

Using a Task/Method Knowledge-Base for Assisting the Development of Declarative Modellers

Emmanuel DESMONTILS & Francky TRICHET

IRIN

Université de Nantes & École Centrale de Nantes

2, rue de la Houssinière - BP 92208

44322 Nantes cedex 03

{desmontils, trichet}@irin.univ-nantes.fr

Abstract

The aim of the *CordiFormes project* is to propose a library of reusable objects and reusable generating methods characterising an ontology of the declarative modelling domain. The indented objective is to facilitate the development of declarative modellers, and more precisely to support the choice of the most relevant technique to be used for generating the objects manipulated by the modeller under construction. In this paper, we present how such an assistance can be supported by a Task/Method Knowledge-Base. A Task/Method Knowledge-Base allows representing explicitly all the relevant characteristics of the components of the *CordiFormes* library and therefore allows developing computer-aided design tools. These tools help the designer when (1) *selecting* the generating methods to be allocated to the objects and (2) *checking the coherence* of the adopted methods according to the relationships between the objects.

Key-Words: Declarative modelling, Task/Method Knowledge-Base, Computer-aided design tools

1 Introduction

The *declarative modelling* (also called *cooperative computer aided design* [1] or *generative computer aided design* [2]) aims at allowing an end-user to create scenes, in the context of a particular domain, by simply giving a set of properties and constraints that the scenes will have to respect [3]. This approach has given rise to a lot of projects [1, 4, 5, 6, 7].

The modellers proposed in these works are based on similar techniques and tools dedicated to the generation of scenes (i.e. exploration of the universe of potential scenes and selection of the ones which correspond to the initial description). However, no standardization work has to date been proposed. The aim of the *CordiFormes project* [8] is to capitalise all this know-how. This capitalisation is based on the development of a library of reusable objects and reusable generating methods characterising an ontology of the declarative modelling domain¹. The indented objective is to facilitate the development of declarative modellers, and more precisely to support the choice of the most relevant technique to be used for generating the objects manipulated by the modeller under construction.

A first proposal of such a library is currently integrated within the programming framework provided by the *CordiFormes project*. This library contains a set of reusable objects which are associated with predefined generating algorithms. In this context, the programming framework allows the designer (1) to define the objects that will be manipulated by the modeller (by construction *ex-nihilo* and/or selection and configuration of reusable components), (2) to allocate a generating method to each object (by definition of specific algorithms and/or selection and configuration of reusable algorithms) and (3) to obtain a prototype of the intended modeller. However, the current framework does not propose guidance when selecting and configuring the reusable components of the library. The different objects and algorithms are simply described by a name, that is supposed to be sufficiently explicit to denote the functionality of the underlying algorithm and/or the semantic of the underlying

¹An ontology is a theory of what entities can exist in a particular domain [9].

concept. They are not described by knowledge related to the selection criteria of the algorithms and/or to the way the objects are represented. Therefore, the one only exploitation of the library consists in presenting all the possible algorithms associated to a selected object. In other words, the current description of the library does not provide sufficient means to help the designer to select the generating methods to be adopted for each object, neither to check if the adopted method is technically usable and relevant for the considered object.

The solution we advocate consists in representing the reusable components of the library by the use of a Task/Method Knowledge-Base. The underlying goals of this Knowledge-Base is to represent explicitly the intrinsic characteristics of the generating algorithms (e.g. the requirements or the complexity).

The *task* notion is used to characterise an allocation objective of a generating method for an object O_i . A task is defined by *preconditions*, which describe the type of the object to be generated (e.g. a *color* type object or a *texture* type object), and a set of *associated methods*.

The *method* notion is used to characterise a generating technique which can be specific to a particular object (e.g. a method dedicated to generate *pale colors*) or generic and reusable for all the objects (e.g. a random type generation). A method is defined by a *selection context* and a *favourable context*, which respectively describe when it is *possible* and *relevant* to use it. Knowledge associated to the selection context describes technical constraints of the algorithm underlying the method such as “*a method applying a generation by enumeration can only be used when the domain of the considered object O_i is bounded and discrete*”. Knowledge associated to the favourable context describes the set of solutions that can be produced by the method. This knowledge is used to take the preferences of the designer into account. For instance, if the designer wants the modeller to be able to produce all the solutions for the object O_i , it is preferable to allocate an enumeration type method to O_i rather than a random type method, because this latter does not guarantee the generation of all the solutions.

A selection mechanism uses this knowledge for assisting the designer when allocating gener-

ating methods. This mechanism can be described as follows. Given an object O_i , the system first identifies a set of candidate methods for generating O_i (use of the *selection context* of the methods as a selection criteria) and then selects eventual relevant methods from the set of candidate ones (use of the *favourable context* as a selection criteria). A method is then advised to the designer, who adopts the recommended method or proposes another one. In this latter case, the system verifies if the proposed method M_i is coherent. The system checks (1) if M_i is applicable to the properties of the considered object O_i (bounded domain, discrete domain, etc.) and (2) if M_i does not introduce incompatibilities with methods previously allocated to objects in relation with O_i (for instance, an object O_i which is defined from the object O_k cannot be generated by an enumeration type generation method if the object O_k is generated by a random type generation method).

