Lecture Notes in Computer Science

A new approach to the Interactive Resolution of Configuration Problems in Virtual Environments

Carlos Calderon[1], Marc Cavazza[1], and Daniel Diaz[2]

[1] School of Computing and Mathematics, University of Teesside,
UK-TS1 3BA, Middlesbrough, United Kingdom
{c.p.calderon, m.o.cavazza}@tees.ac.uk
University of Paris 1
90 rue de Tolbiac, CRI bureau C1405 Paris, 75013, France
Daniel.diaz@univ-paris1.fr

Abstract. Intelligent Virtual Environments integrate AI techniques with 3D real-time environments. As such, they can support interactive problem solving, provided the underlying AI techniques can produce solutions within a time frame matching that of user interaction. In this paper, we describe an intelligent virtual environment based on Constraint Logic Programming (CLP), integrated in a real-time 3D graphic environment. We have developed an event-based approach through which user interaction can be converted in real-time into appropriate solver queries which are then translated back into automatic reconfigurations of the Virtual Environment (VE). Additionally, this framework supports the interactive exploration of the solution space in which alternative solutions (configurations) can be found. We demonstrate the system behaviour on a configuration example. This example illustrates how solutions can be interactively refined by the user through direct manipulation of objects in the VE and how the interactive search of alternative solutions in the VE is supported by these type of systems.

1 Introduction

Virtual Environments (VE) can be used to represent complex situations. It is thus a logical evolution to extend them to carry out problem solving tasks as well. In these systems, called Intelligent Virtual Environments [1] the visual space is directly interfaced to a problem solver. Interacting with objects in the VE serves as input to the problem solver, which outputs new solutions directly as, in this instance, object configurations. This kind of system has many potential applications in design, configuration, situation assessment, etc.

This research aims at the integration of interactive visualisation with interactive problem solving in virtual worlds. A central argument to this paper is to reach a high-level of integration between the visualisation component and the problem-solving component. This can be achieved by selecting a problem solving technique compatible with the nature of user interaction in Virtual Reality (VR) (including its time-scales) and integrating them using an interaction model.

Our problem-solving component is based on Constraint Logic Programming (CLP). Generally speaking, constraint programming is an optimisation technique that provides solutions in terms of relations between domain variables. The solving mechanisms of many constraint systems can be triggered by any modification of variable values. More specifically, physical interaction with the virtual world objects can be translated into real-time input to the CLP solver by selecting the variables whose values have been altered by the interaction. This supports data-driven changes, and provides a strong basis for its integration in an interactive VR system. For instance, when visualising a configuration, the user can alter the position of certain objects which modifies the constraints involving these objects. This triggers the solver on a new set of variables. The solver in turn outputs resulting solutions in the form of new object configurations within the environment, thus seamlessly integrating the solving process in the VR interaction loop.

Previous work using constraint logic programming in 3D environment has essentially been dedicated to the behaviour of individual objects or autonomous agents within the environment. For instance, Codognet [2] has included a concurrent constraint programming system into VRML (VRCC) to specify the behaviour of artificial actors in dynamic environments. Axling et al. [3] have incorporated OZ [4], a high-level programming language supporting constraints, into the DIVE [5] distributed VR environment. Both Axling and Codognet have put emphasis on the behaviour of individual objects in the virtual world and did not address user interaction or interactive problem solving. However, CLP naturally provides solutions for combined behaviours, in this instance, for sets of objects. This is the property we use to implement behaviours for the virtual environment as a whole.

As demonstrated by Honda [6] and Pfefferkorn [7], CLP techniques are particularly appropriate for the resolution of configuration problems. Both have demonstrated the suitability of CLP as an approach based on: the declarative nature of the formalism, which facilitates the description of the problem, and its efficiency for avoiding combinatorial explosion. Our approach extends Honda and Pfefferkorn into the realm of 3D real-time environments and demonstrates that CLP techniques can produce solutions within a time frame matching that of user interaction. Consequently, seen from the user's perspective, the VE reacts to the user's interaction by reconfiguring itself. Furthermore, once an initial solution has been found, the framework we propose in this paper enables the user to interactively search for alternative solutions in the VE.

