A new approach to virtual design for spatial configuration problems

Carlos Calderon and Marc Cavazza
University of Teesside, TS1 3BA Middlesbrough, United Kingdom
c.p.calderon@tees.ac.uk,
Daniel Diaz
University of Paris 1, 90 rue de Tolbiac, CRI bureau C1405, Paris, 75013, France.

Abstract

In this paper, we present a new framework for the use of Virtual Reality (VR) in engineering design for configuration applications. Traditional VR systems support the visual exploration of a design solution but do not assist the user in exploring alternative solutions based on domain knowledge. Extending previous work in the area of Intelligent Virtual Environment, we propose an intelligent configuration system based on constraint logic programming (CLP), integrated in a real-time 3D graphic environment. This type of integration facilitates the expression of design knowledge in the VE and enables the user to interactively solve and/or refine a spatial configuration problem. In the system described in this paper, the user can visually explore configurations, but his interaction with objects of the configuration problem triggers new cycles of constraint propagation from the modified configuration to produce a new compatible solution.

1. Introduction

Spatial configuration problems are visual by nature and they are based on an implicit mapping between the "abstract problem space" (which is searched for solution configurations) and the physical environment in which these configurations are deployed. It is, however, difficult to make this implicit mapping transparent to the user. VR techniques have the potential to solve this problem but, in the current use of VR for engineering design, the knowledge is mainly expressed on the geometrical layout, textures, colours, etc. As a result of this, VR assistance in the use of the underlying design knowledge is restricted. To put it differently, it is currently not feasible to attach much design knowledge to the Virtual Environment (VE); consequently, users cannot use the "natural" interaction mechanisms of VR to interact with it and visualise the "dynamic" consequences of their interactions with the configuration.

In this paper, we present a framework for configuration applications that, while preserving the natural interaction of traditional Virtual Reality systems, support the expression of design knowledge in the VE and the visualisation of the user's interactions with the configuration. In the context of configuration applications this translates into the user being able to navigate and physically interact with 3D objects, but this interaction triggers the automatic reconfiguration of the configuration problem, thus allowing, the dynamic exploration of design solutions.

We claim that this can be achieved by integrating Constraint Logic Programming (CLP) techniques into Virtual Environments, extending previous work in Intelligent Virtual Environments [1] [2]. Consequently and in order to demonstrate the viability of our approach, we have implemented an intelligent configuration system in which solutions can be interactively refined by the user through direct manipulation of objects in the virtual environment.

In the next sections, after reviewing related work, we describe the system's overview, software architecture and the proposed interaction model. We then discuss the formalisation of design knowledge in CLP, its expression in the VE and the interactive exploration of solutions. We conclude by discussing the potential for further applications and generalisation of the approach.

2. Related Work and Background

In the field of virtual design, previous work has described the inclusion of a knowledge level for design applications [3]. In these systems, a Decision Support Systems was added to the virtual environment to validate the design configurations. However, this type of system validates configurations, more as a diagnostic system, rather than enabling the user the visualisation of his interactions with the configuration by, i.e, reconfiguring the configuration. In addition, standard decision support systems, such as rule-based systems, lack flexibility in their inference mechanisms, which prevents their use in a fully interactive system.

Fernando et al. [4][5] have emphasised the importance of constraints in virtual design. However, they have been essentially dealing with graph-based techniques which do not support the interactive generation of alternative design solutions. In other words, graph-based techniques help to interactively find a solution by restricting the solution space but they are limited in terms of the generation of design alternatives once a solution has been found.

The use of constraint programming for the expression of construction knowledge was pioneered by the SEED (Software Environment for Support the Early Phases in Building Design) project at CMU [6]. The SEED project introduced the conceptual basis for the representation of constraints to automatically generate layouts and argued that constraint programming provides a uniform mechanism to handle the domain related knowledge, because spatial configurations could be naturally expressed as constraints. Moreover, constraints have proven to be a useful format to express engineering design knowledge [7]. For instance, much engineering knowledge is stated in terms of constraints: regulations, codes of practice, behaviour models, cost restrictions, and planning strategies all employ explicit declaration of constraints which are easily translated into the "formal" constraints expression of constraint programming. Hence, constraint-based systems have the potential to be one of the most understandable and easiest to maintain of all reasoning systems.

