during reuse the ways in which components can be connected together. While these components constitute basic reuse-oriented building blocks, they are not sufficient to achieve very large applicability of a generic model. Indeed, semantic units and semantic connectors must be related to mechanisms increasing the domain model genericity and its ability to capture the common properties as well as the discriminant ones for a set of applications of the same domain (discrimination principle).

4.2 abstract semantic components

The notion of abstract component is applied both to semantic units and semantic connectors. Generally speaking, an abstract component allows to differ the details of some properties of its descriptor in sub-components. The sub-components are used to emphasise discriminant aspects of a property specified at the abstract level. However, abstract component refers to different meanings according to whether we consider semantic units or semantic connectors.

Abstract semantic unit

Applied to the semantic unit concept, an abstract component is similar to an abstract class in the object-oriented approach (Meyer, 1988). Related to the inheritance link between semantic units, an abstract semantic unit allows to differ the detail of some of its properties (goal, context or scenario) in concrete or abstract sub-units.

In Figure 9, the abstract semantic unit "Payment_Of_Fee" (greyed box) describes the behaviour of a subscriber who has to pay an adhesion every year. This "Adherent" unit has a scenario which is differed in two semantic sub-units:

- The one located on the left of the figure expresses that if a subscriber does not pay his annual fee, he is reminded twice and then his subscription is cancelled.
- The other one located on the right expresses that the subscription is cancelled if after a delay of one month, the subscriber has not paid his subscription.

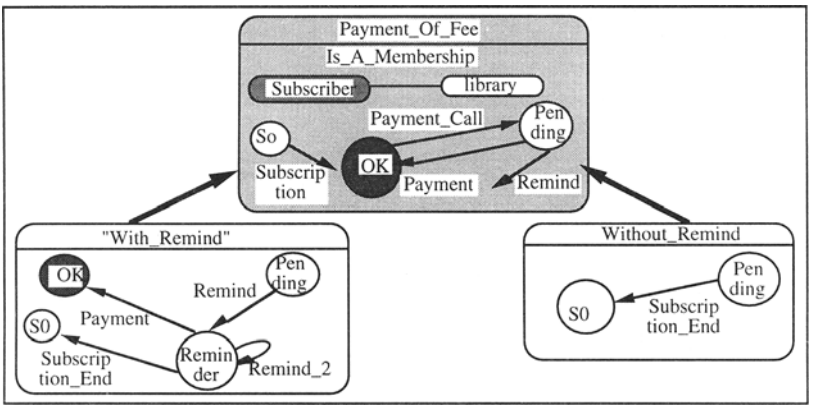

Figure 9 Abstract semantic unit example

The abstract semantic unit and its two semantic sub-units capture at the abstract level common subscription management rules whereas at the lower level the two sub-scenarios show different rules to manage subscribers who do not pay their subscription within the required time. In the illustration below, the sub-units inherit properties from the super-unit.

Abstract semantic connector

Applied to semantic connectors, an abstract component specifies a domain flow which is detailed in several sub-connectors. Related to a composition link defined between semantic connectors, an abstract semantic connector allows to differ the path between the source semantic unit and the sink semantic unit in concrete or abstract sub-connectors.

In Figure 10, the abstract semantic connector "borrow" (greyed box) specifies a context change for a subscriber. The two sub-connectors present two possible paths from the "membership" source context to the "borrower" sink context.

- The first one (in the upper part) means that subscribers can have immediate access to books, and they can borrow a book which is available in the library,
- The second one (in the lower part) means that subscribers have to formulate a request before loaning a book.

The abstract semantic connector and its two semantic sub-connectors capture at the abstract level a domain context change flow whereas at the lower level the two sub-connectors show different paths in becoming a borrower.

