CRC Construction Innovation

BUILDING OUR FUTURE

# Final Report
# DesignWorld: A Tool for Team Collaboration in High Band Virtual Environments

## Research Project No: 2002-024-B


The research described in this report was carried out by:

Project Leader: Mary Lou Maher

Researchers: Robin Drogemuller, Kirsty Beilharz, Andy Dong, John Gero, Mike Rosenman, Thomas Bellamy, Rod Gameson, Willy Sher, Tony Williams, Stephen Egan, Kathryn Merrick, Zafer Bilda, Linda Candy, Mijeong Kim, Tony Shi, Ji Soo Yoon, Figen Gul, Yinghsiu Huang, Sue Sherratt, Adel Ahmed, Owen Macindoe, Robert Shen

Project Affiliates: David Marchant, Carolyn Mitchell, Kanyarat Nemprempree, John Crawford, Lan Ding, Melissa James, Richard Hough, Steve Pennell



Research Program: B
Sustainable Built Assets

Project: 2002-024-B
Supply Chain Sustainability

Date: 30 December 2005


Leaders in Construction and Property Research

## Distribution List

Cooperative Research Centre for Construction Innovation
Authors

## Disclaimer

The Client makes use of this Report or any information provided by the Cooperative Research Centre for Construction Innovation in relation to the Consultancy Services at its own risk. Construction Innovation will not be responsible for the results of any actions taken by the Client or third parties on the basis of the information in this Report or other information provided by Construction Innovation nor for any errors or omissions that may be contained in this Report. Construction Innovation expressly disclaims any liability or responsibility to any person in respect of any thing done or omitted to be done by any person in reliance on this Report or any information provided.



Please direct all enquiries to:

Chief Executive Officer
Cooperative Research Centre for Construction Innovation
9th Floor, L Block, QUT, 2 George St
Brisbane Qld 4000
AUSTRALIA
T: 61 7 3138 9291
F: 61 7 3138 9151
E: enquiries@construction-innovation.info
W: www.construction-innovation.info

# Table of Contents

# List of Figures

# List of Tables

# 1. EXECUTIVE SUMMARY

Current software tools for documenting and developing models of buildings focus on supporting a single user who is a specialist in the specific software used within their own discipline. Extensions to these tools for use by teams maintain the single discipline view and focus on version and file management. There is a perceived need in industry to have tools that specifically support collaboration among individuals from multiple disciplines with both a graphical representation of the design and a persistent data model. This project involves the development of a prototype of such a software tool.

We have identified multi-user 3D virtual worlds as an appropriate software base for the development of a collaborative design tool. These worlds are inherently multi-user and therefore directly support collaboration through a sense of awareness of others in the virtual world, their location within the world, and provide various channels for direct and indirect communication. Such software platforms also provide a 3D building and modelling environment that can be adapted to the needs of the building and construction industry.

DesignWorld is a prototype system for collaborative design developed by augmenting the Second Life (SL) commercial software platform[1] with a collection web-based tools for communication and design. Agents manage communication between the 3D virtual world and the web-based tools. In addition, agents maintain a persistent external model of designs in the 3D world which can be augmented with data such as relationships, disciplines and versions not usually associated with 3D virtual worlds but required in design scenarios.

[1] www.secondlife.com

# 2. INTRODUCTION

Large design projects, such as those in the AEC domain, involve collaboration between designers from many different design disciplines in varying locations. Existing tools for developing and documenting designs of buildings and other artefacts tend to focus on supporting a single user from a single discipline and this is clearly inadequate. Collaboration among different participants in the design of a building involves the ability of the different participants to work on their part of the project using their own particular ways of working yet being able to communicate with the other participants to implement the overall design of the building. The creation of different discipline models, the creation of relationships between the objects in the different models and the maintenance of consistency between the models are essential for a collaborative environment. Collaboration also requires support for both synchronous and asynchronous communication.

A collaborative design environment requires real-time multi-user collaboration by designers in different physical locations. This environment must provide 3D visualisation, walkthroughs and rendering to allow communication of the various views of the design as modelled by the different disciplines. This is of special importance at the conceptual stage of the design since much of the early collaborative decision-making is carried out at this stage. A virtual world environment based on an underlying object-oriented representation of the design is put forward here as an environment for synchronous collaboration in the design of buildings. This is in contrast to the decision made by Lee et al. (2003) to use a commercial CAD system for visualisation. One of the main advantages of virtual world environments is that they allow users to be immersed in the environment, allowing for real-time walkthroughs and collaboration (Conti et al., 2003). Moreover, CAD models contain a great deal of detail which makes real-time interaction extremely difficult.

This document introduces DesignWorld, a prototype system for enabling collaboration between designers from different disciplines who may be in different physical locations. DesignWorld, shown in Figure 1, consists of a 3D virtual world augmented with a number of web-based communication and design tools. Unlike previous approaches which use a single shared data model (Wong and Sriram, 1993), DesignWorld, uses agent technology to maintain different views of a single design in order to support multidisciplinary collaboration. This architecture enables DesignWorld to address the issues of multiple representations of objects, versioning, ownership and relationships between objects from different disciplines.

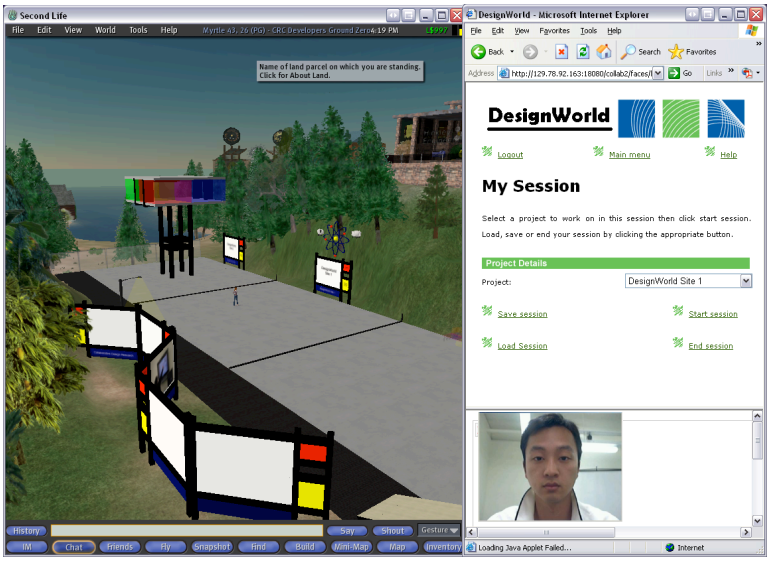


**Figure 1: DesignWorld is a 3D virtual environment augmented with web-based tools.**

# 3. LITERATURE REVIEW

DesignWorld is a system which we have developed in order to support collaborative design across the various disciplines involved in the construction industry. In this section we provide an overview of the current state of research on computer-aided collaborative design, particularly conceptual design, collaborative technologies and multi-view data modelling, and we identify DesignWorld's key advances in these areas. A more comprehensive review of these topics can be found in the review by Maher et al (2003).

Initially, DesignWorld's features were designed to focus on the needs of conceptual design, although there is no reason why this approach is limited to conceptual design. Conceptual design is the early design phase in which the scope of a problem is determined by exploring a range of alternative solutions to a brief or set of requirements. Conceptual design is characterised by a high degree of uncertainty where design ideas are very fluid. Traditionally conceptual design is assisted by pen and paper sketching for rapidly producing and developing solutions. Different disciplines approach conceptual design in different ways. Goldschmidt (1996) established two main characterizations of the design domain; the art-oriented approach and the engineering approach. Architects generally adopt the former, although this is not always the case, depending on the nature of the design problem. Goel (1995) found that architectural design was characterized by semantically dense and ambiguous design representations, generally from sketching, that helped generate a continuous flow of designs. Qin (2003) suggested that CAD systems do not readily support the art-oriented approach to conceptual design because they require complete, rigorous, geometric descriptions since they are optimized for specifying geometry rather than spatial creation and imagery. To address these issues, DesignWorld provides a sketching tool with which designers can rapidly produce design alternatives, whilst maintaining semantic density and ambiguous representation, and uses a 3D virtual world with a direct-manipulation-style building system that affords the kind of use suggested by Qin whilst not being tied down by the complexity and rigor required for a CAD system.

Conceptual design in architecture typically is typically a single person's responsibility and involves more individual work than team work (Lawson, 1996), whereas engineers tend to work as a team in the conceptual stage. In the engineering oriented design professions it is more common to generate ideas within a group of colleagues in, for example, initial brainstorming sessions. The different solutions of individuals are then combined in order to construct the total design. DesignWorld encourages architects to adopt the engineering mode of designing as a team and also supports interdisciplinary design with architects and engineers, whilst still supporting the possibility of working as a lone designer.

DesignWorld's development was influenced by current studies of effective collaborative workspaces and collaborative virtual environments. DesignWorld meets Wang et al's (2002) criteria for effective collaborative workspaces by supporting multiple work modes, asynchronous and synchronous; providing awareness of other users through avatars and visual cues for user actions in the 3D world; and providing for both synchronous communication via voice over IP and asynchronous communication via text messages in the 3D world. It also meets Fenves' (2000) criteria for effective collaborative support systems in supporting concurrent data access for designs in the 3D world, conflict resolution strategies through its releationship manager, data ownership through use of permissions in the 3D world and session management, and data distribution through the web and 3D world interfaces as well as email.

DesignWorld is more than just a collaboration tool, it is a Collaborative Virtual Environment (CVE); a distributed space supporting flexible and multiple viewpoints in which people can interact with other users, agents, and virtual objects. CVEs offer the ability to present large amounts of information in a manner that supports natural information lensing and multiple sensory modalities. Examples of other CVEs include Activeworlds[2], a 3D immersive world with an emphasis on community design, and NetSketch (LaViola et al 1998), a 3D design tool that allows small groups to communicate remotely on a shared virtual model with ambient cues for the actions of others. DesignWorld shares some of the features of these existing products, but uniquely extends their capabilities though its shared sketching tool, web cam, support for multi-disciplinary views, and relationship management system.

[2] http://www.activeworlds.com

The multi-disciplinary view system of DesignWorld is influenced by research into the problem of presenting the data contained in a model of a building so that different users with different stakes in the model are presented with the data tailored to their particular interests. As outlined by Maher et al (2004) and Rosenman et al (2005), what is meant by a view of a model is a way of conceiving of the thing represented by the model that reflects the particular interests of a user with particular stakes in the model. For instance, a structural engineer may see a building as a collection of structures with forces acting upon them, whereas an architect may see a building as a collection of spaces with functional requirements for their occupants. No one view of the model has precedence over another and models may contain some of the same entities, such as a wall that is common between an architect's model and an engineer's model. We have argued in Maher et al (2004) that any attempt to work with a single model composed of "primitives" from which multiple interpretations of the model can be derived cannot succeed, since the elements of the model that would count as "primitive" differ depending on the needs of the user. The CEDAR model developed by Naja (1999) is an example of such an approach to the problem of supporting multiple views, where an OO database maintains a kernel for the multiple representations of the model and information is shared between partial representations. In DesignWorld, by contrast, a model is maintained in an SQL database for each individual view of the building. Relationships between different views can be defined, so that for instance a certain wall in the architect's view may have a corresponding load bearing wall in the engineer's view. There is no monolithic view from which the other views are derived, reflecting our stance on multiple views in Maher et al (2004) and supporting Bucciarelli's (2003) contention that "No participant has at any stage in the [design] process a comprehensive, all-encompassing understanding of the design. No participant has a "god's eye view" of the design." The relevant information for forming the views is selected from the multiple models using agent technology.

In order to investigate the effectiveness of using virtual environments as collaborative tools, a comparative study has been done of designers collaborating face to face, using online sketching, and using an early prototype of DesignWorld. This study was done using Protocol Analysis. Protocol Analysis was developed by Ericsson and Simon (1984) then expanded and validated by an expert group of design researchers at the Delft Protocols Workshop (Cross et al 1996). It is now a standard technique for analysing team design. Protocol Analysis involves encoding time slices of a recorded design session following a pre-defined coding scheme in order to extract quantitative data about tasks performed by the designers. A discussion of the initial experimental design and technological considerations can been found in Candy (2004) and results from the studies can be found in Maher et al (2005) as well as on the project's website.[3] A follow up series of studies on using the version of DesignWorld described in this document for both intra-disciplinary and inter-disciplinary collaboration also appears in Maher et. al. (2005), using surveys, case studies, and Protocol Analysis to examine the kinds of design behaviours that DesignWorld users exhibit.

[3] http://www.arch.usyd.edu.au/~mary/CRCWeb/

# 4. COLLABORATION IN AUGMENTED VIRTUAL WORLDS

The complexity of building design leads to two conflicting requirements: the ability of the different disciplines to work on their part of the project using their own specific models, and the ability to communicate and negotiate with the other disciplines on the synthesis and integration of the different design models. Two approaches for addressing the need for a virtual environment in which designers can coordinate domain-specific and integrated models are: a multi-user CAD system and a multi-user virtual world. While the CAD system approach uses a familiar modelling environment, CAD systems and therefore the CAD models, tend to be specific to one discipline. We propose that a virtual world approach has more potential in providing a flexible approach for modelling and communication that is not discipline specific.

## 4.1 Multidisciplinary Modelling

Collaboration between designers of different disciplines is based on the premise that while there is a common purpose, each discipline has its own needs. Different disciplines have different views of a design object (building) according to their functional concerns and hence create different representations or models of that object to suit their purpose. For example, a building may be viewed as: a set of activities that take place in it; a set of spaces; a sculptural form; an environment modifier or shelter provider; a set of force resisting elements; or as a configuration of physical elements. Depending on the view taken, certain objects and their properties become relevant. For the architects, floors, walls, doors and windows, are associated with spatial and environmental functions, whereas structural engineers see the walls and floors as elements capable of bearing loads and resisting forces and moments. Hence, each will create a different model incorporating the objects and properties relevant to them. Both models must coexist since the two designers will have different uses for their models.

A single model approach to representing a design object is insufficient for modelling the different views of the different disciplines (Rosenman and Gero, 1996, 1998). Each viewer may represent an object with different elements and different composition hierarchies. While architects may model walls on different floors as separate elements, the structural engineers may model only a single shear wall encompassing the three architect's walls. Each discipline model must, however, be consistent vis-a-vis the objects described. While Nederveen (1993), Pierra (1993), Sardet et al. (1998) and Naja (1999) use the concept of common models to communicate between the discipline models, it is never quite clear who creates the common models and maintains the consistency between them and the discipline models. In this project, this consistency will be provided by interrelationships between the various objects in different disciplines modelled by explicit (bidirectional) links from one object to another. Figure 2 shows an example of this approach, with each discipline labelling its objects according to its need and corresponding objects associated with '*correspondsTo*' relationships. While this approach may have the disadvantage of replicating information about the same object in two object models, it saves the complexities of creating the common concepts and allows each discipline greater flexibility in creating its model. The discipline models allow each discipline to work according to its own concepts and representations. The whole model may be seen as the union of the different models.

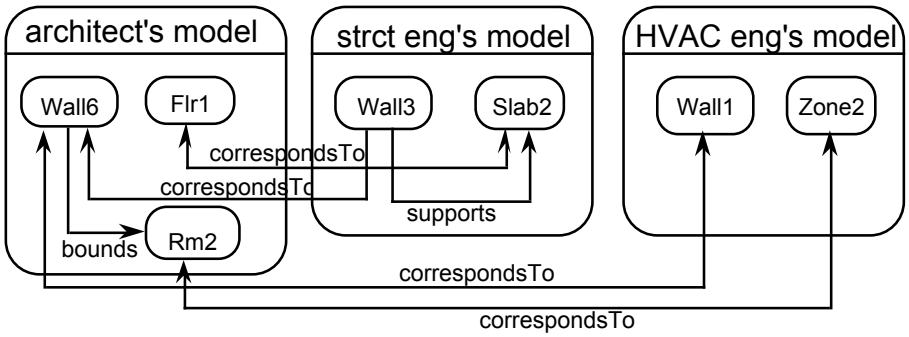


**Figure 2: Discipline models and relationships.**

## 4.2 Augmented 3D Virtual Worlds

A virtual world is a distributed, virtual space where people can interact with other people, objects or computer controlled agents using an avatar. Moreover, the worlds are based on object-oriented modelling concepts that concur with developments in CAD and 3D modeling software. As such, they provide a suitable platform for design and collaboration. DesignWorld uses the Second Life virtual environment as the platform for design and collaboration. However, while virtual worlds such as Active Worlds[4] and Second Life offer tools for creating and modifying virtual buildings and other artefacts, they do not offer features for managing multiple representations, versions or relationships necessary for multidisciplinary design. DesignWorld addresses this issue by augmenting Second Life with web-based tools and using agents to create views and relationships and to manage versions on behalf of designers as shown previously in Figure 1.