2 System Overview

The system is an interactive 3D environment, developed using the Unreal Tournament™ (UT) game engine, in which the user can freely navigate and interact with the world objects. That is, the system initially proposes a first solution (in the form of a configuration of objects) which serves as a starting point for user's exploration of possible configurations. Once the user has explored this configuration, he can interact with it by displacing the constituent objects. The correct allocation of an object in-

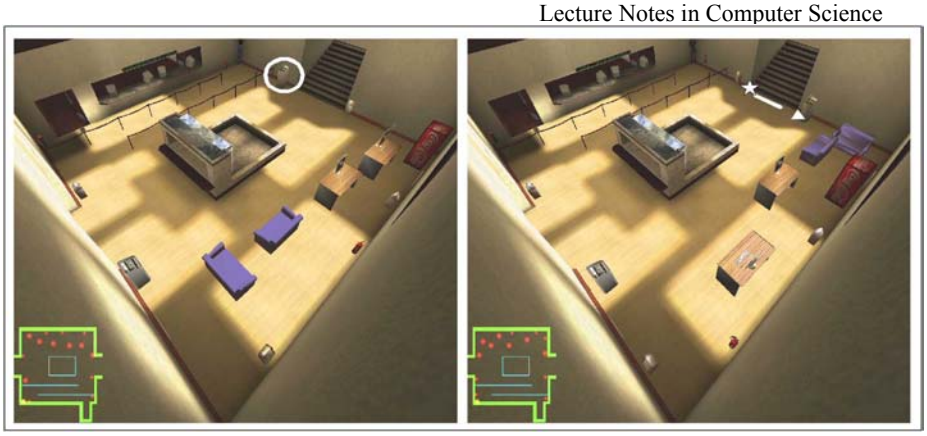

Fig 1. The user refines a given configuration by reallocating an object. As a result of this, the environment reconfigures itself.

stantly triggers new solutions (configurations) from the solver which, in turn, are displayed in the virtual environment (see Figure 1).

The intelligent configuration module (the solver) is based on Constraint Logic Programming (CLP). We have used GNU Prolog [8] as a programming environment, which contains an efficient constraint solver over Finite Domains (FD). This allows the implementation of many different types of constraints which can be represented over a finite domain, i.e. an ordered list of properties. This makes possible to represent "semantic" constraints, i.e. constraints involving object properties such as materials, friction coefficient, resistance to fire, etc. In the following sections, we give a more detailed insight into the implementation considering the specific techniques used.

3 The Interaction Model

The aim of the interaction model is to relate the user interactions which are taking place in the virtual environment to the I/O of the solver, within an interaction rate which is that of the user. The solver works with a specific logic, according to which it is triggered by the addition of new clauses and its results are produced as sets of variable allocations. The interaction model should bridge the gap between these two different logics, from the dynamic perspective of the user's intervention. We thus discuss two aspects: firstly how the solver is made to react to user input, secondly how results produced from the solver should interactively modify the virtual environment.

Unreal tournament supports the real-time generation of events, i.e. those generated by the user's transformation of a configuration displayed in the virtual environment. These events (see Figure 2) are used to generate new "queries" to the solver (using the term query in its Prolog sense). Consider the following example, where from a deployed configuration the user displaces an object, e.g ATM, to a new position. The event is automatically generated from the user's physical interaction. In this case, when the user drops an object using the low-level interaction features implemented in the game engine, this triggers the corresponding event. From this event, a new query

