Go to contents 16

Smart Objects:
Constraints and Behaviors in a 3D Design Environment

EGGINK, Dustin; GROSS, Mark D.; DO, Ellen
Design Machine Group, Department of Architecture, University of Washington
http://depts.washington.edu/dmachine/

We describe a constraint-based three-dimensional design environment called Smart Objects. In Smart Objects, design collaborators (designers, clients and consultants) would engage an architectural design in an interactive three-dimensional environment where they may alter objects in the model and compose formal solutions. Design intentions embedded into objects as constraints are expressed as behaviors when the user moves objects in ways that either violate or meet specified constraints.
Keywords: *Constraints, Collaboration, VRML, Java.*

Introduction

Each new design problem in architecture presents a new set of requirements. A designer must remain aware of these requirements and effectively communicate them to collaborators because the degree to which the requirements are met will determine the success of the solution. Smart Objects is a constraint based three-dimensional design environment we are developing that aims to aid these tasks. In Smart Objects, intentions of an architectural problem are embedded as constraints into the modelled objects that compose a formal solution. The model is presented in an Internet accessible environment in which designers can manipulate and alter objects, composing various arrangements of form. As constraints are either violated or met, object behaviours manifest to ensure that the designer attends to requirements that guide the project. The goal of Smart Objects is to provide a medium for communicating the design principles and guidelines that inform an architectural design to all those involved in the project and ensure that the principles and guidelines are maintained as the design progresses.

Constraints derive from pragmatic functional requirements (e.g., spatial needs, fire codes) as well as more abstract design intentions (e.g., scale, proportion). Constraints are embedded (in the form of linear equations or inequalities) into the objects of a three-dimensional model, which helps collaborating designers communicate the characteristics of a solution. In order to be accessible to all collaborators, Smart Objects is constructed using common Internet technologies.

This paper introduces our first Smart Objects concept demonstration, discusses its limitations, and then describes the current version of Smart Objects. Formal constraints in Robert Venturi's Vanna Venturi House provide a demonstration of Smart Objects' constraint maintenance. The paper concludes with a brief description of work related to Smart Objects and a short discussion of future work.

Smart Objects implementation

The Smart Objects system comprises code written in three languages: VRML, Java and HTML. A VRML (Virtual Reality Modelling Language) world describes the three-dimensional geometric model that Smart Objects manages. VRML is a common 3D Internet language that anyone with a web browser can access. VRML browsers (e.g. Cosmoplayer and Blaxxun) provide an intuitive interface for interaction with the model. Modelling programs such as Form-Z and 3D

StudioMax can export VRML worlds. In projects by Dace Campbell (Campbell, 1998) and Thomas Jung (Jung, et al, 1999), VRML was used to present architectural models and design solutions on the web and as a way for collaborating designers to access and annotate a project.

A Java program asserts the constraints that Smart Objects implements as it observes actions in the VRML world through VRML's External Authoring Interface (EAI). An HTML page acts as a container for the VRML world and the Java applet, which allows the Java program to observe and control actions in the VRML world.

To demonstrate this process, Figure 1 shows a VRML world with five beams spanning between two walls. A designer can drag one wall toward or away from the other. When the wall moves past a certain distance, the depth of the beams increases to accommodate the longer span.

In this concept demonstration, the Smart Objects applet implements constraints using two methods: 1) a **start** method that searches for specially defined nodes in the VRML world and 2) a **callback** method that observes changes in the VRML world. When the HTML page that contains Smart Objects is opened, the applet's first act is to locate the defined nodes of VRML objects using the **start** method. VRML nodes contain several fields that contain values accessible to the Java applet. The values of the fields describe the properties of objects in the world: positions, dimensions, colours and others.