[4] www.activeworlds.com

# 5. DESIGNWORLD

## 5.1 Overview

DesignWorld is an improved version of the CRC Collaborative Designer (CCD) prototype (Rosenman et al., 2005). CCD was implemented using the Active Worlds virtual world platform. This new version, implemented in Second Life, provides facilities for modelling objects in the world and additional programming capability for associating objects in the world with an external data model.

DesignWorld consists of two main components, the client browsers and the web applications. There are two client browsers, the Second Life browser and the Web browser, which provides the extended capabilities to the Second Life virtual environment.

## 5.2 Functional Specification

DesignWorld is a prototype system for multidisciplinary, collaborative design. It allows designers to perform the following key functions:

- Collaborative building in 3D.
- View 3D models by designer discipline.
- Import 3D models from IFC files.
- Create inter-disciplinary and intra-disciplinary relationships within 3D models.
- Create 2D sketches.
- Communicate using video, voice and text.
- Manage projects.

The following sub-sections describe each of these functions in detail. Section 5.3 of this document contains a detailed technical specification of how each function is implemented in DesignWorld.

### 5.2.1 Collaborative Building in 3D

The key functional component for collaborative design is a suitable interface for building representations of design artefacts. Such interfaces should enable both synchronous and asynchronous manipulation and visualisation of objects shared by multiple designers. DesignWorld uses a 3D virtual environment as the tool for 3D design and collaboration. This environment allows users to build 3D models from basic polyhedra in a shared 3D environment as shown in Figure 3. It also supports synchronous and asynchronous text communication through chat and messaging systems. Users are made aware of one another's presence in the world through avatars representing each user and visual cues that show user actions such as typing, editing objects and looking at objects. The 3D virtual environment meets the following criteria specified by Anderson et al (2003) as requirements for a collaborative virtual environment:

- Avatars can convey non-verbal information.
- Suitable interfaces for collaborative manipulation and visualisation of shared objects.
- Scalable and flexible topological construction for virtual environments.
- Synchronous and asynchronous collaboration.
- Persistence in collaborative virtual reality.
- Interoperability with heterogeneous systems (specifically Macintosh and Microsoft Windows platforms.)
- Application specific servers for a representation of the virtual space.

In addition to this functionality the 3D virtual environment, SecondLife, provides for the following:

- Construction of 3D models using rectangular prisms, ellipsoids, cylinders and tori, along with shearing, scaling, hollowing, cutting, top resizing, translating and rotating operations.
- Custom texturing of virtual objects.

- Scripting objects to perform actions in the world such as moving or spawning other objects.
- Connection via XMLRPC to objects in the world to give them commands from external programs.
- Terraforming of virtual terrain.
- Buying and selling in-world goods using a virtual currency.
- Extensive avatar customization options.
- A system of permissions for object and land ownership.
- Integration between in-world communication and external email.
- A physics system modelling forces including gravity and inertia.
- Multiple modes of travel including flight and teleportation.

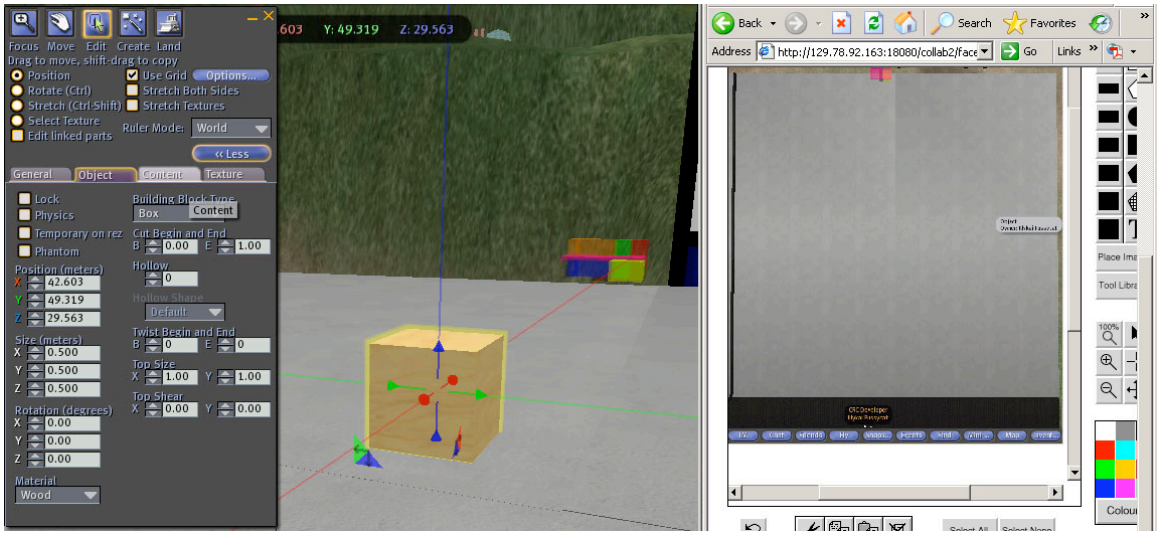


**Figure 3: DesignWorld's design modelling tools: 3D and 2D.**

### 5.2.2 Importing Models from IFC Files

With the extensive range of existing design tools comes the requirement for new collaborative design tools to provide some level of interoperability with existing data formats. The DesignWorld external model is compatible with Industry Foundation Classes (IFCs) (IAI, 2000) providing the potential for models to be uploaded from IFC compatible applications such as ArchiCad for use in collaborative sessions.

### 5.2.3 Creating Relationships in 3D Models

Relationships express the connections or commonalities between objects in a design. DesignWorld offers a relationship management tool which allows users to create corresponds to, decomposes to, supports, is a, adjacent to and bounds relationships between objects modeled in the 3D world.

- The *corresponds to* relationship creates an association between objects in different discipline models that are the same physical object but may have different non-geometric and non-physical properties. For example a wall in the architect's model may be the same as a wall in the structural engineer's model. The wall has the same shape, dimensions and materials but its function for the architect may be to provide privacy to a space whereas its function for the structural engineer may be to support a slab.
- The *decomposes to* relationship provides an association between a complex object and its components. This may also exist between objects in different disciplines. For example, a single wall object in the structural engineer's model may be associated with three walls (one above each other) in the architect's model.
- The *bounds* relationship provides for bounding associations between objects. For example, in the early conceptual design stages, an architect may only create spatial objects, whereas a structural engineer may create wall and slab objects. The relationship between the structural engineer's objects and the architect's object will be through a bounds relationship, e.g. Wall1(engineer object) bounds Space1(architect object).

- The *supports* relationship allows the association of inter or intra disciplinary objects where one object is providing physical support for the other.
- The *adjacent to* relationship allows the association of components that are required to be close to each other. For example, in the architects model of a restaurant, it may be a requirement for the kitchen to be adjacent to the dining area.
- The *is a* relationship allows objects to be grouped into logical structures with their own non-geometric properties. For example, an apartment building might consist of a number of flats which might in turn consist of walls, beams and columns.

### 5.2.4 Constructing Multiple Views of 3D Models

The models created by members of different design disciplines are influenced by the different functional concerns of those disciplines. For example, an architect may be concerned with the design of functional spaces within a building while a structural engineer may be concerned with the position of load bearing walls. DesignWorld offers a selection of viewing tools to enable designers to view the components of a design that are relevant to them:

- The discipline viewer uses object ownership and discipline information model to construct different views of a design with respect to the discipline of its designers.
- The object-property viewer uses discipline and relationship information to display non-geometric properties of objects that are not visible in the 3D virtual world.

### 5.2.5 Creating 2D Sketches

In addition to the 3D modelling tool, DesignWorld offers a tool for collaboratively creating 2D sketches. This allows designers to share their design ideas before committing them to a change in the 3D model. This tool enables designers to draw on a blank page, or over a snapshot of the site or current 3D model. It has the following functionality:

- Editing tools which include different shapes, different sizes, different colours, text input, pan and zoom.
- Ability to upload ASCII DXF files, GIFs and JPEGs
- Ability to export whiteboard background as GIF. Once designers finish discuss session or meet some important information which needs to be saved, they can save current background as a GIF image.

### 5.2.6 Communication

While the 3D virtual world incorporates text based ‘chat’ communication, audio and video teleconferencing can increase the sense of co-presence and provide a ‘hands free’ communication stream in situations where there is a need to convey complex ideas while manipulating objects in the design. Video also allows for real world media such as physical models to be viewed remotely. DesignWorld includes audio and video facilities to meet these needs.

### 5.2.7 Project Management

DesignWorld uses a project system to allow designers to organise their work. Each designer is a member of one or more project teams. Each project has an associated 3D model and may exist for a long period of time. Designers work on projects in multiple short design sessions. A session is associated with a single project only. These sessions may be synchronous, with multiple designers working together in a design session for a particular project, or asynchronous with just a single designer working on the project.

Designers can manage the projects of which they are a member and the project for their current session as well as update their personal details including email and password.

## 5.3 Technical Specification

DesignWorld uses a client-server architecture shown in Figure 4 to provide design and collaboration tools. Designers interact with DesignWorld using a client browser. The client browser, also depicted in Figure 1, has two components, a 3D world window and a web window. The 3D world window is the primary interface through which designers can build representations of design artefacts. We use the Second Life client browser for this purpose. Second Life is an online persistent space, created and changed by its users with built-in content creation tools. The client browser communicates with the Second Life servers (not shown in Figure 4) using TCP and UDP.

While Second Life supports collaborative virtual design with built-in design tools, it does not incorporate the multidisciplinary tools required by 'real world' designers such as architects and engineers. In order to provide designers with these additional design and communication tools, DesignWorld incorporates a web window next to the 3D world window to provide additional functionality and an external model to store additional data. The external model is stored in a MySQL database.

The web window displays Java Server Pages (JSPs). These pages are part of a J2EE web application. The application is hosted on a J2EE container. The container we chose was the Sun Java Application Server Platform Edition 8 included with the Sun Java Studio Creator, the IDE used to develop the web application. However, any other J2EE compliant container such as Apache Tomcat 5.57 or later could be used to deploy and host the web application.

The tools displayed in the web window include interfaces for viewing the non-spatial properties of a design, creating and managing relationships, sketching and audio-visual (AV) communication. The viewing and relationship management tools use forms to gather information about requests from designers for new views or relationships. These requests are carried out on behalf of the designer by agents. Agents are systems which can sense their environment using sensors, reason about their sensory input and act in their environment using effectors. DesignWorld agents can sense and affect the Second Life 3D environment, the JSPs and the external model defined by a MySQL database. DesignWorld agents are implemented in Java and hosted on the Sun Java Application Server. They communicate with the Second Life server using XML-RPC and with the MySQL database using Java Database Connectivity (JDBC). DesignWorld agents and their roles are discussed in detail in Section 5.3.

The sketching and AV communication tools are embedded as applets in the relevant JSPs and served by an Apache Tomcat server. The AV communication tool uses the Java Media Framework (JMF) to capture sound and video. This data is communicated to the Webcam/Audio server via HTTP. The Webcam/Audio server accesses the external model using JDBC to determine the correct recipients of the data then broadcasts the data to those designers. The sketching tool is the commercially available Groupboard tool. It uses the file system of the Tomcat Server to store images.

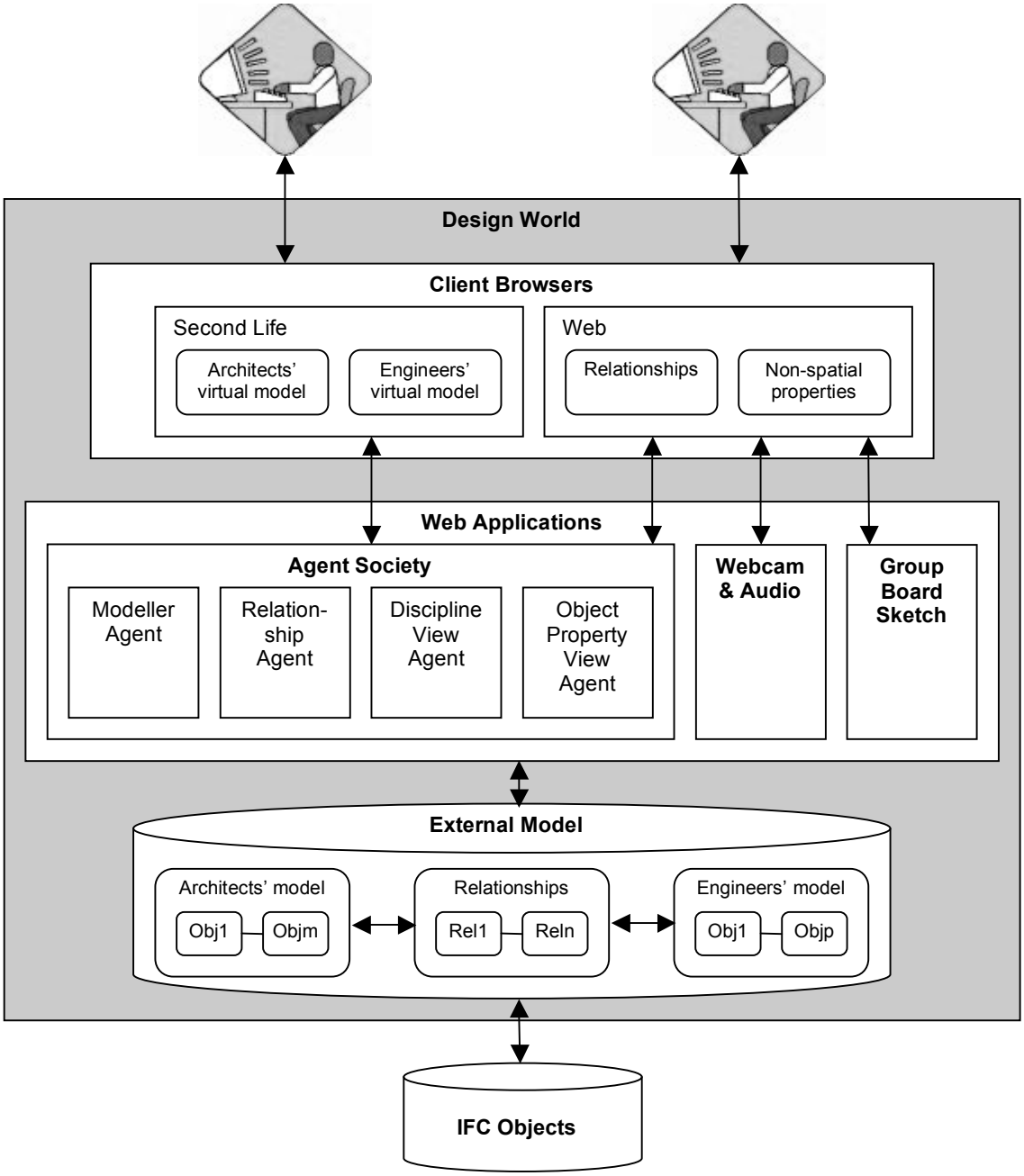


**Figure 4: The DesignWorld system architecture.**

### 5.3.1 The 3D Virtual World

DesignWorld uses the Second Life virtual environment as the tool for collaborative building in 3D. Second Life is a persistent online 3D world developed by Linden Labs. The servers on which Second Life runs are maintained by Linden Labs. The land in Second life is divided into "sims" that each run on separate servers, connected together into a grid. Linden Labs offer exclusive access to sims that are not part of the public grid for a larger fee, allowing for more control over who can visit and use the land and for finer customization of the sim.

Connecting to Second Life requires a free client program available from Second Life's website[5] and also requires registration for the user to activate a personal account. Basic accounts are free, but ownership of land requires the user to pay an ongoing monthly fee to Linden Labs which scales with

[5] http://www.secondlife.com

the amount of land owned. This fee is used to maintain the servers on which Second Life runs and to allow Linden Labs to continue to develop the Second Life code base and release regular patches that users must download. These regular patches add content and functionality to the world and client.

The Second Life client is run as a separate process alongside the DesignWorld browser window. The login details for a user in Second Life are not required to be the same as their DesignWorld details. DesignWorld's browser window communicates with the agent sensors and effectors in Second Life via XMLRPC.