A first proposal of this Knowledge-Base is currently being developed within the DSTM tool. DSTM (Dynamic Selection of Tasks and Methods) is a programming framework dedicated to the construction of Knowledge Based Systems based on a Task/Method architecture [10]. This tool has been chosen because of the flexibility and the explicit representation of knowledge it provides (i.e. explicit representation of the tasks, the methods and the underlying selection mechanisms). In this context, DSTM allows us to simultaneously *model* and *implement* the Knowledge-Base.

The main objective of this paper is to emphasise how a Task/Method Knowledge-Base (classically used in Knowledge Engineering [11]) can be of effective help for assisting the development of declarative modellers.

In Section 2, we briefly present the construction process of a declarative modeller advocated in the *CordiFormes* project and we lay down the basic definitions underlying the putting into practice of our approach for selecting and checking the coherence of the generating methods. In Section 3, we describe the modelling principles of the Task/Method Knowledge-Base and justify the interest of using the DSTM tool as a support for implementing such a base. In Section 4, we dis-

cuss the choice of a Task/Method Knowledge-Base rather than a production-rule Knowledge-Base and describe how we plan to enhance the description of a method in order to refine the selection criteria. We conclude this paper by pointing out the potential interest of a Task/Method Knowledge-Base for adapting the solutions provided by the final declarative modeller to the end-user's wishes.

2 The declarative modelling and the project *CordiFormes*

Within a traditional approach of computer graphics, the end-user has to model and implement by hand (by using abstract specifications provided by the system such as geometrical, physical or topological properties) all the objects he wants to manipulate. The declarative modelling aims at releasing the end-user from such a programming work. Free from technical constraints, the end-user can thus focus his work on high-level tasks [12]. The use of a declarative modellers can be divided into three steps [3, 13]:

1. *the description of the scene*: the end-user describes what he wants to obtain by giving a set of properties²;
2. *the generation of the scene*: the modeller computes the solutions that respect all the properties of the scene to be generated;
3. *the insight of the scene*: the end-user selects (by using appropriated tools) the more relevant solutions provided by the modeller; if no solution fits the needs (i.e. no solution satisfies all the properties of the scene), the end-user can refine the initial description.

These steps have been studied in several works and more details can be found in [14] and [12].

2.1 Constructing a declarative modeller through the *CordiFormes* framework

The construction process of a declarative modeller provided by the *CordiFormes* project con-

²The means used to describe a scene depends on the considered domain. The properties of a scene can be formulated in natural language (oral or written), by the use of drawings, etc.

sists in (1) defining the different objects manipulated by the future modeller and (2) allocating a generating method for each of these objects.

2.1.1 Definition of the objects

The definition of the objects is based on the *concept* notion. One can distinguish two types of concepts: the *terminal concepts* and the *non-terminal concepts*. A *terminal concept* is a concept which is only described by a *domain*. A domain, denoted $[Bm, BM]_u$, is characterised by a list of possible values and a unity. A *non-terminal concept* is a concept defined from a set of *component-concepts*. A component-concept can be a terminal concept or a non-terminal concept.

Figure 1 presents an instance of a hierarchy of concepts. The non-terminal concept “House” is defined from the concepts “Roof” and “Wall”. The concept “Wall” is defined from the concepts “Height”, “Width”, “Texture” (whose domain is {stone, wood, brick}) and “Color”. The non-terminal concept “Color”, defined according to a RGB model, is composed of three terminal concepts called “Red”, “Green” and “Blue” (whose domains are $[0,255]_1$).

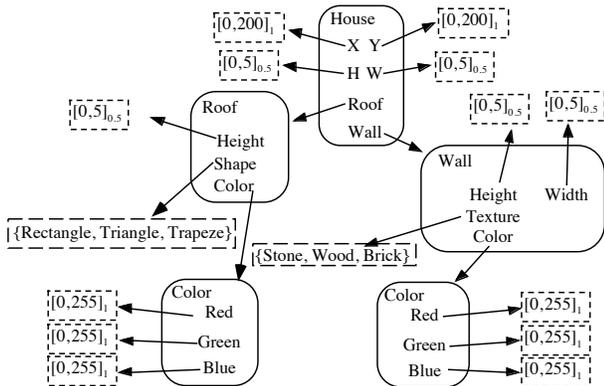


Figure 1: A hierarchy of concepts

The current library proposed by *CordiFormes* contains several predefined concepts classically used in computer graphics (e.g. colors, bounding boxes or classical curves and shape models). However, the considered domain can require the creation of specific concepts which are not managed by the library. For constructing a terminal concept, the designer has to define the domain. For constructing a non-terminal concept, the de-

signer has to define its component-concepts and their relationships. These component-concepts can be constructed by generalising or specialising existing concepts.

2.1.2 Allocation of the methods

Proposal of a new classification

Works on declarative modelling are generally classified according to the *building methods* they are based on: algorithmic exploration [15, 7, 5], inference techniques on rules and facts [16, 17], generative grammars [6, 1, 2], random generation using constraints [4, 18], etc.

Within the *CordiFormes* project, the adopted classification is based on the *intended objectives and the effects expected for the end-user*. In this context, only two classes of generation have been retained: *enumeration type generation* and *random type generation*. A method is member of the enumeration type generation if, given an object k , what would be the object $k + 1$ can be predicted for certain. A method is member of the random type generation if what would be the object $k + 1$ cannot be predicted for certain from the object k .

All the current works on declarative modelling can be classified according to these two classes.

