Submission to the Isac93 conference; rejected by the program commitee

Using Software Engineering Techniques to Build Complex Knowledge Based Systems

Nicolas Revault*, François Pachet, Brigitte Kerhervé****

*LAFORIA
Boîte 169, Tour 46, 2ème ét.
Université Paris VI
4 Place Jussieu
75252 PARIS Cédex 05, France
e-mail : revault@laforia.ibp.fr

**Université du Québec à Montréal
Département de Mathématiques et d'Informatique, LARC
Case Postale 8888, Succursale A
Montréal (PQ) H3C 3P8 Canada
e-mail : pachet@larc.uqam.ca ; kerherve@info.uqam.ca

Area :

Software Engineering

Contact Author

Pr. Brigitte KERHERVE
Département de Mathématiques et d'Informatique
Case Postale 8888, Succursale A
Montréal (PQ) H3C 3P8 Canada
e-mail : kerherve@info.uqam.ca
tel : 1-514-987-6716
fax : 1-514-987-8477

Submission to the Isac93 conference; rejected by the program commitee

Using Software Engineering Techniques to Build Complex Knowledge Based Systems

Abstract

This paper studies the properties of complex systems that are built by reusing an existing object-oriented software in the framework of a knowledge-based system. More precisely, the problem can be stated as : if S is a piece of procedural software that yields some "good" software engineering properties P (such as extensibility and modularity) and we wish to add a knowledge-based component to it (KB(S)), then what are the conditions for the resulting system (the target, S + KB(S)) to retain the properties P of the original software S ? We give a sufficient condition for this integration to be successful, that is based on the uniformity of the language used to express the knowledge with the object language. We exemplify our claim by an example of a complex application that performs the transformation of semantic networks into normal form relations.

Key Words

Reusable knowledge bases, object-oriented rule-based programming, software reuse, pluggability, semantic networks, database design, normalization, meta-modelling, inference mechanism, Smalltalk.

1. Introduction

This paper addresses the issue of building complex *hybrid* systems. Hybrid here means the integration of (1) a procedural system and (2) a knowledge-based component. More precisely, we are interested in the software engineering techniques that should be used for building such systems, and the characteristics of the resulting (target) systems with regards to reusability and extensibility.

If the building of large software is now given a lot of attention, the same cannot be said for complex knowledge-based systems. Except for a few very ambitious projects (such as the Cyc project [Lenat & Guha 90]) this problem is still a research issue because no efficient technique has yet been devised to simplify the reuse of existing knowledge bases. We consider that one promising and more realistic way of building complex knowledge-based systems is to consider procedural systems as *knowledge repositories* (although represented in an awkward fashion) and use software engineering techniques to allow reusing this knowledge and its integration with standard knowledge representation paradigms. Our problem is therefore to study which particular techniques can be used to produce software that supports the integration with knowledge-based mechanisms. The main characteristics we need for such software are of course *extensibility* and *reusability*, but hese notions are inherently fuzzy because they suggest the idea of integrating something "not expected" from the original conceptors. Within the category of "non-expected ordeals" a sofware can endure, we are particularily interested in the ability of a piece of software to support the integration of an inferential component.

The main mechanisms proposed so far to give software good properties (i.e. achieve reusability and modularity) are the mechanisms of object-orientation, namely inheritance, instanciation, encapsulation, as discussed in [Meyer 90]. Those principles yields some essential characteristics for object-oriented programs. Encapsulation is the principle that states that object's structures are not visible from outside (the so-called client-classes), and interaction between objects is achieved only by using the object's interface, i.e. the set of methods that are defined in the object's class and superclasses. In this paper we will focuss on the following characteristics of object-oriented programming : (1) encapsulation, (2) programming by simulation (objets are represented but relations between objects are not), (3) hard-wired control (backward chaining).

On the other hand, AI systems traditionnaly focuss on explicitely representing the structure of the data used for knowledge representation, and avoid, as much as possible, data hiding or procedure opaqueness. In rule-based programming, this is embodied in the notion of *fact*, the basic unit of representation, which represent some property of the world being modelled [Brownston 85]. The characteristics of rule-based programming are therefore exactly opposite to that of object-oriented programming : (1) data explicitation (or should we say "decapsulation"), (2)

programming by relations ("facts" or relations bewteen objects are represented but objects per se are not) and (3) explicit control (forward-chaining).

The traditional ways of integrating these two paradigms (objects and rules) is to establish a correspondance between a fact and an object's attribute. Rules are then expressed in terms of values of objects attributes (as can be seen in the CLIPS system [CLIPS 91] for instance, a typical object + rules system). If this allows to bridge the gap between the two worlds, there is a big disadvantage to doing this with regards to software engineering : explicitely refering to object's attributes simply breaks encapsulation, since the structure of objects is not hidden any more from the "outside". The resulting system therefore loses the properties of the original software.

