Controlling Object Natural Behaviors in a 3D Declarative Modeler

Kwaiter Ghassan Gaildrat Véronique Caubet René
Institut de Recherche en Informatique de Toulouse
Paul Sabatier University; 118 route de Narbonne, 31062 Toulouse Cedex, France
Phone. (33) 5 61 55 83 29, Fax. (33) 5 61 55 62 58
E-mail : {kwaiter,gaildrat,caubet}@irit.fr

Abstract

In this paper, we present an extension of DEM2ONS declarative modeler. DEM2ONS provides the designer with the possibility to easily describe a scene in a high level of abstraction, using various types of constraints. It also integrates knowledge about normal use of objects in order to offer the closest solution to what the designer wants. Constraints are maintained by a new incremental constraints solver called ORANOS, that dynamically satisfies the system when a designer's intervention occurs. It integrates constraint hierarchies to deal with over-constrained problems.

1. Introduction

Declarative modeling is a global approach of the designing process. One can distinguish three phases: description, generation and understanding. First phase allows the designer, using constraints, to describe scenes by only defining a set of relations among objects in a high level of abstraction where the underlying geometrical model of the scene is hidden from the designer's point of view. In consequence, a constraints solver must maintain these constraints and provide the closest solutions to a designer's intention.

In current literature, several approaches have been attempted to describe the scene in a declarative manner. A first approach uses traditional geometrical models of common CAD systems [1]. This approach forces the designer to use imperative methods in a bottom-to-top design methodology; thus it won't be suitable when the design scene becomes more and more complex or when the designer thinks about it in a more abstract and semantic way. A second approach uses iterative approximation techniques, such as classical relaxation or Newton-Raphson techniques, to solve linear and non-linear constraints [2]. However, this approach has a major problem; it can be very slow to converge as the number of constrained variables grows up, and it may give one solution. Recently, some considerable researches in this area have addressed the artificial intelligence domain; an extended technique, which is well known as *Constraint Satisfaction Problem*, either to generate one or multiple solutions [3], or to reduce execution time [4,5].

A truly declarative modeler should provide a rich user interface allowing the designer to easily interact with the studied model. It should contain an incremental constraints solver to dynamically handle designer's interventions. This solver must solve crucial over and under-constrained problems, and take into consideration semantic constraints and relationships among objects. However, most of requirements have generally been ignored in previous declarative modelers works.

In this paper, we present a new declarative modeler called DEM2ONS (DEclarative Multimodal MOdeliNg System). DEM2ONS provides the designer with the following possibilities:

- Describing easily 3D scenes in a high level of abstraction using different type of constraints.
- Delegating to the incremental constraints solver, ORANOS [6], the task to dynamically satisfy the system as a designer's intervention occurs.
- Reacting as closely as possible to the hints of the designer, by integrating knowledge about normal use of objects.

The purpose of this paper is to show a view of our declarative modeler. Some concepts are not exhaustively discussed, references are provided to find more details. The paper is organized as follows: Section 2 gives an overview of the structure of our modeler DEM2ONS. Section 3 presents the design principles of our constraints solver. Section 4 presents various types of constraints. Section 5 defines some of the important object attributes. Section 6 explains our computation method. Section 7 shows an implementation, and finally the conclusion.

2. System overview

DEM2ONS is composed of two main parts: a multimodal interface and a 3-D scene modeler (cf. fig. 1).