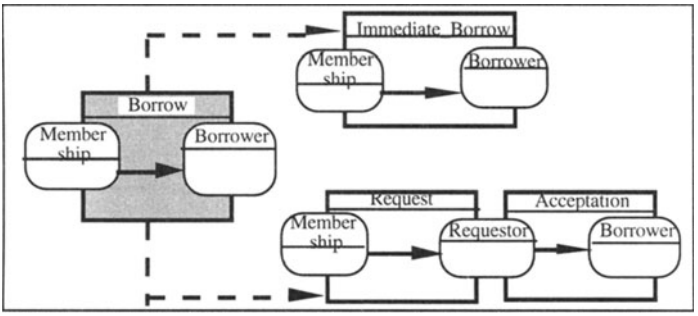

Figure 10 Abstract semantic connector example

4. 3 Generic semantic components

Genericity is a mechanism used in software engineering to customise software modules (Meyer, 88). Applied to semantic units and connectors, genericity allows to generalise domain knowledge. The generic component (or meta-component) describes a class of domain problems. The genericity of semantic components allows a broader reusability of semantic units and connectors. A generic component may be reused in the building of several generic models. Such a component is used by the designer as a structure to be instantiated with domain knowledge.

Generic semantic unit

Applied to semantic units, genericity allows to express a domain object behaviour meta-pattern. A generic semantic unit contains parameters which will have to be instantiated with domain knowledge when effectively used. Figure 12 presents the generic semantic unit "contract" (with the grey border line). Using this unit consists in valuating the parameters (named between [ ]). Three instances are given. They deal with "Subscribers" in contract with a "Library", "Employee" in contract with an "Employer", etc.. This abstraction mechanism allows to reuse all business rules of a "contract" (renewal_request, cancellation, etc.) specified in the scenarios of the generic semantic unit.

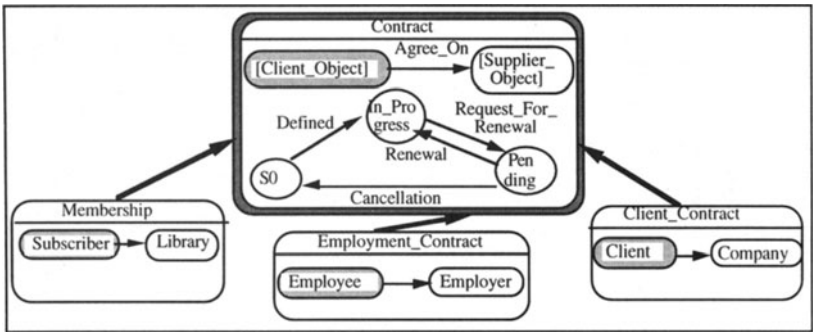

Figure 12 A generic semantic unit

We have used the hierarchical structure of domain abstractions proposed in (Maiden, 1992) and (Maiden, 1993) for identifying generic semantic units. Many generic semantic units have already been identified, among which we can quote :

- The generic semantic unit "Container" describes the evolution of a collection of objects; this unit can be instantiated for a "hotel" (a hotel contains a set of rooms) or for a "centralised room booking agency" (an agency works with a set of hotels).
- The generic semantic unit "Resource" describes the evolution of an object which can become available or allocated. One instanciation may concern "books" in libraries or "rooms" in a hotel or some "shared printers" in a local network of computers.

An important benefit of the concept of generic semantic unit is the extendability, i.e. it is possible to build new generic semantic units in order to resolve new problems.

Generic semantic connector

Applied to semantic connectors, genericity allows to generalize the dependency laws between behaviours. A generic connector can be used for example to generalise a dependency which exists between an object evolution in the context <client, resource> and the same object in the context <client, supplier> . These contexts are respectively represented by the semantic unit "consumer" and "contract". The dependency illustrated by the generic semantic connector "consume" corresponds to the generic situation where a client can only consume if its contract is in progress. Two instanciations are given below (see figure 13):

- In a library where subscribers have to periodically pay a fee, there are the dependency constraints between the states of adhesion contract ("in_progress", "pending") and the fact that the library allows subscribers to borrow books.
- An insurance company refunds victims of injuries according to contract clauses.