Having seen that O.O. programming is inherently contradictory with rule-based programming, we can state the problem of integrating procedural software with a rule-based component with regards to software engineering properties : how to integrate both formalisms without loosing the mains benefits of OOP ?

The main idea of our approach for building complex systems by reusing procedural components within rule-based programming envionments is to :

a- Make full use of "pure" object-oriented techniques for producing procedural software that yields good behavior with regards to reuse, and

b - Use a rule-based language that benefits from these properties, without loosing them.

The central point here is that this can be achieved only if *the rule language is expressed in terms of the object language*, and not as usual attribute-value constraints. The purpose of the paper is to illustrate this approach with the example of a knowledge-based system for database design.

Our claims will be illustrated with an example of a working system built for experimenting with our approach. We will first (section 2) describe the functionalities of the target system (our goal), then (section 3) give a brief description of the two systems that were reused (namely Ackpro and NéOpus), and how their integration was performed. The remaining of the paper (section 4) describes more in detail the characteristics of the implemented target system with regards to its software engineering properties. Finally we conclude on the properties of the resulting system.

2. The Target Knowledge Based System

In order to experiment with our approach for the development of quality complex knowledge-based systems, we focus on an existing knowledge based system dedicated to the database design process. This system, called SECSI (Expert System on Information System Design), has been designed by a research team of the University of Paris VI. For detailed information concerning this system, the reader can refer to [Bouzeghoub 85]. We will briefly describe here the functionalities and components of this system.

The design of a database conditions its consistency as well as the performance of the applications. Thus, in the field of database systems, many research efforts have been devoted to the automation of the database design and have led to the development of specific software products. These products generally take care of three different steps : construction, transformation and generation. The construction step deals with the specification of the application and with the construction of the corresponding semantic representation model. The second step transforms the previous representation model into the data model of the target database system (e.g. hierarchical, codasyl, relational or object-oriented data models). The final step generates the data definition commands in order to create the database in the target database environment.

SECSI follows this approach and the transformation process has been defined as an expert system. This system defines a software environment for building a *database schema* from a "natural" specification given by an user. It presents a *semantic representation model*, called MORSE, that allows to represent complex data structures in the form of *semantic networks*. The transformation process in turn generates the relational data model from this semantic net. The target model is an equivalent data representation consisting of *normal form relations* , thus yielding a database schema of the *relational model*.

This system is interesting for us for two reasons. First, it is a complete knowledge based system since it includes the knowledge representation and the inference process (here a transformation process). Secondly, the expertise of this system has been clearly identified and validated by SECSI. During our work, we intensively used [Bouzeghoub 86] where we found the detailed description of both knowledge and expertise.

3. Using complex systems

As we pointed out in section 1, complex knowledge-based systems cannot be made from scratch. Developping such systems necessarily requires reusing existing software. For the development of our target database design tool, we reused two existing systems : AckPro for the representation of the data models and the building of the environment, and NéOpus for the inference process. In this section we describe the main characteritics of these two systems. Then we show that the mechanisms of object-orientation adopted in AckPro and NeOpus allow to build a complex knowledge based system that yields the same properties than the original system, without breaking consistency.

3.1. The AckPro environment

The first software that we reuse is **AckPro**, an environment which consists of a specification tool for the design and *prototyping* of computer applications. AckPro has been developed at ACKIA[1] in collaboration with members of LAFORIA [Krief 90]. This system is basically an application generator that transforms declarative specifications into a set of classes that satisfy the specifications. Through a main edito AckPro allows to define a framework composed of generic concepts of a certain domain that are linked in an entity-relationship way [Chen 76]. The specification of the concepts and their relationship define a "representational theory" of the domain modelled. This theory is called a *meta-model*. It can be compiled into an "implementational theory" of the domain in the form of Smalltalk classes. These classes are then used to create instances of concepts in a domain specific editor, - or *model editor*. A set of instances so created constitutes what is called a *model* in AckPro. The model editor is composed of both a textual and a graphical view. The characteristics of the graphical view are specified through another editor of AckPro: a *graphical meta-editor*.

The interesting aspect of Ackpro here is that it provides a *vertical* specification system, with which a data base model can be defined, starting from the most general theory (here the Entity-Relation meta-model) down to the model of the given application. Although not necessarily implemented as an instanciation mechanism, it is better seen as an instanciation tree, rather than a hierarchical tree, in which each level is an "instance" of a higher level (see Figure 1). The system supports the transformation of a specification into a set of classes that implement it as well as a set of programming tools (grahical and textual editors) to edit the application.