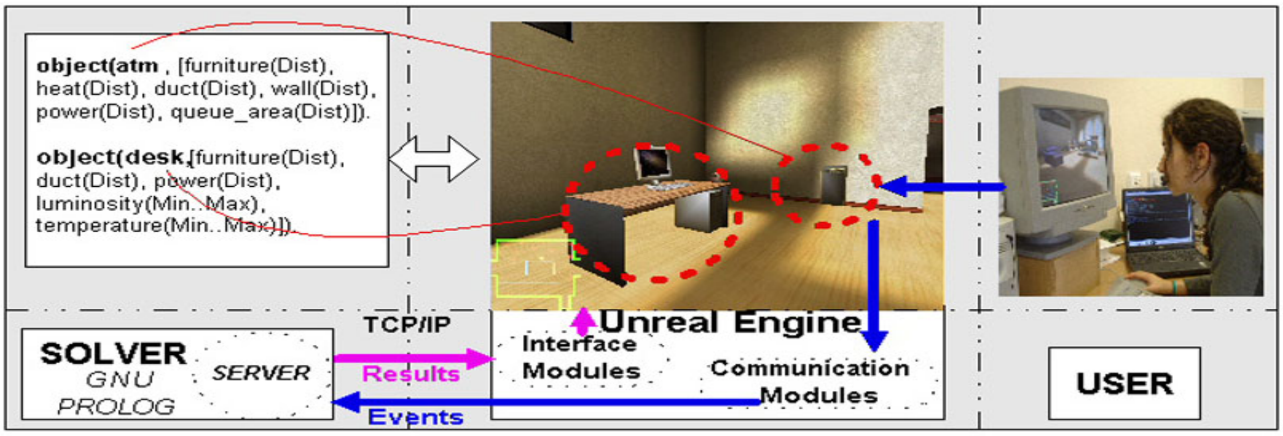
Furthermore, previous research in the area of Intelligent Virtual Environments has proposed the use of constraint logic programming as a supporting mechanism for intelligent object behaviour, its rationale being the seamless integration of symbolic reasoning techniques with the visual and interaction components: Axling et al. [1] and Codognet [8]. Both Axling and Codognet have put emphasis on the behaviour of individual objects in the virtual world. However, CLP naturally provides solutions for the combined behaviours for sets of objects, which is the property we use to implement behaviours for the virtual environment as a whole.

3. System Overview and Architecture

The system is an interactive 3D environment in which the user can freely navigate and interact with the world objects (e.g. by dragging and dropping them). 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 instantly triggers new solutions (configurations) from the solver which, in turn, are displayed in the virtual environment.

The system has been developed using the Unreal Tournament^{TM} (UT) game engine as a development environment. In addition to being an efficient graphics engine, it includes a development environment in which object behaviours and interactions with objects can be development environments, even for immersive systems [9][10]. The UT environment also supports the overall software architecture by allowing integration of external modules via dynamic link libraries or windows sockets. We have used TCP sockets as a a communication mechanism between the visualisation engine and the intelligent configuration system (see Figure 1).

The intelligent configuration module is based on Constraint Logic Programming (CLP). More specifically, the CLP(FD) framework provides all the tools to represent design knowledge, mapping design constraints to "formal" constraints in CLP which express e.g. distance between objects, compatibility between materials, etc. In addition, it enables incremental solutions to be computed in user real-time, which ensures the interactivity of the system as a whole. We have used GNU Prolog [11] 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


```
object(atm , [furniture(Dist),
heat(Dist), duct(Dist), wall(Dist),
power(Dist), queue_area(Dist)]).

object(desk,[furniture(Dist),
duct(Dist), power(Dist),
luminosity(Min..Max),
temperature(Min..Max)]).
```

Figure 1. System Architecture: a GNU Prolog solver is integrated in the Unreal Engine

