A Web-based Approach to Engineering Adaptive Collaborative Applications

This thesis is submitted in partial fulfillment of
the requirements of the award of
Doctor of Philosophy

by
Musbah Sh. Sagar

Oxford Brookes University
School of Technology
Wheatley Campus
Oxford, OX33 1HX
UK

2009

PAGE NUMBERING AS ORIGINAL

Abstract

Current methods employed to develop collaborative applications have to make decisions and speculate about the environment in which the application will operate within, the network infrastructure that will be used and the device type the application will operate on. These decisions and assumptions about the environment in which collaborative applications were designed to work are not ideal. These methods produce collaborative applications that are characterised as being inflexible, working on homogeneous networks and single platforms, requiring pre-existing knowledge of the data and information types they need to use and having a rigid choice of architecture. On the other hand, future collaborative applications are required to be flexible; to work in highly heterogeneous environments; be adaptable to work on different networks and on a range of device types. This research investigates the role that the Web and its various pervasive technologies along with a component-based Grid middleware can play to address these concerns. The aim is to develop an approach to building adaptive collaborative applications that can operate on heterogeneous and changing environments. This work proposes a four-layer model that developers can use to build adaptive collaborative applications. The four-layer model is populated with Web technologies such as Scalable Vector Graphics (SVG), the Resource Description Framework (RDF), Protocol and RDF Query Language (SPARQL) and Gridkit, a middleware infrastructure, based on the Open Overlays concept. The Middleware layer (the first layer of the four-layer model) addresses network and operating system heterogeneity, the Group Communication layer enables collaboration and data sharing, while the Knowledge Representation layer proposes an interoperable RDF data modelling language and a flexible storage facility with an adaptive architecture for heterogeneous data storage. And finally there is the Presentation and Interaction layer

which proposes a framework (Oea) for scalable and adaptive user interfaces. The four-layer model has been successfully used to build a collaborative application, called Wildfmt that overcomes challenges facing collaborative applications. This research has demonstrated new applications for cutting-edge Web technologies in the area of building collaborative applications. SVG has been used for developing superior adaptive and scalable user interfaces that can operate on different device types. RDF and RDFS, have also been used to design and model collaborative applications providing a mechanism to define classes and properties and the relationships between them. A flexible and adaptable storage facility that is able to change its architecture based on the surrounding environments and requirements has also been achieved by combining the RDF technology with the Open Overlays middleware, Gridkit.

Acknowledgments

This thesis arose in part out of years of research that has been carried out since I joined Oxford Brookes University in 2004. Since that time, I have worked with a number of people in the Open Overlays project whose contribution in assorted ways to the research and the making of the thesis deserves to be mentioned. It is a pleasure to convey my gratitude to them all in my humble acknowledgment. In the first place I would like to record my gratitude to Professor David Duce for his supervision, advice, and guidance from the very early stages of this research as well as giving me an extraordinary experience throughout the work. Above all and most importantly, he provided me with steady encouragement and support in various ways. His involvement and originality has triggered and nourished my intellectual maturity that I will benefit from for a long time to come. I gratefully acknowledge Professor Bob Hopgood for his advice, supervision, and crucial contribution. I am grateful to my friend Stefan Thalman, for his words of encouragement and for setting me a deadline for a first draft of this thesis. He has helped me more than he thinks. My thanks also go in particular to Dr. Mohamed Younas, Peter Oriogun and Colin Rainey for using their precious time to read this thesis and giving their comments. Special thanks to Enzian Baur for believing in me and for her kind words of encouragement. I gratefully thank Professor Chris Cooper for his constructive comments and advice in the early stages of this research. An last but not least, I would like to give thanks to Professor Tom Boyle and Professor Andrew Ravenscroft for the support they have given me in providing time and resources to complete my PhD.

Publications

Musbah Sagar, David Duce and Mohammed Younas (2008) ,"The Oea Framework for Class-Based Object Oriented Style JavaScript for Web Programming" , Computer Standards & Interfaces (2008), doi:10.1016/j.csi.2008.03.014

Musbah Sagar, David Duce and Chris Cooper (2005) ,"Advanced Mouse Event Model for SVG" , 4th Annual Conference on Scalable Vector Graphics, SVG Open 2005, Enschede, the Netherlands, August 2005.