The recursive generation

The *recursive generation* is based on the representation of the objects with hierarchies of concepts. This technique consists in constructing a concept C_i from its generating component-concepts³: a solution for each generating component-concept of C_i is first constructed, and then C_i is constructed according to the solutions computed for the component-concepts.

In this context, the generation class of a non-terminal concept C_i depends on the generation classes of its component-concepts. Intuitively, a concept C_i is defined with a random type generation if one of its component-concepts is defined with a random type generation; on the contrary,

³All the component-concepts of a non-terminal concept C_i are not necessary for generating C_i . A component-concept which is essential to generate a concept is called a *generating concept*.

C_i is constructed with a enumeration type generation.

The generating method of a terminal concept (or a non-terminal concept considered as a terminal one from a generation point of view) can be member of the enumeration class or the random class. In both cases, different exploration techniques can be used according to the preferred order for an enumeration class (descending order, growing order, etc.) and the adopted probabilistic rules for a random class.

Problems with the recursive generation

The choice of a generating method is generally influenced by the generating class intended for the considered object. When allocating a generating method to a particular concept C_i , the designer asks himself (in an implicit way) questions such as “*Should the modeller be able to deliver all the solutions for C_i ?*” or “*Should the modeller be able to produce two consecutive solutions relatively different for C_i ?*”. Answers to such questions direct the designer towards the choice of an enumeration type method or a random type method.

However, when allocating the generating methods, the designer can introduce discrepancies between the intended generation classes and the effective adopted buildings methods. For instance, let us suppose that the generation class intended for the concept C_i (defined from C_m and C_n) is the enumeration one and that the generating method adopted for C_m is member of the random class. In this context, a discrepancy occurs because a non-terminal concept cannot be generated by a method member of the enumeration class when one of its component-concepts is generated by a method member of the random class. Such mistakes take their origin from the fact that the manipulated concepts can be complex (deep hierarchies, lots of component concepts, etc.).

Within the current *CordiFormes* framework, a generating method can be allocated to a concept C_i in three different ways:

1. *Implicit allocation* when selecting C_i in the library or creating C_i without preferences about how C_i must be generated. The default generating method is allocated (each concept of the library is associated with a default generating method and when the

designer creates a new concept, a default method is automatically associated).

2. *Explicit allocation by modification* when selecting C_i in the library and modifying the default generating method. This modification consists in (1) allocating a new method for C_i , (2) modifying the generating methods of the component-concepts of C_i or (3) modifying the set of generating concepts of C_i .
3. *Explicit allocation by creation* when creating C_i ex-nihilo. When C_i is a non-terminal concept, the designer can directly allocate a specific generating method (without focusing on the component-concepts) or can first focus on allocating generating methods to the component-concepts (the generation class of C_i would then be defined by the recursive generation).

Whatever allocation approach is used, the designer can introduce discrepancies in the hierarchies of concepts. For instance, in the context of an explicit allocation by modification (second case), the designer can modify the methods of the component-concepts without modifying the generation class of C_i . This can lead to an incoherence which could render the effective generation of the scene impossible.

One can notice that in the context of a more sophisticated conception process (e.g. the draft path design), some discrepancies can have no influence on the effective generation of the scene because some of the objects are not constructed in the same time. For instance, when constructing a dolmen by draft path design (Cf. Figure 2), one first constructs the general shape of the dolmen (by using a method member of the enumeration class). When a satisfactory solution is obtained, one generates the component-concepts by using methods members of the random class. Although these methods are in conflict with the one adopted for the form of the dolmen, this would not conduct to a discrepancy because the specific objects are not generated at the same time than the general one.

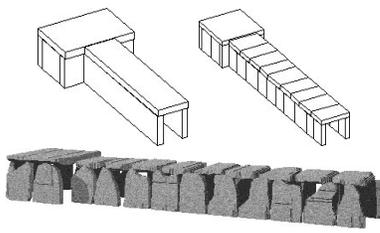


Figure 2: The draft path design of a dolmen [5]

2.2 Needs to integrate new tools in the *CordiFormes* framework

The current architecture of the *CordiFormes* framework is a three-layer one:

1. *The kernel layer.* This layer allows the designer to implement the objects and their associated generating methods. It is composed of (1) the library of reusable and configurable algorithms and predefined objects and (2) a set of programming structures which allows the designer to implement its own algorithms and objects. This layer has been developed above the *Java* language.
2. *The interface layer.* This layer allows the designer to define the interaction with the end-user. It contains a set of predefined dialogues which are used to manage the use of the final declarative modeller, i.e. the *description*, the *generation* and the *evaluation* of the scene.
3. *The prototype layer.* This layer allows the designer to obtain a first proposal of the intended declarative modeller.

However, the different tools provided by this three layers are not sufficient for assisting the construction of a declarative modeller. In particular, no tools are dedicated to assist the designer when selecting and configuring the reusable components of the library (provided by the *kernel layer*) and when checking the coherence of the adopted generating methods. Therefore, the framework has to be enhanced with computer-aided design tools.

The solution we propose consists in constructing a Task/Method Knowledge-Base used to *index* the current library and defining appropriated tools for *selecting* and *checking* the coherence of

the generating methods. This proposal leads to an extension of the current three-layer architecture to a four-layer one (Cf. Figure 3).