Second Life's client has the following minimum system requirements:

***PC/Windows:***

- Computer: Pentium III 800MHz or higher, with 256MB RAM or more
- Operating System: OS: Windows XP (sp2) / 2000 (sp4)
- Video Card: nVidia Geforce 2 (32MB RAM) or higher, or ATI Radeon 8500 (32MB RAM) or higher
- Internet Connection: DSL, cable modem or LAN (256kbps downstream or higher)

***Mac:***

- Graphics Card: nVidia GeForce 2 (32MB RAM) or higher, or ATI Radeon 9000 (32 MB RAM) or higher.
- Computer: 1 GHz G4 or better, 512 MB RAM
- OS: Mac OS 10.3.8 or higher
- Internet Connection: Broadband (DSL/Cable Modem/LAN)

Only meeting the minimum requirements would provide very bad performance, recommended requirements for a PC are:

- Computer: Pentium IV 2GHz or higher, with 1GB RAM or more
- Operating System: OS: Windows XP (sp2) / 2000 (sp4)
- Video Card: nVidia Geforce 4 (64MB RAM) or higher, or ATI Radeon 9000 (64MB RAM) or higher
- Internet Connection: DSL, cable modem or LAN (256kbps downstream or higher)

### 5.3.2 The External Model

DesignWorld extends the concept of a persistent virtual world by maintaining an external model of designed structures in a SQL database in addition to the model maintained by the virtual world server. Use of an external model, makes it possible to store information about designs other than the spatial and rendering properties stored on the virtual world server. The DesignWorld external model contains project management, discipline, versioning and relationship information. The external model is compatible with Industry Foundation Classes (IFCs) (IAI, 2000) providing the potential for models to be uploaded from IFC compatible applications such as ArchiCad for use in collaborative sessions. The external model schema is shown in Figure 5.

The key design oriented components of the external model are the ModelledObject, DesignedObject and Relationship tables. The ModelledObject table stores information about objects that have been constructed by designers in the 3D world. The DesignedObject table stores information about objects that are to be or have been constructed by agents in the 3D world. It is envisaged that in future versions of DesignWorld the DesignedObject and ModelledObject tables will be merged to create a uniform means by which models can be imported from and exported to IFC files.

Designers can also define relationships between objects. These relationships are stored in the Relationship table. Information about designers themselves is stored in the Designer table. Each designer has a single discipline, a number of projects teams of which they are a member and a current project. Discipline types are stored in the Discipline Table. Projects are stored in the Project table. Information about project team membership is stored in the DesignerProject table.

Agent technology is used in DesignWorld to perform some tasks on behalf of Designers. Agents can focus their attention on a particular project by selecting different sensors and effectors. The types of agent sensors and effectors are stored in the AgentStructure table. Information about which sensors and effectors correspond to a particular project is stored in the AgentProject table.

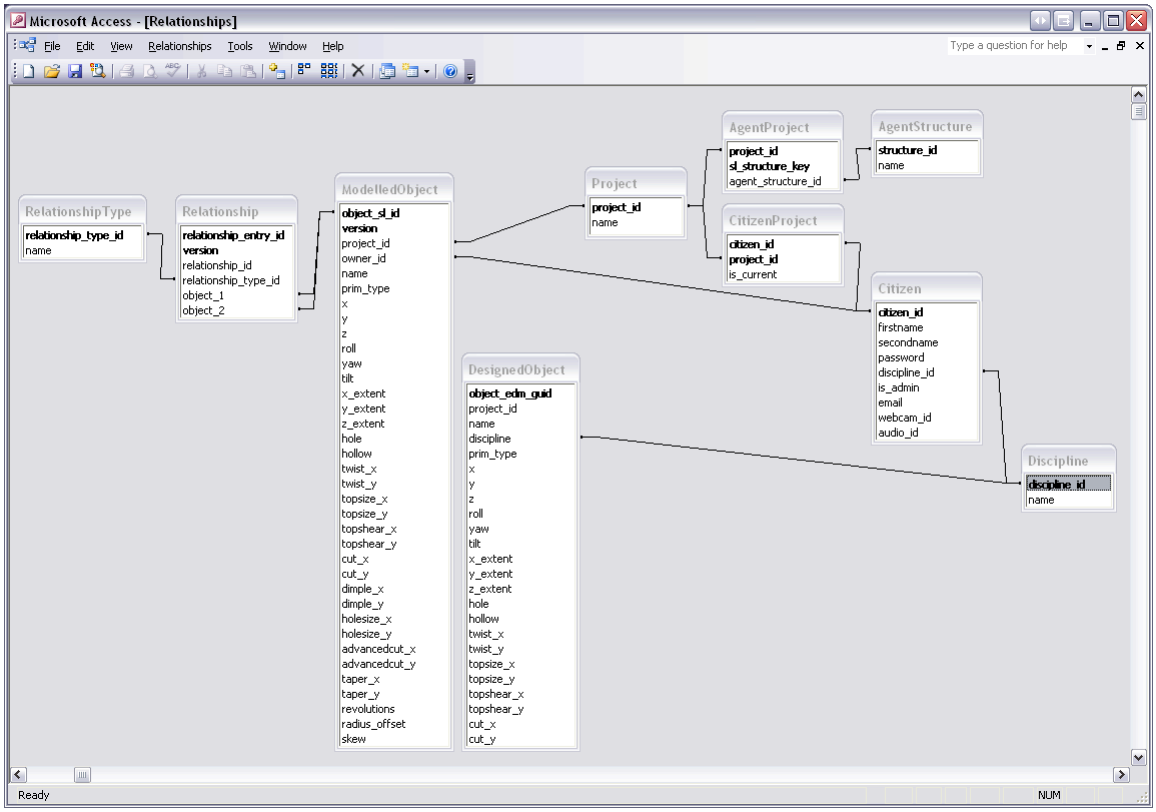


**Figure 5: The SQL external model schema.**

### 5.3.3. Design Tools

The web-based design tools provide designers with additional design functionality not usually associated with 3D virtual worlds. The web pages are the interface through which designers communicate with agents to request them to perform tasks on their behalf.

#### 5.3.3.1 Discipline Viewer

The discipline viewer allows designers to create and display the views of a model in Second Life as relevant to a particular discipline. A user may request a particular view in the web browser and an agent builds the view according to the objects belonging to that discipline. Two views of a tower model are shown in Figure 6. The Discipline Viewer Agent presents different views of a design relevant to designers from different disciplines by retrieving relevant information from the SQL external model and modifying the design displayed in the 3D virtual environment window.

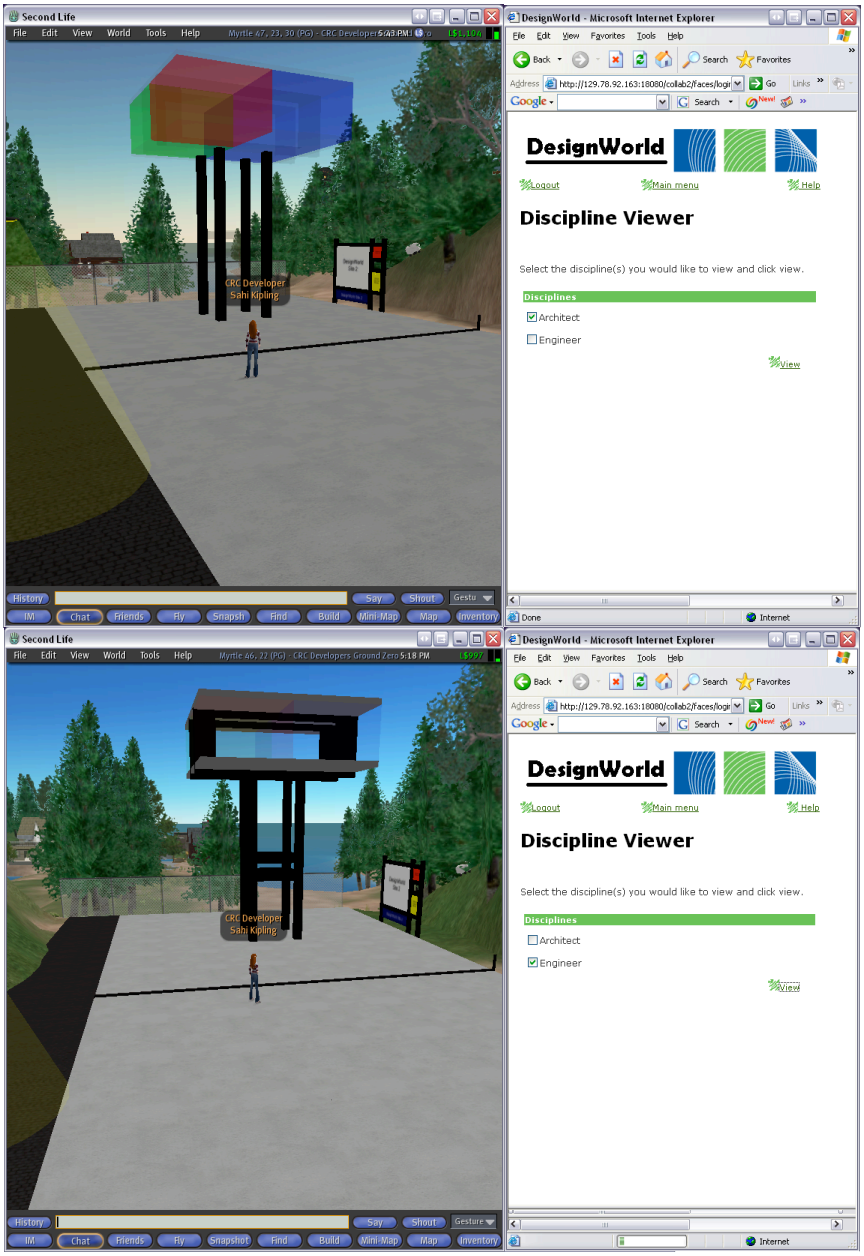


**Figure 6: (Top) The architect's view of a tower, (bottom) the engineer's view of the tower.**

#### 5.3.3.2 Relationship Manager

The relationship manager allows the designers to create and view the associations between different objects. A relationship is created by selecting a set of relating objects, a relationship type, a set of related objects and a number of notification types. Figure 7 shows the DesignWorld interface for creating relationships. On the left is the Second Life window showing a wall in the engineer's model. On the right is the Web browser window showing the creation of a bounds relationship between that wall and a space object in the architect's model. Relating objects form the left hand side of a relationship. Currently, DesignWorld supports corresponds to, bounds, decomposes to, supports, adjacent to and is a relationships. The related objects form the right hand side of the relationship. Notification types are the means by which designers will be notified if another designer moves an object that is part of one of their relationships. Currently the only notification type is a dialog box which pops up in the Second Life window. In the future email notification may also be possible.

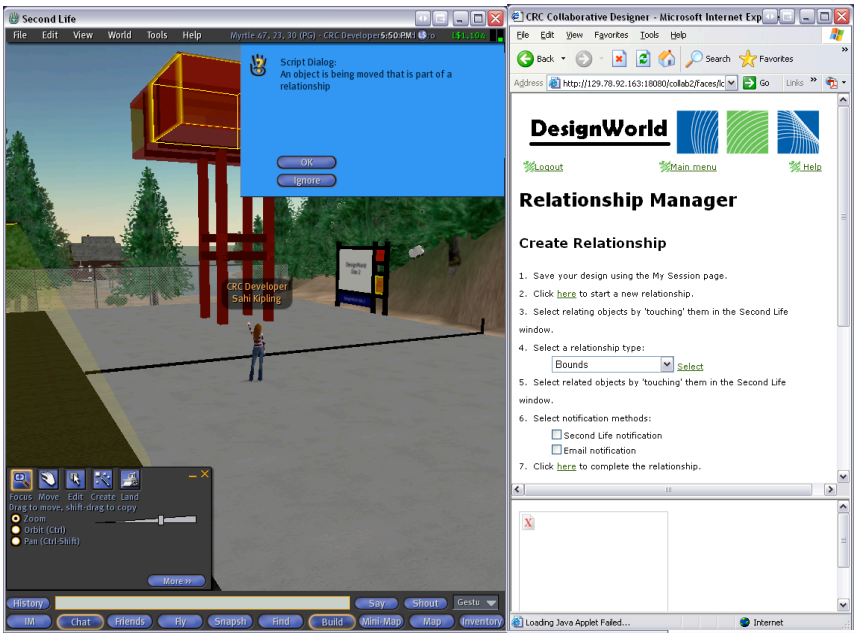


**Figure 7: The relationship manager (right) and a pop-up notification message (left).**

### 5.3.3.3 Object-Property Viewer

The object-property viewer allows designers to view those non-geometric properties of objects which are not visible in the Second Life interface. Designers view non-geometric properties by clicking on the desired DWObject in Second Life then clicking the view button. When an object is selected it turns yellow as shown on the left hand side of Figure 8. The non-geometric properties of the object are then retrieved from the external model and displayed in the web browser. At present, the non-geometric properties that can be viewed are the discipline to which the object belongs and the relationships associated with that object as shown on the right hand side of Figure 8. These properties are attached by DesignWorld. At present, non-geometric properties are not imported from the IFC model but could be in the future.

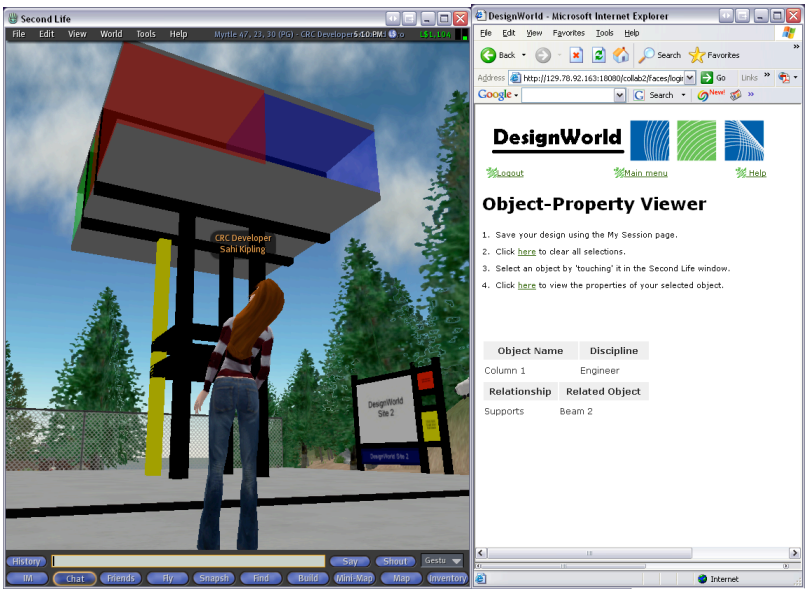


**Figure 8: The object property viewer displays non-geometric properties of objects.**

### 5.3.3.4 IFC File Upload

The purpose of developing an IFC to Second Life converter is to allow designers to work from an existing building model within the collaborative design environment. A building may be modelled in

an external CAD tool such as ArchiCAD, exported to an IFC file, and then converted to the Second Life primitive model. The user interface for the IFC converter and an example of an imported model are shown in Figure 9 and Figure 10. The converter application was implemented in the Java programming language and resides on the web application server. It imports an IFC file into a new model in an EDM database, iterates through each supported building element in the model, and creates entries in a relational database with the required information to create a Second Life primitive for the element. The building agent is then invoked to read the relational database and create the primitive shapes in Second Life.

The converter has been designed to work on the application server without user interaction – excluding initial parameters. It is the responsibility of the system administrator to provide details for the EDM database, the relational database and the location of the IFC file to be converted. The converter can import the IFC file into EDM in one of two ways. It can import it on a temporary basis, in which case there is no need to specify the repository or model name to be used. The model will be removed from EDM after conversion. Alternatively, the repository and model name may be specified, in which case the model is still available after use for other purposes. After conversion, a report may be obtained from the application for each building element type. Each element of that type has an entry indicating its globally unique identifier, whether or not the conversion was successful, and if not, the cause of the failure.

The conversion process attempts to convert all columns, beams, slabs, walls and spaces into a format suitable for Second Life. As discussed below, the primitive model used by Second Life limits the complexity of shapes that may be converted. For most instances of concrete columns and beams, this is not much of an issue, as they are usually represented as rectangular boxes, or cylinders. The other element types are regularly defined as non-rectilinear shapes, and therefore cannot be rendered accurately in Second Life. A bounding box representation is used for these cases.

In detail, a building element is converted using the following steps. Firstly, a check is made to determine if the geometric representation is defined by an extruded area solid. If so, the extruded area profile is checked to determine if it is a circular or rectangular area. If so, an appropriate Second Life primitive may be used. The primitive type is specified – either cylinder or box - and the dimensions of the shape are determined from the area profile and the extrusion depth. In the case of an element specified by an arbitrary profile area, a enclosing rectangular area is calculated and used instead. If the geometric representation is not suitable, for example a boundary representation is used, the converter defaults to converting the bounding box representation. In this case, a box primitive will be used, with the dimensions coming from the bounding box.

The global placement and orientation of each building element must also be determined. The IFC model has a concept of relative placement that is not shared by Second Life. For example, the placement and orientation of a column will usually be relative to its containing element (e.g. building storey), which is relative to the next containing element (e.g. building), and so on. The conversion process firstly traverses the chain of relative placements to determine a global placement and orientation for the element. It is then necessary to convert the vector-defined orientation to the Second Life system of yaw, pitch and roll.