Chris Cooper, *David Duce,* Wei Li, *Musbah Sagar,* Gordon Blair, Geoff Coulson and Paul Grace (2005), "The Open Overlays Collaborative Workspace", 4th Annual Conference on Scalable Vector Graphics, SVG Open 2005, Enschede, the Netherlands, August 2005

Chris Cooper, *David Duce,* Wei Li, *Mohammed Younas, Musbah Sagar,* Gordon Blair, Geoff Coulson and Paul Grace (2005), "The Open Overlays Collaborative Workspace Environment", UK e-Science All Hands Meeting, 2005.

Paul Grace, Coulson Geoff, Gordon Blair, Barry Porter, Wei Cai, David Duce, Chris Copper, Muhammad Younas, Musbah Sagar and Wei Li (2005) ,"Open Overlay Support for the Divergent Grid" , UK E-Science All Hands Meeting 2005, Available at http://csdl.computer.org/comp/proceedings/icdcsw/2003/1921/00/19210382abs.htm

Coulson Geoff, Paul Grace, Gordon Blair, David Duce, Chris Copper and Musbah Sagar (2005), "A Middleware Approach for Pervasive Grid Environments", UK-UbiNet/ UK e-Science Programme Workshop on Ubiquitous Computing and e-Research, 22nd April 2005.

Coulson Geoff, Paul Grace, Gordon Blair, Wei Cai, Chris Copper, David Duce, Laurent Mathy, *Wai-Kit Yeung*, Barry Porter, Musbah Sagar and Wei Li (2005), "A Component-based Middleware Framework for Configurable and Reconfigurable Grid Computing", Concurrency and Computation: Practice and Experience, 2005 (published on-line 17th November 2005, DOI: 10.1002/cpe.981).

Contents

List of Figures

1

Introduction

1.1 Collaboration

Collaborative applications help people to cooperate and work together usually on shared data to better perform a common task. Each participant in a collaborative task contributes by altering the shared data usually indirectly through a common memory. The collaborative application is responsible for updating the views of all participants. For example, whiteboard applications allow people to cooperate collectively to visualise an idea or outline a design.

In the world in which we live today, there are many natural and man-made circumstances that require high levels of cooperation between people in order to complete a common task that overcomes a challenging event such as a natural disaster, a terrorist attack, fighting a fire, a traffic accident or a military operation. It is envisaged that computer-based collaborative applications can play a vital role in reacting to events and enable the personnel involved to carry out the work necessary to overcome the disaster more effectively. For instance, professional people, such as police officers, soldiers, and fire fighters, can use collaborative applications to see the position of an incident and the location of other team members such that they can coordinate their operations appropriately. They can also be used to exchange information received from sensing devices such as weather monitors, bomb detection devices, and seismic

monitoring instruments. In addition, collaborative applications can be used for sharing information in scientific activities such as simulation experiments which involve complex operations and large volumes of data.

Computer Supported Cooperative Work (CSCW) is the area of computer science that specialises in designing and developing collaborative applications to support cooperative group work. Development methods used in the area of CSCW research at present produce collaborative applications [Jacobson and McCanne, 1994] [Summers, 1998] that are exceedingly inflexible, work only on a single platform, are only capable of running on homogeneous networks, operate on a single device type (i.e. desktop computer, laptop, hand-held PDA, etc.) and are incapable of handling data without previous knowledge of its data schema. These methods make many assumptions about the environment in which these collaborative applications are to operate and the requirements they need to meet. For instance, assumptions are made prior to the development of the software about the network infrastructure (reliability, availability, bandwidth, etc.) or the device type. Undoubtedly, these methods of design and development have limitations. This research proposes a Web-based engineering methodology to address such shortfalls.

The remainder of this chapter is structured as follows. The motivation and the Application Scenario of this research are described in Section 1.2. This is followed by Section 1.3 describing the different dimensions of heterogeneity recognised from the Application Scenario. This highlights the challenges that future collaborative applications need to tackle to overcome the limitations mentioned above, and provides an account of an exemplar collaborative application to demonstrate and evaluate the proposed methodology. The aims and objectives of this research will be presented in Section 1.4, followed by a summary of the research's main contributions in Section 1.5. The Open Overlays project which provided a context for this research will be

introduced in Section 1.6. Section 1.7 gives a summary of the publications arising from this work with highlights of the major contributions they make to this research. Finally the chapter concludes in Section 1.8 with an outline of the remainder of this thesis.