[1] *ACKIA, 47 rue de Bellevue, 92100 Boulogne, France*

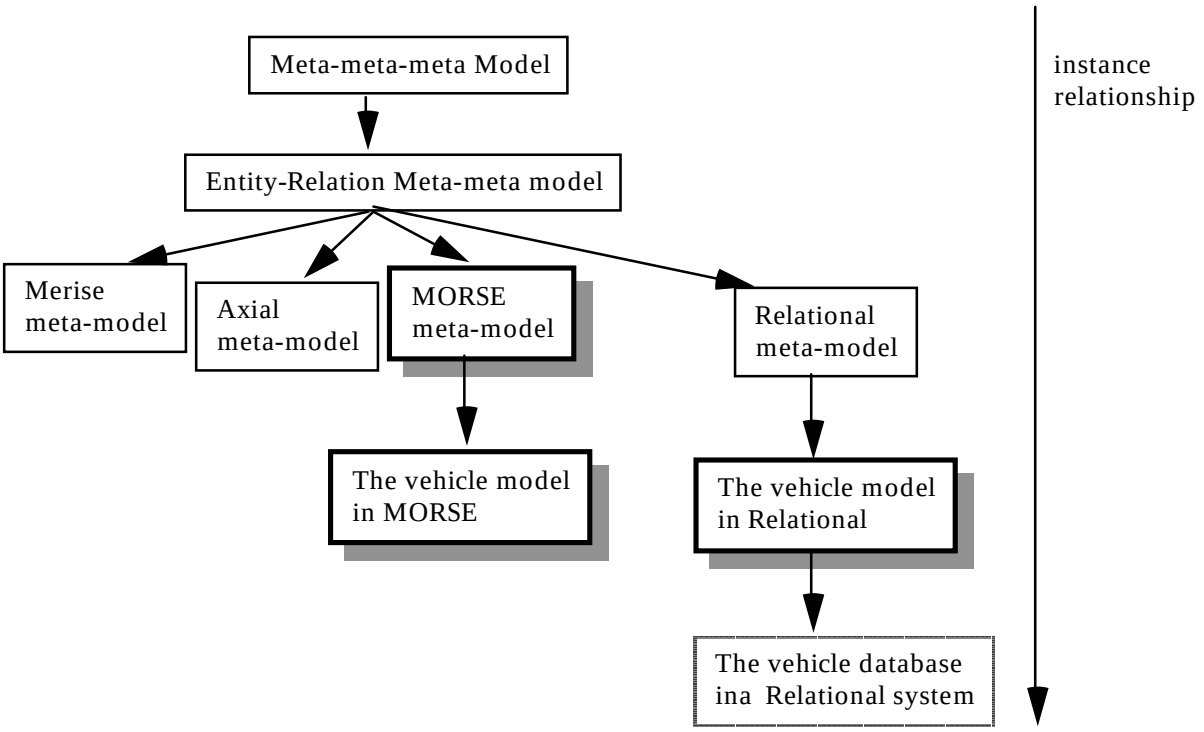

Fig 1 : the vertical instanciation hierarhy

It is therefore important to note that the system we reuse is actually the system *generated by* Ackpro, and not AckPro itself (although AckPro is in a way generated by itself, but this is another issue). By construction, an application generated by AckPro is conform to the standards of OO programming : the set of classes that composes the application respect encapsulation, defining complex messages that may be redefined by subclassing the AckPro classes, if needed.

3.2. NéOpus

The second development tool we used is the **NéOpus** system [Pachet 91, 92]. It is basically an *inference mechanism* integrated in the Smalltalk-80 environment. It combines first-order *production rules* in forward-chaining with Smalltalk objects. It is pluggable to any Smalltalk class application modelling a given domain, as our work will illustrate. It has been developped at LAFORIA and is now commercialized by HumanWare [2] .

This system integrates production rules to the "autochtonous" Smalltalk objects in a very natural way, because (1) any objects may be used in a rule, and (2) any Smalltalk expression may be used in conditions as well as action parts. In this context, the notion of *fact*, traditionally used in rule-based languages, disappears, as well as the notion of *fact-base*. The object-oriented rule-based programming, in NéOpus, is exclusively based on side-effects : the action part of a rule does not *assert* new facts, but modifies some objects by sending them messages.

A rule base written in NéOpus may be seen as an *horizontal process*, that transforms a set of objects into an other set of objects (possibly including new ones created during the reasoning process).

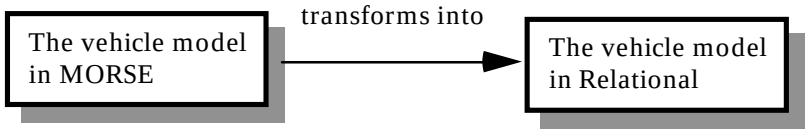

Fig 2 : the horizontal transformation process

[2] *HumanWare, 66 Avenue des Champs Elysées, 75008 France*

3.3. Integration

Integrating the two systems (the system generated by AckPro with a knowledge base written in Néopus) may therefore naturally be seen as the construction of a two-dimensional space, whose vertical axis represents the instanciation relationship between instances and classes, and the horizontal one represents the transformation of set of objects by the reasoning process.

It is to be noted here that there is no integration problem per se here, since both systems are written in the same language (Smalltalk), and that language is totally uniform, i.e. represents all entities as objects. So the problem of defining an interface between the two systems is irrelevant : the objects manipulated by NéOpus are exactly the objects generated by AckPro.

