The 5th World Multi-Conference on Systemics, Cybernetics and Informatics, SCI 2001, July 22-25

Intelligent Virtual Environment for Building Design

Carlos Calderon and Marc Cavazza
University of Teesside, TS1 3BA Middlesbrough, United Kingdom,
c.p.calderon@tees.ac.uk, m.o.cavazza@tees.ac.uk

ABSTRACT

This paper presents the first prototype of a reconfigurable Virtual Environment (VE). The objective of the system is to link 3D Intelligent Virtual Environments to interactive planning systems. This type of system makes possible interactive solutions where the user refines a possible configuration and enables the system to generate a complete new solution enforcing all the design constraints, previously programmed. In this first prototype we link our constraint solver with the visualization engine so that the solution produced by the constraint solver is displayed in a VE.
Keywords: Intelligent, Reactive, Virtual Environment, Spatial configuration tasks.

1 INTRODUCTION

Many applications of Virtual Reality have been proposed for design tasks. In previous work, the authors have explored the use of Virtual Reality (VR) in construction [1], especially for building design. However, it is unpractical to experiment many configurations within the virtual environment by systematic manual exploration, using only the visual display to assess the value of a specific resource allocation. In other words, the visual components of a VR system are not sufficient to solve the design/configuration task. On the other hand, automated solutions based on optimisation techniques are generally not interactive. Hence, we are investigating a 3D reactive system which will incorporate both functionalities: 3D interactive visualisation and interactive constraint solving.

The objective of this paper is to present the first prototype of Reconfigurable Virtual Environment (VE). The goal of a 3D-Reconfigurable Virtual Environment is to support building design by displaying how the resources available within a Virtual Environment can be re-arranged.

The emerging area of Intelligent Virtual Environments explores the integration of Artificial Intelligence techniques into Virtual Reality systems. One particular research topic is to couple interactive AI systems (scheduling, planning, etc.) to virtual environments. The VE serves as an interface to the interactive planning system. Because the nature of the task is mainly spatial, the user directly manipulates the objects from the virtual environment to create input configuration for the system. This makes possible interactive solutions where the user refines a possible configuration and enables the system to generate a complete new solution enforcing all the design constraints.

To date, two main systems have attempted this integration. The first one it was a basic interface between Oz programming and the DIVE VR software, a toolkit for building distributed VR applications [4] [5]. Though Oz supports constraint programming, the implementation reported its use as a generic high-level programming language rather than for planning (scheduling) or design applications. More recently, Codognet [6] has developed a generic constraint package, VRCC, which is fully integrated into VRML and can define high-level behaviours through the specification of constraints. Previous work in the SEED project at Carnegie Mellon [7] has also demonstrated the viability of interactive constraint solving. The goal of the SEED project is to provide support, in principle, for the preliminary design of buildings in all aspects that can gain from computer support. This includes using the computer not only for visualization, analysis and evaluation, but also more actively for the generation of designs describing conceptual design alternatives and variants. In other to achieve this, SEED make use of Artificial Intelligence techniques such as constraint propagation and support tools such as ILOG solver (a constraint solver).

The paper is organized as follows. The next section introduces the technologies use to create a reactive and intelligent VE for exploring spatial configuration tasks. The third section describes how the prototype has been implemented. The fourth section explains the applicability of the work in the construction industry. Finally, the paper closes with a brief discussion and further work.

2 TECHNOLOGY BASELINE

2.1 3D Interactive Visualization Engine

We decided to use the Unreal Virtual Machine for three main reasons: extensibility, an extremely powerful rendering engine and cutting-edge networking capabilities. The Unreal Virtual Machine is a real-time system that consists of several components: The server, the client, the rendering engine and the engine support code. Real-time systems are time triggered in the sense that their overall behavior is controlled by a recurring clock tick.