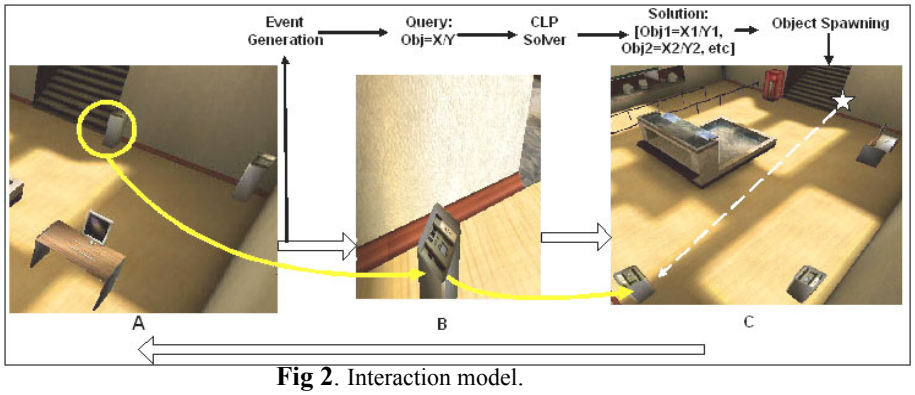

Fig 2. Interaction model.

is passed to the solver which takes the form of a new Prolog fact about the altered object (this would replace older facts about that same object). This new fact is passed as a structured message via a TCP/IP socket.

The solver is triggered by the reception of the new query via the socket: a listening mechanism, a server embedded in the solver, translates the received message into an actual query, that is able to trigger a new solution. The solving mechanism produces solutions one by one, but the search is interrupted after the production of the first solution, as solutions have to be proposed interactively A solution is a set of object positions, whose variables are the targets of the constraint solver.

The set of new object positions is passed as another message via the same TCP socket. In the virtual environment, the reception of this message triggers the corresponding series of low level events specifically implemented in the engine. The end result of these events is the transformation of the accepted message into a new configuration in the virtual environment.

It must be noted that, according to our results, the communication time for the overall cycle is on average less than 20ms, which is fully compatible with the user interaction (as the user is not navigating when interacting with objects).

4 Example

An intelligent configuration system is used as an application example. The data used in this configuration scenario is derived from a simple yet realistic example which uses real-world design knowledge in terms of building interior design for offices (a bank agency in our case). More specifically, the data used for both objects and constraints was drawn from real specifications [9].

In our intelligent configuration system, the spatial relationships between the objects in a layout configuration are all known and the constraints whose formulation depends on these relations reflect those specific spatial relations. Moreover, the objects involved in the configuration have been divided into non-movable objects (e.g. ventilation ducts, sources of heat, etc) and movable objects (e.g furniture: sofas, desks, etc). This is a purely semantic distinction which can be easily reversed. In our case, this means that whilst all objects take part in constraints specifying the design requirements, the user will only interact with the movable objects: the furniture. Consequently, when the user decides to reallocate a movable object, this, in turn, disrupts the imposed constraints in the configuration and forces the system to re-allocate the

remaining movable objects to generate a solution compatible with all the design requirements.

In our case, the movable objects are: one vending machine, two desks (which represent the customer attention area), two sofas (waiting attention area), two automatic teller machines (ATMs), three fire extinguishers and four bins. This constitutes a subset of 14 objects: considering the size of the environment and that the overall size of the available set of constraints for each object is eleven, the corresponding search space (abstract problem space) is substantial and indeed impossible to search systematically, even less so in real-time.

4.1 Problem representation in CLP

The problem knowledge (design requirements in our case) is expressed in the topological description of the 3D environment (non-movable objects or building elements) and in a set of constraints on the objects' attributes (movable objects or furniture, see Figure 3). There are two types of constraints on the movable objects: *local* and *global* constraints. Both types incorporate geometrical as well as more "semantic" attributes such as lighting and temperature levels. That is, those constraints without a geometrical component.

The *topological constraints* are inherited from the 3D environment and are transformed into Prolog facts which describe room's topological characteristics. Consequently, from the user's perspective, there is a perfect matching between the topological characteristics of the 3D environment and the Prolog facts implemented in the