Our problem here is to study the properties of the resulting system, with regards to extensibility and reusability. Our hypothesis to build a target system that yields good properties is that the rule language be exactly the object language. We will now describe more in detail the implications of this principle.

4. The Database Design Tool Revisited

The target knowledge-based system aims at helping a database administrator who has to design the database. In section 2, we presented the approach followed by SECSI, which is based on (1) an adequate representation for knowledge and (2) an effective transformation process. We explained that this system transforms MORSE semantic nets to normal form relations. Each of these two kinds of objects (semantic nets and normal form relations) has its owns data representation. [Bouzeghoub 86] has shown that a data representation expressed in one of these two forms has an equivalent expression in the other form.

In this section we present the development of the target knowledge based system. We successively examine the three components of this system : (1) the knowledge representation, (2) the transformation process and (3) the control architecture. For each point we show how we reuse existing software (e.g AckPro and NéOpus).

Henceforth, we will indifferently use the MSN abbreviation for 'MORSE semantic nets', NFR for 'normal form relations', RM for 'relational model' and TP for 'transformation process'.

4.1. Knowledge Representation

In order to implement the two kinds of data representations, we had to study the *representation model* for each of them: MORSE for MORSE semantic nets, and the relational model for the normal form relations. As we will see, there exists a relation between these two models. Once this context is defined we will be able to describe the transformation process.

4.1.1. MORSE Concepts

MORSE [Bouzeghoub 84] is a generic semantic network that allows to describe a real world data representation. Three concepts are present in MORSE : objects (nodes of the network), links between nodes and constraints both on objects and links. More precisely, four categories of objects can be distinguished : *atomic objects* representing "leaf nodes" of networks; *data-types* specifying a domain for atomic objects; *molecular objects* aggregating atomic objects and representing "non-leaf nodes" of networks; and *roles*, qualifying molecular objects.

There is a number of links that objects can entertain with each other in MORSE, such as : the notion of *atomic aggregation*, that allows to constitute a molecular object from atomic objects; there is a link of *molecular aggregation*, for building a molecular object from other molecular objects; there is a *generalization* link that gives the possibility of refining a specification of a molecular object by an inheritance protocol of atomic objects; there is also a *domain* link between atomic objects and data-types; finally, there is a link of *functional dependency* between atomic objects.

In addition, MORSE gives the possibility of expressing some constraints on the objects or on the links of the network. First, the links of atomic aggregation and of molecular aggregation are valued by cardinality constraints, like in the entity-relationship model [Chen 76]. Concerning objects, the main constraint is one about atomic objects. An atomic object - or a combination of some atomic objects - can be considered as a *primary key* of the molecular object it contributes to compose.

4.1.2. An example

The next figure shows an example of a graphic representation for a MSN, generated by AckPro. To limit the complexity of the example, we did not represent the data-types of atomic objects, and of course, neither the domain

links. Rectangle representations represent atomic objects and ellipses molecular objects. The labels on the links show their types: 'p' stands for atomic aggregations, 'o' for molecular aggregations, 'g' for generalization links, and 'df' for functional dependency links.

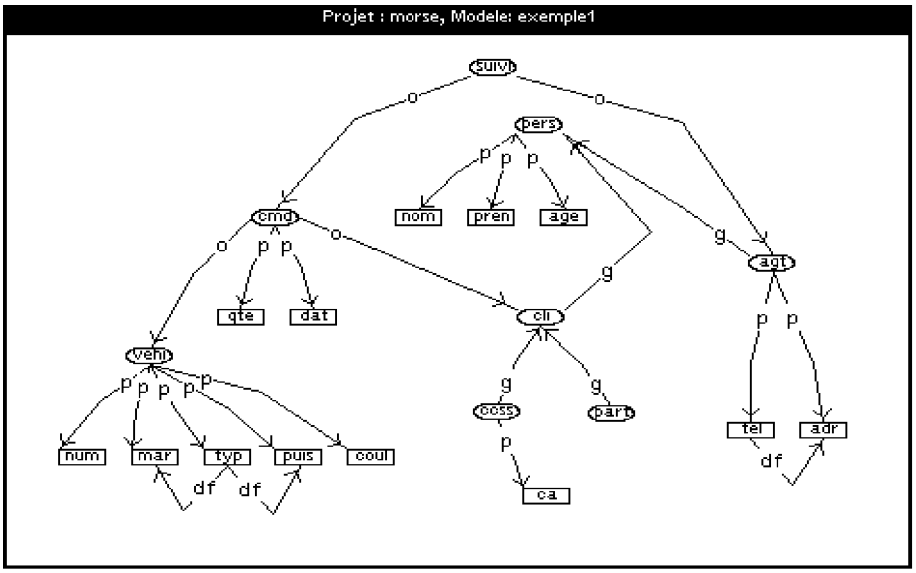

Figure 3: an example of MSN

This example models a simple data structure for a vehicle supplier. CLIENTS (cli) and AGENTS (agt) are two kinds of PERSONS (pers) to be represented. An ORDER (ord) is binding VEHICLES (vehi) to clients and a FOLLOW UP (fol) is binding agents to orders. An agent is characterized by, say, his localization, composed of a TELEPHONE NUMBER (tel) and an ADDRESS (adr) in functional dependency. Concerning the constraints, we can say that the notion of VEHICLE is aggregating the notion of NUMBER in a one to one link. That means that a VEHICLE has exactly one NUMBER, and that a NUMBER is the one of exactly one VEHICLE.