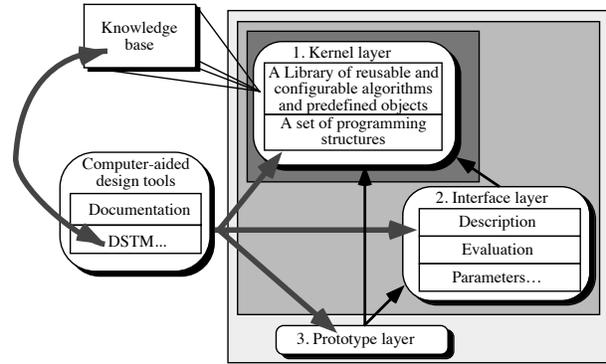


Figure 3: From a three-layer to a four-layer architecture of the *CordiFormes* framework

3 Constructing the Knowledge-Base with the DSTM framework

3.1 Interest of the DSTM framework

DSTM (Dynamic Selection of Tasks and Methods) is a programming framework dedicated to the construction of Knowledge Based Systems (KBS) based on a Task/Method architecture [10]. A KBS constructed with DSTM is composed of (1) a set of tasks and methods (which describe the problems to be solved and the different possible means to solve them) and (2) a set of selection mechanisms which allows the putting into practice of opportunistic behaviours (dynamic selection of the most relevant task to be studied according to the current solving context, and then dynamic selection of the most favourable method to achieve the selected task).

What makes DSTM originality (in comparison with related work in Knowledge Engineering such as Lisa [19] or MML [20]) is that this framework does not impose built-in definitions of what must be a task and a method and how tasks and methods must be selected. DSTM allows the customisation of the task and method definition, i.e. the set of slots that are used to describe the different characteristics of a task (resp. method) such as *Objectives* or *Selection context*, and then

the selection mechanisms. This leads to a better representation of the studied expertise.

In the context of the *CordiFormes* project, the expertise we want to model is related to the know-how of the computer graphics experts concerning the generating methods classically used for the objects. In this context, we have used the DSTM framework to construct a preliminary sketch of the Task/Method Knowledge-Base. The flexibility and the explicit representation of knowledge provided by this tool (explicit representation of the tasks, the methods and the underlying selection mechanisms) allows us to test different versions of the base before fixing the final one. In other words, DSTM has been used both for studying how to model and then implementing the Task/Method Knowledge-Base.

3.2 Modelling the Knowledge-Base

3.2.1 The adopted task and method modelling primitives

The modelling choices adopted for the task and method primitives have been influenced (1) by the technical constraints of the current generating methods (e.g. “*a generating method member of the enumeration class can only be used for concepts whose domains are bounded and discrete*”) and (2) by our will to take into account the objectives intended for the declarative modeller (e.g. “*the modeller would be able to generate all the possible solutions for a concept*” or “*two solutions asked by the end-user would be relatively different*”).

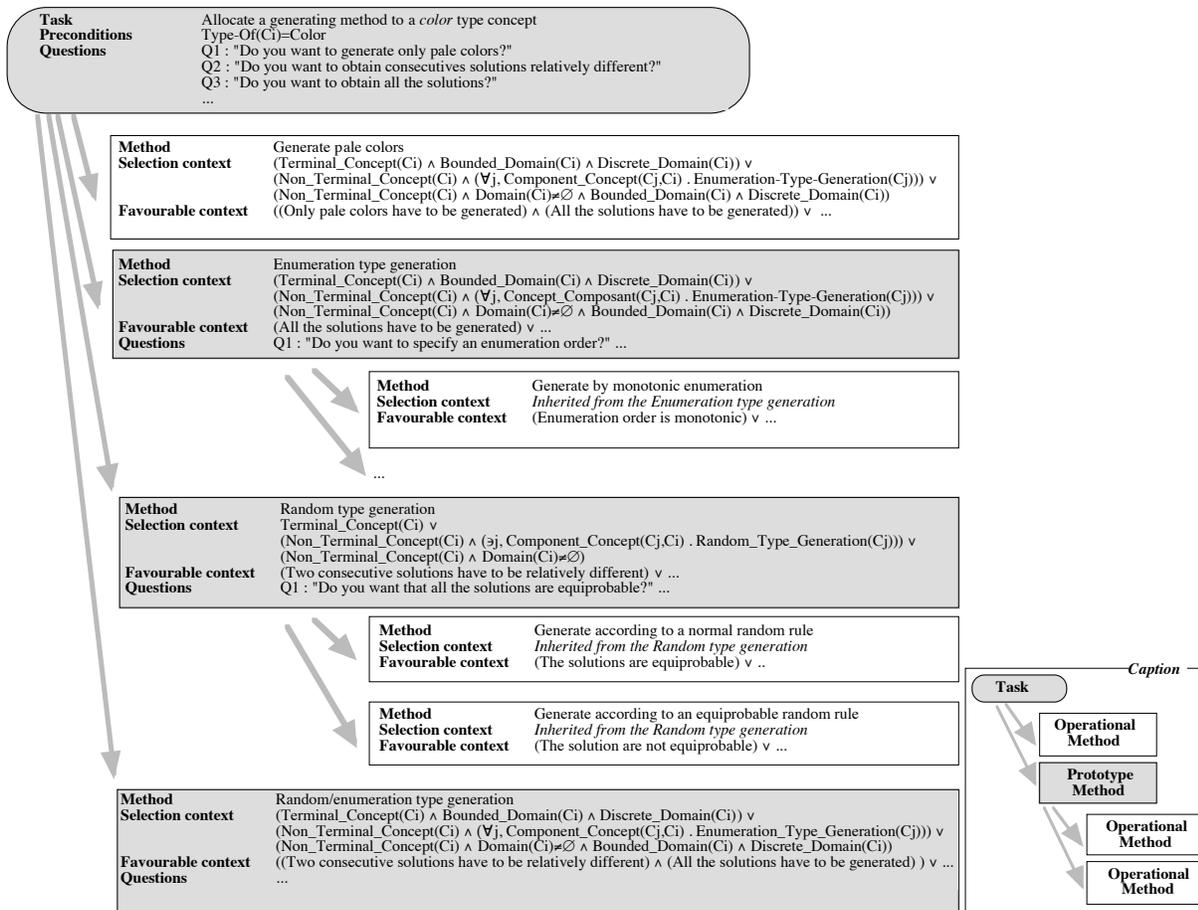
The *task* notion is used to characterise an allocation objective of a generating method for an object O_i . A task is defined by *preconditions* which describe the type of the object to be generated (e.g. a *color* type object or a *texture* type object), and a set of *associated methods*. For each object identified in the library corresponds an allocation task.