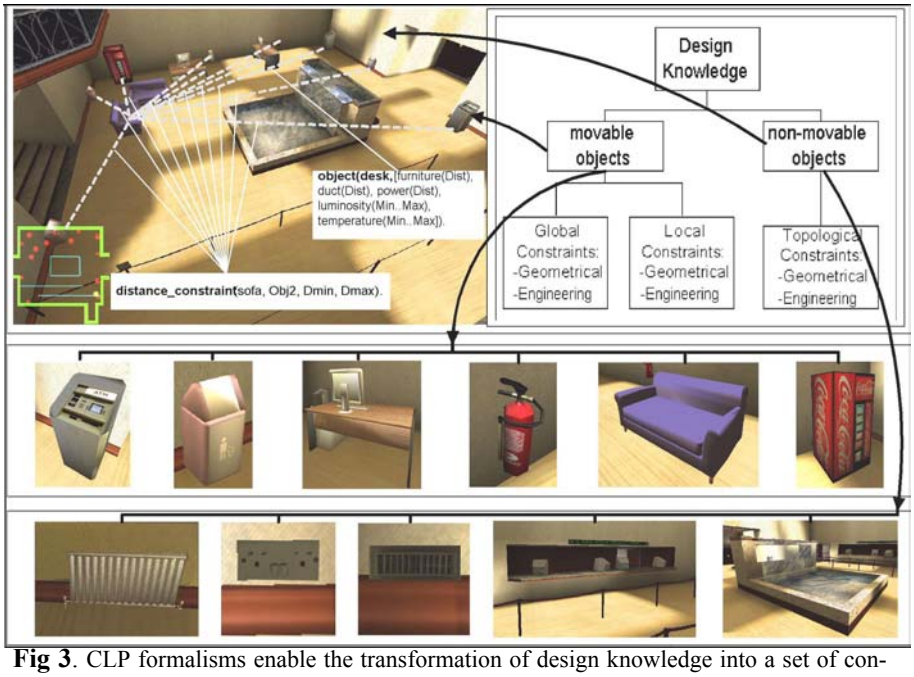

Fig 3. CLP formalisms enable the transformation of design knowledge into a set of constraints between objects.

solver. For instance, sources of heat or radiators and different lighting levels are visually apparent to the user in the 3D environment. Therefore, both characteristics have been formalised as Prolog facts:

```
source_of_heat([X0/Y0,X1/Y1,X2/Y2])
luminosity( [lightingvalue = Area0, lightingvalue =
Area1, lightingvalue = Area2,])
```

These facts define the coordinates of the sources of heat (a list of points X/Y in the search space) and the regions where the lighting level is less than 200 lux (a list whose each element defines a lighting value and an associated rectangle –$Area_i$- in the search space).

In the example, there are also definitions for the location in the 3D environment of the following elements: power points, ventilation ducts, special features (e.g. the central fountain), queuing area, counters, walls, luminosity and temperature levels.

Local constraints are constraints on the attributes of a single object and specify how the object relates to the topological characteristics of the virtual environment. For instance, let us imagine that the user wanted to reallocate the object desk. The new object location would be constrained by the object's attributes (or design requirements) expressed in the corresponding Prolog clause: `object(desk, [features_min(6), duct_min(3), power_max(3), luminosity(300..500), temperature(19..24)].` This clause reads as follows: a desk should be placed at a minimum distance of 6 units from any special feature (e.g. the central fountain), at a minimum distance of from any ventilation duct, at a maximum distance of 4 from a power point, inside a region whose luminosity is between 300 and 500 flux and whose temperature is between 19° and 24°.

Global constraints are constraints whose formulation involves more than one object and therefore, are imposed on the configuration. These constraints relate, for example, objects of each kind, objects of two different types, all the objects and so on. Consequently and following with the reallocation of a desk object, this not only disrupts its local properties but also the properties that link that object to the rest of the configuration. For example, the following constraint: `distance_constraint (desk,atm,6,12)` enforces the minimum and maximum distance between 2 objects: a desk and an atm in this case. Hence, if the new allocation is nearer than six units or further than 12 it will force the atm to be reallocated which, in turn, will force any other object linked to the atm to behave in the same fashion. In this case, constraint propagation serves as the basics for interactive problem solving, as it solves the configuration problem created by the user by displacing an object.

Global constraints are particularly relevant to express design requirements which involve group of objects. For instance, the following requirement: fire extinguishers and bins need to be distributed in the room to comply with health and safety regulations has been implemented in a similar fashion.

4.2 Sample run

In this section, we give a step-by-step description of a system's run on an example, and for the sake of clarity we will only detail the internal mechanisms on a sub-configuration extracted from the full application.