4.1.2. The Relational Model

The relational model (RM) is due to [Codd 70]. It was introduced to palliate the inadequacies of the old models of non inferential, formatted data systems. It consists of a "model based on n-ary relations, where a normal form for data base relations, and a concept of a universal data sub-language are introduced". According to Codd, a database schema is in a normal form if each of its composing relations is "defined on simple domains, i.e. domains whose elements are atomic (non-decomposable) values".

In the RM terminology, NFR are constituted with named attributes, each being of certain domain. This is indeed the theoric specification of the physical implementation of a database: tables, composed of fields, each of a certain data-type.

4.1.3. Using Ackpro

The concepts defined by MORSE consist of nodes and links as in semantic net frameworks. Our solution consists in defining an object oriented - Smalltalk-80 - implementation of these concepts, on which a set of production rules representing the transformation process will be aplied. To get a computer representation of the MORSE semantic nets, we used AckPro, in which the meta-model is MORSE itself.

The next figure shows the simple meta-model we have built for MORSE in AckPro. We note on this figure that all the MORSE concepts we have presented in the section 4.1.1 are represented as entities. We can see that there are cardinality constraints on each relation that consist of a description of a certain MORSE type of link.

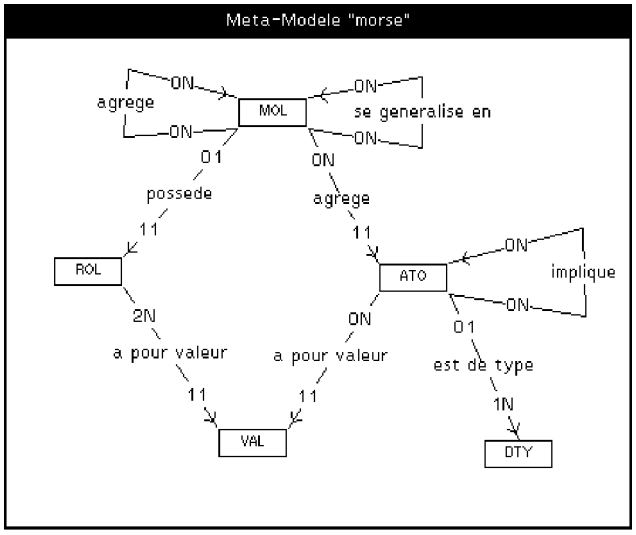

Figure 4.a: graphical view

Meta-Modele "morse"

Meta-Individus	Meta-Relations M-I
[ATO] objet atomique	1 - ATO -> VAL (normale)
[DTY] data type	1 - ATO -> ATF (normale)
[MOL] objet moleculaire	2 - ATO -> ATP (normale)
[ROL] role	2 - ATO -> MOL (normale)
[VAL] valeur	----------
Rubriques M-I	**Rubriques M-R**
NOM (normale)	----------
CAR (normale)	ALL (normale)
----------	----------
Sous-Rubriques M-I	**Sous-Rubriques M-R**
----------	cardPMax (A 1)
cle (A 1)	cardPMin (A 1)
type (A 20)	cardAMax (A 1)

Definition

UNDEF

Figure 4.b: textual view

Figure 4: the meta-model for the MORSE semantic nets

The same could be said for the meta-model supporting the result of the transformation (the relational model). However, as we will see below, the relational meta-model has not been explicitely defined, since it is a subset of the MORSE model. This allows us to use the same formalism for the initial networks than for the result of their transformation. So a graphical view of a network in its initial state can also represent its final state as well as all its intermediate states. The view can be kept open during the transformation process, showing an animation that represents the transformation in a continuous way.

4.2. The Transformation Process

The goal of SECSI is to transform a MORSE semantic net into normal form relations (NFR). These NFR in turn yield directly the schema of a database of the relational model (RM). So the relational model constitutes the representation model for the normal form relations. In this section we present our implementation of the transformation process. We first describe the relation between the MORSE semantic network, then we explain the principles of the transformation process and we conclude the section with the description of this process using NeOpus.

4.2.1. Relation between MORSE and RM

MORSE is a generic semantic network and it is important to point out that the RM is actually implemented as a *sub-model* of MORSE, although they may be considered as two differents meta-models. As a matter of fact, the MORSE formalism -and also its graphical notation- is well suited for representing NFR. The notion of data-types in MORSE exactly matches the one of simple domain described by [Codd 70]. So if we consider a molecular object just bound to some atomic objects, each of it being linked to a certain data-type, we can see it as representing a NFR, whose attributes are represented by the atomic objects.