Networking Architecture. Earlier distributed interactive virtual environments such as DIVE [8] were based on peer-to-peer approach with no centralized server. This means that each participant is equal and that each participant has to have synchronized its input and timing with the others. The main advantage of this approach is its simplicity but there are major disadvantages such as lack of constant frame rate and participant scalability due to the fact that each participant has to have synchronized its input and timing with the others. The Unreal Virtual Machine introduces a new approach termed the generalized client-sever model [9]. In this model, the server is still authoritative over the state of the system. However, the client maintains an accurate subset of the system state locally, and can predict the system flow by executing the same code as the server, on approximately the same data, thus minimizing the amount of data that must be exchanged between the two machines.

It is important to realize that if the network bandwidth were unlimited, the network architecture would be quite simple: at the end of each tick the server could send to *all* the clients *all* the data so that any client could render/have locally an exact view/copy of the system state. However, the reality of the Internet is quite different and there is not enough bandwidth to communicate the complete system state updates. Furthermore, this will not improve as Internet connections become faster because the amount of data to transmit in order to update the system state will be itself larger (better graphics, more AI, etc).

The Unreal Virtual Machine tackles this problem by providing an architecture where the server is in control of the system state. Clients only run *approximations* of the state of the system. It is up to the system's programmer to decide what is an approximation of the shared "reality" between server and clients.

This is a key point in order to have a "real-time" interactive system. Previous research projects that attempt to implement reactivity at real-time (at the next tick) recommended to "take care of avoiding to slowdown the system by heavy computation in the TCC part (constraint solver part)" [6]. On other words, the computations needed to solve the "constraints" would bring down the frame rate. Unreal provides us with mechanisms to play with the tradeoff between the bandwidth and the amount of data we want to ship to all the clients. This means that pending on how the constraint solver is implemented it might not be possible to get a real-time reactive environment but we will be able to move around at real-time. For instance, spawning actors (objects) as a result of a user's interaction with the system is an expensive operation that could be very slow if it is not designed efficiently.

Rendering engine and subsystems. Unreal rendering engine rich and powerful set of features has allowed us a variety of effects. For instance, Unreal's multicolored dynamic lighting and shadow mapping (see figure 1) has been used to create surfaces with shadowing. Unreal's parametric, fractal, and animated texture mapping has allowed us to create realistic wavy water and animated materials (see figure 1). Human users are *represented* by graphical objects called "bots" a specific type of actors. Bots could have associated any mesh, skins or animations depending on the kind of human behavior we want to represent (see figure 1).

Figure 1. Unreal's set of features.

Unreal also supports a very powerful digital sound system (software Dolby Surround encoding for full 360-degrees) that we have used to simulate environment sounds.

2.2 Interactive Constraint Solving

Our approach is based on a combination of Constraint Logic Programming over Finite Domains (CLP(FD)) and 3D real-time interactive technology.

Our choice (CLP(FD) was made based on a series of factors: expressivity, the combination of search and incremental constraint solving capabilities, the short development time while exhibiting an efficiency comparable to imperative languages, and .the fact CLP(FD) is fact enough to react in "real-time" to the user's input configuration.

In our initial building design constraint examples, the spatial relationships between the objects in a layout configuration are all known. That is, the spatial relations between objects remain invariant in the various combinations of possible values and the constraints whose formulation depends on these relations reflect those specific spatial relations. This set of spatial relations is called a spatial configuration.

The constraint formulations at the "building design level" lead to a system of conjunctive atomic constraints in variables that are intervals because the alternatives available to satisfy disjunctive constraints have been reduced to Finite Domains (FD). In this initial prototype we are concerned with building design tasks in which a feasible solution for allocating several objects that preserves the spatial configuration has to be generated.