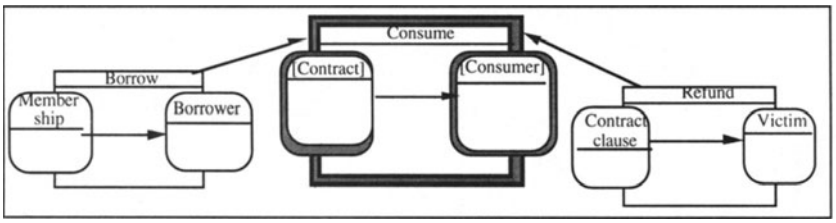

Figure 13 A generic semantic connector

From a reuse point of view, the genericity is a powerful mechanism allowing to build semantic units and connectors for different domains.

5 CONCLUSION

This paper presents a domain engineering approach to build generic models. The domain modelling approach we propose is based upon two knowledge basic building blocks : semantic units and semantic connectors. We argue that semantic units and semantic connectors capture domain knowledge; both require specification in a generic model, both can be reused. By including semantic units as well as semantic connectors in a generic model we achieve much richer domain models. Furthermore, such domain models provide the reuser a discipline to compose components in developing a specific system.

Future works to be investigated concern the refinement of the set of concepts supporting both domain analysis and generic model organisation. The problem-oriented view of the domain leads to propose a detailed description of the goal part of the semantic unit. Futhermore, we believe that the well-defined set of domain analysis concepts provides a powerful means to hierarchically organise the semantic components. Some domain modelling constructs (such as goal and context) concentrate on problem-level aspects whereas others are closer to solution aspects (scenario and transition). Such a domain-oriented organisation is typically appropriate to conceptual modelling where initial situations are characterized by an informal definition which is subsequently refined into a more formal description.

REFERENCES

Allen, R. and Garlan, D. (1994) Formalizing Architectural connection, *in Proc. 16th International Conference on Software Engineering*

Arango, G. and Freeman, P. (1985) Modeling knowledge for software development, *in Proc. 3rd Int. Workshop Software Specification and Design,* Washington, DC:IEEE Computer Society Press

Arango, G. (1989) Domain Analysis -From Art Form to Engineering Discipline *in Proc.of the 5th Int.Workshop on Software Specifications and Design*, , pp 152-159.

Arango, G.and Prieto-Diaz, R. (1991) Domain Analysis Concepts and Research Directions *in Domain analysis and Software systems Modeling*, ed. R. Prieto-Diaz and G. Arango,

Bellinzona, R., Fugini, M.G. and de Mey, V. (1993) Reuse of Specifications and Designs in a Development Information System, *Proc of the IFIP Working Group 8.1 working conf. on Information System Development Process (A-30),* Como, Italy

Booch G. (1987) Software components with Ada: structures, tools and subsystems, Benjamin/ Cummings

Castano, S., de Antonellis, V., Francalanci C., and Pernici, B. (1994) A Reusability-Based Comparison of Requirements Specification Methodologies, *Proc. of the IFIP WG 8.1 Conf. CRIS 94,* Maastricht, The Netherlands

Coad, P. and Yourdon E. (1990) Object-Oriented Analysis, Yourdon Press, Prentice Hall, Englewood Cliffs, NJ

Deutsch, L.P. (1989) Design Reuse and Frameworks in the smalltalk-80 System, *in Frontier Series: Software Reusability: Volume II: Applications and Experience.* Biggerstaff T.J., and Perlis A.J., Eds. ACM Press, New York, pp. 57-71

Garlan, D. (1993) Domain specifications Require First Class Connectors, *in Proc. of the IEEE Int. Symposium on Requirements Engineering*, San Diego

Hall, P.H.V. (1992) Overview of Reverse Engineering and Reuse Research, *Information and Software Technology,* Vol 34, No 4

Iscoe, N. (1991) Domain modelling: Evolving Research, *Proc. of the 6th Knowledge-based Software Engineering Conference*, Syracuse NY