1.2 Research Motivation: Application Scenario

In order to establish a rationale for this research, this section describes a scenario based on wildfire management. This scenario was chosen as it involves complex collaborative applications working on heterogeneous devices, platforms, data and network technologies. The scenario is used to set the context for system experiments with the aim to develop a prototype for field development.

This scenario was developed with the help of the department of Geography of Royal Holloway, University of London based on a study of the Brazilian savannah (cerrado) of central Brazil. For further details, see the paper entitled "Assessing Fire Potential in a Brazilian Savannah Nature Reserve" [Mistry and Berardi, 2005] which assesses the potential of fire in an ecological reserve in the Brazilian savannah.

The scenario is set in the Brazilian savannah cerrado where the conditions are harsh and limited resources are available for fighting fire. A fire may start at any time due to the increase of potential causes [Mistry and Berardi, 2005]. Therefore, the fire brigade has to be prepared at all times. Means to fight the fire are restricted to primitive methods such as fire-breaks and hand-beaters. The boundaries of a given fire are hard to determine and the communication between fire fighters is only verbal.

We speculate how technology could aid fire fighters in these conditions. We suggest that communication between teams involved in fighting the fire could be boosted with the use of wirelessly networked Personal Digital Assistant (PDA)-like devices that are capable of presenting graphical information. This enables fire fighters to communicate among themselves and with on-site controllers who coordinate the

operation. Global Positioning System (GPS) devices are given to fire fighters while out fighting the fire to enable location tracking. Sensors to observe environmental parameters such as wind speed, direction and humidity level of the surroundings are also positioned to drive fire spread predictions using a fire simulator.

1.3 Highly Heterogeneous Environments (HHEs)

The Application Scenario above posits a challenging environment for collaborative applications to work in. The environment exhibits many dimensions of heterogeneity that require a high level of adaptability. The descriptions of four dimensions of heterogeneity recognised from the Application Scenario above are presented here:

(1) *Network Heterogeneity*: The networking infrastructure requirements vary in the Applications Scenario. Wireless ad hoc networks can be used to connect the sensors (wind, humidity, etc.) that are placed around the fire site. Fixed networks can be used to connect the fire simulation (running on a high-end computer) and other computers used by the controllers. Wireless networks can be used to connect the PDA-like devices used by the fire fighters. The connection between the fixed network used by the controllers and the wireless network used by the PDA-like devices of the fire fighters is maintained through a bridge so that the controllers who run the fire simulation also retain communication with the fire fighters. In practice, a variety of wireless technologies (including satellite communication) might be deployed, as per the IEEE 802.1 family, with perhaps technologies such as Bluetooth in low power devices.

(2) *Device Type Heterogeneity*: The Application Scenario implies that a collaborative application is used to facilitate communication among fire fighters and controllers. It must operate on different devices (desktop computers used by the controllers and PDA-like devices used by fire fighters) and therefore it should be able to adapt to the differences between these various device types.

(3) *Data Heterogeneity*: The Application Scenario is rich with data and information collected from different sources such as sensors, GPS devices, fire simulation and the graphical annotations and data created by the fire fighters and the controllers as they communicate. Also, data needs and availability might change as the scenario evolves.

(4) *Architectural Heterogeneity*: The data used by the collaborative application is to be stored and shared among the participants in a collaborative task. The harsh conditions presented by the Application Scenario are not predictable and can change at any moment. These changes, in addition to the network heterogeneity, call for a flexible approach to setting and adjusting the architecture. Reliability (ability to work in normal and unexpected circumstances) might be favourable at one point whereas scalability could be preferable in another circumstance. These changing requirements require a flexible approach to the data storage architecture.