The next figure illustrates this point: the left part of it shows a MORSE molecular object representing a vehicle. This vehicle aggregates atomic objects, each of them being linked to a certain data-type (represented by left-pointing pentagons). The right part of the figure shows a textual description of a relation called "vehicle" composed of named attributes with their domain. It clearly appears that the two parts of the figure represent the same data structure.

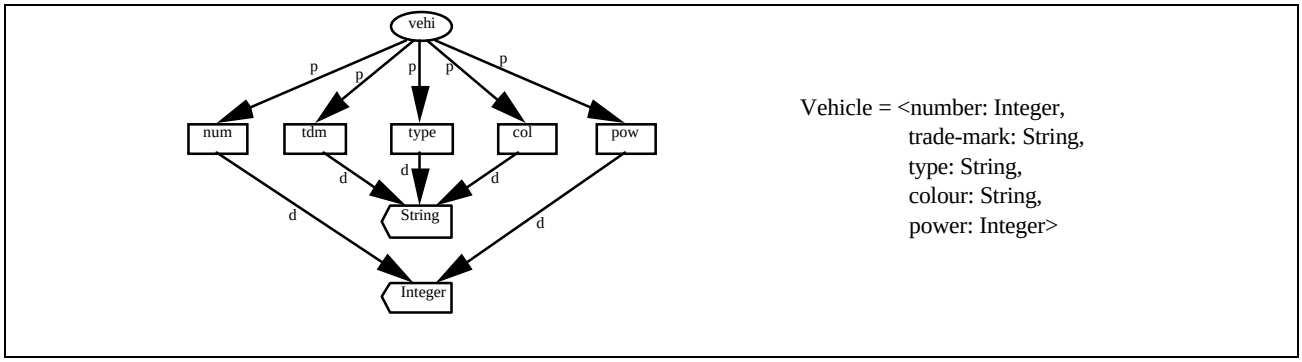

Figure 5: Equivalence between MORSE molecular objects and RM relations.

Finally, it is possible to establish equivalences between some of the MORSE concepts and some of the RM concepts: the notion of data-type is equivalent to the notion of domain, the notion of atomic object to the one of attribute, the notion of molecular object only aggregating atomic objects to the one of relation.

4.2.2. Principle of TP

The principle of the transformation process (TP) from MORSE semantic nets (MSN) to normal form relations (NFR) is described by Bouzeghoub in the form of an expert system [Bouzeghoub 85, 86]. It is described as a set of production rules that can be organized in different subsets according to their aim. Each rule is responsible for an elementar, - local- transformation of a MSN. This expert system is directed according to a control strategy that sequentially processes various phases in a predetermined order. In summary, the MORSE semantic nets to normal form relations transformation process consists of modifying an initial network to an equivalent set of connected components.

4.2.2.1. The types of rules

[Bouzeghoub 86] distinguishes four types of rules in the transformation process : simplification rules, intermediate rules, direct rules and rules for normalization.

The "*of simplification*" rules are aimed to eliminate redundancy in a MSN. That is typically the case for a simple rule supposed to cut a cycle of functional dependency links between atomic objects (cf. figure 4).

The second type of rules called " *intermediate*", are dedicated to reduce a MSN structure. Especially, they are responsible for the suppression in a MSN of all the MORSE concepts that have no direct equivalent in the RM. That is the case for generalization links between molecular objects, which can be eliminated by two different methods: either by moving component atomic objects from a general molecular object to a more specialized one (cf. figure 5), or by constraining some atomic aggregation links according to a role value of a general molecular object.

The third type are of *"direct"* rules. These rules are concerned with expressing the equivalencies between MORSE objects and those of the RM. Except for one rule that suppresses molecular aggregation links, the direct rules are not effective in our work because we do not change the representation formalism between an initial MSN and the result of its transformation, some NFR.

The fourth and last type of rules is called *"normalization*". The rules of this kind are here to ensure that the final result of TP is composed of fifth normal form relations [Codd 72].

4.2.2.2. Examples of rules

To illustrate this notion of local transformation rule, Figure 6 shows an example of the local transformation due to a simplification rule that cuts a cycle of functional dependency links by removing a redundant edge of a network graph (i.e. a 'df' link in our terminology). Figure 7 illustrates the first kind of suppression of generalization links described above.