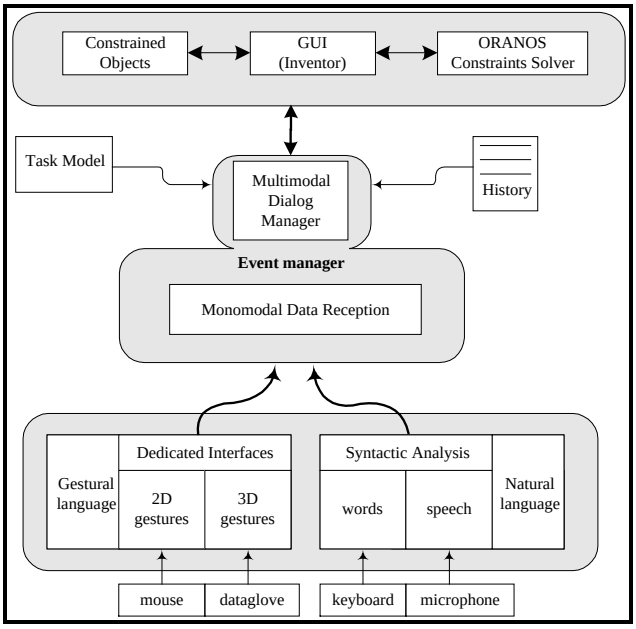

Figure 1. Architecture of DEM²ONS.

2.1. Multimodal interface

The DEM2ONS multimodal interface allows the designer to easily communicate with the system throughout multiple combined ways. Several *input devices* (such as dataglove, speech recognition system, spaceball, mouse...) are provided. *Syntactic analysis* and *Dedicated Interface modules* [7,8] analyze and control the low-level events in order to merge them into multimodal events. These events are collected and treated by an *Event Managing module* that provides a single command to be sent to the modeler. Moreover, it uses a *Historical List* to solve anaphora problems. In this way, the designer will be able to interactively add predefined objects, define the constraint's type, apply constraints among objects and change the strength of a constraint. Consequently, the designer can combine different input modes in order to build a single command. For example: "Put the console against the wall", "Move it to the left", and "Put the view point in front of them", where "it" refers to the same console retrieved by the system from the dialogue history.

2.2. The 3D scene modeler

The 3D constrained objects are instantiated from several abstract classes using object oriented techniques and are modeled and rendered using Inventor ToolKit [9]. ORANOS handles constraints passed either through the Inventor or the Events Manager. It maintains the system, and transmits the evaluated variables to Inventor in order to update constrained objects attributes.

3. ORANOS constraint solver

This section outlines the various design principles and different phases of our constraints solver. However, in order to give a global view of ORANOS, some related concepts are not deeply explained, they refer to their references.

3.1. Design principles

A number of design principles are based on the fact that ORANOS must be integrated in a declarative modeler, and used in an interactive application.

Generally: ORANOS is a domain independent approach; it can be applied in declarative modelers, Computer Aided Design Systems (CAD), and layout aspect in a user interface. It extends the constraint satisfaction problem [10] by using an interval propagation technique to satisfy both linear equality and inequality constraints. For instance, it supports geometric constraints expressed by linear equality equations (such as distance constraint, parallel constraint and horizontal constraint); as well as topologic constraints that correspond to the relative positioning of objects expressed by linear inequality equations (such as object *A* is to the left of object *B*).

Avoiding inconsistency: in declarative applications, halting and error signaling are not normally acceptable when the constraints solver encounters an over-constrained or an under-constrained problems. ORANOS is based on constraint hierarchies [11] in order to deal with over-constrained problem. It allows the designer to attach arbitrary strengths to the constraints, indicating how strongly the designer wants particular constraints to be satisfied. Given an over-constrained hierarchy, ORANOS leaves the weakest constraints unsatisfied in order to satisfy the strongest ones. If under-constrained problem occurs, DEM2ONS proposes only one solution, and the designer can as well ask the modeler for other possible solutions. It controls the produced solution either by changing the strength of existing constraints or by adding additional ones.

Incrementally: one important technique in declarative applications is the dynamic changing of the relationships among the objects. In such applications, user would be able to add more constraints on objects, or remove some constraints from the objects to release them to a certain degree. He would also have the possibility to weaken the strength of a constraint to satisfy the other stronger ones, or inversely, strengthen a particular unsatisfied constraint to emphasize it over the others. For all these requirements, ORANOS re-satisfies dynamically the constraints set when

new constraints are added, existing constraints are removed, or a constraint's strength is changed.

3.2. ORANOS phases

This section sketches out the high levels of ORANOS algorithms. The reader should refer to [12] for more details.