1.3.1 Collaborative Workspace Environment (CWE)

In order to meet the above requirements of the Application Scenario this research develops, implements and validates a tool called CWE. CWE is a graphical tool that supports effective communication using graphical annotations on a shared work surface. It serves as an exemplar of Adaptive Collaborative applicaTions (ACTs), a class of computer collaborative applications built to overcome the challenges posed by HHEs. CWE provides various facilities including:

1. It supports collaboration through graphical communication between users (i.e. fire fighters and controllers).
2. Users of the CWE graphically communicate among themselves by making sketches on a shared workspace in order to share ideas, issue commands and help in decision making and post event analysis.

3. Information from different sources including sensory data, fire simulation predictions and location information are stored and presented visually to aid users.

4. Controllers run a fire simulation in order to predict the spreading of the fire and the information collected from the sensors is used to drive the fire simulation engine and to seed 'what if' scenarios.

1.4 Aim and Objectives

The general aim of this research is to discover an engineering methodology for designing and constructing ACTs that can work in HHEs. In order to achieve this aim, the following objectives are defined:

1. To investigate and develop a method for constructing adaptable user interfaces that work on different device types (i.e. desktop computers, laptop with different sizes, hand-held devices, etc.).

2. To develop a method that can be used to mix data, information and knowledge from various sources into a unified knowledge repository without prior knowledge or awareness of the structure or the model of the data.

3. To establish a suitable model for a flexible and adaptable architecture for a knowledge storage system (knowledge repository) that can operate in heterogeneous and changing environments.

4. To implement a system in order to test and demonstrate the approach.

A key pre-condition of this research is the use of Web technologies. We impose some constraints on the solution by proposing to use a Web-based approach that combines Web technology standards such as Scalable Vector Graphics (SVG) [Ferraiolo, Duce et al.], Resource Description Framework (RDF) [Manola and Miller] [Prud'hommeaux and

Seaborne], Resource Description Framework Schema (RDFS) [Brickley and Epinions] and SPARQL Query Language for RDF (SPARQL) [Prud'hommeaux and Seaborne, 2006] with component-based Grid middleware using the Open Overlays approach [Grace, Hughes et al., 2008].

1.5 Research Contribution

Web technologies have been used in this research to address the various issues presented in the previous section (Aim and Objectives) because of their pervasive nature. These technologies (i.e. SVG, RDF, RDFS, SPARQL, etc.) were not originally designed to tackle the broad challenges faced by this research; however, we have addressed and resolved many of their problems and limitations in this respect. This thesis proposes a new generic architectural framework, a novel methodology and a set of engineered technologies for building ACTs as described below:

- The first contribution of this research is Collaborative RDF (CoRDF), an engineering approach addressing the data and knowledge heterogeneity aspect of building ACTs. This contribution has two parts:
    - A novel way of data modelling for building ACTs that uses RDF technologies to express data types and the relationships between data for any kind of information as opposed to using the type system of the programming language chosen for the development. This approach allows various kinds of information from different sources such as simulations, sensors, annotations, etc. to be mixed in a data repository without previous knowledge (from the application point of view) of its structure, model or the relationships that govern it, and then be able to begin querying this data.
    - A novel solution to building a data and knowledge repository for collaborative applications called Knowledge Base (KB). The KB is a

flexible data storage architecture that can run on devices with different capabilities to accommodate the storage requirements of ACTs. The characteristics of the KB are that it is adaptable and has a flexible architecture such that it can work in different modes including: centralised, distributed and replicated to cope with requirements changes (see Section 6.4).

- The second contribution of this work is a novel generic approach to building adaptable user interfaces using SVG technology. This method allows user interfaces to be built that can adapt to various device types. The contributions that have emerged from the development of this method are:

  1. An enhanced set of APIs for manipulating graphics in SVG.
  2. A user-friendly new mouse event model that simplifies handling mouse events in SVG and solves the notorious out-of-sync problem that affects many SVG applications.
  3. A wide set of Graphical User Interface (GUI) components that are reliable, flexible, extensible and easy to use.
  4. An approach that enables programmers to import applications from other Object Oriented Programming (OOP) languages (e.g. from Java, C++, etc.) by using a Class-based Object Oriented model called ClassBJS in JavaScript.

