# Working Modes with a Declarative Modeler

Christian Colin, Emmanuel Desmontils, Jean-Yves Martin, Jean-Philippe Mounier

# WORKING MODES WITH A DECLARATIVE MODELER

**Christian COLIN** [†]
**Emmanuel DESMONTILS** [††], **Jean-Yves MARTIN** [††]
**Jean-Philippe MOUNIER** [†††]
[†] Ecole des Mines de Nantes
4, rue Alfred Kastler
BP 20722
44307 NANTES CEDEX 3
FRANCE
Christian.Colin@emn.fr
http://www.emn.fr/dept_info/perso/colin

[††] Institut de Recherche en Informatique de Nantes
Faculté des Sciences et des Techniques
2, rue de la Houssinière
BP 92208
44322 NANTES CEDEX 3
FRANCE
Emmanuel.Desmontils@irin.univ-nantes.fr
Jean-Yves.Martin@irin.univ-nantes.fr
http://www.sciences.univ-nantes.fr/info/recherche/mgii

[†††] CERMA
Ecole d'Architecture de Nantes
rue Massenet
44300 NANTES
FRANCE
mounier@cerma.archi.fr

## ABSTRACT

This paper presents the point of view of a designer about declarative modelers. We propose a model of this point of view. Our approach is based on the work we have done to develop some of our declarative modelers. It takes into account the main functions used by a designer in these modelers. We explain also the scene-building process and we present working processes that can be used with those modelers.

**Key Words**: Declarative modeling, Design, Understanding, Description, Modelers.

## INTRODUCTION

Research in Declarative Modeling started some ten years ago. The declarative modelers which have already been realized shown the interest of such an approach. They helped us to experiment with some of the mechanisms involved in that kind of modeling. Our main purpose is to create scenes only giving only a set of properties and constraints that the scenes will have to respect. The computer is in charge of exploring the universe of potential scenes and selecting the ones which correspond to the given description. The designer has only to choose the ones he

prefers, with appropriate tools. He can concentrate on the process of creation, instead of calculating the scene.

Due to its great experience in declarative modeling, the GEODE group is currently writing a "state of the art" in this domain. Within this framework, a group composed of the authors of this paper began work to define what a declarative modeler is for a designer, and to model a "User Model of a Declarative Modeler". The purpose is to describe the functionalities a user can see. It is devoted to the intended user or intended developer of a declarative modeler. This model doesn't include any detail on the implementation because this one doesn't belong to the user's universe. However, it is based on the modelers developed by the members of the GEODE group since 1988. The model is a generalization of these modelers. Furthermore, it is the seed of the new generation of modelers we began.

At the beginning of this paper, we will compare conventional and declarative modelers to highlight their differences. Then, we will show the main modules of a declarative modeler. At last we will present the working modes a designer can use with a declarative modeler.

## COMPARING CONVENTIONAL AND DECLARATIVE MODELERS

When an user wants to solve a design problem with a conventional modeler, he first has an idea which can lead him to a solution to his problem (Fig. 1). He clarifies it by writing specifications that have to be respected. Using them, he mentally conceives a quite precise object. After that, he has to deduce the list of elementary operations which are authorized by the modeler and necessary to design the object. He interactively and gradually builds the object, following the elementary operation's planning. He then checks the validity of the object he built, using a set of validity tests. Negative results to a test mean that the mental object the designer conceived doesn't answer the specifications. So, he must change this mental object before he interactively updates the modeled object. These steps are repeated until the object passes the set of validity tests ("negative tests" loop on Fig. 1). When the object is finished, the designer can also modify the specifications in order to get a best answer to his problem. This corresponds to the design process ("design process" loop on Fig. 1). He has to repeat all steps in the "negative tests" loop to build a new scene corresponding to the new specifications.

Designing with a declarative modeler is quite different. According to his idea, a designer describes properties and constraints that must be verified to solve his problem. He can have a "mental image" of an object that, for him, could be a solution. However, this image doesn't take part in the design process, and a declarative modeler has no use for it. From the description given by the designer, the modeler automatically builds one or more scenes which are in accordance with his request. It is obvious that it isn't useful to apply validity tests to these scenes. However, as the modeler built the scenes, the designer doesn't have a precise idea