First running the system results in the solver producing a set of variable allocations satisfying all the design constraints (see left side of Figure 1). These variables are translated in the virtual environment in terms of object types and positions, which spawns all furniture objects at their respective locations, thus constituting a first design solution.

Once the initial configuration has been deployed, the user can explore this first solution by navigating in the virtual environment and test variants of the configuration by changing objects' positions.

For instance, let us image that the user is not satisfied with the initial allocation of an object (e.g ATM is too close to the stairs). Consequently, the user seizes the ATM object and proceeds to reallocate it while he explores the 3D environment. Once, a suitable location has been found the user will drop the object. In our implementation, the user's actions trigger the corresponding Unreal events. For instance, when the object is dropped an unreal event is triggered which sends the object's location to the solver in the appropriate query format (e.g atm=34/9.).

The solver has an embedded server which deals with all the incoming queries. In other words, the server detects the incoming unreal events and transforms their content into logical terms which, in turn, prompt the main clause in the solver and triggers a new solution search. Consequently and continuing with the example, when the user seizes the ATM object he is disturbing both the local and the global constraints attached to it. As shown in figure 4, an ATM object, can only be allocated away from

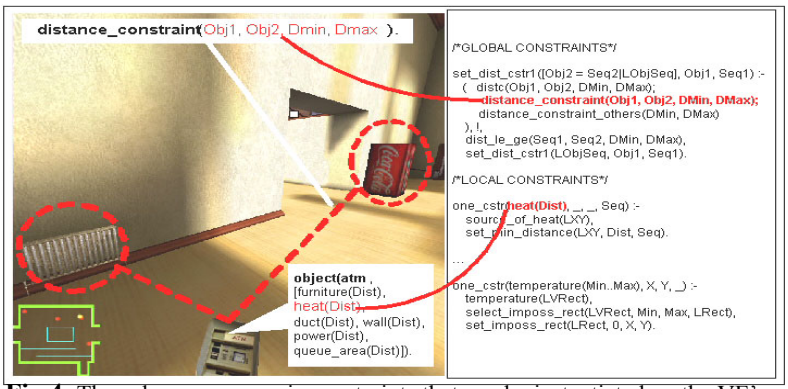

Fig 4. The solver uses generic constraints that can be instantiated on the VE's objects

a source of heat (`heat(Dist)`) and, similarly, it needs to be away from any other object of the configuration a specified distance (`distance_constraint(Obj1, Obj2, DMin, DMax)`).

As a consequence, when the user decides to reallocate the object to a new position, this in turn disrupts the imposed constraints in the configuration and forces the system to "propagate" all the constraints and to generate a solution compatible with the de-

sign requirements. This propagation and a non-deterministic search are the basic mechanisms for interactive problem-solving. Thus, since the resolution process is invisible to the user, seen from his or her perspective, the 3D environment automatically reconfigures itself as a consequence of his/her interactions (see right side of Figure 1).

Let us now describe the resolution phase as seen from the variables standpoint. The resolution phase is divided into three stages: the Finite Domains (FD) of variables are defined, then the constraints are imposed *(constraint propagation)*, and finally the labeling stage (*search*) finds concrete solution(s).

For instance, let us think of a specific object: a vending machine (vm). Initially the finite domain of its spatial variables is: vm=_#[0..35/0..35]. Then, the constraints are imposed on the FD variables till no further domain reduction (due to constraint propagation) is possible: vm = _#3(1..3 :11..13 :16..18 :24..25 :27..28) / _#25(1..2 :10..12 :23..25). This example illustrates the power of constraint propagation and its importance to support the integration of the solving process in the VR interaction loop.