Once the applet finds the nodes, the **callback** method waits for the designer to make changes to the model. Designers interact with Smart Objects through the VRML interface while the Smart Objects applet records the distance an object has moved in the X, Y and Z directions (object translations). The **callback** method runs each translation change through a set of "if, then" statements. As the changes meet or violate the conditions, the applet implements a change back to the VRML world, thereby expressing the constraint through object behaviour. To revisit the above example, the **start** method looks for the beam nodes called "beam0," "beam1," etc. and the translation node of the wall called "move." The **callback** method observes the coordinates of the "move" node. The constraint asserts: **if** the X coordinate of "move" is greater than 3 **then** "increase size of 'beams0-4' "

This wall and beam scenario expresses design constraints by changing the size of objects. This is one of several object behaviours available to alert the designer of a constraint. Other scenarios we built alter an object's colour or position when constraints are met or violated. The variety of behaviours are useful to express constraints that are not absolute requirements, but rather design preferences.

However, the limited examples of the concept demonstration applet do not approach the complexity of constraints found in full architectural programs. Also, the applet required each constraint and its consequence to be manually coded in the Java program as an "if, then" statement and compiled into the applet. These concerns called for the more robust and flexible implementation of Smart Objects that is described in the following section.

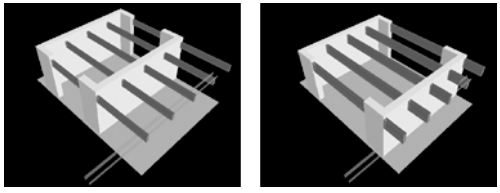

Figure 1. Two walls and a series of beams that re-size to accommodate the longer span between the supporting walls.

Cassowary constraint solving engine

Originally conceived in the early 1960's (Sutherland, 1963) constraint programming has resurfaced as an active area of research over the past fifteen years. In constraint programming, a problem is described as a set of relationships, or constraints, on a set of variables. The computer's task is to find values of the variables consistent with the constraints. Constraint programming has been used to solve

problems in scheduling, production planning and networking, as well as in graphics, natural language processing and database applications (Bartak, 1999).

The current implementation of Smart Objects uses the Cassowary constraint solving engine (Badros and Borning, 1998) to add, remove and resolve constraints. Cassowary enables Smart Objects to handle many constraints in a single design and allows constraints to be added, removed and altered while the program is running.

Besides relying on Cassowary for managing constraints, the principal difference between the concept demonstration applet and the Cassowary-based applet is that the former enforces constraints by altering specifically coded nodes in the VRML world, whereas the latter (more generally) assigns the nodes of a VRML world to an array that the Cassowary solver can access. Leaving constraint management to Cassowary, the Smart Objects applet performs three principal tasks: 1) instantiate a series of object classes and a Cassowary solver, 2) define constraints, and 3) exchange information between the VRML world and Cassowary. These three tasks are performed by the two methods (**start** and **callback**) described in the Smart Objects concept demonstration applet along with a third **constrain** method. The following describes how Smart Objects utilises Cassowary to implement constraints and express behaviours.

When Smart Objects is launched, its **constrain** method first instantiates the Cassowary solver and an array of object instances (obj[ ]) . Object instances describe the positions and colours of objects modelled in the VRML world. Each object in the array describes an XYZ translation node of a modelled object or colour node of RGB values. The length of the obj[ ] array is double the number of modelled objects in the VRML world: the first position (obj[0]) contains the XYZ translation of the first modelled object, and the second (obj[1]) contains the RGB colour of the first modelled object, obj[2] and obj[3] contain the translation and colour of the second modelled object and so on. Each constraint is asserted by the **constrain** method as a linear equation or inequality between the XYZ (or RGB) component assertions of an object class. (A future implementation will not rely on coding constraints into the applet). The Cassowary solver adds the XYZ and RGB components of each object to its database of variables that can be constrained. In the database, the linear equations and inequalities asserted establish the constraint. For example, a line of code that calls upon Cassowary ("solver") to equate ("new LinearEquation") an object's X coordinate (defined as "obj[0].X()") to another object's X ("obj[1].X()") would be written as:

```
solver.addConstraint(new
LinearEquation(obj[0].X(), obj[1].X()))
```

The **start** method is no different than that of the demonstration prototype except that a variable specifies nodes in the VRML world. Objects are defined in the VRML world as "Object0", "Object1, "Object2" and so forth. When a designer activates an object in the VRML world by dragging it with a mouse, the **callback** method loops, copying the translation and colour values of every node in the VRML world into the array that is presented to the Cassowary solver. Cassowary then adjusts the values in this array using its solving techniques, and returns a new set of values to the Smart Objects applet, resolved to satisfy all the constraints. Smart Objects then displays the new state by sending the computed values to the VRML world via the **callback** method. Figure 2 diagrams this process.