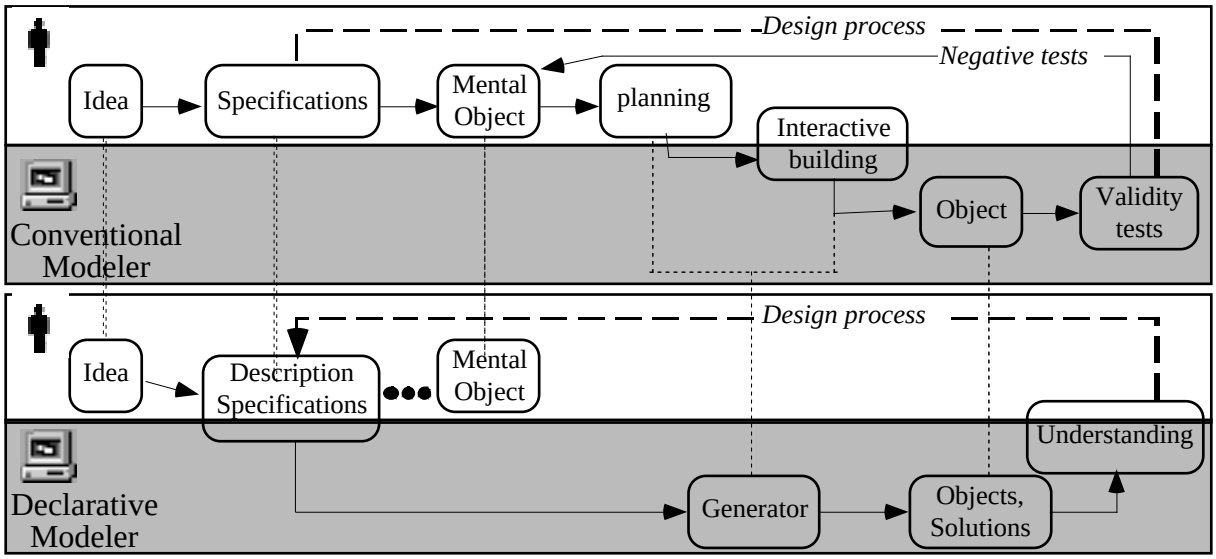



Fig. 1: Comparison between conventional and declarative modelers

of their aspect. So, it is very important to give him some efficient tools allowing him to understand the solutions. After seeing them, he may want to change the description he gave at the beginning. This is the design process (Fig. 1).

Fig. 1 shows the building scene's process with conventional or declarative modelers. The user's work corresponds to the white areas whereas computer's work is in the gray ones. The dots stress the correspondence between steps in the two kinds of modelers. The location of the boxes shows who, the man or the computer, does the action. When the box is in the both areas, it shows how much work each of them has to do. This Figure points out one of the main differences between the two approaches: the amount and the kind of work done by a man. Declarative modelers are in charge of a more important part of the work than conventional modelers. That allows a designer to restrict his work to higher-level tasks.

## MAIN PHASES IN A DECLARATIVE MODELER

### Overview of a declarative modeler

Using a high level of abstraction, a declarative modeler is very simple for a designer. The initial process allows him to describe what he wants (Fig. 2). After a computing time, he can discover and understand the produced scenes with specific tools. Because of the definition of declarative modeling, the scenes are in accordance with the description made by the designer. They are called "solutions".

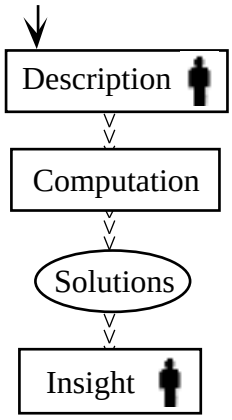



Fig. 2: Overview of a declarative modeler

On figures 2, 4 and 5 we use the same notation. Actions are represented by rectangles (Fig. 3) and informations by ellipses. An action can be made of sub-processes and data. An arrow on a rectangle shows which action is executed first. The actions the user can interact with are marked with a small human icon. A lozenge means a test. An arrow made of chevrons ('>') means oriented data exchange between two actions. This arrow doesn't imply any notion of sequence.

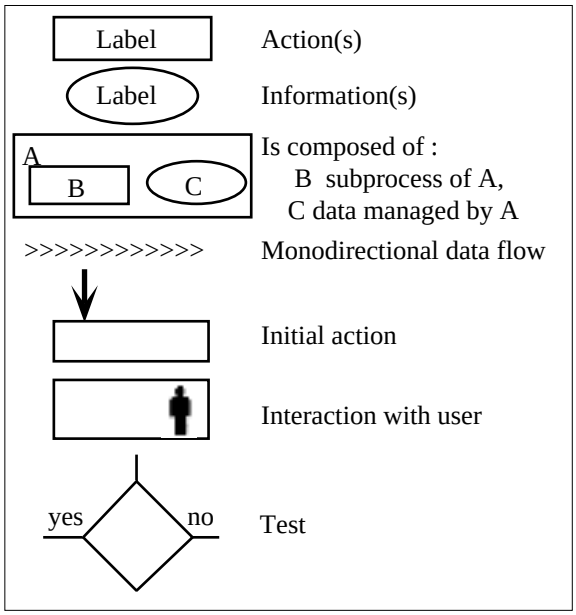



Fig. 3: Figures notation

The different schemes stress only data exchanges between the tasks. So, each diagram remains valid whether or not the implementation is sequential. The differences a designer can see according to the implementation relate to his comfort level with the work and, especially, the execution time. For instance, with a purely sequential implementation [Chauvat94, Colin90, Liege-Hegron97, Martin-Martin93, Martin90, Poulet-Lucas96], the designer has to input the whole description before the modeler can start to build any solution. He discovers the computed objects only when all of them are produced. So, the waiting time can be very long. With a concurrent implementation, the modeler begins to compute scenes as soon as the description contains enough informations. The designer can see objects as soon as the first solution is built [Champciaux97].