Jackson, M. and Zave, P.(1993) Domain Descriptions, *in Proc. of the IEEE Int. Symposium on Requirements Engineering*, San Diego,

Jarke, M. and Pohl, K. (1993) Establishing visions in context: towards a model of requirements processes, *Proc of the 12th Intl. Conf. Information Systems*, Orlando,

Johnson, R. and Wirfs-Brock, R. (1991) Object-Oriented Frameworks, Tutorial Notes. *In Proc.of ACM OOPSLA*

Krueger, C.W. (1992) Software Reuse, *ACM Computing Surveys,* Vol. 24, No 2

Maiden, N. (1992) Generic Domain Models *in Software Engineering, AAAI'92*, System Design Workshop

Maiden, N. and Sutcliffe A. (1993) The Domain Theory: Object System Definition *Nature Report CU-93-OOA*

McIlory, M. Mass-Produced Software Components *in Software Engineering Concepts and Techniques, 1968 NATO Conf. Software Engineering*, J.M. Buxton et al.(eds), 1976.

Meyer, B. (1988) Object-Oriented Software Construction, Pub. Prentice-Hall Int. Ltd, Hernel Hempstead

Mili, H , Mili, F. and Mili A. (1995) Reusing Software: Issues and Research Directions, *IEEE Transactions on Software Engineering*, Vol 21, No 6

O'Connor, J., Mansour, C., Turner-Harris, J. and Campbell, G. H. (1994) Reuse in Command-and-Control Systems, *in IEEE Software*

Pernici, B. (1990) Objects with Roles, ACM/IEEE *Conference on Office Information Systems*, Boston, MA

Pree, W. (1994) Design Patterns for Object-Oriented Software Development, Addison-Wesley

Prieto-Diaz, R. (1990) Domain Analysis: An Introduction, ACM SIGSOFT, *Software Eng. Notes*, Vol 15, No 2

Reubeinstein, H.B. and Waters, R.C. (1991) The Requirements Apprentice: Automated Assistance for Requirements Acquisition, IEEE *TSE* 17(3), 226-240

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991) Object-Oriented Modeling and Desing, Prentice-Hall, Englewood Cliffs, NJ

Simos, M.A. (1990) The domain-Oriented life cycle: Towards an extended process model for reusability software reuse: emerging technology, ed. Will Tracz, IEEE CS Press,

Wartik, S., and Prieto-Diaz, R. (1992) Criteria for Comparing Reuse-Oriented Domain Analysis Approaches, *Int. Journal of SEKE*, 2(3)

Wirfs-brock, R. and Wilkerson, B.(1989) Object-Oriented Design: A Responsability-Driven Approac, *In Proc. of OOPSLA'89 Conference*, SIGPLAN Not. (ACM) 24, 10

Wirfs-brock, R. and Johnson, R. (1990) Surveying Current Research in Object-Oriented Design, *Communications of the ACM*, Vol 33, No 9

7 BIOGRAPHY

Corine Cauvet holds a Thesis in Computer Science from University of PARIS VI P. and M. CURIE. She is currently a lecturer of Computer Science in the Department of Mathematics and Informatics at the University of PARIS I Panthéon/Sorbonne. Her research interests lie in the areas of information modelling, object-oriented analysis and design, analysis and design methodologies, CASE tools and reuse in information systems design. She has published a number of papers in these areas. Her research work has been supported by fundings of the MRT (Ministry of Research and Technology) and the ANVAR (French Association for Research Development). She has participated, in co-operation with the industry, in the development of a CASE tool which is now commercialized. She has participated to the ESPRIT project "BUSINESS CLASS" and also has played an active role in the definition of an object-oriented model for analysis and modelling of business applications in a collaborative project between University of Paris I and Alcatel ISR Co.

Farida Semmak received the DEA degree in computer science from Dauphine University. She has prepared her thesis at Paris university and her research works include domain analysis and reuse at RE level.