The details for each element are stored in a relational database table as outlined below:

- **Designed_guid**: The Globally Unique Identifier for the element.
- **Designed_prim_type**: The primitive shape type that the geometrical representation will be based on (0 for a box, and 1 for a cylinder).
- **Designed_x**: The location of the element in the X plane.
- **Designed_y**: The location of the element in the Y plane.
- **Designed_z**: The location of the element in the Z plane.
- **Designed_yaw**: The number of degrees that the element should be rotated in the Yaw direction.
- **Designed_roll**: The number of degrees that the element should be rotated in the Roll direction.
- **Designed_tilt**: The number of degrees that the element should be rotated in the Tilt direction.
- **Designed_x_extent**: The X dimension of the element.
- **Designed_y_extent**: The Y dimension of the element.

- **Designed_z_extent**: The Z dimension of the element.
- **Designed_hole**: Unused.
- **Designed_hollow**: Unused.
- **Designed_name**: Human readable name for the element.
- **Designed_category**: Which design category the element belongs to (e.g. engineer, architect).
- **Designed_project**: An indicator of the project to which the element belongs.
- **Designed_twist_x**: Unused.
- **Designed_twist_y**: Unused.
- **Designed_topsize_x**: Unused.
- **Designed_topsize_y**: Unused.
- **Designed_topshear_x**: Unused.
- **Designed_topshear_y**: Unused.
- **Designed_cut_x**: Unused.
- **Designed_cut_y**: Unused.

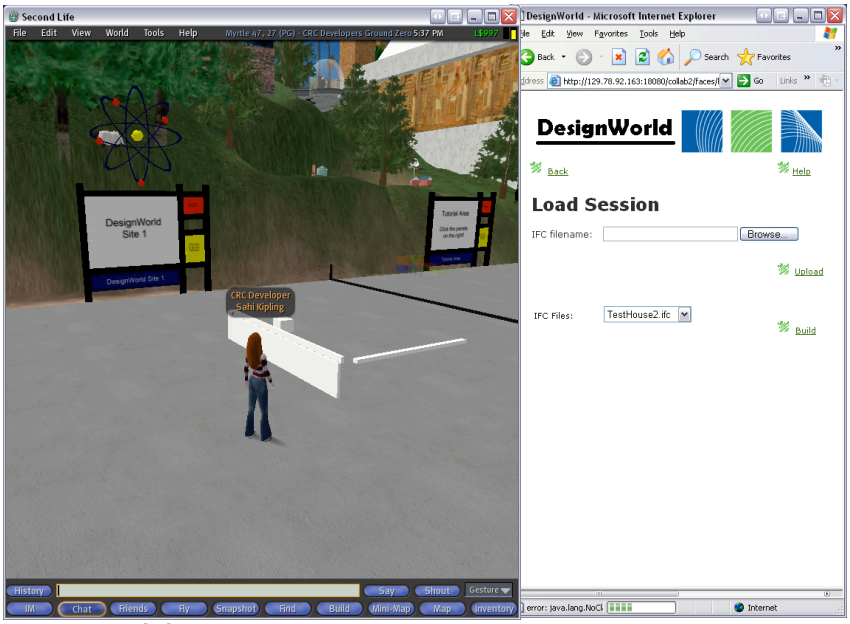


**Figure 9: The IFC file upload page (right) and a partially uploaded model (left).**

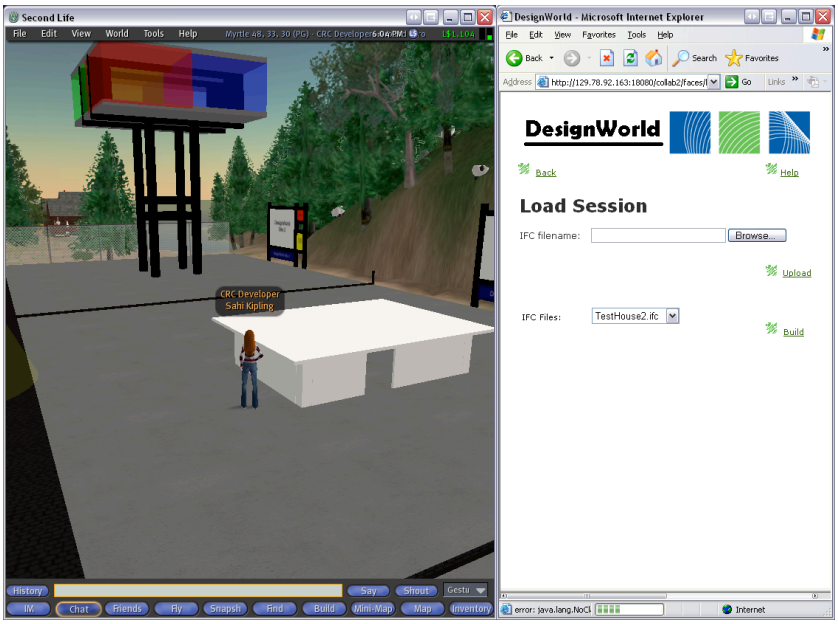


**Figure 10: The completed model house.**

The Second Life platform currently uses primitives and Boolean operations for the creation of content. While it is anticipated that Second Life will support meshes for content creation in the near future, the current version imposes limitations on what can be converted from IFCs successfully, and on how accurate a geometrical representation can be presented. To reduce the amount of data that needs to be sent to a Second Life client, elements in the CAD model are represented as primitives (i.e., box, cylinder, sphere, etc.) with very few defining parameters. Typically, a CAD model will contain extremely complex geometric representations for all but the simplest of elements. Consequently, only the simplest elements may be converted with any degree of accuracy. For elements with any complexity in their geometry, the bounding box representation is used instead.

There are also limitations due to the use of Boolean operations in Second Life. Although Second Life does support the concept of cuts and openings, the flexibility of the parameters to define them are so simplistic that they are not useful for converting CAD models. Therefore, all Boolean operations are ignored in the current conversion process. There are other limitations relating to the builder agent in Second Life that are due to the use of public land rather than purchasing a sim for building modelling purposes only. For example, elements may only be created within a ten meter radius of the building agent. It is therefore necessary to scale down a building model as part of the conversion process. This scaling factor limits the size of a building model that may be imported. Another limitation to the size of the building model is due to the speed with which elements may be created in Second Life. As the Second Life virtual world is publicly accessible, the time for a scripting command to complete is artificially high, to reduce the effect of malicious users. This results in a building element taking approximately 20 seconds to be created. A modestly sized IFC model may contain around one hundred applicable elements. For practical reasons, the converter has only been used for very simple proof-of-concept models. A more useful converter from IFC to Second Life depends on the use of a special purpose sim and the ability to create content using meshes rather than primitives.

It is quite feasible that a Second Life model could be converted to an IFC model, without losing any geometrical data. This would be useful to allow designers to use their model as a starting point within a CAD application after a session in DesignWorld, when they move on from the early sketching phase. Although the geometric representations can be automatically converted to the IFC format, non-geometric data would need to be manually entered by the Designer. For example, there would be no way to tell what type a building element is or what building storey it belonged to without user interaction, or an extension to the builder agent.

#### 5.3.3.5 2D Sketch Tool

DesignWorld uses the commercially available Groupboard Designer[6], an java based graphic annotation/mark-up tool for enabling multi-user 2D sketching. Groupboard consists of a C++ server, running under Windows NT/2000/XP or UNIX, which communicates with a Java client running in the web browser using TCP/IP. The client is written in Java JDK 1.0 and is compatible with most web browsers. Currently, a new trial version written in JDK 1.1 is also available.

On connection, the client opens up a direct socket connection to the server. If this fails (e.g. because of a firewall), then it opens up a tunnelling connection instead. Firewall tunnelling is handled by integrating a module into the web server which listens for requests to a specific URL and passes them on to Groupboard. The client makes a request every 1-2 seconds to receive any data waiting on the server and send out any data waiting to be transmitted. There is a module within the Groupboard which maintains persistent state information for these discrete connections. Although this is a lot less efficient than using a direct connection, it does mean that it will work with any firewall or proxy server.

In terms of bandwidth, Groupboard compresses the drawing data so that it is usable with a modem connection even when a few people are drawing at once. The average bandwidth used by Groupboard is 250kbps, assuming a maximum of 300 simultaneous users.

Because DesignWorld already incorporates a login function, the Groupboard login functionality is disabled by setting the value of “NEW_PASSWORD” to false. Groupboard is embedded into an html page by setting the value of “ALWAYS_FLOAT” as false. The Groupboard WIDTH and HEIGHT parameters are set to 330 and 480. The chat window is disabled as chat is already facilitated by the

[6] www.groupboard.com

webcam and Second Life. Multiple users can connect to the same board at the same time and multi-user pan is enabled to allow all users to control image and background movement.

### 5.3.4 Communication Tools

DesignWorld includes audio and video communication via a webcam. The webcam component is a Java applet that is based on Sun's Java Media Framework 2.1.1e (JMF). The Java Media Framework API enables audio, video, and other time-based media to be added to applications and applets built on Java technology.

There are three main components to the webcam applet: a Transmitter component, a Receiver component and a Graphical User Interface (GUI) applet component. The Transmitter component captures video and audio frames from the local device, encodes the data into some form of raw real time protocol (RTP) format and then transmits the encoded data over the network using an RTPManager. The Receiver component on the other end receives the encoded data from the network and renders it on the webcam applet GUI. The Webcam applet monitors both the transmitter and the receiver and handles user interaction.

When designers start a session on a particular project, they are able to see and hear all other designers who currently have sessions on the same project. In addition, designers are able to see themselves. This makes it easier for designers to position physical artefacts in front of their webcam for viewing by other designers.

### 5.3.5. Project Management Tools

The pages shown in Figure 11 allow users to start, end and save a design session. Users can manage their projects and personal details via the web-page shown in Figure 12.


**Figure 11: The MySession page allows users to start, save and end sessions on a project.**


**Figure 12: User Preferences for managing personal details and projects.**

### 5.3.6 Agents

Agents are software systems that can sense their environment using sensors, reason about sensory input using some characteristic reasoning process and act in their environment using effectors. This general model is shown in Figure 13. The general function of an agent is to act for or represent another. While this does not preclude an agent from acting on its own behalf, an agent must have some capacity and propensity for acting on behalf of another.

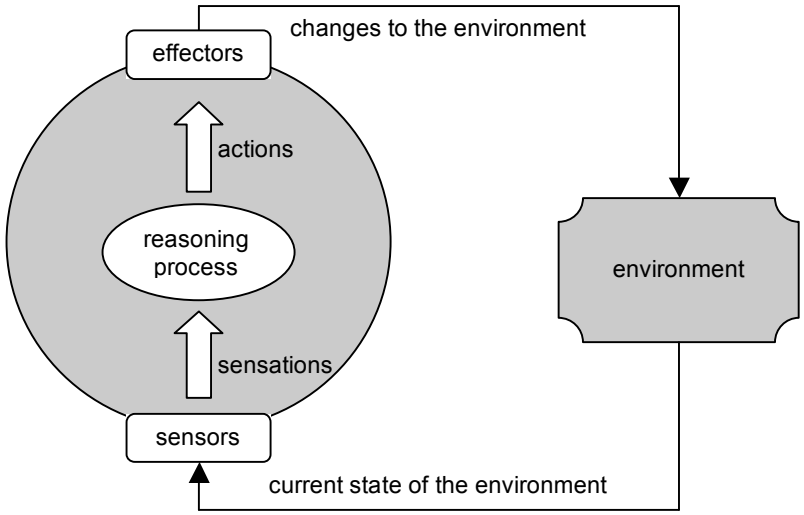


**Figure 13: The general agent model.**

An environment is a set of circumstances surrounding an agent. The environment for DesignWorld agents comprises three sub-environments as shown in Figure 14: a MySQL database, the Second Life virtual environment and the web-browser (HTTP) environment. These environments represent the three key components of the DesignWorld system architecture. DesignWorld agents use a reflexive reasoning process to select actions based on sensory input. They use four structures: sensations, states, rules and actions. These structures are connected by two reasoning process: sensation and action. The sensation process groups data sensed from the agent's sub-environments into a single description of the current state of the environment. The action process uses pre-programmed rules to produce an action in response to the state of the agent's environment.

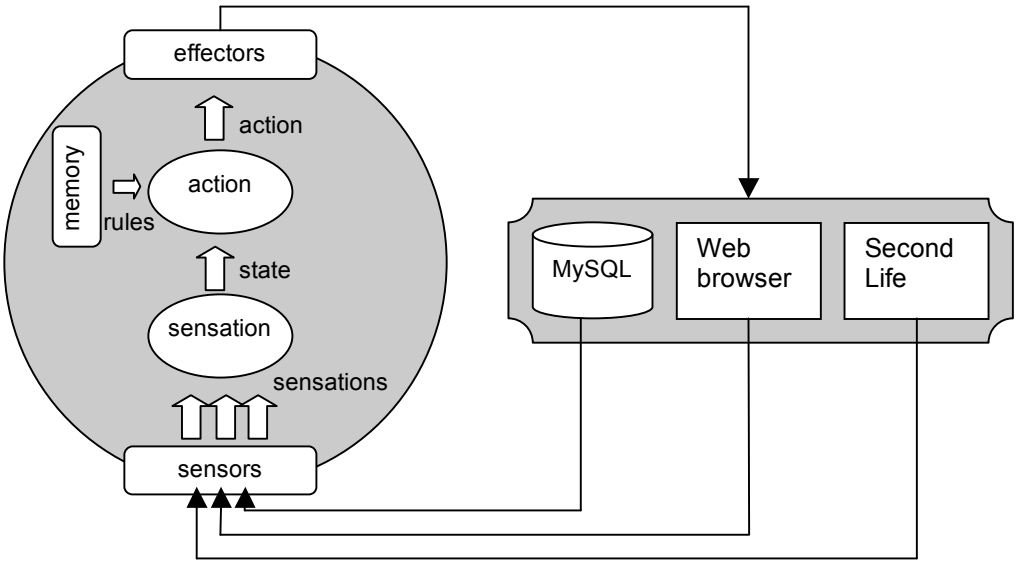


**Figure 14: The DesignWorld agent model.**

#### 5.3.6.1 Sensors and Effectors

Agents receive information about the state of their environment using sensors while effectors are the means by which actions are achieved in the environment. DesignWorld agents have three types of sensors and effectors to enable them to sense or affect the Second Life, HTTP and SQL environments.

***Second Life Sensors and Effectors***

The sensors and effectors that allow a DesignWorld agent to sense or affect the Second Life environment have two parts, a Java component and a Linden Scripting Language (LSL) component as shown in Figure 15. These components communicate via XML-RPC. In addition to the LSL component of each sensor and effector, each object in a design also contains an LSL script which contains rules defining how the object responds to messages from sensors and effectors. Objects communicate with the LSL components of sensors and effectors by broadcasting chat on hidden channels.

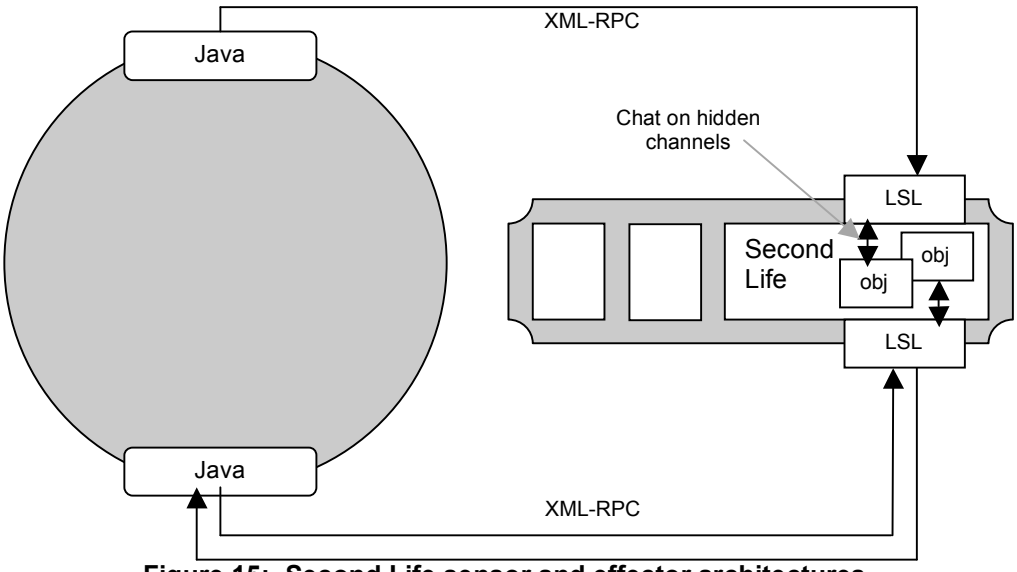


**Figure 15: Second Life sensor and effector architectures.**

Second Life sensors are active sensors. The agent controlling them must actively send a message to its sensor which then scans the Second Life environment and returns data describing it. Only small amounts of data (256 characters) can be transferred between the LSL and Java sensor components after

any scan. Thus individual sensations contain only a partial description of the environment. The sensation process is responsible for combining individual sensations into more complete, higher level, descriptions of the current state of the world. In addition to the limited amount of data which can be sent in a single transfer, Second Life also imposes a server side delay on a number of operations such as the XML-RPC response, chat and listen operations. These delays are for security purposes but have the undesirable side-effect of making some sensor and effector operations quite slow.