### The description phase

With a declarative modeler, a designer must be able to enter and to modify the description using many natural ways. That is why a modeler

offers a set of high level tools allowing the input data from text, a graphics or other ways. These software tools may use any device connected to the computer. The description items are saved and checked before they are used to create scenes (Fig. 4). Indeed, they can contain contradictions and inconsistencies. We make distinctions between problems which are detected during the description, and those which can be found only during the exploration of the possible scenes universe. Checking description items consists of looking for inconsistencies of the first kind.

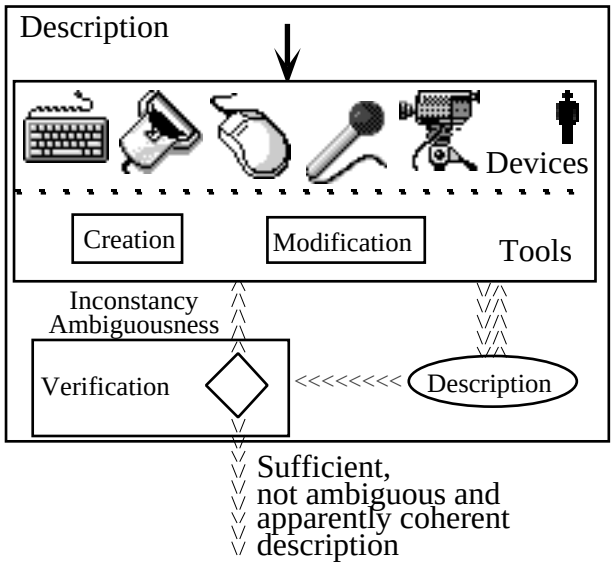



Fig. 4: Description phase

We seek to clear up ambiguities during the checking phase. In fact, the description can be ambiguous because there are several ways to interpret it. Each meaning can lead to a different concept. For instance, the use of the negation form can be understood differently [Pacholczyk -Desmontils97]. If the designer specifies “not big”, does he only want small objects, or does he want objects which aren't big ? Or is there another meaning ? In such a case, the modeler seeks to automatically remove ambiguities taking into account the context of the description. It can also ask the designer for the precise meaning of the description. When there is no more anomaly, the description is sent to the generation module. The sending way of the description depends on the implementation. The modeler can parse the consistence and send data to the generation module immediately after the input of each description item or, on the contrary, only when the input of the description is totally finished.

**Insight phase**

As the description given by the designer is usually vague and sketchy, more than one solution can be found. This diversity is an advantage for the designer because he obtains several replies to his problem with only one description. He can freely choose the one he prefers. But, sometimes there are a great number of solutions. This variety becomes a difficulty because the designer can't see all generated scenes. That is why declarative modelers have to offer specific tools allowing the designer to navigate in the solution space. Currently, we can divide these tools into three types: refinement tools, browsing tools, and controlled alteration tools (Fig. 5).

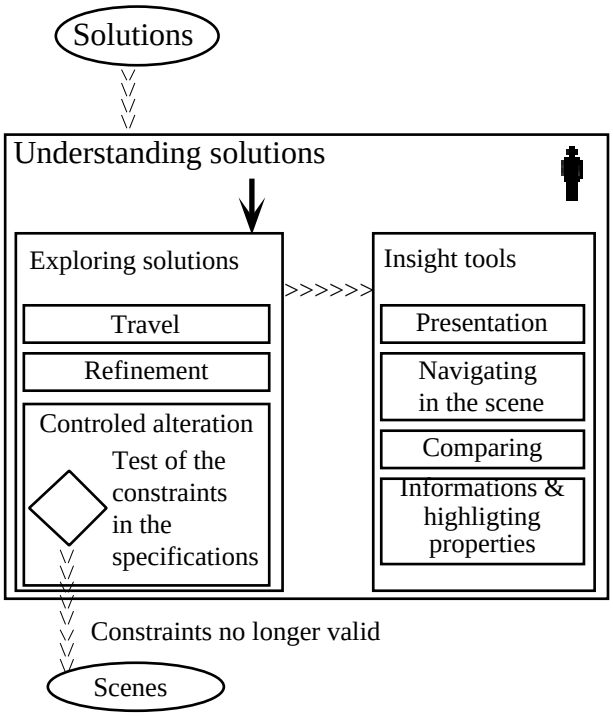



Fig. 5: Insight phase

As all the solutions are in accordance with the designer's specifications, each of them is a reply to the design problem. Sometimes, the designer wants just one solution, any one. So, he will keep the first one that the modeler presents. But since he can choose, he can be more selective. In that case, he can use refinement tools which act like filters to select solutions from new criteria. To obtain a subset of solutions, he adds, for a short time, more restrictive properties to the specifications. For example, these properties can be “the presence of properties which aren’t necessary”, “solutions that maximize a property”, etc. The modeler