The *method* notion is used to characterise a generating technique which can be specific to a particular object (e.g. a method dedicated to generate *pale colors*) or generic and reusable for all the objects (e.g. a random type generation method). A method is defined by a *selection context* and a *favourable context*, which respectively describe when it is *possible* and *relevant*

to use it. Knowledge associated to the selection context describes technical constraints of the algorithm underlying the method such as “*a non-terminal concept is necessarily generated by a method member of the random class when one of its component-concepts is generated by a method member of the random class*”. Knowledge associated to the favourable context describes the set of solutions that can be produced by the method. This knowledge is used to take the preferences of the designer into account. For instance, when the designer wants the modeller to be able to deliver all the solutions for an object O_i , it is preferable to use an enumeration type method rather than a random type method, because this latter does not guarantee the production of the exhaustive set of the solutions.

Figure 4 presents the task dedicated to the generation of a *color* type concept and the different generating methods currently identified for such a purpose. The method *Generate pale colors* is specific to the task *Allocate a generating method to a color type concept*. The methods *Enumeration type generation* and *Random type generation* are techniques that can be used whatever the considered object is (these methods are associated to all the tasks of the Knowledge-Base).

As one can see from Figure 4, a set of questions is associated to each task. The designer is presented with these questions when the task is selected (e.g. when the designer is searching a generating method for the concept C_i underlying the task). Answers to these questions enhance the selection context (initially composed of knowledge related to the properties of the studied concept C_i such as “ *C_i is a non-terminal concept defined from the concepts C_m and C_n* ” or “ *C_i is a terminal concept whose domain is continuous*”) with knowledge related to the designer’s choices. Some questions are shared by all the tasks because they do not concern the semantic of a particular concept but the way to generate concepts (e.g. “*Do you want to obtain all the solutions?*”). Others are more specific and are used to characterise special features of a concept. For instance, the question “*Do you only want to generate pale colors?*” is specific to the task dedicated to the allocation of a method to a *color* type concept. This question permits the orientation of the ad-



Terminal_Concept, Bounded_Domain, Component_Concept, etc. are primitives which manipulate the description of the concepts.

Figure 4: An example of a task and its associated methods

vice towards a method that focuses on the property *saturation* (in our case, the method *Generate pale colors*).

Two kinds of methods are distinguished: *prototype* methods, which characterise a generation class, and *operational* methods, which characterise effective building algorithms. A prototype method is the generalisation of operational methods (*Specialisation/Generalisation* mechanism). For instance, the operational methods *Generate according to a normal random rule* and *Generate according to an equiprobable random rule* are specialisations of the prototype method *Random type generation*. A set of questions is associated to a prototype method. The designer is presented with these questions when the prototype method is considered as a *candidate method* (when the knowledge associated to its selection context is verified) or a *favourable method* (when it is a candidate method and the knowledge as-

sociated to its favourable context is verified) for generating the considered concept C_i . Answers to these questions enhance the selection context conditioning the choice between methods members of a same generation class. For example, the question "Do you want to specify an enumeration order?", associated to the method *Enumeration type generation*, allows the designer to define its own enumeration order.

Questions associated to the tasks and the prototype methods allow the system (1) to dynamically acquire the knowledge required to select the most relevant generating method and (2) to dynamically construct the interaction with the designer. All the questions are not asked at the same time (but only when they are relevant to be asked). In other words, resources are mobilised only when they are required. These two aspects are put into practice by the selection mechanism.

3.2.2 The adopted selection mechanism

The selection mechanism is based on the following actions:

- (1) selection of an allocation task (use of the *Preconditions* of the tasks as a selection criterion),
- (2) refinement of the selection context by interacting with the designer (use of the *Questions* associated to the selected task),
- (3) identification of candidate methods (use of the *Selection context* of the methods as a selection criterion),
- (4) identification of favourable methods (use of the *Favourable context* of the methods as a selection criterion),
- (5) refinement of the selection context by interacting with the designer (use of the *Questions* associated to the selected prototype methods),
- (6) selection of the generating method (use of the *Favourable context* of the operational methods as a selection criterion),
- (7) justification (to the designer) of the selected generating method.