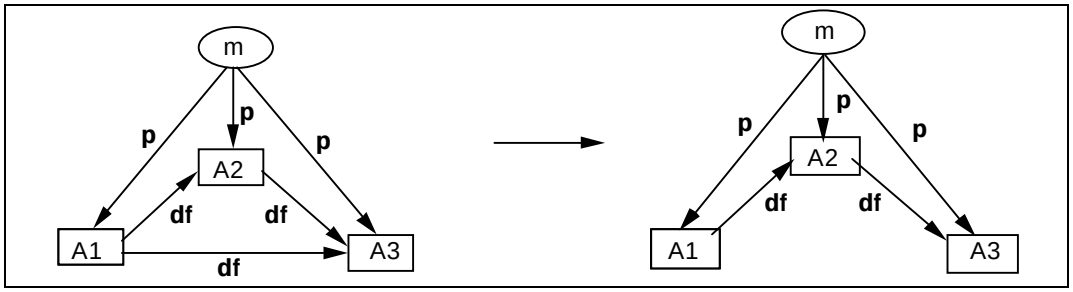

Figure 6: Example of a "simplification" rule

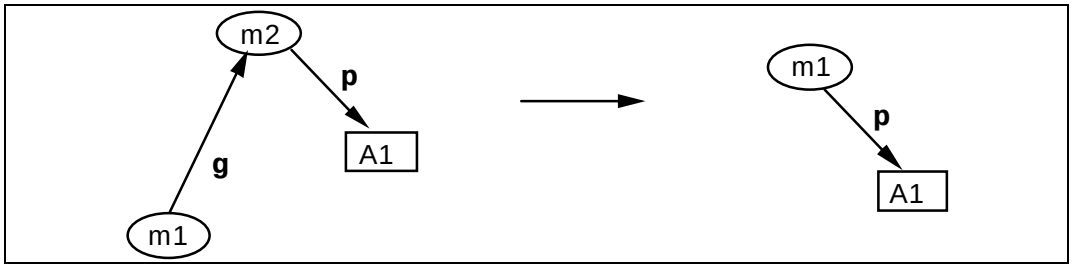

Figure 7: A so-called "intermediate" rule

4.2.3. using NéOpus

The Smalltalk classes generated by AckPro constitute our implementation basis for the MORSE semantic nets. They are instantiated (through a model editor, also generated by PckPro) for creating a particular network. In order to implement the transformation process (TP) in the form of an expert system, we had to express production rules on this class structure. We here describe in detail some rules of our rule-base, and show how their expression does not break the encapsulation provided by the application classes.

NéOpus allows to express production rules on any set of classes. Rules consist in a declaration part where variables are declared according to their class, followed by conditions and action part. Figure 6 shows the text of an example of a NéOpus rule. It corresponds to implementation of the rule of figure 6. "ATO" is the name of the class generated by AckPro, according to the description of atomic objects, as a result of the compilation of the MORSE meta-model.

```
eliminateDf
        | ATO a1 a2 a3. Global GraphicalController |
        a3 linkedInDfTo: a2.
        a2 linkedInDfTo: a1.
        a3 linkedInDfTo: a1.
      actions
      "update the structure through the model editor"
        GraphicalController getRidOfLink: #ATF from: a3 to: a1.
```

Figure 8: Smalltalk code for the rule of figure 6

On this example, we notice how NéOpus allows to use our implementation of a MORSE network, *without breaking the encapsulation principle* . The conditions recognize a local net configuration to modify, by specifying a set of Smalltalk expressions consisting of Smalltalk messages understood by the corresponding objects (here the message*linkedInDfTo:)*. It is essential to note here that the structure of the objects are completely ignored in the text of the rule. The action part is operating the transformation itself by modifying objects denoted by the variables that have been matched when firing the rule. The objects are modified by sending them appropriate messages.

4.3. The Control Process

Reasoning methods in forward chaining raise control problems. These problems are inherent to the mechanism itself: at a certain time of the inference cycle, several rules can be applied, and the choice of one of them conditions all the following reasoning. Most of the existing systems use a procedural control mode, that is to say described by the implementation of the inference engine.

Recently, it has been agreed that the control of a rule base is a crucial, and non-trivial process. It can need both general knowledge (general strategy for the choice of a rule) and domain specific one (e.g.. the notion of goal in OPS5). The idea that the evaluation of a rule-base is in itself a knowledge-based (or a meta-knowledge-based) process has led to find declarative manners to represent control. Some research works just express the need to dispose of declarative means of control [Batali 88], [Clancey 83]. Others propose general representation frameworks like the blackboards [Engelmore & al. 88], [Hayes-Roth 85]. But the problem of the representation of declarative control is still young in A.I.. Nevertheless, is seems to be accepted that control have to be explicit.

[Bouzeghoub 86] uses a classical method for the application of his transformation rules. The control of the transformation process, as he defines it, consists in using the notion of *packages of rules.* Each package corresponds to a step in the general TP. Each step is dedicated to a certain task of the TP and obeys a particular strategy. For instance, the generalization hierarchy is suppressed in a MSN thanks to rules written to ensure it is done with a depth-first method. Such a suppression defines one of the steps of the transformation process.