checks “a posteriori” the objects it produced and keeps only the ones that are in accordance to the new properties. At any time, the designer can take back the initial specifications or, on the contrary, he can validate the new constraints which will be permanently added to the specifications.

With browsing tools, the designer defines the presentation order of the scenes he selected with the refinement tools. For instance, he wants to see solutions in ascending order of their size. In spite of the refinement he made, the number of selected solutions can still be very large. Thus, he can first discover the most representative scenes. This is a kind of ordering solutions [Martin-Martin93]. Thus, he doesn’t have to inspect all the scenes in the selection. The representative solutions, in accordance with the criteria, are shown following the defined order.

Controlled alteration is the third way to explore the solution space. While the designer is examining a solution, he sometimes wants to change a detail, to alter the scene interactively, with respect to the specifications. As the solution space contains all the solutions, the altered scenes belong to this set. The interactive alteration of a solution amounts to looking for scenes and displaying those corresponding to the alteration described by the designer. This is a particular way to explore the solution space. The alteration the designer wants isn't always possible in accordance with his specifications. So, the modeler has to prevent the designer for any constraint violation and to allow him to exceed the specifications. But in this case, the modeler produces a scene which is no longer a solution to the problem. In that case, it gives to the designer a measure of the error with respect to the specifications.

From his point of view, the designer has the set of all solutions in accordance with the specifications. He can explore it as he wants, using techniques we mentionned previously. However, according to the modeler implementation, the solution set isn't always totally computed before the beginning of the insight phase [Champciaux97, Liege-Hegron97, Martin-Martin93, Poulet-Lucas96]. This is the case when the display of scenes takes place simultaneously with the phase of scene creation [Champciaux97]. With these three kinds of tools, the designer defines the kind of exploration he wants. Therefore, he induces a classification on the displayed solutions. Some of them are selected and studied before others. If all the solutions aren't built yet, the generation phase has to be dynamically controlled to first compute the solutions which will be shown to the designer. This problem is complex and really hasn’t been resolved yet.

Another important difficulty for a designer is to understand solutions. He knows exactly the description of the solution. However, he doesn’t know their aspect because he didn't build them by himself. It is very difficult to handle unknown scenes only with conventional tools [Colin90]. Usually, with any high level system, displaying a solution without any explanation isn't sufficient. So, a set of tools which helps the designer understand solutions are available in declarative modelers. These help him understand the structure and properties of solutions [Lucas-Desmontils95]. Currently, four kinds of such tools are proposed in declarative modelers: displaying tools, moving tools, highlighting tools and comparison tools.

Displaying tools handle all kinds of display found in conventional modelers. For instance, hidden lines algorithms, photorealism methods, and superimpositions of dimensions belong to these kinds of tools.

One advantage of a declarative modeler is that it has a detailed knowledge of the produced scenes. Particularly, it knows their description, how the scenes were created and why they are built like this. It can use the whole data set to offer high-level tools. Thus, it can automatically highlight properties the designer wants to observe. The highlight is obtained by choosing a good viewpoint and using a suitable display mode [Colin90, Sellinger-Plemeos96]. For example, a hole inside a scene can’t be shown only with the computation of the observer’s location. In this case, the scene can be cut by a plane which intersects the hole. A suitable display mode is to show an exploded view of the scene and to display the hole with a specific color [Colin90]. The highlight of properties is also useful as well for properties defined in the description as well as for those which are fortuitous.

The moving tools allow the designer to move inside the scene. The usual tools which handle a scene in real time through devices, like a mouse or a spaceball, belong to this kind of tools. High-level tools allow the designer to describe, with a language close to his own, the type of the view or kind of moving he wants for a better understanding of the scene. For instance, he indicates to the modeler that he wants to “walk slowly inside the scene” which represents a

town [Mounier96]. In that case, the modeler computes a view or an animation which respects his request.

The last kind of tools allows the designer to compare several solutions or several views of one solution [Lucas-Desmontils95]. The purpose is to highlight differences and similarities among several views of one or several scenes. For example, two views are superimposed and the common areas are displayed differently.

## DESIGNING WITH A DECLARATIVE MODELER

The previous figures stress the main functionalities of a declarative modeler from the point of view of a designer. But, they can't show the succession of steps the designer has to take to obtain solutions to a problem. The purpose of this part is to model the design process when the modeler is a declarative one.