In previous research projects was argued that when using Constraint Logic Programming techniques it was not easy to force the system to produce a feasible solution within the specified bounds [7]. However, today's extended Prolog languages such as GNU Prolog, Prolog III and, Sicstus Prolog provide us with Finite Domain solvers, which extends Prolog with constraints over FD, by using single (low-level) primitive to define all

(high-level) FD constraints.
In our system we have used GNU Prolog, which contains an efficient constraint solver over FD that supports arithmetic constraints (priceless to define spatial configurations) and labeling constraints. The latter help us to discover inconsistency as soon as possible, thus avoiding futile search through inconsistent alternatives., and therefore a more efficient constraint solver.

3 EXAMPLE

This section explains how we have used the 3D-real time visualization engine and constraint logic programming to solve spatial configuration tasks. Such a solution has always two parts to it. The starting point is to formulate the *problem* in terms of constraints (using GNU Prolog). Then the next step is to link our constraint solver with the visualization engine (Unreal Virtual Machine).
The whole problem is about Spatial Configuration Tasks related to building design. However, for the present demonstration purpose, the chosen example application allocates coke machines in the reception hall of a bank.
The illustrative example presented, "Coke machine allocation", is formally equivalent in its non-interactive form to a traditional "n-queens" problem, as it consists in searching a space to minimise the number of conflicts. It is of course a well-described problem, on which very different methods can be applied, including branch-and-bound search, constraint programming and heuristic repair. Heuristic repair consists in allocating a complete, but possibly inconsistent set of variables and reduce inconsistencies incrementally, for instance by using a "maximum conflict" heuristic, i.e. moving the object which is responsible for the greatest number of conflicts. Recently, heuristic repair and local search have been shown to be possible alternatives to constraint programming in the context of similar applications in virtual worlds [10].
This is what makes it precisely interesting for comparing methods, not so much on their algorithmic properties as on their expressivity, defined as their knowledge representation capabilities.
The convergence, or the formal equivalence, of different techniques is misleading, as formal equivalence is often the negation of expressivity. To illustrate the problem of expressivity, we have solved the "Coke machine allocation" example by both methods. Within search, we have defined an instance of the conflict criterion as Y1-Y=/= (X1-X). Within constraint programming, the constraint had a natural declarative definition, for example X-Xm+Y-Ym<Dmax. On trivial examples, search and constraint solving are equivalent. However, as our application gets more realistic and complex, expressivity of the constraint formalism, that is the ease of expressing a variety of operations in a simple, declarative and powerful way, is increasingly more important.
Paradoxically, while finite domains are not a very powerful knowledge representation formalism, they are still much more expressive than heuristics, especially when several criteria have to be taken into account. It is the case that scalability inevitably involves heterogeneous constraints not only geometrical ones. It is possible to devise ordered sets of domain concepts to use with the finite domain approach, i.e. the ordered set of building material shininess, cost, strength, fire resistance, thermal isolation, etc.
Therefore, the use of constraint programming in this application is thus justified by its expressivity and the declarative nature of the formalism. As it has been mentioned in the introduction, the SEED project has already demonstrated the viability of interactive constraint solving.
The processing pipeline of the example application is the following: firstly the virtual space is discretised then the constraint solver generates a solution. This solution is transformed so that it can be, finally displayed in the Virtual Environment. Figure 2 shows the result of this process.

Figure 2. Coke-machine allocation

The next sub-section explains in more detail how the application has been implemented.

3.1 Description

In this our example application, the *Constraints* imposed on the coke-machines are the following:

a) Sources of heat (i.e. radiators) should stay a minimum distance away from the machines.
b) The machines must not obstruct ventilation ducts
c) The coke machines must stay clear from exits and thoroughfares.
d) Finally, the allocation of coke machines must take into account the existing furniture and decoration in the hall.

In this our example application we have regarded the constraints *b*, *c* and *d* as fixed topological constraints whereas *a* is considered as a movable object. On other words, if the user were to interact with constraint solver through the VE by changing the position of the coke machine, topological constraints restrict the number of possible positions whereas movable objects re-allocate themselves within the VE to comply with the constraint. It must be said that the criterion used to consider an entity as a something movable or as a part of the topology of the VE is totally arbitrary. There is nothing that could

prevent us from considering, for instance, ventilation ducts as movable objects.