DesignWorld agents may make use of any of the Second Life sensors shown in Table 1 or the effectors shown in Table 2.

| SL Sensor | Description |
|---|---|
| Object Sensor | Senses objects in the Second Life environment. |
| Selected Object Sensor | Senses objects that have been 'touched' in the Second Life environment. |
| Chat Sensor | Senses chat in the Second Life environment. |

**Table 1: Second Life sensor capabilities.**

| SL Effector | Description |
|---|---|
| Chat Effector | Broadcasts text on a specified channel. |
| Unselect Object Effector | Unselects all objects. |
| Relate Effector | Relates objects |
| Add Object Effector | Adds an object to the 3D world. |

**Table 2: Second Life effector capabilities.**

The objects with which users construct their designs are called DWObjects and contain an LSL script that defines how they should respond to messages from sensors or commands from effectors. The object script listens for messages from effectors on a hidden chat channel. The channel on which the message occurs indicates the rule which should be fired. The message itself contains the parameter values required for the rule to be executed. Table 3 lists the effector rules contained in the object script. The object script can also fire rules in response to user actions in the Second Life environment in order to register information with the LSL component of a sensor in preparation for messages from the Java component of that sensor. Table 4 lists the sensor rules contained in the object script.

| Channel | Parameters | Rule |
|---|---|---|
| Visible | objectID | If objectID equals the objectID of this object then it set its transparency to zero. |
| Invisible | objectID | If objectID equals the objectID of this object then set transparency to the maximum level. |
| Relate | objectID | If objectID equals the objectID of this object then set Related to true. |
| Un-relate | objectID | If objectID equals the objected of this object then set Related to false. |

**Table 3: Object script rules for communicating with effectors.**

| Channel | Parameters | Description |
|---|---|---|
| Rez | objectID | When a new object is created (rezzed), then it registers its existence with the ObjectSensor by broadcasting its objectID on the Rez channel. |
| Object | objectID | If objectID equals the objectID of this object then send details of location, orientation, etc to the ObjectSensor by broadcasting them on the Object channel. |
| Select | objectID | When an object is 'touched' and it is not selected, then it becomes selected by sending its objectID to the SelectedObjectSensor on the Select channel. |
| Unselect | objectID | When an object is 'touched' and it is selected, then it becomes unselected by sending its objectID to the SelectedObjectSensor on the Select channel. |

**Table 4: Object script rules for communicating with sensors.**

### *HTTP Sensors and Effectors*

The sensors that allow a DesignWorld agent to sense the HTTP environment are written in Java. In contrast to Second Life sensors, HTTP sensors are passive sensors. This means that the agent passively waits for a message to be sent to these sensors rather than actively requesting data. DesignWorld agents may make use of the HTTP sensors shown in Table 5 and effectors shown in Table 6.

| HTTP Sensor | Description |
|---|---|
| Parameter Sensor | Senses parameters with a label and value as follows:<br>`param(label, value)` |

**Table 5: HTTP sensor capabilities.**

| HTTP Effector | Description |
|---|---|
| Property Effector | Displays the non-geometric properties of an object on a web-page. |

**Table 6: HTTP effector capabilities.**

### *SQL Sensors and Effectors*

Like HTTP sensors, the sensors and effectors that allow a DesignWorld agent to sense or affect the SQL environment are written in Java. SQL sensors are active sensors. DesignWorld agents may make use of the SQL sensor shown in Table 7 or the effector shown in Table 8.

| SQL Sensor | Description |
|---|---|
| Query Sensor | Senses entries in an SQL database defined by a SELECT query. |

**Table 7: SQL sensor capabilities.**

| SQL Effector | Description |
|---|---|
| Update Effector | Modifies a SQL database using an ADD, DELETE or UPDATE query. |

**Table 8: SQL effector capabilities.**

#### 5.3.6.2 Reasoning Processes

DesignWorld agents use four structures: sensations, states, rules and actions. These structures are connected by two reasoning process: sensation and action. The sensation process groups data sensed from the agent's sub-environments into a single description of the current state of the environment. The action process is a reflexive process which uses pre-programmed rules to produce an action in response to the state of the agent's environment. The rules available to an agent define the role it plays in DesignWorld. Currently, there are four DesignWorld agents: the Modeller, the Relationship Manager, the Discipline Viewer and the Object-Property Viewer agents.

### *Modeller Agent*

The Modeller agent facilitates the presentation of different views of a design by constructing and maintaining a data model of the design artefacts in the SQL external model. This persistent model is capable of describing more properties of an object than can be represented in the 3D environment. For example, in Second Life an object may have an owner but the SQL external model might additionally specify a project and a design discipline to which the owner and the object belong. The rules of the Modeller Agent are shown in Table 9.

| State | Rule |
|---|---|
| Message from HTTP environment requesting model of project P | Sense IDs of SL sensors S for project P in external model. |
| IDs of sensors S for project P | Sense state of SL using sensors S to sense objects O. |
| Objects O | Sense non-geometric properties O' of objects O in external model |
| Objects O' | Affect SQL environment by deleting oldest version of project P then affect external model by inserting objects O'. |

**Table 9: Modeller Agent rules.**

### *Discipline Viewer Agent*

A specific model of an object is a representation of that object resulting from taking a particular view. Given a design object, such as a building, there are many views that may be taken, leading to different conceptual interpretations. For example, a building may be viewed as a set of activities that take place in it, as a set of spaces or as sculptural form. Depending on the view taken, specific properties and descriptions of the object become relevant. Architects will model certain elements such as floors, walls, doors and windows associated with the spatial and environmental qualities with which they are concerned. Structural engineers, however, see the walls and floors as elements capable of bearing loads and resisting forces and moments. Both models must coexist since the two designers will have different uses for their models. The Discipline Viewer Agent presents different views of a design relevant to designers from different disciplines by retrieving relevant information from the SQL external model and modifying the design displayed in the 3D virtual environment window. The rules of the Discipline Viewer Agent are shown in Table 10.

| State | Rule |
|---|---|
| Message from HTTP environment requesting view V of project P | Sense objects O from view V of project P in the external model. |
| Objects O | Affect the SL environment by making all objects invisible then affect the SL environment by making objects O visible. |

**Table 10: Discipline Viewer Agent rules.**

### *Object-Property Viewer Agent*

The DesignWorld external model is capable of representing more information about a design object than it is possible to display in the 3D virtual environment window. For example DesignWorld associates disciplines, projects and relationships with objects. The Object-Property Viewer Agent displays non-geometric information about design objects by retrieving relevant information from the SQL external model and displaying it in tabular form in a web-browser. The rules of the Object-Property Viewer agent are shown in Table 11.

| State | Rule |
|---|---|
| Message from HTTP environment requesting view V of project P | Sense objects O' from view V of project P in the external model. |
| Objects O' | Affect the HTTP environment by displaying non-geometric properties of objects O' in a browser window. |

**Table 11: Object-Property Viewer Agent rules.**

### *Relationship Manager Agent*

When several views of a design exist, there may be relationships between the elements that compose one view and the elements that compose another. There may also be relationships between the elements that compose a single view. Relationships express the associations between objects in a design. For example, one object may bound another object, one object may be correspond to another object or one object may decompose to several other objects. The Relationship Manager Agent takes information specified by a system user and uses it to create a persistent record of design relationships in the SQL database. The rules of the Relationship agent are shown in Table 12.

| State | Rule |
|---|---|
| Message from HTTP environment requesting unselect on project P | Affect SL environment by unselecting all objects on project P. |
| Message from HTTP environment requesting relationship type T | Sense all selected objects in the SL environment on project P then affect SL environment by relating all selected objects on project P then affect SL environment by unselecting all objects on project P. |
| Message from HTTP environment requesting relate on project P | Affect SL environment by relating all selected objects on project P then affect the SQL environment by inserting relationship records. |
| Message from HTTP environment requesting delete relationship R on project P | Sense SQL environment to retrieve objects O in relationship R then affect the SQL environment by deleting R then affect the SL environment by un-relating objects O. |

**Table 12: Relationship Manager Agent rules.**

### *Builder Agent*

The Builder Agent reads data from the external model and transforms it into a model in the 3D world. Data is uploaded to the external model by the IFC converter tool in response to a user action on the Load Session web-page. The Builder Agent currently produces a structure composed of 'drone' objects which do not have the functionality of a DWObject. In addition, they are limited by the Second Life rez() function to building models within a 10 metre radius. Models which are larger than this must be scaled down before they are imported into the SQL database.

| State | Rule |
|---|---|
| Message from HTTP environment requesting IFC to be built | Sense objects O' of project P in the external model. |
| Objects O' | Affect the SL environment by building objects O'. |

**Table 13: Builder Agent rules.**

# 6. DESIGNWORLD IN PRACTICE

Adel Andrews, an architect with the firm Urban Vibe, and Mary Gilman, an engineer with the firm Stadthoffer Technik, are collaborating using DesignWorld to design an exclusive restaurant tower for a

developer in Sydney. The brief stipulates that the restaurant proper has to be positioned with harbour views. Adel and Mary arrange via email to work on the brief.

Adel logs into DesignWorld and sees on the webcam that Mary is already waiting for him. They exchange greetings, log into the 3D world, and start the design session.

In the 3D world a full-scale mock-up of the real life site has been created, so that Adel and Mary can get a sense of the surrounding landscape and the height required for the harbour view. Adel and Mary fly up with their avatars to 60 meters in the air as shown in Figure 16 and satisfy themselves that the harbour view from that height is acceptable.

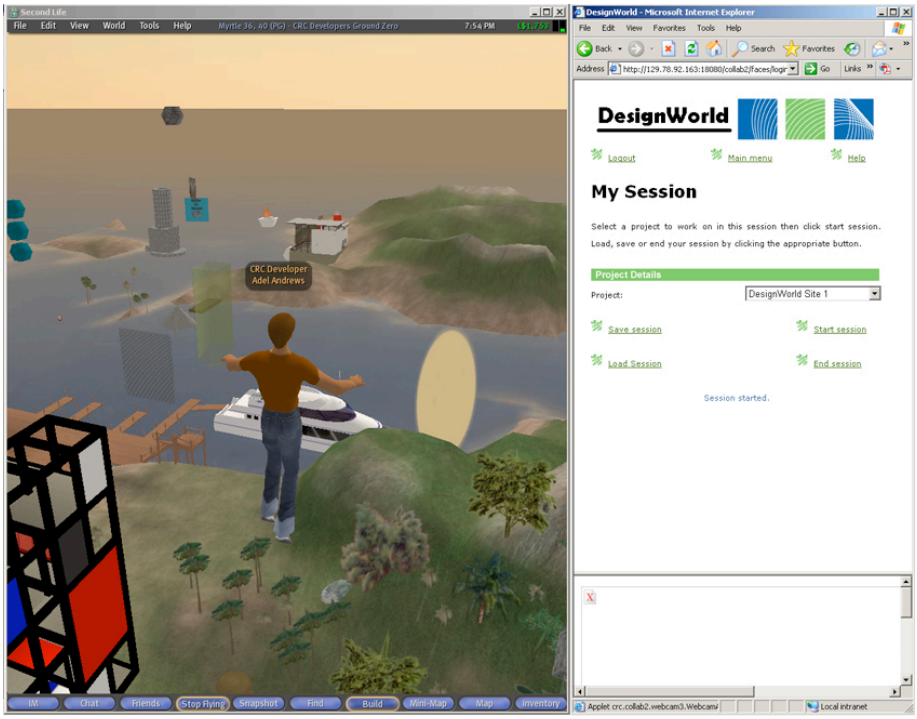


**Figure 16: Adel inspects the view of the harbour.**

Having established the height that they want their restaurant tower to reach, Adel and Mary open the sketching tool as shown in Figure 17 to draw and discuss possible layouts for the restaurant, finally settling on a design with two elevators to access the restaurant. Mary then proposes a cross-sectional view of the tower using the sketching tool, which Adel agrees to after modifying the supporting struts to extend above the ceiling to produce the aesthetic that he's looking for.

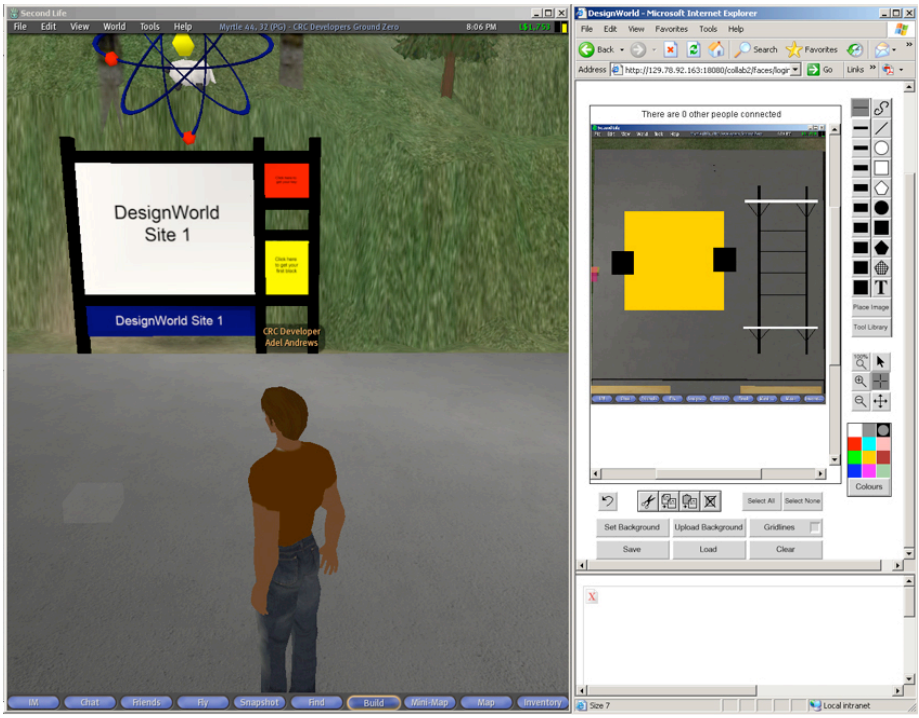


**Figure 17: Mary and Adel complete their sketch.**

Keeping the sketching tool open as a reference, Mary begins to build the supporting beams, columns, and elevator shafts in the 3D world as shown in Figure 18. Adel simultaneously creates transparent blocks in the 3D world to represent the foyer area and the restaurant proper as shown in Figure 19.