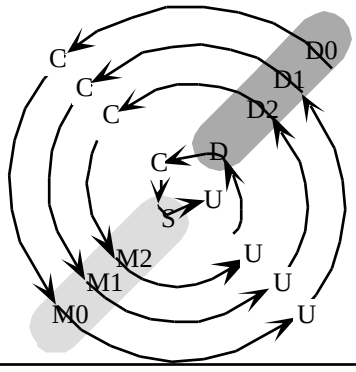
One of the foundations of declarative modeling is to allow a designer to give an incomplete description [Lucas91]. This allows him to explore of a set of possibilities and begin the design without detailed specifications. The initial description, which is usually incomplete, can be insufficient to automatically obtain solutions. The modeler doesn't have enough data to produce any scene. To solve this problem, a first approach is to let the modeler automatically complete data by choosing values according to a default strategy. Another approach is to ask the designer for additional informations, so he can control the choice of missing data and act on the modeler to guide it in searching for solutions. Before it asks for any additional data, the modeler presents the construction model with appropriate tools. We will call this a partial model. This model can be pieces of scenes, intermediate scene or objects which do not belong to the scene universe but that take part in the solution creation. The designer gets an insight into them and completes the knowledge of the modeler. So, the scene generation phase can progress until the solutions are built or until another piece of data is missing. With his intervention, the designer gives directions to follow or to reject during the solution search.

When the designer doesn't want strong control over the missing data, he lets the modeler apply a strategy to choose default values. Each further piece of information is given by the designer or chosen by the modeler. The designer can adjust the functioning mode of declarative modelers:

- in automatic mode, the modeler computes solutions without asking the designer anything. If it doesn't have enough data, it chooses values with a default strategy.
- in manual mode, the designer has to give any missing data to the modeler.
- in any mode between the two previous ones., according to the kind of scenes, the designer gives some particular data and lets the modeler choose the rest.

Each insight phase allows the designer to analyze the solutions and the intermediate scenes which are displayed. During this phase, he can note if scenes don't respond to the problem as well as he would like. This situation would mean that the description isn't really suitable. So, he can modify, update or change this description.

The process can be represented by Fig. 6. It stresses the succession of designer's actions between the initial description (D0) and the final solutions (S). Each Di represents further informations given by the designer or a new release of specifications. Mi are intermediate models or solutions (complete models) produced by the modeler which the designer discovers during the insight phase. The spiral converges to the final solutions kept by the designer.




| | | |
|---|---|---|
| Di | : | Description |
| Mi | : | Solutions or partial model according to Di |
| C | : | Computation |
| U | : | Understanding |
| D | : | Description of the final solution set |
| S | : | Final solution set |

Fig. 6: Design process of solutions

This design process model for declarative modelers is valid even for conventional

modelers. There is no opposition between the two kind of modelers. Declarative modelers are extensions of the conventional ones. In fact, when declarative modelers are used with a very manual functioning mode, they behave like conventional modelers because the designer has to give all specifications. The main difference on the spiral is the nature of available tools. The level of declarative tools is higher than that of conventional tools. They allow the designer to work with a higher level of abstraction.

## WORKING MODES

### Introduction

The design process we modeled in the previous part allowed us to use declarative modelers in different ways [Colin et al. 97]. So, a user can conceive different scenes using different approaches, different working modes. These working modes depends on the designer's uses, the declarative modeler's possibilities, and the kind of problem the designer has to work on. With the first declarative modelers, we noted that the designers were usually using two working modes: an "automatic design" mode and a "draft path design" mode. In this part, we will describe these two modes and propose another one which is more general, more convenient and more powerful. When we take into account this new working mode, we can bypass the limitations of current declarative modelers. This mode is the seed of the new generation of modelers we develop.

### Working mode: "Automatic design"

The designer gives his description to the modeler in one step. The generation process uses only the informations given at the beginning. The designer doesn't interact with the program until the computation is done. If the modeler needs more informations for its computation, it has to use default strategies for these missing data. The modeler never stops between the description and the obtaining of the solution set, to ask for any additional data (Fig. 6). The designer cannot interact with the program to guide it for searching solutions. When the computation of the solution set is done, he can change his description and restart the generation. This is the design process loop.

This working mode was especially used with the first declarative modelers [Colin90, Martin-Martin93, Martin90]. They allowed designers to create scenes which had a very simple description. However, with more recent modelers, the designer can still use this approach when objects aren't essential to his scene, or when he doesn't want a hard control on their creation. For example, when a designer is creating many people in the background of a scene, he doesn't need to control the generation of each person.