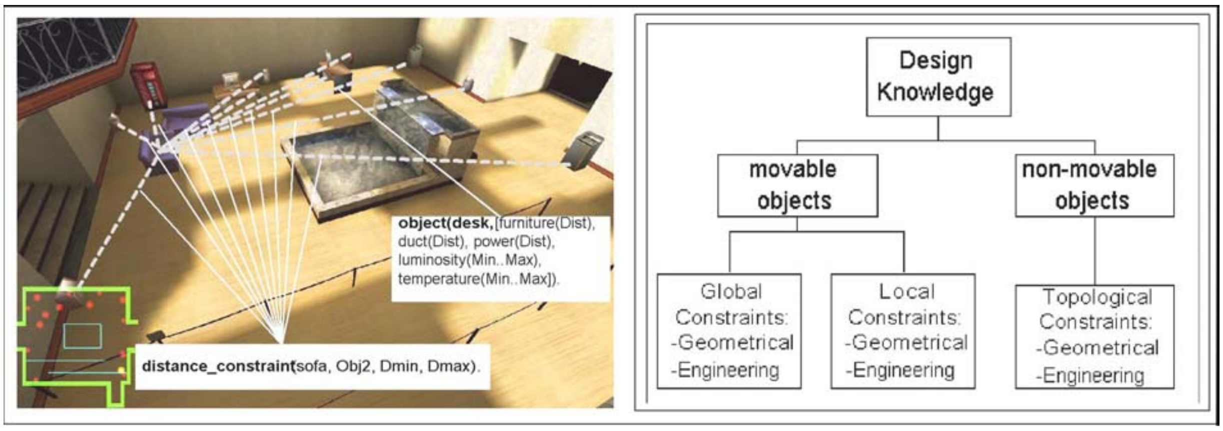

object(desk,[furniture(Dist), duct(Dist), power(Dist), luminosity(Min..Max), temperature(Min..Max]).
distance_constraint(sofa, Obj2, Dmin, Dmax).
Design Knowledge
movable objects
non-movable objects
Global Constraints: -Geometrical -Engineering
Local Constraints: -Geometrical -Engineering
Topological Constraints: -Geometrical -Engineering

Figure 3. CLP formalisms enable the transformation of design knowledge into a set of constraints.

coefficient, resistance to fire, etc. In the next section, we give a more detailed insight into the implementation considering the specific techniques used.

It must be noted that, according to our results, the communication time for the overall cycle is on average less than 15ms, which is fully compatible with the user interaction (as the user is not navigating when interacting with objects).

4. An Intelligent Configuration System

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 [12][13].

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

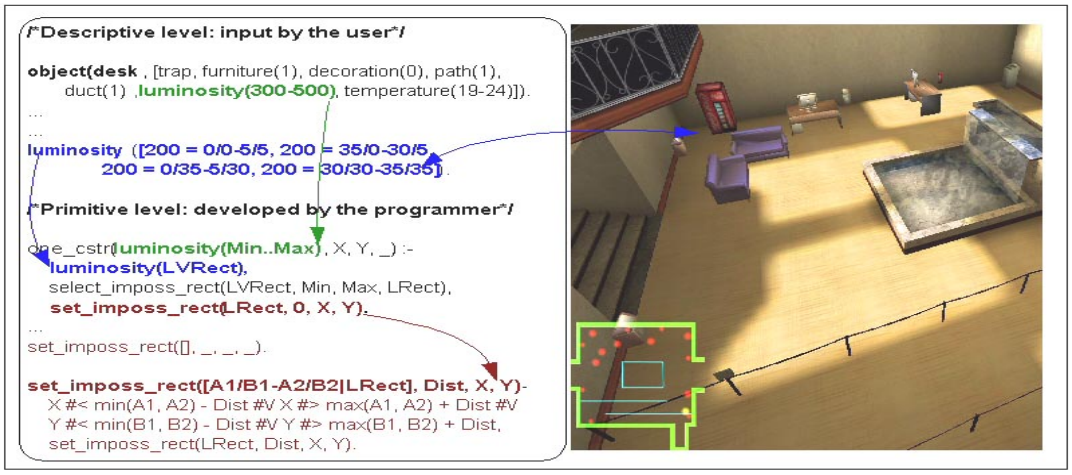
In our case, the movable objects are: one vending machine, two desks (which represent the customer attention area), two sofas (waiting attention area), two automatic teller machines (ATMs), three fire extinguishers and four bins. This constitutes a subset of 14 objects: considering the size of the environment 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 Formalisation of design knowledge in CLP

In order to acquire and encapsulate the design knowledge into the appropriate formalism: CLP, the characterisation of building design requirements proposed by the SEED project has been adopted [6]. This characterisation proposes two intertwined levels for the formulation of design requirements: design unit and functional unit level.