At the end of step (4), several cases are possible:

- Case 1. Only one method is favourable and this method is an operational one *or* only one method is candidate (none is favourable) and this method is an operational one. Steps (5) and (6) are omitted and the system justifies the selected method (step 7).
- Case 2. Only one method is favourable and this method is a prototype one *or* only one method is candidate (none is favourable) and this method is a prototype one. Steps (5), (6) and (7) are sequentially performed.
- Case 3. Several methods are favourable and all these methods are operational ones *or* several methods are candidate (none is favourable) and all these methods are operational ones. Steps (5) and (6) are omitted.

The system presents (and justifies) the different candidate and/or favourable methods and the designer is asked to select one (step 6). When there exists both candidate and favourable methods, although the system gives priority to the favourable methods, it also accepts the choice of a candidate method.

Case 4. Several methods are favourable and some of them are prototype ones and others operational ones *or* several methods are candidate (none is favourable) and some of them are prototype ones and others operational ones. The system pursues the selection mechanism by only considering the operational methods (Cf. Case 1 and Case 2). This heuristic takes its origin from our will to give more importance to the designer's choices which characterise specific features of the considered concept. For instance, in a context where the designer wants to obtain all the solutions of pale colors, the methods *Generate pale colors* and *Enumeration type generation* are both favourable methods (knowledge associated to their *Favourable context* is verified). However, priority must be given to the method *Generate pale colors* which totally satisfies (and not partially as this is the case for the method *Enumeration type generation*) the wishes of the designer (Cf. Figure 4).

Case 5. Several methods are favourable and all these methods are prototype ones *or* several methods are candidate (none is favourable) and all these methods are prototype ones. The system selects the prototype method which *better* satisfies the designer's choices. Steps (5), (6) and (7) are then sequentially performed. For instance, let us suppose that, for a color concept type whose domain is bounded and discrete, the designer wants to obtain all the solutions with the constraint that two consecutive solutions must be relatively different. In this context, the methods *Random type generation* and *Random/enumeration type generation* are both favourable methods (Cf. Figure 4). However, priority must be given to the method *Random/enumeration type generation*

which satisfies all the wishes of the designer (the method *Random type generation* does not guarantee the getting of all the solutions).

One can underline that the taking into account of the designer's wishes can lead to some discrepancies. For instance, let us suppose that one of the choices is to permit the generation of all the solutions of the concept C_i , whose domain is non-bounded. In this context, the fact that C_i domain is non-bounded imposes a *random type generation*, whereas the choice of the designer requires an *enumeration type generation*. In these cases, the retained solution is (1) to inform the designer of the expected incompatibility and (2) to give priority to the respect of the technical features (e.g. the *Selection context* has priority on the *Favourable context*).

Then, given a concept C_i , the designer may want to use its own generating method. In order to detect such a situation, the question "*Do you want to allocate a personal generating method?*" is asked first and the selection mechanism is modified from the step (3) as follows:

- (3_b) acquisition of the features of the new generating method proposed for C_i ,
- (4_b) verification of the coherence of the new generating method proposed for C_i .

The aim of step (3_b) is to acquire information required for adding the new method in the Knowledge-Base. This information concerns the requirements of the method ("*Can your method be used to generate a concept whose domain is bounded?*") and the set of solutions that it can produce ("*Can your method produce all the solutions?*"). The objective is to fill in the *Selection context* and the *Favourable context* of the method. Moreover, in order to point out the eventual specificities of the new method, the designer can formulate questions which will be associated to the task underlying C_i . The aim of step (4_b) is to check if the proposed method does not introduce incoherences with the methods previously allocated to concepts in relation with C_i .

To sum-up, at the end of a selection cycle for a concept C_i , different cases can occur:

1. the method proposed by DSTM satisfies all the wishes of the designer and he adopts it,
2. an incoherence has been detected and the designer follows the advice of DSTM (he adopts the proposed method),
3. an incoherence has been detected and explicitated to the designer who does not want to relax his constraints and/or adopt another method than his. In this case, a negotiation process is proposed, from which the designer can modify his previous choices about the generating methods and/or the definition of some concepts.

3.3 Integration of the Knowledge-Base in the CordiFormes framework

A first version of the Task/Method Knowledge-Base is currently being implemented within the DSTM framework. DSTM is based on the object-oriented layer *Ceyx* of the *LeLisp* language. Therefore, this current Knowledge-Base will only be manipulated from a *LeLisp* environment. However, a current project aims at developing DSTM above the *Java* language [21]. This will facilitate the integration of the Knowledge-Base in the *CordiFormes* framework, which has also been developed above the *Java* language. The architecture we propose is presented Figure 5.