### Working mode: "draft path design"

Describing a scene that has many details is often tedious. Giving descrtiptions of all items in one step can be very hard. To avoid this problem, the designer builds his scenes step by step. He first describes a rough scene, giving a more precise description at each step. This means that the first description is usually a very general one. Then the modeler shows him a set of "level 1" solutions, that correspond to his request (Fig. 7). These are very rough, based on the final solution he is looking for. If the scenes he obtains do not correspond to those he would like, that means that the description he gave isn't really what he wanted. In that case, he can modify the description he gave at that level and build new scenes. When he is satisfied with his solution set, he selects one solution that will be the seed of the scenes he wants to build. This solution is called a *draft*. Using this selection, the designer guides the search for solutions, because he rejects the other ones, and each solution issued from them. Then, he gives a new description which consists of the draft and more precise description items. The modeler computes a "level 2" solution set, which correspond to this new description. The scenes which are build are more precise than those at level 1. The designer selects one "level 2" solution which becomes the new draft. He gives a more precise description which will lead to a higher step scene, a more detailed scene. The process is repeated until he obtains the final scenes. The process which computes a "level i" solution set according to a "level i" description and also to the designer's wishes corresponds to a problem of declarative design using an "automatic design" working mode.

This approach consists of building scenes according to several levels of detail. At each step, the level of detail increases, and scenes are built with a higher level. This working mode with drafts can be applied only with modelers which manage levels of detail or with a set of different modelers in whiwh each works on one level of detail. In this case, a particular module acts as a "conductor" and deals with the chain of modelers. If there is no conductor, the designer is in charge of chaining modules.

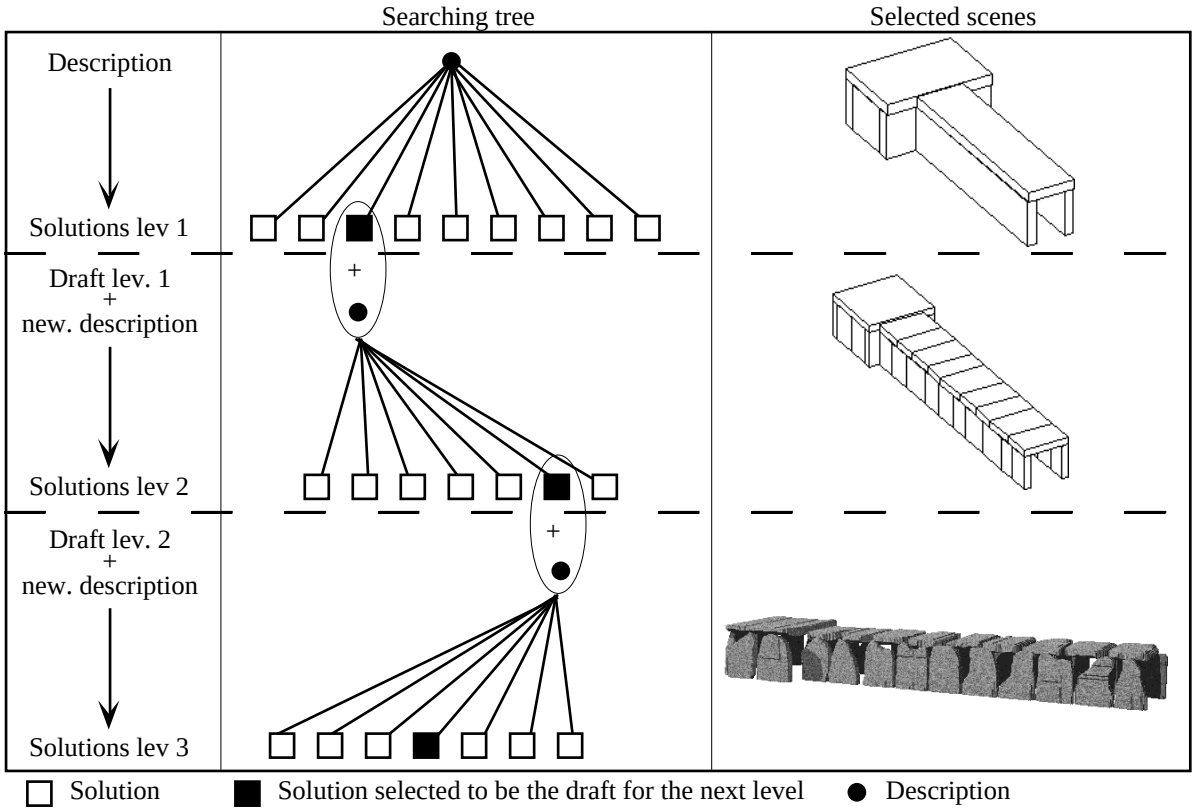



Fig. 7: Working mode “draft path design”

The first design programs using the working mode “draft path design” we developed consisted of a set of modelers. Each of these modelers used an “automatic” mode. The designer wasn't allowed to interact between the description phase and the corresponding solution set building phase. He was only allowed to interact in selecting the drafts, and to give more precise description. Usually, however,a designer wants to control a bit of the scenes building to limit the scope of the solution set research. So, he often agrees to give additional data to the modeler, when it hasn't got the ones it needs. Therefore, the modelers using this working mode must complete missing data using any default choice strategy, and using dialogues with the designer.