For the software architecture, to *link* our constraint solver with the visualization engine, we have used Unreal client-server architecture, as described in previous section. The system can be described as follows:

On the client side, we have implemented, by using GNU Prolog, two modules the constraint solver and a client socket that allocates (yields a solution) coke machines in the reception hall of a bank. Both modules run on a dedicated machine distinct from the system on which Unreal is running. The solution produced by the constraint solver is then passed through a bi-directional socket to the server in the form of a byte stream. The communication protocol used on top of the primitive data stream is TCP/IP (Transfer Control Protocol and Internet Protocol). In our current implementation remote machines communicate with the Unreal server via socket-based-connections

On the server side, we have implemented a series of new packages (modules) in Unreal. These were coded using Unreal Script. Firstly, the server creates a socket through which receives the solution produced by the constraint solver. Then, this is parsed and transformed into UnrealScript commands in order to create the corresponding object configuration which the Unreal Virtual Machine displays in real-time (see figure 2). Figure 3 illustrates the system architecture

To conclude this section, we must say that in this our first prototype, for simplicity and suitability to the problem at hand, the constraints or relations between objects the objects (which are treated as a dumb objects) are stated/programmed in the constraint solver. For spatial configuration tasks we are not concerned with having to process critical information within critical times. However, it would be possible to solve the problem from a different approach. By endowing the objects with behaviors (using built-in and scripted methods), these would behave as agents that could communicate. Hence, the objects/agents could send and retrieve messages to the constraint solver by using Tell/Ask/BlockingAsk operations. This approach would require a different point of view to deal with the concurrency problem and this is outside of the scope of this paper

4 BUILDING DESIGN AS APPLICATION DOMAIN

This section intends to give the reader an idea of the applicability of the work in the building design domain. In order to so, this part of the paper presents, briefly, an abstract modeling framework.

There are four basic concepts that need to be defined: design units, functional units, requirements, and technology. *Design units* are spatial or physical structure of a building. *Functional units* are a combination of functions to be satisfied by a single design unit and collects the requirements for that unit. *Requirements* consists of one or more constraints and criterion. A constraint indicates a restriction on an attribute of an associated design unit or several units (for example, minimum area requirement for a room). A criterion indicates an attribute of an associated design or collection of design units that can have values of varying degrees of desirability. For instance, the distance from a room to another room that should be as short as possible. A constraint and a criterion may be given for the same attribute (i.e distance between two rooms). *Technology* is a set of computational mechanisms to create design units that satisfy the requirements of associated functional units in a design context [7].

Building design is a complex process well beyond of the scope of this paper. However, it is important to understand that building designers (Architects) are usually given a minimum set of requirements to come up