3.2.1. Data structures: A *constraint* object class is specified by its strength, a finite set of constrained variables, methods, and a history field containing a set of variables whose bounds are reduced when this constraint is satisfied. A *variable* object class has boundary values that represent the lower and upper bounds of a closed continuous interval and an instant of the variable. A *method* object class has an input field containing a list of the input variables, an output field containing a list of the output variables, and a code field that implements this method. Finally, the system uses three global lists whose elements are in descending order depending on the constraint's strengths. *Satisfied Constraints List* that contains the consistent constraints. *Unsatisfied Constraints List* that contains the inconsistent constraints. *Retracted Constraints List* containing the weakest strength constraints that will be retracted from the Satisfied Constraints List in order to satisfy constraints with greater strengths.

3.2.2. The entries phase: ORANOS is called when a designer's intervention occurs to add, remove, or change the strength of a constraint. Since ORANOS saves the modified variables domains for a constraint after satisfying it, only a local sub-constraints set will be treated when a new constraint is added, or existing constraint is removed.

Adding constraint: If ORANOS is invoked to add a new constraint, it retracts from the *Satisfied Constraints List* the constraints whose strengths are less than the strength of the newly added constraint. It puts them with the new constraint in the *Retracted Constraints List* in a descending order. This order guarantees that no satisfied constraint will be retracted again to satisfy any following constraint. Then, the *Retracted Constraints List* is passed to the refinement phase.

Removing constraint: If ORANOS is invoked to remove a satisfied constraint, it is possible to satisfy some other unsatisfied constraints. This possibility will be given to those constraints that share variables with the removed one. Thus, ORANOS retracts from the *Satisfied Constraints List* the constraints whose strengths are less than the strength of the removed one. It puts them with the related unsatisfied constraints in the *Retracted Constraints List* in a decreasing order, to be passed to the refinement phase. However, if ORANOS is invoked to remove an unsatisfied constraint, it simply removes this constraint from the *Unsatisfied Constraints List*.

Changing the strength of a constraint: If ORANOS is invoked to change the strength of a constraint, it distinguishes two cases:

- Strengthening an unsatisfied constraint: where this constraint may get a chance to be satisfied, therefore, ORANOS should retract this constraint from the *Unsatisfied Constraints List* and put it into the *Retracted Constraints List*.
- Weakening or strengthening a satisfied constraint: in this case, ORANOS retracts this constraint from the *Satisfied Constraints List* with the constraints whose strengths are less than this one, putting them all into the *Retracted Constraints List*. Weakening a satisfied constraint may give a chance to other constraints in the *Unsatisfied Constraints List* to be satisfied, so only unsatisfied constraints that are equivalents or stronger than this constraint will be added to the *Retracted Constraints List*. Finally, the *Retracted Constraints List* is passed to the refinement phase, in both cases.

3.2.1. The refinement phase

In this phase, ORANOS attempts to satisfy all the retracted constraints in the *Retracted Constraints List*; it removes each constraint from this list and calls a *Global Propagation Filtering* algorithm to satisfy it [13]. The Global Propagation Filtering algorithm tries to satisfy the given constraint by tightening the bounds around its variables employing the method associated with the constraint; and then propagates these variables to the constraints that have already been satisfied in the *Satisfied Constraints List*. If the constraint is satisfied, it is added to the *Satisfied Constraints List*, otherwise, it is added to the *Unsatisfied Constraints List*. The algorithm iterates for all the constraints in the *Retracted Constraints List*, removing them from the list and adding them to either Satisfied Constraints or Unsatisfied Constraints lists. When the algorithm terminates ORANOS passes to DEM2ONS only the variables whose interval domains are being reduced; thus, a few object attributes will be updated; i.e. a local change will take place on the scene.

4. Constraints

In DEM2ONS, constraints may be applied to an object from different sources: either during object modeling, or as a result of its relations with other objects. A distinction between two constraint types can be made: *internal* and *external* constraints.

Internal constraints are applied on one object and used to define its intrinsic properties such as its position, orientation, or fixed dimensions without any relation to other objects. They may be applied implicitly (like a dimension constraint, considered as an internally required

constraint that the designer can't violate), or explicitly (like position and orientation constraints).