1.6 Open Overlays Project

The research reported in this thesis was carried out within the Open Overlays project [Blair, Coulson et al.], a joint project between Oxford Brookes University and Lancaster University funded by the EPSRC (Engineering and Physical Sciences Research Council) fundamental computer science for e-Science Programme. The Open Overlays

project addressed the issue of network and platform heterogeneity in the Grid middleware architecture. The aim is to support building applications that can run over an increasing number of network technologies and operating systems. The idea is that applications are built from components that can be configured through a third party. Open Overlays offer a configurable and reconfigurable framework that supports the layering of multiple overlay networks to allow the creation of composite protocols and network services [Grace, Hughes et al., 2008].

Open Overlays builds on the concept of configurable and reconfigurable lightweight components, OpenCOM [Coulson, Blair et al., 2004]. This concept of Open Overlays is realised as a Grid middleware platform called Gridkit [Grace, Coulson et al., 2004].

This research was not concerned with the development of the Gridkit middleware itself but rather with addressing the concerns of building ACTs using Gridkit as the middleware and layering other elements of the solution on top of this. Nevertheless, experiences in using Gridkit in this way provided feedback on the Gridkit architecture and had an influence on the design.

1.7 Overview of Publications

There have been a number of papers (seven in total) which were published as a part of this research work. The author of this thesis has been the major contributor to three of them. Some of the author's work reported in these papers is fundamental to this thesis and is described herein. Other work was peripheral to the thesis and is not described here. These publications are (in reverse chronological order): Paper A: The Oea Framework for Class-based Object Oriented style JavaScript for Web Programming which was published in the Journal of Computer Standards and Interfaces in 2008, Paper B: Advanced Mouse Event Model for SVG which is available online and was

presented at the 4th Annual Conference on Scalable Vector Graphics, SVG Open 2005 and Paper C: The Open Overlays Collaborative Workspace, which was also presented at SVG Open 2005 and available online. The three papers are described here:

1. Paper A: The Oea Framework for Class-based Object Oriented style JavaScript for Web Programming describes the differences between Prototype-based and Class-based OOP languages and introduces an approach to writing JavaScript code following a Class-based style. The new approach is called ClassBJS and it allows JavaScript developers to write programs using a Class-based style in a way that is easy to use, has a syntax style that resembles that of the Java Class-based approach and has high performance and reliability. The paper also presents a survey of the most widely used Class-based techniques for JavaScript - at the time of writing - describing their methods and highlighting their shortcomings. It also compares my new approach with this earlier work where the ClassBJS enables better performance than the rest. The author was responsible for work behind the concept of the ClassBJS model and the actual design, implementation, testing and the comparison with other similar techniques. The contributions of the Co-authors were editorial. This paper makes a significant contribution to the Oea framework described in Chapter 7. The Oea framework was developed to build adaptable and scalable Web-based interfaces that could be used in building Web applications and ACTs.

2. Paper B: Advanced Mouse Event Model for SVG describes the work on a new mouse event model built on top of Document Object Model (DOM) Level 3 Event Model [Le-Hors, Wood et al., 2004]. The new mouse event model (called domMouse) which was written in JavaScript is concerned with defining a higher level of abstraction particularly for handling mouse events to ease the process of developing SVG applications and to inspire changes to the current SVG mouse event model in order to make it easier to develop interactive applications for SVG.

The paper also explains elements of the Oea framework such as svgDraw2D and svgSwing. The key ideas, designs and the development work in this paper is my work. The contributions of the Co-authors were editorial. This paper makes an important contribution to the Oea framework described in Chapter 7.

3. Paper C: The Open Overlays Collaborative Workspace introduces CWE described in Section 1.3.1 in its early stages of development being built using SVG and RDF. The paper describes the approach to develop CWE with regard to all the information in a collaborative workspace as an annotation of the workspace that can be represented as RDF triples. Information for display can then be selected by querying these triples. Audit trails of events in the workspace can be replayed by querying the triples. I was the major contributor to the key ideas and design and implementation of CWE.

1.8 Outline of the Thesis

Figure 1-1 illustrates the structure of this thesis. Paper C is generic and does not relate to a specific chapter.

Chapter 2 provides an introduction to Computer Supported Cooperative Work (CSCW) and a critique of the current collaborative systems in regards to addressing the requirements presented in Section 1.3.