For our purpose, the encapsulation of design requirements was adopted at functional unit level: movable and non-movable objects (see Figure 3). The constraints or design requirements on those objects (functional units) have been classified in three groups: *topological, local* and *global constraints*. These constraints incorporate geometric as well as more "semantic" attributes such as lighting and temperature. Consequently, to assess a proposed design of a design unit (e.g the furniture layout of a room), the designers select and input the attribute values on the functional units (objects) and their "performance" is interactively evaluated by the user through the virtual environment.

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 solver. For instance, sources of heat or radiators and different lighting levels are visually apparent to the



```
/*Descriptive level: input by the user*/

object(desk , [trap, furniture(1), decoration(0), path(1),
     duct(1) ,luminosity(300-500), temperature(19-24)]).
...
...
luminosity ([200 = 0/0-5/5, 200 = 35/0-30/5,
        200 = 0/35-5/30, 200 = 30/30-35/35]).

/*Primitive level: developed by the programmer*/

one_cstr(luminosity(Min..Max), X, Y, _) :-
   luminosity(LVRect),
   select_imposs_rect(LVRect, Min, Max, LRect),
   set_imposs_rect(LRect, 0, X, Y).
...
set_imposs_rect([], _, _, _).

set_imposs_rect([A1/B1-A2/B2|LRect], Dist, X, Y)-
   X #< min(A1, A2) - Dist #V X #> max(A1, A2) + Dist #V
   Y #< min(B1, B2) - Dist #V Y #> max(B1, B2) + Dist,
   set_imposs_rect(LRect, Dist, X, Y).
```

Figure 4. Constraints implemented at descriptive and primitive level.

user in the 3D environment. Therefore, both characteristics have been formalised as Prolog facts as follows:

`source_of_heat([`X_0/Y_0, $X_1/Y_1/$,X_2/Y_2`])`
`luminosity([`lightingvalue = $Area_0$, lightingvalue = $Area_1$, lightingvalue = $Area_2$`])`

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 300 lux (a list whose each element defines a lighting value and an associated rectangle 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, 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 desk object. The new object location would be constrained by the object's attributes (or design requirements) expressed in the corresponding Prolog clause:

`object(desk`
[furniture_min(6), duc_mint(3), power_max(4), luminosity(300…500), temperature(19..24)]

This clause reads as follows: a desk should be placed at a minimum distance of, for instance, 6 meters from any furniture (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:

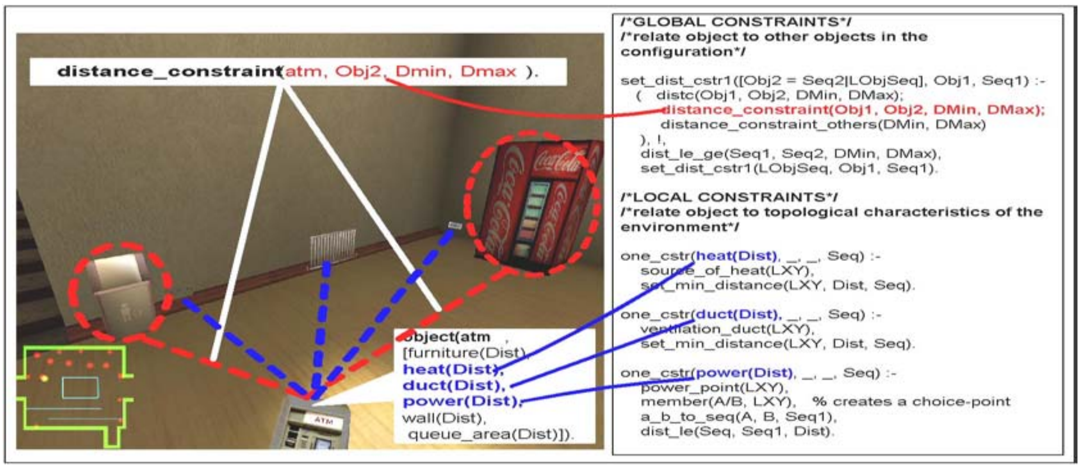
`distance_constraint(desk,atm,6,12)`

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.

It must be noted that there are two implementation levels for either local or global constraints: *descriptive and primitive*. In the descriptive level the user of the system (e.g the designer) states the constraint, or what it needs to be solved, without being concerned about how it is resolved. Hence, constraints can be easily asserted or retracted from the constraint solver. On the other hand, the primitive level is concerned with the optimization of the resolution process. That is, at a primitive level the main concerned is to find, or define, the most appropriate finite domains predicates which assure an efficient/fast solver.