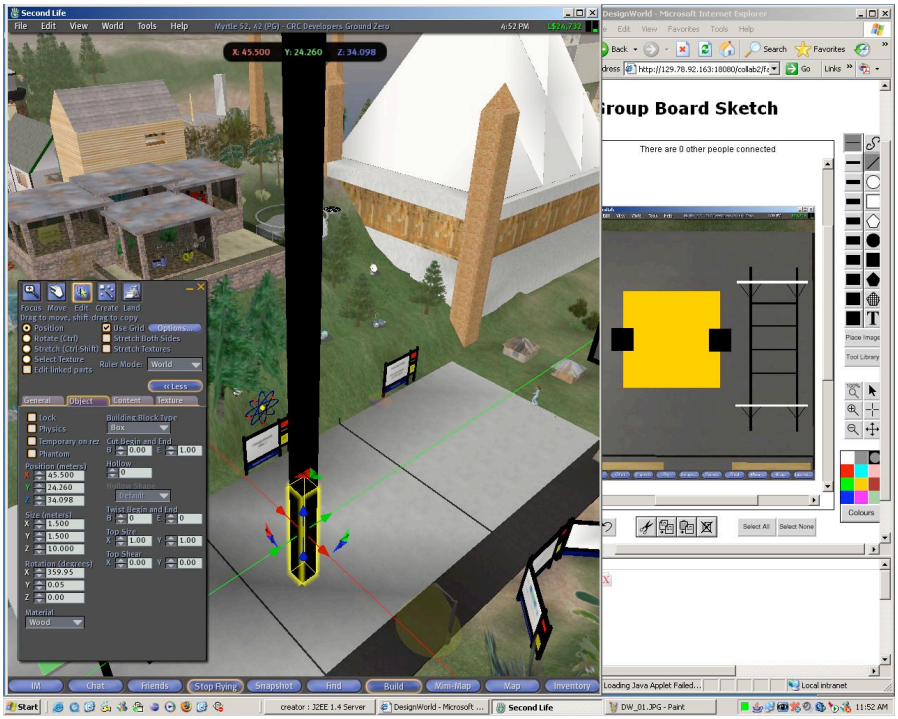


**Figure 18: Mary starts work on the columns...**

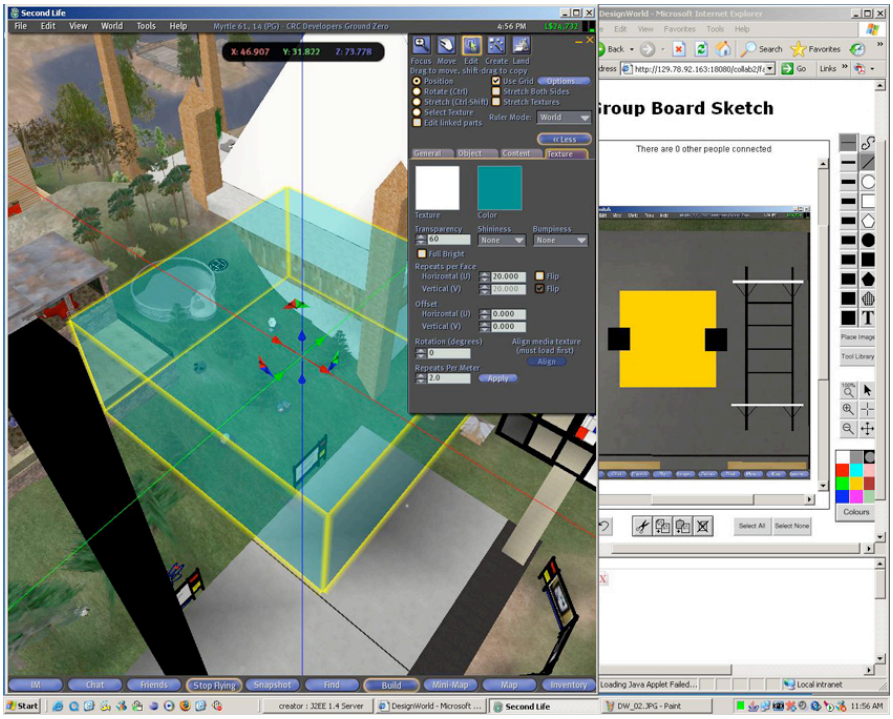


**Figure 19: …Whilst Adel creates the restaurant space.**

As he finishes placing the spaces and watches Mary building, Adel realises that building regulations for the part of Sydney in which the building is being designed require emergency fire exists which the design is lacking. He alerts Mary to this and they go back to the sketching tool, adding two stairwells to access the restaurant. Mary quickly duplicates the elevator shafts in the 3D world and changes their description so that they represent the stairwells and places them, working off the modified sketch.

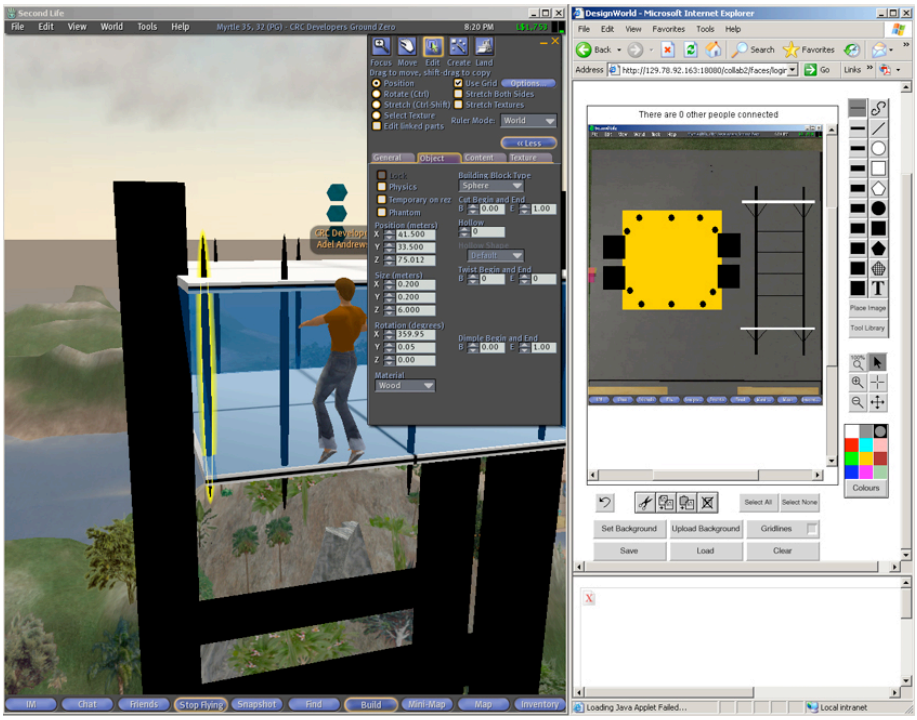


**Figure 20: Adel adds the columns following the new sketch.**

Adel explains to Mary that he wants a series of columns around the windows of the restaurant to produce the aesthetic that he has in mind, marking up the sketch to show where he envisages them as shown in Figure 20. Mary says that those columns would not be load bearing, so Adel builds them as part of the architect model whilst Mary adds ceilings and floors to the spaces built by Adel.

Adel has to leave in order to attend a meeting before the remainder of the detail can be put into the model, so the two save their work. After Adel leaves for his meeting, Mary defines relationships between the elements of the model that bound the spaces in the model as shown in Figure 21. She also defines relationships to show which elements support other elements so that if either of them modify some part the model in the future DesignWorld will alert them that their change impacts on other parts of the model.

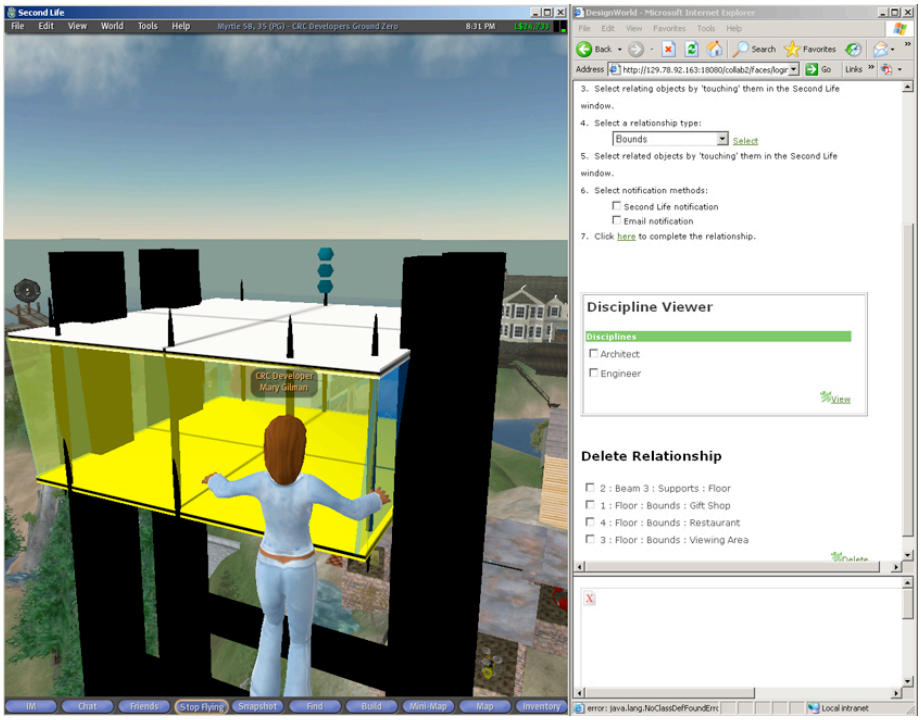


**Figure 21: Mary creates a bounding relationship between the floor and the restaurant.**

Before finishing the design session Mary inspects the model once more using the discipline viewer from an architect's view and an engineer's view, checking that it matches the sketch and that each part is owned by the correct person, as shown in Figure 22. She then sets the model back to the default view, logs out of the 3D world and ends her DesignWorld session.

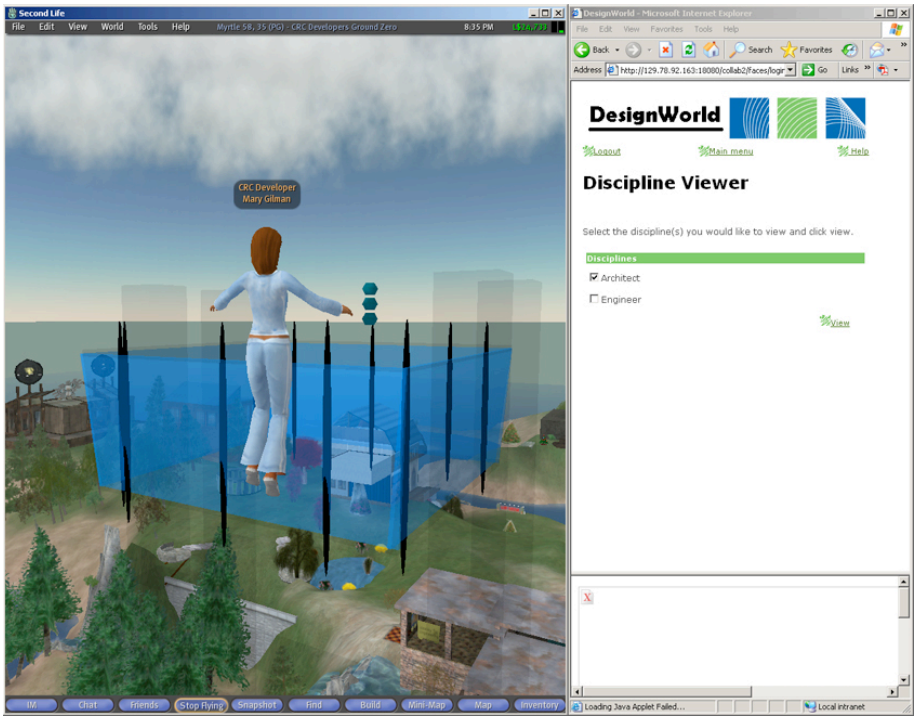


**Figure 22: Mary inspects the architect view of the completed tower.**

# 7. BENEFITS OF DESIGNWORLD TO INDUSTRY

In the AEC industry, most major projects involve design teams who have come together for that project and need to collaborate to arrive at a design solution. The members of the team are, in many cases, in disparate locations spread around the world. At present, collaboration is carried out through infrequent face-to-face meetings which are expensive and time-consuming or video-conferencing which only allows partial sharing of design information. It is not currently possible to share large CAD files in real time. DesignWorld allows participants in the design of a project to successfully collaborate towards a conceptual design. DesignWorld takes into account the needs of all participants, clients, architects, consultants and contractors, through its ability to allow the modelling of multiple views of a design object. The lack of multi-view modelling capability has been an impediment to the ability of the various disciplines involved to work on their part yet work together in a collaborative mode. By providing an immersive collaborative environment responsive to the participants' needs, it allows participants distributed over distant locations to synchronously communicate and collaborate on a design project. DesignWorld provides the necessary tools for such industry collaboration, 2D sketching and 3D modelling, the ability to visualise in real-time 3D and video and audio communication in addition to text communication. It is envisaged that once industry familiarises itself with the concepts and the technology, it will find that DesignWorld provides an appropriate environment for multidisciplinary design collaboration.

# 8. REFERENCES


Anderson, L., Esser, J., and Interrante, V. (2003) A virtual environment for conceptual design in architecture, *Workshop on Virtual Environments, Zurich, Switzerland*.

Bucciarelli, LL: 2003, Designing and learning: a disjunction in contexts. *Design Studies*, **24**(3):295-311.

Conti, G, Ucelli, G and De Amicis, R: 2003, JCAD-VR – a multi-user virtual reality design system for conceptual design, in *TOPICS. Reports of the INI-GraphicsNet*, 15:7-9.

Cross, N., Christiaans,H. and Doorst, K.(eds.) (1996) *Analysing Design Activity*. John Wiley and Sons, Chichester, West Sussex.

Ericsson, K.A. and Simon, H.A.(1993) *Protocol Analysis: Verbal Reports as Data*, MIT Press, Cambridge, MA, Revised Edition.

Fenves, S. J., Rivard, H., and Gomez, N., (2000) SEED-Config: a tool for conceptual structural design in a collaborative building design environment," *Artificial Intelligence in Engineering*,14 (3), pp. 233-247.

Goel, V. (1995) *Sketches of Thought*, MIT Press, Cambridge MA.

Goldschmidt, G. (1996) The designer as a team of one. In N. Cross, H. Christiaans, and K.Doorst (eds.) *Analysing Design Activity*, John Wiley and Sons, Chichester, West Sussex.

IAI (2000). Industry Foundation Classes-Release 2x, IFC Technical Guide.
http://www.iai-international.org/iai_international/Technical_Documents/documentation/IFC_2x_Technical_Guide.pdf

LaViola, J.J., Holden, L.S., Forsberg, A.S., Bhuphaibool, D., and Zeleznik, R.C. (1998) *Collaborative Conceptual Modeling Using the SKETCH Framework.* In *Proceedings of the IASTED International Conference on Computer Graphics and Imaging*, 154-158, June 1998.

Lawson, B. (1996) *Design in Mind.* Butterworth Architecture, Oxford

Lee, K, Chin, S and Kim, J: 2003, A core system for design information management using Industry Foundation Classes, *Computer-Aided Civil and Infrastructure Engineering*, **18**:286-298.

Maher, ML, Bilda, Z., Candy, L., Dong, A., Kim, M., Rosenman, M., Shi, T., Yoon, JS., Marchant, D.,: 2003, Literature Review, CRC Technical Report, Project 2002-24-B/Report 01-2003.

Maher, ML, Bilda, Gu, N., Gul, F., Huang, Y., Kim, MJ., Marchant, D., Namprempree, K.: 2005, Collaborative Processes: Research Report on Use of Virtual Envionments, CRC Technical Report, Project 2002-024-B/Report 02-2005-02-04.

Maher, ML and Gero, JS: 2002, Agent models of 3D virtual worlds, *ACADIA 2002: Thresholds*. California State Polytechnic University, Pamona, pp. 127-138.

Naja, H. (1999) Multi-view databases for building modelling, *Automation in Construction* 8 (5): 567-579.

Nederveen, SV: 1993, View integration in building design. *in* KS Mathur,, MP Betts and KW Tham, (eds). *Management of Information Technology for Construction*. Singapore:World Scientific, pp. 209-221.

Pierra, G: 1993, A multiple perspective object oriented model for engineering design. *New Advances in Computer Aided Design & Computer Graphics*, Zhang, X. ed. Beijing: International Academic Publishers, pp. 368-373.

Qin, S. F., Harrison, R., West, A. A., Jordanov I. N. and Wright, D. K. (2003) A framework of web-based conceptual design, *Computers in Industry* 50(2) pp. 153-164.

Rosenman, MA and Gero, JS: 1996, Modelling multiple views of design objects in a collaborative CAD environment. *CAD Special Issue on AI in Design*, **28**(3): 207-216.

Rosenman, MA. and Gero, JS: 1998, CAD modelling in multidisciplinary design domains, *in* I. Smith (ed.), *Artificial Intelligence in Structural Engineering*, Springer, Berlin, pp. 335-347.

Rosenman, MA, Smith, G, Ding, L, Marchant, D and Maher, ML: 2005, Multidisciplinary design in virtual worlds, *in* B. Martens and A. Brown (eds), *Computer Aided Architectural Design Futures 2005*, Springer, Dordrecht, Netherlands, pp. 433-442.

Sardet, E, Pierra, G, Poiter, JC, Battier, G, Derouet. JC, Willmann, N and Mahir, A: 1998, Exchange of Component Data: The PLIB (ISO 13584) Model, Standard and Tools, *Proceedings of the CALS EUROPE'98 Conference*, 1998, pp. 160-176.

Savioja, L, Mantere, M, Olli, I, Ayravainen, S, Grohn, M and Iso-Aho, J: 2003, Utilizing virtual environments in construction projects, ITCon, **8**:85-99, http://www.itcon.org/cgi-bin/papers/Show?2003_7

Wang L., Shen W., Xie H., Neelamkavil J. and Pardasani A. (2002) Collaborative conceptual design—state of the art and future trends *Computer-Aided Design* 34(13) pp. 981-996.

Wong, A and Sriram, D: 1993, SHARED An information model for cooperative product development. *Research in Engineering Design*. **5**:21-39.

# Appendix 1: DesignWorld Developers' Guide

## A1.1 Introduction

DesignWorld is a 3D virtual world augmented with web-based communication tools and agents for managing non-graphic properties of the 3D virtual world. There are two client browsers in DesignWorld, the Second Life browser and the Web browser which provide the extended capabilities to the Second Life virtual environment. Second Life provides the environment where the different designers meet as avatars and construct their design models. The Web browser provides access to the relationships browser and the extended communications facilities. This guide covers the design of the non-3D component of DesignWorld, namely the web component.

DesignWorld web component is designed as a J2EE web application. The whole application is hosted on a J2EE container running on some physical machine. The container of our choice was Sun Java Application Server Platform Edition 8. The reason for this choice was that this application server was included with the Sun Java Studio Creator, the IDE that used to develop the web application. However, any other J2EE compliant container can be used to deploy and host the web application, such as Apache Tomcat 5.57 or later.