Chapter 3 presents our proposed model to develop ACTs for HHEs as described in Section 1.3. This model has four layers, and these are from the bottom up: Middleware, Group Communication, Knowledge Representation and Presentation and Interaction.

Chapter 4 describes the Middleware layer of the model presented in Chapter 3, providing the necessary background knowledge about middleware infrastructures and other related technologies.

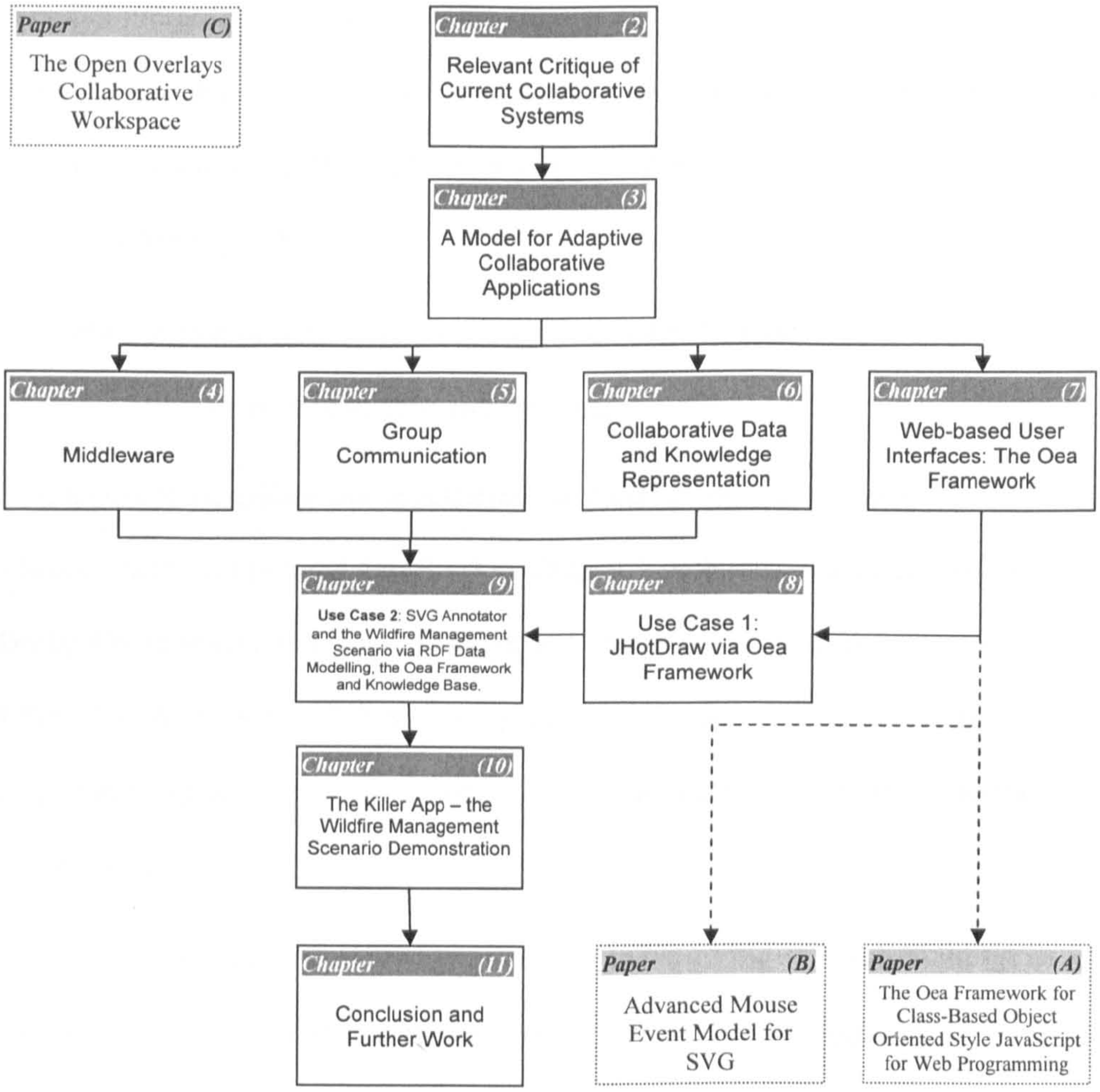


Figure 1-1: Thesis Structure

Chapter 5 describes the Group Communication layer of the model presented in Chapter 3. Group communication facilitates collaboration and data transfer for ACTs.