On the other hand, an external constraint is applied between two objects and has a unique representation:

" N_{target} is spatial preposition $N_{landmark}$ with Strength "

Where subject N_{target} refers to the located object and the subject $N_{landmark}$ refers to the site object. Strength refers to the constraint's strength, and spatial preposition refers to the locative prepositions used. A designer can use several locative prepositions available in DEM2ONS, such as: in front of, behind, against, on the left of, or a distance x of, in order to express a relationship between two objects.

However, some spatial prepositions have also an implicit *functional constraint* [14]. The functional constraint comes simply from the functional factors of the objects themselves and their interactions with other objects. For example, constraints such as: "put the cupboard against the wall", or, "put the chair in front of the table" have both geometrical and functional constraints. The first constraint is applied on the position attribute of the object (cupboard, chair), and the second one is applied on its orientation. Thus, a constraint must integrate the functional role of the object, and reflect this functionality in a geometric way.

Object-oriented techniques, with encapsulation and inheritance, provide DEM2ONS with important mechanisms to define various types of constraints. Each constraint refers to a particular class that contains a particular method to satisfy it. Thus, meanings of each constraint are accomplished by using constraint constructors. Moreover, DEM2ONS uses *constraint inheritance* to deal with complex constraints. For instance, figure (2) shows a hierarchy of constraint classes.

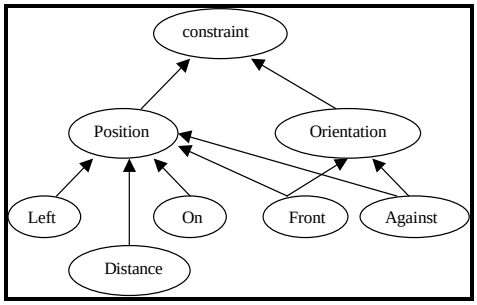

Figure 2. Constraint Classes.

The class "Constraint" is an abstract class that contains information about target and landmark objects, and strength of the constraint.

5. Objects

Each object refers to a particular class that describes the properties of each instance objects. Thus, we can differentiate between objects that are always landmarks (like floor or walls), and objects that might be both landmarks and targets. Shared attributes, such as type of object, position, volume, and bounding box are inherited from an ancestor class. In this section, we only treat of the orientation and positioning zone attributes.

Orientation attributes: *intrinsic* and *deictic* orientations are some of the essential attributes, they define the functional level of an object. The orientation processes, which is greatly conditioned by functional features, consist of making correspondence between an abstract orientation and a concrete one; i.e., the intrinsic front of a computer is derived from the usual position of its screen.

Three attributes are attached to an object to express the three intrinsic orientations, if exist: up, front, and left orientations. An object is always described in its own local coordinate space. The upper direction attribute coincides with the gravitational upper direction (for instance: bottle, chair and table). The front and left directions are defined from a tandem or mirror canonical use of objects (for instance: the front direction of a TV is well known).

Furthermore, the designer's location is required to define the deictic orientations of the landmark object, and in consequence to define the position and the orientation of the target object. In the following example (Figure 3), for the constraint: "the chair in front of the table", the table has only up intrinsic orientation, so the modeler uses the designer's location to define the chair's position and orientation. A new constraint "the chair in front of the table" is applied on another chair according to a new designer's position.

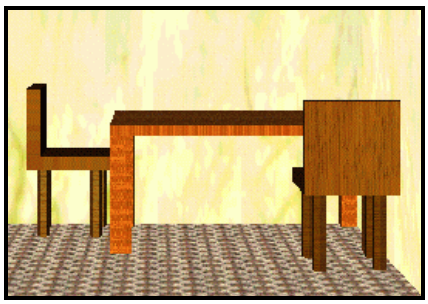

Figure 3. Deictic positioning of a target object.

Positioning zone of a target object: by definition: zone of space that satisfies all the constraints applied on this object, considering it as a target object. Each constrained object in the scene has a positioning zone, since it is constrained somehow; for example: table, vase, and chair always have a landmark object that supports them.

In mathematical terms, the positioning zone corresponds to three variables, each one is an interval of continuous values that have a lower and upper bounds.

$$\text{Positioning zone} = \{[X_{min}, X_{max}], [Y_{min}, Y_{max}], [Z_{min}, Z_{max}]\}$$

Each interval presents the possible positions of an object according to each axis. Thus, relative positioning of an object into the scene corresponds to a set of values for all the interval variables.