## A1.2 Application Structure

As mentioned above our web application is J2EE compliant. The deployed application structure is shown in Figure 1.

```
collab2
    |
    |---  WEB-INF
    |         |---  lib              External JAR files
    |         |---  classes          Application beans
    |         |     managed-beans.xml  J2EE required files
    |         |     navigation.xml     for a web application
    |         |     sun-web.xml
    |         |     web.xml
    |     index.jsp                  Web interface pages
    |     ...
```

**Figure 23: The structure of the deployed J2EE web application.**

The application folder contains the `WEB-INF` folder. This folder holds the J2EE specific `XML` files that describe the web application. Sun Java Studio Creator IDE uses a different `DTD` than commonly used by other IDEs. This is why all JSP files are written in Sun Java Studio Creator specific `XML`, which is translated to `HTML` by the application server at execution.

The project source folder structure is shown in Figure 2. The `build` folder holds the distribution copy of the project. Java Sun Studio Creator copies them into this folder once the "Build" command is issued from within the IDE.

Any external libraries that are added to the project from within the IDE interface are copied into the `lib` folder.

The Sun Java Studio Creator saves its book keeping records and information in the `project-data` folder.

The source code of the project resides in the `collab2/src` folder. The source code is split into two parts, the Java source and the Web interface. The java source resides in the `collab2/src/src` folder. All packages and classes belonging to the web application can be found in this folder. The web interface and pages for the web application reside in `collab2/src/web` folder. This folder also contains the `WEB-INF` folder that is required by the J2EE web application specification.

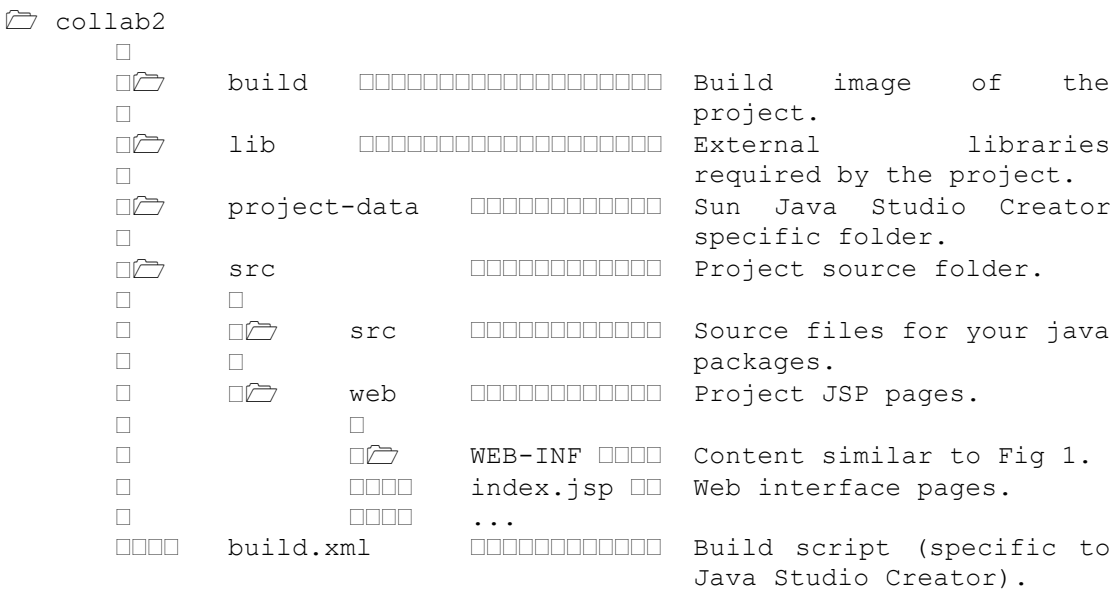


**Figure 24: The collab2 source folder.**

A web application can be viewed as a collection of web pages, a set of page-behind code and some navigation rules. The web application server acts as a glue for the three mentioned component of the web application.

Through out this document sufficient knowledge of the Sun Java Studio Creator IDE is assumed. While aspects of the IDE will not be covered, some IDE representation to various pieces of application code will be referenced.

## A1.3 Application on a Fine-grain Level

The smallest web element of our web application is a `JSP` web page. Every `JSP` web page has a corresponding `JAVA` class attached to it. Checking the code folder you will find that `collab2/src/web/index.jsp` file has its corresponding `JAVA` page bean at `collab2/src/src/collab2/index.java` and so on with the rest of the `JSP` files.

Each `JSP` page is strongly bound to its `JAVA` page-behind bean, that is, every web component in the `JSP` page is bound to a `JAVA` component in the page bean.

Here is an example related to the `login.jsp` page. Figure 3 shows a portion of the login page. This portion contains three `textField` web component and two `linkAction` web component. The `XML` code for the `textField` on the right of the label "First name" looks like

```
<h:inputText              binding="#{login_area$login.textField_userFirstName}"
id="textField_userFirstName" style="height: 24px; width: 200px"/>
```

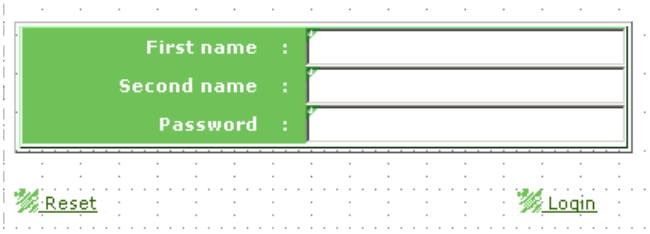


**Figure 25: A portion of the login page.**

This web component is bound to the `textField_userFirstName` property of the `login.java` file

```
private HtmlInputText textField_userFirstName = new HtmlInputText();

public HtmlInputText getTextField_userFirstName()
{
     return textField_userFirstName;
}

public void setTextField_userFirstName(HtmlInputText hit)
{
    this.textField_userFirstName = hit;
}
```

Note that the name of the java bean for the `login.jsp` page is `login_are$login` as registered in the `managed-bean.xml` file

```
<managed-bean>
   <managed-bean-name>login_area$login</managed-bean-name>
   <managed-bean-class>collab2.login_area.login</managed-bean-class>
   <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Similarly, the "Login" `linkAction` web component is described by the following `XML` statement in `login.jsp`

```
<h:commandLink
   action="#{login_area$login.linkAction_login_action}"
   binding="#{login_area$login.linkAction_login}"
   id="linkAction_login">
      <h:outputText
         binding="#{login_area$login.linkActionText_login}"
         id="linkActionText_login"
         value=" Login"
      />
</h:commandLink>
```

Which is bound to the `login_area$login.linkAction_login` property of the `login.java` bean

```
private HtmlCommandLink linkAction_login = new HtmlCommandLink();

public HtmlCommandLink getLinkAction_login()
{
    return linkAction_login;
}

public void setLinkAction_login(HtmlCommandLink hcl)
{
    this.linkAction_login = hcl;
}
```

The action of this web component is bound to an action method in the bean, namely the `login_area$login.linkAction_login` method in the `login.java` bean.

```
public String linkAction_login_action(){...}
```

The navigation for this action is described in the `navigation.xml` file

```
<navigation-rule>
    <from-view-id>/login_area/login.jsp</from-view-id>
    <navigation-case>
        <from-outcome>controlPanel</from-outcome>
        <to-view-id>/user_area/control_panel.jsp</to-view-id>
    </navigation-case>
    ...
    ...
</navigation-rule>
```

All the other JSP files and their respective components follow a similar structure.

## A1.4 Database connection

DesignWorld database is hosted on andy.arch.usyd.edu.au on a mySql DBMS. JDBC is used as the database connector. For our programming convenience a DatabaseProxy.java class was implemented to handle database connections. This class contains two methods one for querying and the other for updating the database.

```
public class DatabaseProxy
{
    public ResultSet query(String query){}
    public void update(String query){}
}
```

## A1.5 Session and Application Beans

The web application maintains several session scope and application scope beans. One example of user session scope bean is

```
public class user_sessionBean extends AbstractSessionBean {}
```

this session scope bean contains the necessary book keeping code for different user sessions that are being served by the application server. Properties like session user name and preferences are kept in this bean.

Application scope beans, as the name implies, works on the application level and is used to save some global states of various aspects of the application.

## A1.6 Special Deployment Requirements

In our implementation we use XML-RPC package to communicate with Second Life server. Since our application server is behind a firewall we need to inform the server JVM about the proxy server name and port number. The following steps describe how this is done.

Step 1: Deploy your application through the Java Sun Studio Creator IDE. This step will overwrite the old application settings.
Step 2: Logon to the server administration page, usually running on port 14848 of the server.
Step 3: Click on the "JVM Settings" tab on the main control panel window
Step 4: Click on the "JVM Options" tab
Step 5: Add the two following options

```
-Dhttp.proxyHost=www-cache.usyd.edu.au
-Dhttp.proxyPort=8080
```

Step 6: Restart the server, now this part is tricky. You may stop the server through this web page by clicking on the "Restart required" link that appears on the top right corner of the page. But in order to start the application server you need to have a physical access to the server to manually start the server. One way around this is to have some sort of VNC server running on the machine and access the server desktop through the service and manually start the server.

The server runs the web application on a default port 18080.  Make sure that this port is accessible and not blocked by your local firewall.

# Appendix 2: Technical Specifications for Communications

DesignWorld includes audio and video communication via a webcam. The webcam component is a Java applet that is based on Sun's Java Media Framework 2.1.1e (JMF). The Java Media Framework API enables audio, video, and other time-based media to be added to applications and applets built on Java technology. This optional package, which can capture, playback, and stream multiple media formats, extends the Java 2 Platform, Standard Edition (J2SE) for multimedia developers by providing a powerful toolkit to develop scalable, cross-platform technology. (http://java.sun.com/products/java-media/jmf/index.jsp)

There are three main components to the webcam applet: a Transmitter component, a Receiver component and a GUI applet component. The Transmitter component captures video and audio frames from the local device, encodes the data into some form of raw RTP format then transmits the encoded data over the network using an RTPManager. The Receiver component on the other end receives the encoded data from the network and renders it on the webcam applet GUI. The Webcam Applet monitors both the transmitter and the receiver and handles user interaction.

## A2.1 Software requirements

The webcam applet is a solution that uses the JMF API and thus the webcam applet requirements are similar to the system requirements of JMF 2.1.1e. The three components of the webcam applet were bound together with Java 1.5 and therefore JVM 1.5.x or higher is required.

## A2.2 Hardware requirements

Since the webcam applet was intended to run along side Second Life in any one session, a reasonably powerful processor with sufficient amount of memory is also required. Testing with Pentium III produced undesirable results, most of the time JMF was rendering an empty frame and was not able to handle video. A minimum of a Pentium IV 2GHz machine running Windows XP Service Pack 2 with 512MB of RAM is recommended.

The webcam code was tested with Logitech Webcam Pro 4000. Although JMF can support different camera hardware we have hard coded "`vfs://`" compatibility into the webcam applet and thus our code only works with devices that identify themselves to Windows as "`vfw:Microsoft WDM Image Capture (Win32):0`" devices. This also reduces the running environment of our applet to Windows platform as "`vfs://`" interface are not available on other known platforms.

A similar coding decision was taken with audio capture devices. We have hard coded "`DirectSoundCapture`" as the audio device that the webcam applet uses. This limits the availability of audio only on those machines that expose their audio devices through DirectX interface.

To make the webcam applet generic and platform independent some extra code can be added that searches and uses the available video and audio devices. A sample can be found at David Fischer's Java Programming Examples (http://www.mutong.com/fischer/java/usbcam/TestQuickCamPro.java).

## A2.3 The Transmitter component

The following pseudo code summarizes the tasks that are undertaken by this component:

- Initialize JMF, which involves;
  - Connecting to appropriate hardware
  - Setting up proper references to those hardware devices
  - Creating appropriate `DataSource` objects for each device

- Setting up a `Processor`, which involves;
  - Connecting the processor with appropriate codec.
  - Setting up proper audio and video formats that are compatible with `RTP`
- Setting up `RTPManagers`, which involves;
  - Creating the managers and hooking them up with our `Processor`
  - Binding the managers to transmission and reception ports on both the local and the remote machines.
  - Starting up the processor and the managers

Once the above is done, we may add or remove destinations to/from the managers accordingly. That is, adding new destination whenever a new participant arrives and removing destination whenever a participant leaves the video conferencing session. Details of each step will be presented next.

### A2.3.1 Hardware and JMF

The first thing needed for the transmitter component is to determine the hardware devices that will be used to capture audio and video and initialize the JMF engine. Our implementation uses the `MS-WDM Image Capture` device for video and `DirectSoundCapture` for audio, but more generic and intelligent code could be written that probes the hardware list and fetches the available devices. A list of available hardware devices can be obtained manually by running the JMF Registry Editor and checking the entries on the capture device tab as shown in **Error! Reference source not found.**1.

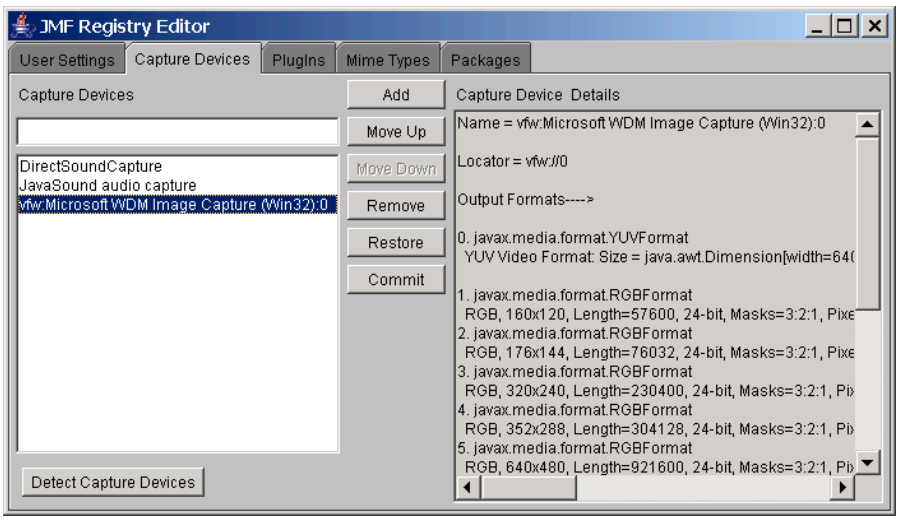


**Figure 26: JMF Registry Editor. The capture device name displayed in the right panel can be used to obtain a soft reference the device.**

In the transmitter component code the following two lines identify the devices that are to be used for audio and video capture.