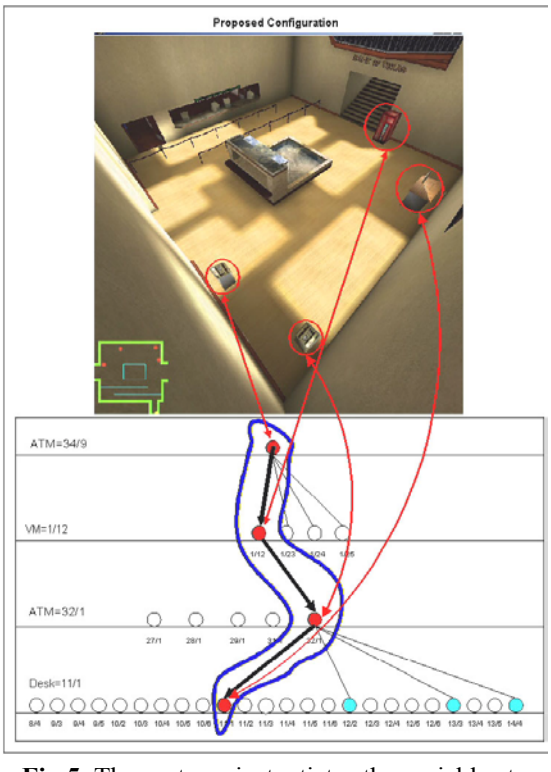

Fig 5. The systems instantiates the variables to propose a solution to the user: labeling stage.

Consequently, when no further reduction is possible, the system must explicitly instantiate the variables to reach or "search" a solution: enumeration or labeling stage (see Figure 5). In this case, the choice of heuristics for labeling stage is: first-fail –the variable with the smallest number of elements in its FD is selected- for the choice of variable and "random" for selection of value (which gives us good execution times and offers a good visual aspect). Thus, this means that the CLP Solver outputs a list of predicates which contains the objects' types and spatial coordinates: [Obj1=X1/Y1, Obj2=X2/Y2, ….]. This list is sent to the 3D engine where a series of low-level system events transform this incoming symbolic list into the configuration of objects displayed in the virtual environment. In our example, a possible solution would be the following: [vm=1/12, atm=34/9, atm=32/1,desk=11/1] which corresponds to the configuration displayed to the user in the virtual environment (see Figure 5).

Once the proposed configuration has been displayed, the user can explore all the feasible alternatives. In our case, in order to enable the user to explore the solution space a specific interaction mechanism has been implemented: when the user presses a specified key, an implemented low-level event is generated by the visualization engine which, in turn, produces a new "query" for the solver. Consequently and following the implemented interaction model previously explained, this new query triggers the deployment in the virtual environment of the next solution (see Figure 6). The concept of next solution is intimately related to the heuristics used to explore the search space. In other words, the heuristics used in the enumeration (search) stage determined how the next solution is found. For instance and continuing with the example, once a solution is proposed: [vm=1/12, atm=34/9, atm=32/1,desk=11/1]. The user could request the next one which, according to the selected heuristic, is the following: [vm=1/12, atm=34/9, atm=32/1,*desk=12/2*].

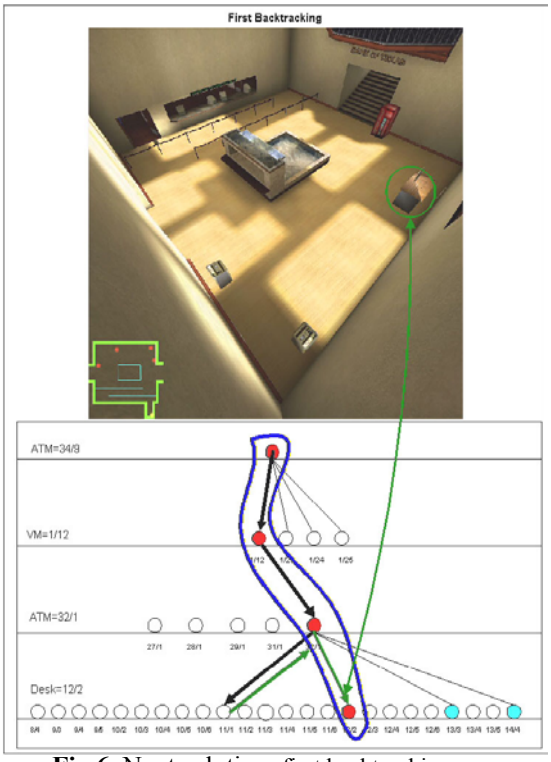

Fig 6. Next solution: first backtracking.