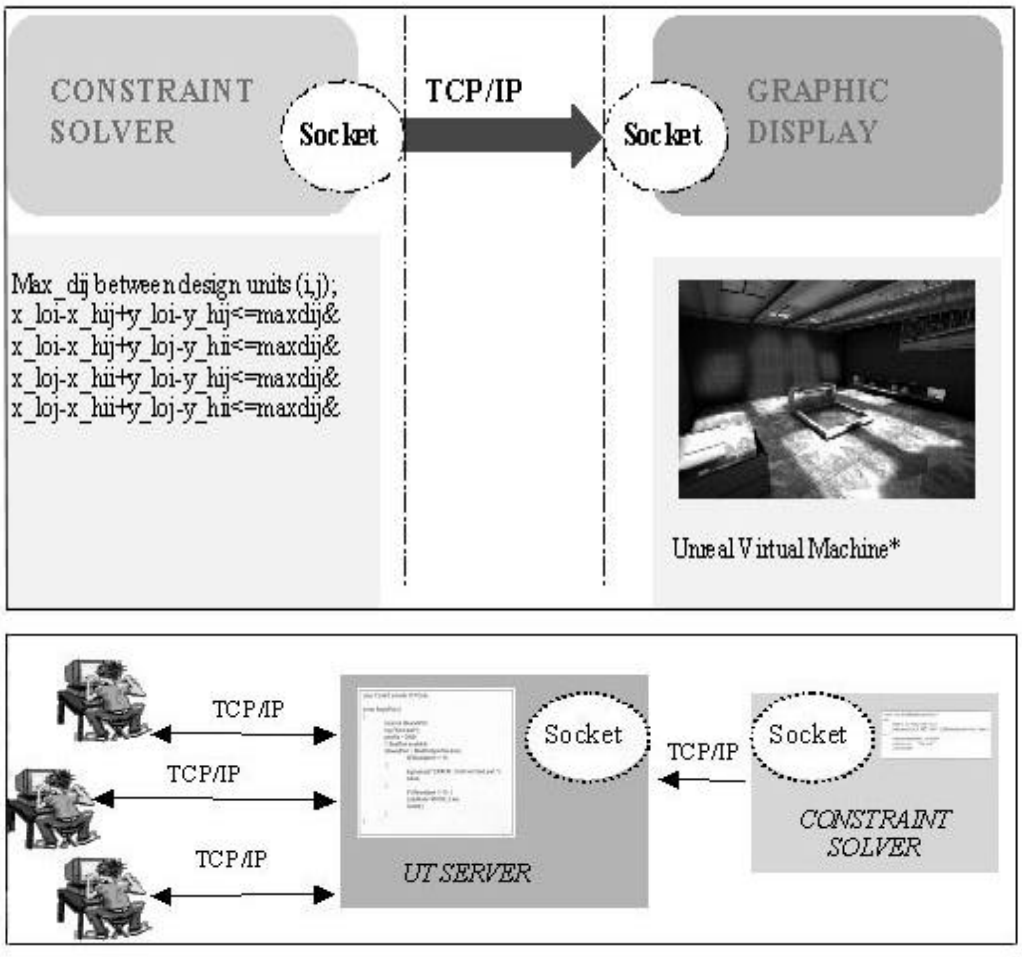

Figure 3. System Architecture

with a "good" design. This needs the client's validation or approval.

In this paper we do not deal with issues such as the transformation of functional level constraints into design level constraints, or how the requirements can be incorporated into the generation process that creates design units, or how design units interact with functional units. We recommend the interested reader to read the document on constraint handling in SEED-Layout [7].

However, our system, based on a combination of CLP and 3D real-time interactive *technology*, could be used by a user/client to interactively validate a *design unit*. In other worlds, the Architect would be responsible for entering the appropriate design constraints that comply with the client's criteria and, of course, regulations and standards. The architect would also have to decide which "objects" are movable or can be reconfigured when the client/user interacts with the design of a particular design unit. That is, the user refines a possible configuration and lets the system to generate a complete new solution enforcing all the design constraints.

5 FURTHER WORK AND CONCLUSIONS

At the moment of writing this article a second prototype is under development. We are currently working on the system to make it fully interactive so that a user can interact with the constraint solver through the 3D real-time environment. As it has been mentioned previously, CLP over a Finite Domain (FD) is fast enough. This is vital to address the well-known technical issue of response time in reactive planning. That is, the system analyzes the user's input configuration and yields a "response" in "real-time". Figure 4 illustrates this process.