```
String camDevice = "vfw:Microsoft WDM Image Capture (Win32):0";
String audioDevice = "DirectSoundCapture";
```

### A2.3.2 Initialization of JMF

The initialization of JMF is done in the initJMF() method. The process involves obtaining a CaptureDeviceInfo object from the CaptureDeviceManager. For more information on CatureDeviceManager please refer to JMF documentation and online tutorials provided on Sun Microsystems' website. Then we obtain the locator objects from those CaptureDeviceInfo objects, one locator for each device (audio/video). Once the locators are ready, we need to create DataSource objects for both locators. The last step in the initialization phase is to merge the two DataSources into one. This step is necessary because we need to encode the stream coming out of those DataSources into a suitable format that can be transmitted using RTP over the network.

This concludes the initialization process. This is the only part of the code that deals with hardware devices and sets up proper references to appropriate devices. The next phase is the creation of a Processor. The processor will encode the streams produced by the merged DataSource into a format suitable for transmitting over RTP. This is done in the setupProcessor() method of the transmitter component.

### A2.3.4 Creating a Processor

The processor is used to trans-code the data coming out of the `DataSources` into a format that is appropriate for transmission over the network using `RTP`.

We use the `Manager` class once more to create a `Processor` object. The processor, according to JMF documentation, needs to be configured and then realized. This may take some time, depending on your hardware speed and configuration, and we need our code to wait for these states to be reached. The method `private synchronized boolean waitForState(Processor p, int state)` contains all the necessary code for doing that and we use it for waiting until the processor is configured. Once the processor is configured, we can access the tracks inside the processors. Then we set the `ContentDescriptor` of the processor to *`RAW_RTP`*. Then we need to choose the tracks in the processor that we will be trans-coding for transmission. The `checkForVideoSizes()` method makes sure that for *`JPEG`* the width and height are divisible by 8 and for *`H.263`*, by default JMF support some specific sizes (128×96, 176×144, or 352×288).

Once the tracks are configured we need to wait for the processor to reach the `Realized` state. The last step in setting up the processor is to se the JPEG quality of each video frame. You may experiment with different values. The `setupJPEGQuality()` method loop through the controls to find the Quality control for the JPEG encoder and sets the given value.

### A2.3.5 Creating RTP Managers

Transmission of real-time audio and video is done through the JMF `RTPManagers`. During the execution of the code, setting up the `RTPManagers` will be delayed until at least one destination is known. In JMF, any device that produces a stream of data, like the camera or the transmitter, can be considered as a `PushStream` object while any object that consumes a stream of data, like a player or a receiver, can considered as a `PullStream` object. This concept will be clearer when we examine setting up of the `RTPManagers` next. It is important to note that the appearance of exceptions in the `initJMF()` method will be due to incorrect configuration of JMF on the machine or due to the absence of appropriate hardware. Whereas exceptions in `setupProcessor()` method indicate the absence of *`RTP`* compatible codec.

As mentioned above, we need at least one destination to initialize the `RTPManagers` in our transmitter component. Typically we need one manager per stream and each manager uses two ports (new work connections) to transmit the stream to the destination. Once connection is used to send data and the other is used to send control information.

Several assumptions were made during the design phase; among them were the choice of port numbers that will be used for transmitting and receiving data. With our current implementation we chose port 22220 to be the base port for transmission and port 33330 as the base port for reception of real-time streams. Since we have one audio and one video stream, our `RTPManagers` will use 4 ports, two for each stream. Therefore one requirement for successful running of the applet is the availability of ports 22220, 22221, 22222, 22223 and ports 33330, 33331, 33332, 33333 on the local machine. Firewalls should be configured accordingly.

Setting up the `RTPManagers` involves the following; first we fetch the list of all `PushBufferStreams` from the processor, then for each stream we create an `RTPManager`. In each iteration of the above loop we create an RTPManager, bind the manager to a local transmission address and port, then bind the manager to the remote address and port, and finally, start the manager stream to commence transmission. The same `RTPManagers` can be used to transmit to multiple destinations. All we need is to add the appropriate target using `addTarget()` method of the

RTPManager. This is done in the last part of the startTransmittingTo() method. For book-keeping purses our code maintains a Vector<RTPManager>() of managers.

### A2.3.6 Handling participants

Adding new participants was discussed above and the same code can be used to add as many participants as needed. At the same time the program must devise a way to find out if a participant had left the conference or not and remove the appropriate destination machine form the RTPManagers accordingly. In our implementation we rely on the Webcam applet that polls the user database frequently to find out who is logged into the DesignWorld system and who is not. The applet maintains a participant lists and as soon as a poll indicates the absence of a previously existing participant, the applet calls the public void stopTransmittingTo(InetAddress dest) method of the transmitter.

In this method, a target, specified by the dest argument of the method, is removed from each RTPManager.

### A2.3.7 Cleaning up at the end

It is important that the transmitter releases all the hardware devices upon exiting otherwise the devices may be permanently engaged and in some circumstances, during our testings, we had to reboot the system to release the devices.

The close() method is provided for this purpose. First, all destinations must be removed from each manager and then each manager must be disposed. Up till now, the hardware is still engaged. Recall that the processor object is directly hooked up with the capture devices through some DataSources. So we need to close the processor and de-allocate it to release the hardware. There is no need to close and de-allocate the other DataSources that were created in the initJMF() step. The call to this.myProcessor.deallocate() seems to do that automatically for all the DataSources that are hooked up with the processor.

## A2.4 The Receiver Component

The receiver component does not interact with any special capturing hardware. It only receives RTP streams from the network and renders audio and video on their respective devices. Thus no JMF initialization, like what was done in the transmitter component, is required. However, to enable the component to be aware of network activity it must implement several interfaces and a proper implementation of those interfaces is essential to achieve the required results.

### A2.4.1 Required interfaces

The receiver component in our design implements three interfaces, namely, ReceiveStreamListener, SessionListener, and ControllerListener. The RTPManagers assign a session to each participant. Session information is exchanged by managers on both sides, the transmitter side and the receiver side. Because a session is instantiated and controlled by the transmitter, the receiver is informed with changes in the session by various events.

The SessionListener interface generates the callback for all SessionEvents. These events are LocalCollisionEvent that pertains to the local participant and NewParticipantEvent that informs the listener of every new/unique participant that joins the session. In our implementation we are interested in the NewParticipantEvent and we ignore the other events.

The receiver must also keep track of the state transitions of incoming *RTP* streams. This is done by using the ReceiveStreamListener interface.

The ReceiveStreamListener is an interface that generates the callback for all RTPSessionManager Events. This interface generates callbacks for events that pertain to a

particular RTPRecvStream. This interface also generates callbacks that pertain to state transitions of active/inactive of a passive participant as well i.e. Active, Inactive, Timeout, ByeEvent are also generated for passive. In our implementation we associated different behavior with each or the four events mentioned above.

The last interface we need to implement is the ControllerListener interface. This is required because our receiver component will handle several Player objects that will render audio and video on their respective devices. To control those players our component makes use of the ControllerListener interface and handles ControlEvents generated by those players.

The Controller interface, which extends Clock, provides resource-allocation state information, event generation, and a mechanism for obtaining objects that provide additional control over a Controller.

ControllerListener is an interface for handling asynchronous events generated by Controllers. In our case they are the players.

### A2.4.2 Initializing the receiver component

The receiver component, like the transmitter component, maintains a list of RTPManagers that handle the task of receiving *RTP* streams over the network. These managers provide soft references to various objects required by the players to render the audio and video data on their respective devices.

At least one source is required to create RTPManagers for both audio and video streams. The code for manager initialization is similar to the one used in the transmitter implementation.

In our application we know that we have two streams, audio and video. We create two managers in the initReceiver() method.

In each iteration (there are only two), we create a new RTPManager. One for audio stream and one for video stream. Register our component for listening to session events. Register our component for listening to ReceiveStream events. Create a SessionAddress object with which the manager will be bound. Recall that we use base port 22220 for transmission and base port 33330 for reception of *RTP* streams. In this step we specify that we will be receiving *RTP* streams on port 33330 at the local machine. Bind the manager with the specified local address and port for receiving *RTP* streams . The next step is necessary for fine tweaking the managers. We have used the buffer length 350 which was recommended in the JMF tutorials published by Sun Microsystems. Those tutorials recommend experimenting with different buffer sizes to achieve better performance depending on the underlying network configuration.

Then we bind the manager with the remote machine address and port. This is the step where we specify where the *RTP* stream is coming from. According to our implementation we are interested in streams coming from the base port 22220 of the remote machine. For book keeping purpose, we maintain a Vector<RTPManager>() of managers.

### A2.4.3 Event handling

As stated earlier, our component listens and reacts to different session, stream, and controller events. A detailed discussion on this behavior is presented next.

SessionEvents are handled in the update(SessionEvent) method. We do not perform any significant task other than displaying a debug message which enables us to keep track of streams during our testing phase. We have kept this piece of code in our component for future reference. It may be modified if new requirements, concerning sessions, arise in the future. ReceiveStreamEvents are handled in the update(ReceiveStreamEvent) method. This method contains the behavior of our component depending on the events that arrive through out the lifetime of the component. Before anything else, we obtain the reference to the ReceiveStream that generated the event. According to JMF tutorials and JMF API documentation, this reference could be null. An appropriate

check for a null pointer is recommended. We did not handle the `RemotePayloadChangeEvent` in our implementation. According to our design, we assume that the transmitter will never change its *RTP* format that it initially used at the beginning of transmission. In the event of a new stream we perform several tasks. Most of them are for book keeping purpose.

A discussion of the most significant steps is presented next. As soon as a new stream is detected we first obtain the reference to its `DataSouce` object, then we create a player for this `DataSource`. It should be noted that the Player, like the Processor in the transmitter component, needs to be configured and realized. The player will generate events when different states are reached. We are interested in the `Realized` state. To enable our component to listen to those events we need to register our component as a listener to the `Player` object before calling the `realize()` method on the player.

The events generated by the player will be handled in our component by the `controllerUpdate(ControllerEvent)` method implementation. Detailed discussion of this method will be presented later in this document. The third stream event we are interested in is the `ByeEvent`. This event will be generated when a participant stops transmitting *RTP* streams to the receiver machine.

In our implementation we inform the applet, the GUI component of the webcam, to remove the appropriate panel and release the screen real-estate that was occupied by this player. The receiver component maintains a list of players and their visual panels. In the following code segment `psp` is an instance of the class that contains this information.

The `TimeoutEvent` can be handled. We just print out a debug message for tracking the session during our testing phase. Typically an implementation may perform some kind of polling with the transmitting machine. Testing showed that timeouts are not significant to the execution of our webcam applet.

The `InactiveReceiveStreamEvent` informs the component that a particular stream was inactive for some period of time, and hence the name of the event. In our implementation we locate the player that is associated with the stream and stop it form playing. A player can be started again once it is stopped. Stopping a player reduces the load from the system processor, which is desirable in any application especially with our DesignWorld application due to the existence of many processor hungry components running side by side. The `ActiveReceiveStreamEvent` is generated when a stream becomes active again. Our implementation locates the player that is associated with the stream and starts playing it once again.

### A2.4.4 Handling Video Conferencing Participants

It should be noted that we use the terminology "participants" in two different contexts on two different levels of abstraction. The first is the RTP streams level, and the second is the video conferencing level. JMF `RTPManagers` associate every new stream with a `Participant` object that maintains relevant information about that particular RTP stream. On a higher level of abstraction, the webcam applet keeps a list of people participating in a video conference session. A list of conference participants is maintained and the DesignWorld database is frequently polled to maintain that conference participant list. Reception is started or stopped depending on the joining and leaving of participants respectively.

When the applet discovers the existing of a new conference participant, it calls the `startReceivingFrom(InetAddress)` method of the receiver component. In this method a new target is added to all the `RTPManagers`. Remember that the receiver component maintains a `Vector<RTPManager>()` of mangers. On the other hand, when the applet detects the leaving of a conference participant from the conference it calls the public void `stopReceivingFrom(InetAddress)` method of the receiver component and we remove the target from target from all the `RTPManagers`.

### A2.4.5 Cleaning up at the end

The receiver component was originally designed to implement some dynamic behavior and hence it `extended` the `Thread` object. But later we realized that no such behavior was required for our particular application. The idea was dropped but the `Thread` structure of the class remained. This is why the `run()` and `terminate()` methods exist in the source code.

The `terminate()` method is used to initiate the clean-up code and closing the receiver component. As with the transmitter component we need to release and dispose all the `RTPManagers`. The following code segment from the `close()` method is the more significant part. The rest of the `close()` method code clears up all the book-keeping data.

## A2.5 The GUI Component (Webcam Applet)

The DesignWorld specification dictated that the webcam component should work as an integrated part of the DesignWorld web application. The DesignWorld web application is designed to work inside a browser environment. Thus the natural and logical choice for integration would be by using the Java Applet technology.

The java applet handles three major tasks:

- Create and maintain a transmitter component
- Create and maintain a receiver component
- Poll the DesignWorld database periodically and coordinate with the transmitter and receiver components to maintain the conference session.

The applet implements the `Runnable` interface and executes as a thread. This facilitates the periodical maintenance of the conference information.

### A2.5.1 Initialization

The applet declares and maintains a global reference to a transmitter object and a receiver object. These objects are initialized in the `init()` method of the applet. Recall that we had implemented the receiver component as a thread and hence we need to start the thread. There are some applet security issues that need to be resolved. In our particular application these issues revolve around the fact that applets cannot acquire the local host address nor can they acquire access and control to some local hardware, like capture devices. More discussion on how to overcome these issues will be presented later in this document. For now we will assume that our applet is granted permissions similar to a stand alone java application.

### A2.5.2 Coordination with Transmitter and Receiver Component

The applet maintains a list of conference participants. Initially the applet fetches a list of all participants with the help of the `DatabaseProxy` object, which is declared as a global object reference. In the `run()` method, which is executed periodically through out the lifetime of the applet, the applet finds all the DesignWorld users who are logged into the DesignWorld system and are working on the same project as the current user. Then the applet coordinates with the transmitter and the receiver components to set up the RTP sessions. The SQL query below performs this task. This is how the applet maintains the conferencing session. The `citizen_current_project` value in the DesignWorld database determines whether a user should be considered to be "in" a conference session or not.

```
ResultSet results = this.proxy.query("
    SELECT citizen_webcam_id
    FROM citizen
    WHERE citizen_current_project =
    (
        SELECT citizen_current_project
        FROM citizen
        WHERE citizen_webcam_id LIKE \"" + myIP + "\"
    )
    AND
```

```
    citizen_webcam_id NOT LIKE \"\"
");
```

The list of participants returned by the above query will be added as targets and sources for the transmitter and the receiver components respectively.

The next section of the `run()` method builds a list of new participants who joined conference recently and another list of participants who left the conference. This information is conveyed to the transmitter and the receiver components accordingly. The following code segment starts transmitting to and receiving from newly joined participants. The applet thread sleeps for 30000 milliseconds or 30 seconds. This value can be changed as desired.

The `public synchronized void addWebcamPanel(Panel panel, String name)` method and the `public synchronized void removeWebcamPanel(Panel panel)` method are used by the receiver component to add and remove video panels whenever a conference participant joins or leaves the conference session respectively.

### A2.5.3 The Location of the DesignWorld Database

The assumptions taken by the `DatabaseProxy` object are listed below.

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
String url =
   "jdbc:mysql://andy.arch.usyd.edu.au:3306/crc2?removeAbandonded=true";
String user = "tutor";
String password = "kn0wl3dg3";
String host = java.net.InetAddress.getLocalHost().getHostName();
Connection conn = java.sql.DriverManager.getConnection(url, user, password);
Statement stmnt = conn.createStatement();
return stmnt.executeQuery(query);
```

The `DatabaseProxy` class is implemented to use the mySQL `JDBC` connector and therefore inclusion of the mySQL `JDBC` libraries in the underlying JVM class path is required. We tested our code with `mysql-connector-java-3.2.0-alpha-bin` version of the library that was available at the time of development.

### A2.5.4 Handling Applet Security Limitation

It is well known that applets run in the browser environment and certain resources are not available to them. This includes the capture devices that our applet needs to perform its task successfully. One solution is to use the `SecurityManager` and modify specific properties of the underlying JVM. This requires the involvement of DesignWorld user to change the default setting of the browser and the underlying JVM which in our experience is not desirable by the users themselves due to security reasons. Therefore we opted to use a different solution, namely, Signed JAR files.

We reverted to producing and using a signed JAR file for our webcam applet. When the user first logs into the DesignWorld, the webcam applet is loaded. A security warning is displayed asking the user whether to trust the applet or not along with other relevant information like the publisher of the applet … etc. Once the permission is granted by the user, the applet will execute normally. This way we prevent the user from following some what lengthy procedure to reconfigure the browser setting every time they want to use the DesignWorld application. Two steps are needed to be carried out to produce a signed JAR file:

- Generate a key store.
- Sign the JAR file using the key store

The generation of a key store is done by using the `keytool` program, which is a part of the JDK. The relevant data of the key store is written to a destination, a file in our case. The following command was used to produces a key file for our DesignWorld webcam applet. The `$` sign indicates the shell command prompt.

```
$keytool -genkey -alias webcam3 -keystore webcamstore -keypass CRC
2005 -dname "cn=Adel Ahmed" -storepass CRC2005
```

Once the key store is created, it can be used to sign the JAR file. Don't forget to produce a JAR file for the webcam applet that includes `mysql-connector-java-3.2.0-alpha-bin` library. The reason for this is because our webcam applet uses the `DatabaseProxy` object to poll the DesignWorld database and this object uses `JDBC` connectors in mySQL library to do so. Please refer to the redistribution notice of the mySQL libraries on their website before copying the whole library into your JAR file. Another remedy would be to add the mySQL JAR file into your underlying JVM class path.

Once the JAR file is ready it needs to be signed using the key store that was created earlier. This can be done using the `jarsigner` utility that comes with the standard JDK. The following command line was used for our DesignWorld webcam applet.

```
$jarsigner -keystore webcamstore -storepass CRC2005 -keypass CRC2005
-signedjar signed_webcam3.jar webcam3.jar webcam3
```

This method will produce a signed JAR file that has a 30 day expiry tag on it. To increase the expiry period please refer to SUN Microsystems' website and `jarsigner` documentation for further details.

### A2.5.5 Execution Environment Setup

The webcam applet is integrated with the DesignWorld web application as a separate frame on the application webpage. The applet JAR file and the applet HTML file must be served by some web server. We used an instance of the Apache web server running on a machine that has a public IP, so that the applet can be accessible to virtually everyone on the Internet.

## A2.6 Communication Implementation Issues

This version of the webcam applet cannot function across firewalls and proxy servers.

**Cooperative Research Centre for Construction Innovation**

9th Floor, L Block
QUT Gardens Point
2 George Street
BRISBANE QLD 4001
AUSTRALIA

Tel: +61 7 3138 9291

Fax: +61 7 3138 9151

Email:
enquiries@construction-innovation.info

Web:
www.construction-innovation.info

Established and supported under the Australian Government's Cooperative Research Centres Program