First, the designer defines the hierarchies of concepts (which will be manipulated by the modeller) by using the tools provided by the *kernel layer* (Cf. Section 2.2). These hierarchies are represented as *Java* objects. In order to allocate a generating method for each concept, the hierarchies are explored within an bottom-up approach (e.g. from the terminal concepts to the non-terminal ones). Given a concept C_i , a first diagnosis is made in order to define the technical features of C_i . This diagnosis is contextual because it takes into account the different allocations that could be done previously. For instance, for the concept C_k (Cf. Figure 5), this diagnosis can be interpreted as follows: "*C_k is a non-terminal concept and one of its component-concepts is generated by a method member of the random class*". Then, the dynamic selection of tasks and methods is performed. According to the situation (i.e. conflict or consensus), DSTM directly

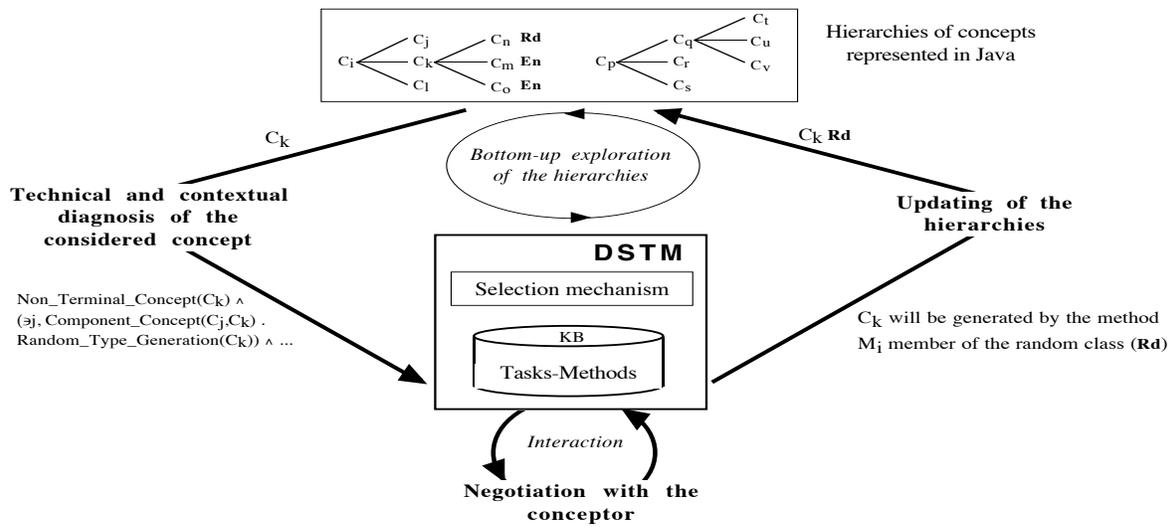


Figure 5: Integration of the Knowledge-Base in the *CordiFormes* framework

allocates the proposed method to C_i or advocates a negotiation phase. This negotiation can conduct to modify the previous allocated methods or the *Java* hierarchies. The current Knowledge-Base (implemented within the original DSTM) does not yet provide this negotiation process and does not yet manage the configuration of the reusable components.

4 Discussion

4.1 Task/Method Versus production-rule Knowledge-Base

The objectives underlying the construction of the Knowledge-Base are (1) to allow the system to justify the proposed method (this requires the explicitation of all knowledge that plays a role in the selection process) and (2) to facilitate the updating and the upgrading of the base. A rule-base formalism is not adapted for such purposes. A production rule is generally composed of a set of premises P_i , where each premise denotes a part of the conditions required to select the method underlying the rule. However, these premises are not structured according to the nature of the domain knowledge they represent. For instance, the following rule (that could be used to represent *when* the method *Generate pale colors* can be used) does not explicitly differentiate that the premises P1, P2, P3 and P4 denote technical selection criteria whereas the premises P5 and P6

denote selection criteria related to the choices of the designer. Therefore, the system can not distinguish two premises of different nature and, consequently, is not able to perform a specific selection mechanism, in particular a mechanism that can differentiate candidate methods and favourable methods (Cf. Section 3.2.2). Moreover, modifying a production-rule Knowledge-Base is generally a complex process [11] because of the difficulty to keep in mind a synthetic understanding of the general behaviour from a set or isolated low-level rules. A Task/Method representation formalism is more convenient for expliciting, upgrading and updating a Knowledge-Base.

If	Type-Of(C_i)=Color	P1
	Terminal_Concept(C_i)	P2
	Bounded_Domain(C_i)	P3
	Discrete_Domain(C_i)	P4
	Only pale colors have to be generated	P5
	All the solutions have to be generated	P6
Then	"Generate pale colors" is a candidate and favourable method	

4.2 Towards the taking into account of new knowledge as selection criteria

A concept C_i can have several sets of generating concepts (Cf. Section 2.1.2). A set of generating concepts characterise a way to represent C_i (i.e. a representation model for C_i). However, a representation model can influence the set of solutions that can be computed for C_i . For instance, a color type concept can be represented

with multiple models such as the RGB model or the TLS model. These models greatly influence the generation step as they do not lead to the same set of solutions.

Therefore, these different point of views on the same concept must be taken into account when allocating a generating method. We propose (1) to associate to each method a description of the representation model of C_i it is based on and (2) to define specific questions (associated to the tasks) which will allow the designer to specify a particular representation model for C_i . We also plan to enhance the description of a generating method with knowledge related to its complexity and its reliability [2, 15].