6 Conclusions

We have presented a novel framework for the use of virtual environments in interactive problem solving. This framework extends visualisation to serve as a natural interface for the exploration of configuration spaces and enables the implementation of reactive virtual environments.

This implementation is based on a fully interactive solution, where both visualisation and the generation of a new solution are under the control of user. In other words, our approach extends VR towards fully interactive environments by introducing the concept of reactive environments which react to the user's interaction whilst preserving the user-centred aspects of VR: exploration and interaction in a 3D real-time environment. Additionally, our approach supports the interactive exploration of the solution space in which alternative solutions can be found.

The system has potential for extension in different directions. For instance, in terms of mechanisms of user interaction, we envisage offering yet more interactivity to the user for more efficient object manipulation [10][11][12]. For instance, it is

fairly simple to “lock” some objects in the virtual environment which would ensure that an object will remain at the some location after the user has interacted with the configuration. That is, the user could select a rectangle (an X and Y coordinates) where he wants a given object to be located and the object will remain there. It is also easy to dynamically redefine parameter for the objects, e.g, minimum/maximum distances amongst objects. As well, taking advantage of the incremental capabilities of the solver, we could give the user the possibility of adding objects on-the-fly and to choose the constraints for that objects from a set of predefined constraints.

In its current form, the system is still faced with a number of limitations, the most important being the absence of an explanatory module that would provide the user for justifications for the proposed solutions. Such a module is even more important to explain why there exist no acceptable solutions for some object positions proposed by the user. Further work will be dedicated to providing more feedback from the configuration system.

References

1. Aylett, R. and Cavazza, M.: Intelligent Virtual Environments - A State-of-the-art Report. Eurographics (2001).
2. Codognet, P.: Animating Autonomous Agents in Shared Virtual Worlds. Proceedings DMS'99, IEEE International Conference on Distributed Multimedia Systems, Aizu, Japan, IEEE Press (1999).
3. Axling, T., Haridi, S, and Fahlen, L.: Virtual reality programming in Oz. In Proceedings of the 3rd EUROGRAPHICS Workshop on Virtual Environments, Monte Carlo, February (1996).
4. Smolka, G, Henz, M and Wurtz, J.: Object-Oriented Concurrent Constraint Programming in Oz. Research Report RR-93-16, Deutsches Forschungszentrum fur Kunstliche Intelligenz, Stuhlsatzenhausweg 3, 66123 Saarbrucken, Germany, April (1993).
5. Andersson, M. Carlsoon, C. Hagsand, O, and Stahl, Olov.: DIVE –The Distributed Interactive Virtual Environment, Tutorials and Installation Guide. Swedish Institute of Computer Science, March (1993).
6. Honda, K. and Mizoguchi, F.: Constraint-based Approach for Automatic Spatial Layout planning. Conference on Artificial Intelligence for Applications, IEEE Press (1995).
7. Pfefferkorn, C.: A heuristic problem solving design system for equipment or furniture layouts. Communications of the ACM, 18(5):286-297. (1975).
8. Diaz, D. and Codognet, P.: Design and Implementation of the GNU Prolog System. *Journal of Functional and Logic Programming*, Vol. 2001, No. 6. (2001).
9. British Educational Communications and Technology agency . Health and Safety: planning the safe installation of ICT in schools. http://www.becta.org.uk / technology / infosheets / html / safeuse.html (last visited 4/06/2002). (2002).
10. Bukowski, W. R. and Séquin, H. C. Object Associations. ACM Symp. On Interactive 3D Graphics, Monterey, CA, USA, (1995).
11. Kallman, M. and Thalmann, D. Direct 3D Interaction with Smart Objects. *ACM International Symposium on Virtual Reality Software and Technology*, *VRST 99*, London. UK, December, (1999).
12. Stuerzlinger, W. and Smith, G. Efficient Manipulation of Object Groups in Virtual Environments. *Proceedings of the IEEE VR 2002, March 24-28, Orlando, Florida.* (2002).