6. Positioning task

When a designer applies a constraint, DEM2ONS creates an instance of this constraint, according to its type, and passes it to ORANOS that tries to satisfy it by executing its method. The method determines the target object's positioning zone, in light of landmark object's position and orientation, applying the following steps:

- First, it transforms the target object's positioning zone into the landmark object's local coordinate.
- Second, it refines each interval variable in the target object's positioning zone, if needed.
- Third, an inverse transformation is applied to obtain the refined target zone in the world coordinate system.

Figure (4) explains the method for a constraint such as "Object *A* is in front of Object *B*".

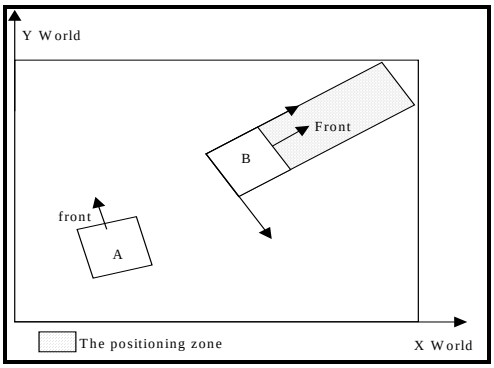

Figure 4. Positioning zone of a target in light of the landmark's location.

If the constraint is satisfied, every interval's variable of the target object's positioning zone is not empty. ORANOS will pass a message to DEM2ONS containing the new refined interval domain for all the variables whose intervals have been affected by this constraint. Enumerating all the possible solutions is a *NP-hard* problem. So DEM2ONS gives only one solution in order to satisfy the entire *Satisfied Constraints List*. Then, DEM2ONS re-instantiates only the variables whose interval's domains are reduced and whose previous instantiations are not included in the new refined intervals. Thus, few object attributes will be updated and a local change will take place on the scene. Moreover, the designer can as well ask DEM2ONS for other possible solutions, control the produced solution either by changing the strength of an existing constraint or by adding a new constraint.

Others approach as [2,15] have main drawbacks: First, all objects are always orthogonal oriented, thus, it does not take into account natural orientations of objects in the scene, and consequently relative positioning is not natural. And, expressing each constraint by three primitive constraints is memory-consuming and makes the constraints solver relatively slow, when the number of constraints grows up.

On the other hand, our approach deals efficiently with these drawbacks: it takes into account the orientation of the landmark object to naturally place the target object and uses only three variables to represent the object's positioning zone.

7. A DEM2ONS implementation

Objects have various uses depending on the context. Dealing with any context is a too large domain to be studied. So, we restrict our study to the fitting-out of a particular scene: the founded model will be, in a great part, transposable to another scene type. For instance, a chair and a table have always the same function wherever they are placed.

We chose to fit-out a living room because it is a familiar scene, giving to the designer a large freedom to place objects as he wants, respecting their normal use. The living room is initially composed with the floor, the walls, and the roof. The designer tries to furnish the living room simulating natural processes in the real world. He adds and removes objects, applies constraints to them, strengthens unsatisfied constraints in order to make them satisfied.

Figure (5) shows the previous scene with two added objects: a small table and a lamp. The designer applies three constraints:

1- Put the small table on the floor
2- Put the lamp on the small table
3- Put the small table at the right of the table

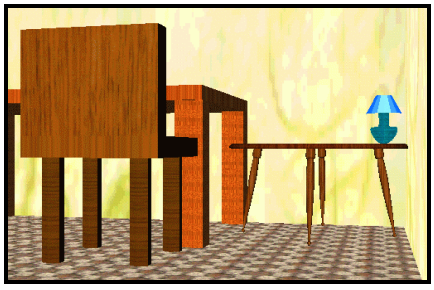

Figure 5.

In this example, when the designer adds the small table into the scene, an available positioning zone must be

found. For that, the bigger table is moved by the solver to give the possibility to the small table to be placed. If the solver cannot found enough space, the constraint is unsatisfied.