Constraints in the Vanna Venturi House

The Vanna Venturi House by Robert Venturi provides a demonstration of Smart Objects. According to Venturi (Venturi 1966), symmetry, balance and scale were among fundamental architectural parameters that dictated the final form of the house. In this example constraints were applied to a model of the house to reflect Venturi's design intentions. Smart

Objects maintains these design relationships as the user moves objects in the model by automatically altering the position or colour of other related objects. For example, Figure 3 illustrates the relationship between the two halves of the front façade: they are opposite and equidistant from the centre-line. When the designer moves one wall, the other moves an equal distance in the opposite direction.

Figure 4 illustrates a design preference for centring the stairwell in the composition. As it moves off of the centre-line, the stairwell becomes more translucent to signal that a design preference is being violated; the opacity of the object indicates the degree of constraint satisfaction.

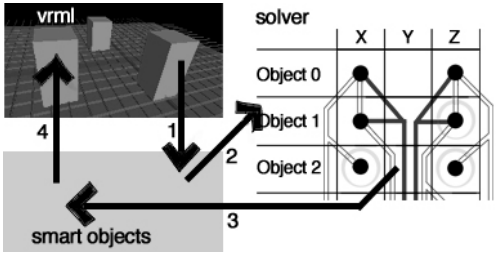

Figure 2. The Cassowary implementation of Smart Objects: (1) Changes in the VRML world are sent to the Smart Objects applet. (2) Smart Objects sends the new coordinate sets to a Cassowary solver as object instances, which places values from the objects into a matrix. (3) The solver uses the linear equations and inequalities drawn between the object class coordinates to solve constraints and sends the updated values back to the Smart Objects applet which(4) implements the changes to the VRML world.

Figure 3. Symmetrical movement of two halves of the Vanna Venturi House front façade.

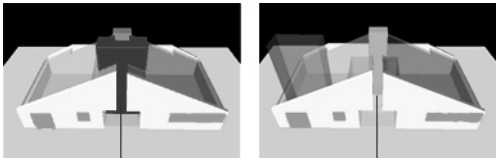

Figure 4. The stairwell becomes more transparent as it is moved away from the centre-line.

Through interaction with the constrained model of this house, a designer can better understand the key relationships that Venturi set up to guide the design process. Design alternatives can be created that satisfy the absolute constraints of a design while softer constraints can be met to varying degrees.

Related work

Constraints and parametric variation has been an active area of investigation in computer aided architectural design for over fifteen years (Gross, 1986; Frazer, 1987). Recently some of this functionality has become available in commercial CAAD applications, notably the recently released software Revit (Revit Technology Corporation, 2000), which employs a parametric change engine to manage relationships between objects in a building model. As users make changes to objects in the model, Revit propagates values to other objects according to these relationships.

Arvin and House (1999) use a relaxation solving method (similar to that used in Sutherland's Sketchpad (1963) system in which constraints are expressed as forces that act upon objects until the constraints are satisfied. Frazer (1987) showed how constraints could be used to express the rules of physical and spatial structure of architectural designs, which he called "plastic modelling." Gross (1986) described design as a process of exploring constraints, and articulated an architecture for CAAD that employed constraint programming to support the management of design relationships. A two-dimensional implementation was demonstrated in the Co-Draw system (Gross, 1992). Kolarevic's Re-Draw system (Kolarevic, 1997) developed these ideas in a two-dimensional drawing environment that uses guidelines to position and dimension walls. Smart Objects extends this explorative use of constraints in two ways: by expressing constraints as object behaviours and by extending the design environment into one that is three-dimensional.