4.3 Interest of a Task/Method Knowledge-Base for adapting the solutions to the end-user's wishes

When constructing a declarative modeller with the *CordiFormes* framework, the designer allocates one and only one generating method to a concept, and this choice is definitive. In other words, the designer imposes his choices to the end-user. In order to relax this constraint, we plan to offer the designer the possibility to define several potential generating methods for a same concept. These methods will be used to dynamically adapt the generated solutions to the end-user's wishes. For instance, for generating a *house*, it could be interesting to propose two methods: one which generates detailed solutions (time consuming and resources heavy method) and another one which generates crude solutions (economic method). These methods, similar in functionality, will allow the modeller to adapt itself to the end-user's wishes. These wishes can be expressed in an explicit way in the *description* (e.g. "*I want a coarse house rapidly*") which influences the modeller to choose the second generation method) or in a implicit way and therefore interpreted by the modeller. For example, when the end-user says "*On the foreground, I want the house that I have previously described and on the background, a white house*", the modeller interprets the fact that the end-user does not want to focus on the white house and therefore selects the method which generates crude solutions. One can notice that these choices can be

reviewed according to the adopted visualisation point of view. For instance, in the previous example, if the end-user uses virtual reality tools (as for instance VRML), the solution advocated by the system is not a satisfactory one because the end-user can move in the scene and therefore can focus on all the objects of the scene. In this context, the background house has to be generated by a method which produces detailed solutions.

This example underlies the interest of using a Task/Method Knowledge-Base during the *use* of a declarative modeller. Indeed, providing more than one method per object increases the flexibility of the modeller and allows it to dynamically adapt the solutions (e.g. dynamically selects the more relevant generating method) to the end-user's wishes.

Acknowledgments: We are grateful to J.-Y. Martin, P. Tchounikine and L. Hartley for providing a number of useful comments on earlier versions of this paper.

References

- [1] S. Kochhar. CCAD: A Paradigm for Human-Computer Cooperation in Design. *IEEE Computer Graphics and Applications*, 14(3):54–65, 1994.
- [2] R. F. Woodbury. Searching for Designs: Paradigm and Practice. *Building and Environment*, 26(1):61–73, 1991.
- [3] M. Lucas, P. Martin, D. Martin, and D. Plemenos. Le projet *ExploFormes* : quelques pas vers la modélisation déclarative de formes. In BIGRE, editor, *Acte des journées AFCET-GROPLAN*, 67, pages 35–49, France, 1989.
- [4] D. Chauvat. *Le projet VoluFormes : un exemple de modélisation déclarative avec contrôle spatial*. PhD thesis, Université de Nantes, 1994.
- [5] F. Poulet and M. Lucas. Modelling Megalithic Sites. In *Eurographics'96*, pages 279–288, France, 1996.
- [6] A. Rau-Chaplin, B. MacKay-Lyons, and P. F. Spierenburg. The LaHave House Project: Towards an Automated Architectural Design Service. In *Cadex'96*, pages 24–31, 1996.
- [7] L. Khemlani. GENWIN: A Generative Computer Tool For Window Design in Energy-Conscious Architecture. *Building and Environment*, 30(1):73–81, 1995.
- [8] E. Desmontils. *Le projet CordiFormes : une plateforme pour la construction de modeleurs déclaratifs*. PhD thesis, Université de Nantes, 1998.
- [9] J. Charlet, B. Bachimont, J. Bouaud, and P. Zweigenbaum. Ontologie et réutilisabilité : expérience et discussion. In Cépadues-Editions, editor, *Acquisition et ingénierie des connaissances*, pages 69–87, 1996.

- [10] F. Trichet and P. Tchounikine. Reusing a Flexible Task-Method Framework to Prototype a Knowledge Based System. In *9th International Conference on Software Engineering and Knowledge Engineering (SEKE'97)*, pages 192–199, Spain, 1997.
- [11] J.M. David, J.P. Krivine, and R. Simmons. *Second Generation Expert Systems*. Springer-Verlag, 1993.
- [12] C. Colin, E. Desmontils, J.-Y. Martin, and J.-P. Mounier. Working with Declarative Modeler. In *Computographics'97*, pages 117–126, Portugal, 1997.
- [13] M. Lucas. Equivalence Classes in Object Shape Modeling. In *IFIP TC5/WG 5.10 Working Conference on Modeling in Computer Graphics*, pages 35–49, Japan, 1991.
- [14] M. Lucas and E. Desmontils. Les modeleurs déclaratifs. *Revue Internationale de CFAO et d'infographie*, 10(6):559–585, 1995.
- [15] C. Colin. *Modélisation déclarative de scènes à base de polyèdres élémentaires*. PhD thesis, Université de Rennes, 1990.
- [16] D. Martin and P. Martin. Declarative Generation of a Family of Polyhedra. In *GraphiCon'93*, Russia, 1993.
- [17] D. Plemenos. *Contribution à l'étude et au développement des techniques de modélisation, génération et visualisation de scènes : le projet MultiFormes*. Prof. thesis, Université de Nantes, 1991.
- [18] E. Giunchiglia, A. Armando, P. Traverso, and A. Cimatti. Visual Representation of Natural Language Scene Description. *IEEE Transaction on Systems, Man and Cybernetics*, 26(4):575–589, 1996.
- [19] I. Jacob-Delouis and J.P. Krivine. LISA: un langage réflexif pour opérationnaliser les modèles d'expertise. *Revue Intelligence Artificielle*, 9(1):53–88, 1995.
- [20] V. Guerrero-Rojo. MML, a modelling language with dynamic selection of methods. In *The Knowledge Acquisition for Knowledge-Based Systems Workshop (Banff'95)*, Canada, 1995.
- [21] Z. Istenes, F. Trichet, G. Camilleri, and A. Fortes. *Modelling Knowledge Structures and Patterns: Capturing World Subsets through Language Structured Subsets*. Research Report IRIT/98-06-R, Institut de Recherche en Informatique de Toulouse, 1998.