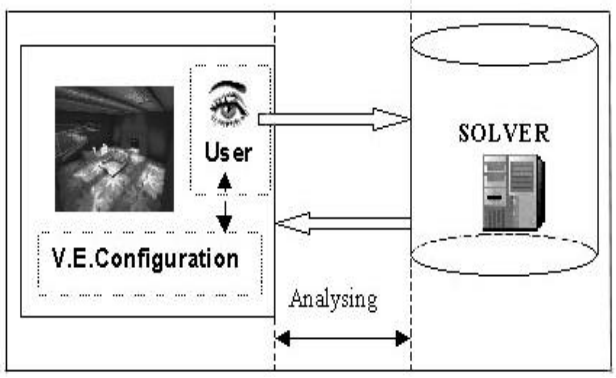

Figure 4. Response in real-time

However, CLP over a Finite Domain (FD) as representation formalism has some limitations. FD constraints are constraints on integer valued variables rather than real valued variables. Real interval constraints are better suited to deal with trigonometric and non-linear arithmetic constraints. In GNU Prolog, FD expressions are not restricted to be linear. However, non-linear arithmetic constraints, for instance R3=R1*R2-I1*I2, usually yield less constraint propagation than linear constraints. That is, when a non linear constraint is encountered during computation, then it is delayed until it becomes linear.

Sometimes it is difficult to draw a line between what is Unreal Engine and therefore, Unreal technology and Unreal game content. This can be quite confusing and hamper the development since packages like engine share technology with content. It is expected that in future versions of the engine this will be improved.

We learnt from previous experiments [11] that "typical" walk-throughs aid users to understand building design in a better way than traditional CAD tools. However, we also found that the visual components of a VR system are *not* sufficient to solve a design/configuration task. In this paper we have presented the first step towards an innovative approach in which the VE serves as an interface to the interactive planning system. Because the nature of building design tasks are mainly spatial, the user should be able to directly manipulate the objects from the virtual environment to create input configuration for the planning system. This makes possible interactive solutions where the user refines a possible configuration and enables the system to generate a complete new solution enforcing all the design constraints.

6 ACKNOWLEDGEMENTS

We would like to express our gratitude to Professor Takeo Ojika for welcoming us to Virtual Systems Laboratory (Gifu, Japan) where this work was developed.

Many thanks to Mr H. Robert Berry, Jr for his technical assistance regarding UnrealScript.

Many thanks to Daniel Jason Patton (Spooge) for letting us use his fantastic map.

7 REFERENCES

[1] Calderon, C,. van Schaik, P,. and Hobbs, B (2000a). Is VR an effective communication medium for building design? *Proceedings* of *Virtual Reality International Conference 2000*. Laval 18-21 May 2000 (pp 46-55).

[2] Chambers, D. and Reisberg,D. (1985). Can mental images be ambiguous? *Journal of Experimental Psychology*: Human Perception and Performance. 11, pp 317-328.

[3] Kirsh, David (1995). The Intelligent Use of Space. *Artificial Intelligence 73*(1-2): pp 31-68.

[4] Axling, Tomas, Haridi, Seif, and Fahlen, Lennart (1996a). Virtual reality programming in Oz. *In Proceedings of the 3rd EUROGRAPHICS Workshop on Virtual Environments*, Monte Carlo, February 1996

[5] Axling, Tomas And Haridi, Seif. (1996b). A Tool For Developing Interactive Configuration Applications. *J. Logic Programming* 26(2), pp 147-168.

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

[7] 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.
[8] Magnus Andersson, Christer Carlsoon, Olof Hagsand, and Olov Stahl (1993). *DIVE –The Distributed Interactive Virtual Environment, Tutorials and Installation Guide*. Swedish Institute of Computer Science, March 1993.
[9] Sweeney, Tim (1999). *Unreal Networking Architecture*. http://unreal.epicgames.com/Network.htm, 27/03/2001
[10] Codognet, Philippe (2001). Behaviours for virtual creatures by Constraint-based Adaptive Search. *Proceedings, Artificial Intelligence and Interactive Entertainment, Papers from the 2001 AAAI Symposium. 25-30*. Stanford, USA.
[11] Calderon, C., van Schaik, P., and Hobbs, B (2000). How to evaluate VR applications *Proceedings of VSMM 2000. Gifu 4 6 October 2000* .