The last examples figure (6) shows a complete scene with various constraints applied to the objects.

Figure 6.

8. Conclusion

This paper has presented a new version of our declarative modeler DEM2ONS that offers to the designer an easy way to describe a scene in a high level of abstraction using various types of constraints. It integrates a new constraints solver, ORANOS, based on an efficient refinement model that supports linear equality and inequality constraints organized in constraints hierarchy to deal efficiently with over-constrained problem.

Our implementation deals with a particular scene. Even the proposed context will be, in a great part, transposable to another scene type. We hope to explore the ability of our declarative modeler to support objects with large domain contexts.

Acknowledgments

The authors would like to thank other members of the team: Frederic Rubio, Olivier Balet, for their contributions in implementing several parts of our modeler. Thanks to Olivier Lhomme who has been generous with his time in explaining the inner functionality of his global consistency filtering algorithms. Thanks to Jean-Christophe Aurisset and Fouad Jennawi, for discussions and comments on an earlier draft of this paper.

References

[1] Light R., and Gossard D., "Modification of geometric models through variational geometry", *Computer Aided Design*, V. 14, N. 4, PP. 209-214, July, 1982.

[2] Donikian S., and Hégron G., "A Declarative Design Method for 3D Scene Sketch Modeling", *EUROGRAPHICS'93*, V. 12, N. 33, PP. 223-236, 1993.

[3] Liege S., and Hégron G., "An Incremental Declarative Modelling applied to Urban Layout Design", *WSCG'97*. V. 2, PP. 272-281, February, 1997

[4] Plemenos D., and Tamine K., "Increasing the efficiency of declarative modelling", Constraint evaluation for the hierarchical decomposition approach. *WSCG'97*. V. 2, PP. 414-423, February, 1997

[5] Champciaux L., "Declarative Modelling : speeding up the generation", *CISST'97*, Las Vegas, Nevada, USA, June 30-July 3, PP. 120-129,1997.

[6] Kwaiter G., Gaildrat V., and Caubet R., "ORANOS : Un Solveur Dynamique de Contraintes Hiérarchiques et Géométriques", *MICAD'97, International Journal of CADCAM and Computer Graphics*. V.12, N. 1-2, PP. 139-152, 1997.

[7] Caubet R., Djedi N., Gaildrat V., Rubio F., Pérennou G., and Vigouroux N., "NOVAC: A drawing tool for blind people", *Interface to real and virtual worlds*, France. 1992

[8] Balet O., Gaildrat V., and Caubet R., "Le geste comme moyen d'expression", *Proceedings of 3IA'94*, PP. 177-185, April, 1994.

[9] Strauss S., and Carey R., "An Object-Oriented 3D Graphics Toolkit", *Computer Graphics*, V. 26, N. 2, PP. 341-349, July, 1992

[10] Mackworth A. K., "Consistency in networks of relations", *Artificial Intelligence*, V. 8, PP. 99-118, 1977.

[11] Borning A., Benson B-F and Wilson M., "Constraint Hierarchies. Lisp And Symbolic Computation", *An International Journal*, V. 5, PP. 223-270, 1992.

[12] Kwaiter G., Gaildrat V., and Caubet R., "Dynamic and Hierarchical Constraints Solver with Continuous Variables", *JFPLC'97.,Sixth French Conference on Logic Programming and Constraint Programming*, PP. 1-11, Orlréans, 26-28 may, 1997

[13] Lhomme O., "Consistency Techniques for Numeric CSPs", *Proceeding of the 13th international Joint Conference on IA '93*, PP. 232-238, 1993.

[14] Aurisset J.C., "Interpretation of Spatial Orders in Natural Language : Integration of Functional and Pragmatical Factors", *Fifth International Conference on the Cognitive Science of Natural Language Processing.* Dublin, 2-4 September, 1996.

[15] Ligozat G., "Toward a General Characterization of Conceptual Neighbourhoods in Temporal and Spatial Reasoning", *Proceeding. of the AAA-94, Workshop on Spatial and Temporal Reasoning*, PP. 55-59, Seattle, Washington, USA, 1994.