Figure 4 shows an example of this. In this case, the


```
distance_constraint(atm, Obj2, Dmin, Dmax ).
```

```
object(atm ,
[furniture(Dist),
heat(Dist),
duct(Dist),
power(Dist),
wall(Dist),
 queue_area(Dist)]).
```

```
/*GLOBAL CONSTRAINTS*/
/*relate object to other objects in the
configuration*/

set_dist_cstr1([Obj2 = Seq2|LObjSeq], Obj1, Seq1) :-
   (   distc(Obj1, Obj2, DMin, DMax);
       distance_constraint(Obj1, Obj2, DMin, DMax);
       distance_constraint_others(DMin, DMax)
   ), !,
   dist_le_ge(Seq1, Seq2, DMin, DMax),
   set_dist_cstr1(LObjSeq, Obj1, Seq1).

/*LOCAL CONSTRAINTS*/
/*relate object to topological characteristics of the
environment*/

one_cstr(heat(Dist), _, _, Seq) :-
   source_of_heat(LXY),
   set_min_distance(LXY, Dist, Seq).

one_cstr(duct(Dist), _, _, Seq) :-
   ventilation_duct(LXY),
   set_min_distance(LXY, Dist, Seq).

one_cstr(power(Dist), _, _, Seq) :-
   power_point(LXY),
   member(A/B, LXY),   % creates a choice-point
   a_b_to_seq(A, B, Seq1),
   dist_le(Seq, Seq1, Dist).
```

Figure 5. The solver uses generic constraints that can be instantiated on the VE's objects.

descriptive level is concerned, firstly with matching the lighting levels defined in the solver, using the topological constraint lumninosity [LAreas], to those on the 3D environment; and secondly with defining the acceptable levels for a particular object, in this case a desk. At a primitive level the constraint *set_imposs_rect(LRect, 0, X, Y)* ensures that an object X/Y, in this case the desk, cannot belong to a given rectangle A1/B1-A2/B2, defined by the 2 diagonal coordinates, in which the lighting levels are inferior to imposed minimum threshold or acceptable level. Thus, this further level of description maintains, at a descriptive level, the declarative nature of CLP as well as assuring an efficient solver.

4.2 Interactive exploration of solutions

First running the system results in the solver producing a set of variable allocations satisfying all the design constraints. These variables are translated in the virtual environment in terms of object types and positions, which instantiates all furniture objects at their respective locations, thus constituting a first design solution (object configuration). 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 wants to refine and/or further explore the configuration (e.g an ATM is too close to the queuing area). Consequently, the user seizes the ATM object and proceeds to reallocate it while h/she 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=1/12.).

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 5, an ATM object, can only be allocated away from a, e.g, source of heat (heat(Dist)) and, similarly, it needs to be away from any other object of the configuration a specified (distance_constraint(Obj1, Obj2, DMin, DMax). Thus, when the user decides to reallocate the object by dragging and dropping it 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 design requirements. This propagation and a non-deterministic search are the basic mechanisms for interactive exploration of solutions.

Consequently, the user utilises the "natural" interaction mechanisms of VR to interact with the configuration problem and, the automatic reconfiguration of the configuration problem (see Figure 6) enables him to visualise the consequences of his interactions on the configuration.

5. Conclusions

We have presented a novel framework for the use of virtual environments in interactive virtual design. For design applications, this framework supports the expression of design knowledge in the VE and the exploration of new design solutions by refining previous ones, which would appear a natural process to many users. In other words, this framework supports the interactive exploration of the solution space of a spatial configuration problem.