Future work and discussion

The following identifies specific tasks to be done in the Smart Objects project.

Although Cassowary allows objects to be added, removed and altered on the fly, Smart Objects currently relies on equations coded in the Smart Objects applet to assert constraints for Cassowary to manage. In a future version, designers would interactively apply constraints to objects either graphically or through a text interface.

Currently the position of objects are constrained by a single translation vector from their initial position in the VRML world. If Smart Objects could convey to Cassowary the coordinates of any point on an object, the range of constraint types would be richer and better accommodate the architectural process (eg., a face of an object could be related to the face of another).

Changes made to a model cannot be saved. A next step will be to save altered objects and constraints as a new VRML file with an accompanying Java applet.

Finally, in order to bring a VRML world built in a modelling application into Smart Objects, the VRML file must be pre-processed. This consists of adding and defining translation nodes and renaming object nodes. Currently we code this by hand. An automated process for transforming models into Smart Objects-ready worlds is clearly needed.

With these additions, Smart Objects can become a means to explore design through constraints, a method to communicate purposes behind formal design solutions, and a tool to analyse and demonstrate design principles.

Acknowledgements

Funding for this project has been provided by the Valle Scholarship and Scandinavian Exchange Program at the University of Washington. Thanks must also be extended to Thomas Jung for his extensive technical expertise and support with VRML and Java.

References

Arvin, S. A. and House, D. H.: 1999, Making Designs come Alive: Using Physically Based Modeling Techniques in Space Layout Planning. Computers in Building, Proceedings of the CAADFuture's '99 Conference, pp 245-262.

Ayagen, Z. and Fleming, U.: 1998, Classification of Precedents: A Hybrid Approach to Indexing and Retreiving Design Cases in SEED. CAADRIA '98 Proceedings of the Third Conference on Computer Aided Architectural Design Research in Asia, pp 435-444.

Badros, G. J. and Borning, A.: 1998, The Cassowary Linear Arithmetic Constraint Solving Algorithm: Interface and Implementation. Technical Report UW-CSE-98-06-04, Department of Computer Science and Engineering, University of Washington, http://www.cs.washington.edu/research/constraints/cassowary/.

Bartak, R.: 1999, Constraint Programming: In Pursuit of the Holy Grail, Proceedings of WDS 99, http://kti.ms.mff.cuni.cz/~bartak/constraints/index.html

Campbell, D. A.: 1998, VRML In Architectural Construction Documents: A Case Study, VRML 98 Monterey, Proceedings of the 1998 VRML Conference, pp 115-120.

Frazer, J.: 1987, Plastic Modelling – the flexible modelling of the logic of structure and spaces, CAADfutures '87, Proceedings of the Second International Conference on Computer Aided Architectural Design Futures, pp 199-208.

Gross, M. D.: 1986, Design as Exploring Constraints, Ph.D. Dissertation, MIT.

Gross, M.D.: 1992, CoDraw "Graphical Constraints in CoDraw", IEEE Workshop on Visual Languages, Seattle, WA, pp 81-87.

Jung, T., Do E. Y., Gross, M. D.: 1999, Immersive Redlining and Annotation of 3D Design Models on the Web, Computers in Building, Proceedings of the CAADfutures '99 Conference, pp 81-98.

Kolarevic, B.: 1997, Relational Descriptions of Shapes and Form Generation, CAADRIA '97, Proceedings of the Second conference on Computer Aided Architectural Design Research in Asia, pp 29-39.

Revit Technology Corporation: 2000, An Introduction to Revit, http://www.revit.com/cornerstone/.

Schwartz, F. (ed.): 1992, Mother's House, Rizzoli International Publications, Inc., New York.

Sutherland, I.E.: 1963, Sketchpad, A Man-Machine Graphical Communication System, Ph.D. Dissertation, MIT.

Venturi, R: 1966, Complexity and Contradiction in Architecture, The Museum of Modern Art, New York.