Of course, the working mode is in accordance with the one in Fig. 6. The design process of final scenes needs several steps. Each of them corresponds to a part of the spiral. The first turns (Fig. 8) represents the design of level 1 solutions. E1 is the selected draft validated by designer. It is a descriptive item for the design of scenes at level two (bold turns in the spiral). More generally, Ei is the level i validated draft, and it is used to build scenes at the higher level i+1. The last turns in the spiral correspond to the final scenes building.

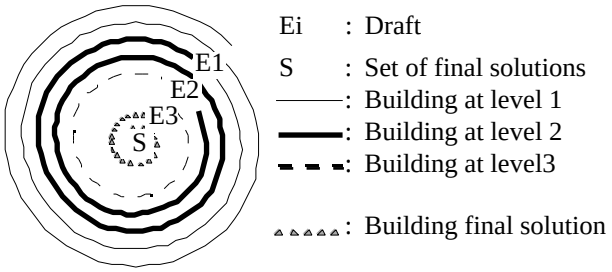



Fig. 8: draft path design spiral

**Working mode “Guided design”**

Even if the “draft path design” working mode is a real improvement for a designer according to the ‘automatic design” working mode, it retains important limitations. Therefore, we propose a new mode for declarative design: “Guided design”. This working mode, which is more general and more convenient, is the seed of the new generation of modelers we are currently developing. It also allows a designer to use modelers with to the two previous working modes, if he so desires.

The draft path design working mode allows us to keep control of the scene building during specific steps. However, this control is quite

rigid. Indeed, the whole scene is modified when we go to the next level of detail. That means that a designer must give the whole description of the scene at each level. Furthermore, all objects in the scene must have the same level of detail. However, sometimes the designer likes the fact that he can change the details of the objects in his scene according to his will, instaed of the modeler's decision. To allow such a mode, the modeler must control objects with different levels of detail. This can include different modeling of objects. For example, he may want to develop one part of the scene and leave the other part as draft. It also implies changing items in the scene according to different modes:

- an imperative mode to add items built by other modelers,
- an automatic mode to let the modeler propose solutions without taking care of the building,
- a draft path mode to control an item each time the level of detail changes,
- a guided mode to get a better control on the building.

**The Guided Design.**

To get a first draft of the scene he wants, a designer gives a description. But, contrary to the draft path design working mode, the designer can give a good description of some items, and leave others as drafts. The description includes the building mode (imperative, automatic, draft path or guided) of each of them. The modeler proposes a set of scenes that respect the description, and also the building mode. This means that objects built using the automatic modes are displayed when they are completely finished. Those using the draft path mode will be displayed at a given level of detail. Among the scenes that are displayed, the designer selects one, completes its description of every one (or some) of the items in the scene, and makes the program go on. He can also freeze some items he likes [Liege-Hegron97]. These items will not be modified during the next generation. Freezing all items in the scene except one, he can explore what is possible for this element. Insight tools can help him to understand which part of the scene has changed, compared with the previous version.

The designer changes the description according to his rhythm. He has great control of the research of solutions, and of building of the scenes. This working mode is very close to the one he uses with a conventional modeler. The main difference is that the modeler proposes solutions. The research is faster, more precise. However, the possibilities to build unexpected scenes are poorer. The designer must find a good compromise between a strong control and any originality he may want. Of course, this depends on the designer, on the kind of problem he is working on, and on the way he wants to solve it. This working mode a very close to "Computer aided design" in the truest sense of the word "aided".

A scene is build by describing its items step-by-step. At each time ("t"), a scene is made of items using levels of detail which may or may not be the same. In Fig. 9, the tree represents the scene at a given level. Nodes are representations of each item at different levels of detail. At time ("t"), the scene is made of the leaves of the tree (black nodes in Fig. 9) which are items at there current higher level of detail. However, it is still possible to display the items at lower levels of detail. To do this, one of the ancestors of the leaf must be displayed instead of the leaf. A display will show the scene, made of different levels of detail, to the designer. Note that in Fig. 9, the designer didn't want to describe the second node of the first level of the tree more precisely. He can do so when he wants.