The system has a 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. For instance, it is fairly simple to "constrain" some objects in

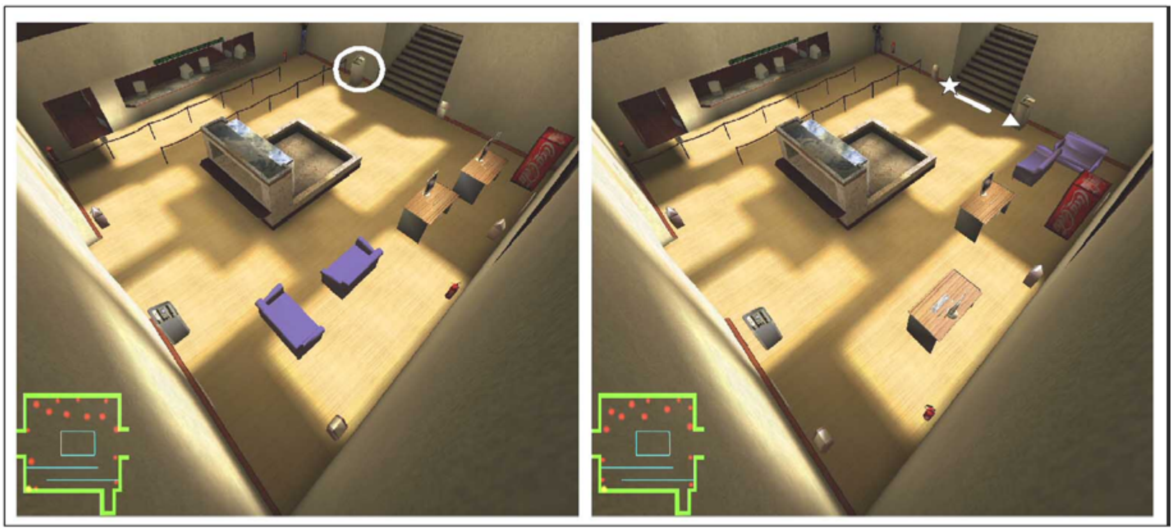


Figure 6. The user refines a given configuration by reallocating an object. As a result of this, the environment reconfigures itself.

the virtual environment what it would ensure that an object will remain at the some location after the user has interacted with the configuration. 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.

6. References

[1] Axling, T., Haridi, S, and Fahlen, L. (1996). Virtual reality programming in Oz. *In Proceedings of the 3rd EUROGRAPHICS Workshop on Virtual Environments*, Monte Carlo, February 1996.

[2] Aylett, R. and Cavazza, M. Intelligent Virtual Environments - A State-of-the-art Report. *Eurographics 2001*

[3] Nomura, Junji, Hikaru Ohata, Kayo Imamura, Robert J. Schultz. (1992). Virtual Space Decision Support System and Its Application to Consumer Showrooms. *Matsushita whitepaper*.

[4] Fernando, T., Murray, K., Wimalaratne, P. (1999). Software Architecture for a Constraint-based Virtual Environment, , *ACM International Symposium on Virtual Reality Software and Technology*, *VRST 99*, London. UK, December, 1999.

[5] Fa, M., Fernando, T., and Dew, P.M (1993). Interactive Constraint-based Solid Modelling using Allowable Motion, *ACM/SIGGRAPH Symposium on Solid Modelling and Applications, May 1993, pp 243-252*

[6] Fleming, U., Coyne, R., Fenves, S., Garrett, J., Woodbury, R. (1994). SEED –Software Environment to Support the Early Phases in Building Design. *Proceedings of IKM94*, Weimar, Germany, pp 5-10

[7] Lottaz, C., Clément, D., Faltings, B. and Smith, I. (1999).Constraint-Based Support for Collaboration in Design and Construction, *Journal of Computing in Civil Engineering*, Vol. 13, No. 1, jan, 1999, pp. 23-35.

[8] Codognet, P. (1999). Animating Autonomous Agents in Shared Virtual Worlds, *proceedings DMS'99, IEEE International Conference on Distributed Multimedia Systems,* Aizu, Japan, IEEE Press 1999.

[9] Jacobson, J and Hwang, Z. (2002). Unreal Tournament for Immersive Interactive Theater. Communications of ACM, Vol. 45, No. I, January 2002. pp39-42.

[10] Lewis, M and Jacobson, J (2002). Games Engines in Scientific Research. *Communications of ACM*, Vol. 45, No. I, January 2002. pp27-31.

[11] Diaz, D. and Codognet, P. (2001). Design and Implementation of the GNU Prolog System. *Journal of Functional and Logic Programming*, Vol. 2001, No. 6, Oct 2001.

[12] European Commission – Joule Thermie Programme-(2000). Tax Office Extension: Enschede (The Netherlands). http://erg.ucd.ie / EC2000 /EC2000_PDFs / repo_enschede.pdf (last visited 4/06/2002)

[13] British Educational Communications and Technology agency (2001). 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).