In NéOpus, the problem of control can be treated in two different ways. One way is to use the implicit control protocol the engine furnishes, but that is quite always insufficient. The best way to do is to use rule bases for the control itself. This has been initiated by [Pachet 92], according to the recent research works on the control problem, leading to seeing rule base control as a process that requires some particular expertise. Such rule bases are called *meta-bases* and their rules, *meta-rules* [Pachet&Dojat 92]. NéOpus environment includes a certain number of meta-bases for standard. Those meta-bases use some explicit control objects as described in [Pachet 92]. One of the principal advantages of this architecture is the reusability of control knowledge bases.

The control we used for implementing our transformation process of MORSE nets is indeed a NéOpus standard one, using the notion of *agenda*. It is based on *rule packages* that manage an agenda (a kind of collection of rule packages). It implements a simple sequential strategy where rules are applied by packages. Each rule package we defined corresponds to a certain step of the transformation process described by [Bouzeghoub 86].

In other words no particular work was done to implement the control strategey of SECSI in NéOpus. We simply picked one of the control rule base "off the shelf" and applied it to our problem.

5. Conclusion

Building a big system is not difficult. What is difficult is to build a big system that yields good software properties. We advocated that encapsulation is the key mechanism of object-orientation that helps in producing reusable software. But this encapsulation is jeopardized by the very nature of artificial intelligence systems, and particularily rule-based programming. We claim that in the particular case of object-oriented rule-based programming, this problem is solved by using a rule language that allows to express rules in the object language itself.

The resulting system is therefore by nature extensible, in the sense of object-oriented programming : new classes can be added to the system without breaking its consistency, the implementation of existing classes may be redefined without having to change all the rules that access them, and new rules can be added to the rule base without having to reconsider the design of the classes.

Future work will focuss on the application of our approach to other types of transformations, including transformation from relational database models to object-oriented database models.

6. Bibliography

[Batali 88] Batali J. "Reasoning about self-control". In meta-level Architectures and Reflection. P. Maes et D. Nardi eds. North Holland, 1988.

[Bouzeghoub 84] Bouzeghoub M. "MORSE: A functional query language and its semantic data model". Proc. of 84 trends and application conf on databases. IEEE-NBS Gaithersburg (USA), 1984.

[Bouzeghoub 85] Bouzeghoub M., Gardarin G., Métais E. "Database design tools: an expert system approach". VLDB conf. Stockholm, 1985.

[Bouzeghoub 86] Bouzeghoub M. "SECSI: "an expert system on information system design". PhD thesis, Université Paris VI. Paris, 1986.

[Brownston 85] Brownston L. & al. Programming Expert systems in OPS5. An Introduction to Rule-Based Programming. Addison-Wesley Publishing Company ,1985.

[Chen 76] Chen P.P. "The entity relationship model -Toward a unified view of data". ACM TODS V1, N1. March 1976.

[Clancey 83] Clancey W. "The advantages of abstract co ntrol knowledge in expert system design". Report n° STAN-CS-83-993. Stanford university, 1983.

[Clips] CLIPS reference manual, v. 5.1, NASA document JSC-25012, Huston, TX, September 1991.

[Codd 70] Codd E.F. "A relational model of data for large shared data banks". Communications of ACM, vol 13, nb 6. 1970.

[Codd 72] Codd E.F. "Further normalization of the database relation model". In Database system, Rustin ed., Prentice Hall Englewood Cliffs 1972.

[Cox 86] Codd E.F. "Object oriented programming: an evolutionary approach". Addison Wesley publishing company, 1986.

[Engelmore & al. 88] R. Englemore, T. Morgan. "Blackboard systems". Addison-Wesley Publishing Company, 1988.

[Goldberg&Robson83] Goldberg A. & Robson D. "Smalltalk 80: the language and its implementation". Addison-Wesley, 1983.

[Hayes-Roth 85] Hayes-Roth B. "A blackboard architecture for control". Artificial Intelligence N° 26, pp. 251-321, 1985.

[Krasner&Pope 88] Krasner G.E., POPE S.T. "A cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk 80". ParcPlace Systems, 1988.

[Krief 90] Krief Ph. "M.Pv.C - Un système interactif de construction d'environnements de prototypage de multiples outils d'interprétation de modèles de représentation". Thèse de l'Université Paris VIII. Paris, 1990.

[Lenat & Guha 90] D. B. Lenat & R. V. Guha. Building Large Knowledge-based Systems. Representation and Inference in the Cyc Project. Addison-Wesley, 1990.

[Meyer 88] Meyer B. Object-Oriented Software Construction. Prentice Hall, 1988.

[Pachet 91] Pachet F. "Reasoning with objects: The NéOpus environment". EastEurOOpe. Bratislava, 1991.

[Pachet 92] Pachet F. "Knowledge Representation by objects and rules : the NéOpus system". PhD Thesis, Université Paris VI. Paris, 1992.

[Pachet&Dojat 92] Pachet F & Dojat M. NéoGanesh: Towards a Generic Knowledge-Based System for the Control of Mechanical Ventilation. 14 th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, October 29-Novembre 1st, Paris, (1992).