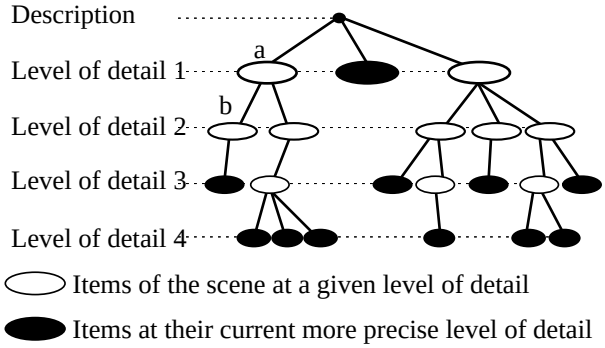



Fig. 9: Scene at time ("t")

A node is an item at a given level of detail. The tree doesn't represent a hierarchic scene, but a hierarchy of levels of detail. The children of a node aren't a decomposition of the node, but a more precise representation of it. This can be obtained by adding more precise details (node 'b' and its son) or by changing some items (node 'a' and its children). The type of the node can be different from its children' one. For example, a designer can first describe some boxes to define the position of the items. Then he can give a more precise description. The children of this node can be a table and chairs.

This working mode includes the draft path design working mode, and the automatic

working mode. For the first one, we have to force the building of the scene by levels. All the items are build with the same level of detail. The building of each of them is done with the same mode. The automatic working mode consists in giving solutions only when they are completely built. As the designer must not interact, an automatic building mode is chosen for each item.

## CONCLUSION

Using the programs already developed, we proposed a model of declarative modeler according to a designer. This model takes into account the aspects already implemented in the current experimental applications. Nowadays, none of the modelers which were developed has all the functionalities of this model. However, we also want to point out several advantages of this model:

- it shows the main functionalities of declarative modelers and the advantages in their use. It offers a general view of a declarative modeler to a devoted user as well as to a developer.
- It is a guide to develop declarative modelers. It shows developers how a declarative modeler works, and what main functionalities he has to program.
- Currently, one of our activities is developing generic declarative modelers. This will help a developer build his declarative modeler by specifying the domain and the items he is working on. In this scope, our model is a good framework for these applications.

We also presented three working modes which the designer can use with declarative modelers. The first two have already been implemented. The third one is the one designers dream of, but current declarative modelers don't allow the full guided design mode yet. Between the two last modes, there are intermediate modes and some of our modelers are of that type.

Finally, note that the description of this mode will help to build a new generation of declarative modelers, more convenient and more powerful.

## ACKNOWLEDGEMENTS

We are grateful to Judy Ellen Corcoran and Vidal Martin for reviewing previous version of the present paper.

We also want to thank people of the GEODE group for their help and their remarks.

## REFERENCES

Champciaux L. 1997 *Declarative Modeling, speeding up the generation*, CISST'97, July.

Chauvat D. 1994 *Le projet VoluFormes: un exemple de modélisation déclarative avec contrôle spatial (the VoluFormes Project, an example of declarative modeling with spatial control)*, PhD dissertation, p. 225.

Colin C. 1990 *Modélisation déclarative de scènes à base de polyèdres élémentaires (Declarative modeling of scenes using basic polyhedra)*, PhD dissertation, p.266.

Colin C., Desmontils E., Martin J.Y., Mounier J.P. 1997 *Modèle Utilisateur d'un Modeleur déclaratif (User model of a declarative modeler)*, JMG 97, Grenoble.

Kochhar S. 1990 *Cooperative Computer-Aided Design: A paradigm for automating the design and modeling of graphical objects*, PhD dissertation, p.117

Liège S., Hegron G. 1997 *An incremental declarative modeling applied to urban layout design*, WSCG'97, February, Pilsen.

Lucas M. 1991 *Equivalence classes in object shape modelling*, IFIP TC95/WG 5.10, Working Conference on Modeling in Computer Graphics, Tokyo, April.

Lucas M., Desmontils E. 1995 *Les modeleurs déclaratifs (Declarative modelers), Revue internationale de CFAO et d'informatique graphique*, Vol. 10, no 6, December.

Martin P., Martin D. 1993 *Declarative generation of a family of polyhedra*, Graphicon'93, September.

Martin J.Y. 1990 *Synthèse d'images à l'aide d'automates cellulaires (Image synthesis with cellular automata)*, PhD dissertation, p. 228.

Mounier J.P. 1996 *Modélisation déclarative de parcours pour l'analyse des ambiances architecturalkes et urbaines (Declarative modeling of courses for architectural and urban environment analysis)*, AFIG'96, November.

Pacholczyk D., Desmontils E. 1997 *A qualitative approach to fuzzy properties in scene description*, CISST'97, July.

Poulet F., Lucas M. 1996, *Modeling Megalitic sites*, Eurographics'96, September.

Rau-Chaplin A., MacKay-Lyon B., Spierenburg P. 1996 *The LaHave House Project : Towards an Automated Architectural Design Service*, Cadex'96 pp. 24--31

Sellinger, Plemenos D. 1996 *Integrated geometric and declarative modeling using cooperative computer aided design*, 3IA'96, Limoges.

Woodbury R. 1991 *Searching for Designs : Paradigm and Practice, Building and Environment*, Vol 26, no 1, pp.61-73