Chapter 6 describes the Knowledge Representation layer of the model presented in Chapter 3. The work done on this layer is called CoRDF and it includes two parts: using RDF and RDFS for data modelling and the KB for flexible knowledge storage facility.

Chapter 7 introduces a new approach to developing flexible and adaptable user interfaces using SVG. This work corresponds to the Presentation and Interaction layer

of the model presented in Chapter 3. This chapter also includes brief details of some of the work that was done to realise the new approach to developing adaptable Web-based user interfaces such as svgDraw2D, svgSwing and domMouse. Further details of this work are included in paper A and B.

Chapter 8 demonstrates the use of the Oea framework described in Chapter 7 as a valid method to construct adaptable user interfaces.

Chapter 9 describes the application of CoRDF devised from Chapter 6, the flexible user interfaces method described in Chapter 7 with the work on the Middleware and Group Communication layers described in Chapter 4 and Chapter 5 respectively, to build the SVG Annotator, CWE and its specific application, Wildfmt. The purpose of building these applications was to demonstrate and evaluate our new approach to developing ACTs.

Chapter 10 is dedicated to demonstrating the working of CWE/Wildfmt in an emulated environment in order to further evaluate and validate our approach.

Chapter 11 gives the conclusions and outlines possible future work based on this research.

2

Critique of Current Collaborative Systems

The aim of this chapter is to review and critically analyse current collaborative systems within the context of the requirements presented in Chapter 1 (see Sections 1.3). These requirements address the multi-dimensional heterogeneity of the environment described in the Application Scenario (see Section 1.2) such as Network Heterogeneity, Device Type Heterogeneity, Data Heterogeneity and Architectural Heterogeneity (see Section 1.3). For example, it analyses Whiteboard applications which are supported by two widely used tools such as Microsoft NetMeeting Whiteboard (NetMWB) and LBL Whiteboard (WB). The reason these tools were chosen is because they represent extreme styles of architecture. The prime focus of this chapter is how these systems are built but some secondary observation about their core functionality has also been included. The overlapping functionalities of these two systems are common in most similar Whiteboard applications and we are going to make CWE support these functionalities.

2.1 Computer Supported Cooperative Work (CSCW)

Collaborative Systems, Workgroup Computing, Groupware, and Cooperative Work

Support are all terms that fall into the well-known research area of Computer Supported Cooperative Work (CSCW). CSCW has attracted considerable attention from people all over the world since it was first introduced by Irene Greif and Paul Cashman in 1994 [Grudin, 1994]. The main concern of this research area is the use of computer systems to support people in their work activities. However, a commonly accepted definition of CSCW remains elusive. Irene Greif, one of the founders of CSCW, has provided the following definition for the term [Greif, 1988]:

> *"An identifiable research field focused on the role of the computer in group work"*

A more suitable definition of the field appropriate to this thesis was provided by Bannon, & Schmidt [Bannon and Schmidt, 1989]:

> *"An endeavour to understand the nature and characteristics of cooperative work with the objective of designing adequate computer-based technologies".*

The term Groupware was first introduced in the 1988 ACM conference during a panel discussion [Bannon, Ehn et al., 1988] where a need for the development of computer systems to support group activities was recognised. Many recognise the difference between Groupware and CSCW [Wilson, 1991] in that CSCW prefers to assess how humans function prior to designing the computer support for the group working while Groupware tends to be more technology oriented. However, some researchers regard them as synonyms.

As shown in Figure 2-1, there are many classifications that one can apply to collaborative applications; Time and Location organisation is the most common one (the horizontal line for time and vertical line for location as shown in Figure 2-1). This classification was presented for the first time by Robert Johansen [Johansen, 1988] in

1988 and has been used by many others ever since. Following this categorisation, users of a collaborative application can work in the same place and at the same time (face to face), or in the same place but at a different time, at the same time in different places or at different times and different places.

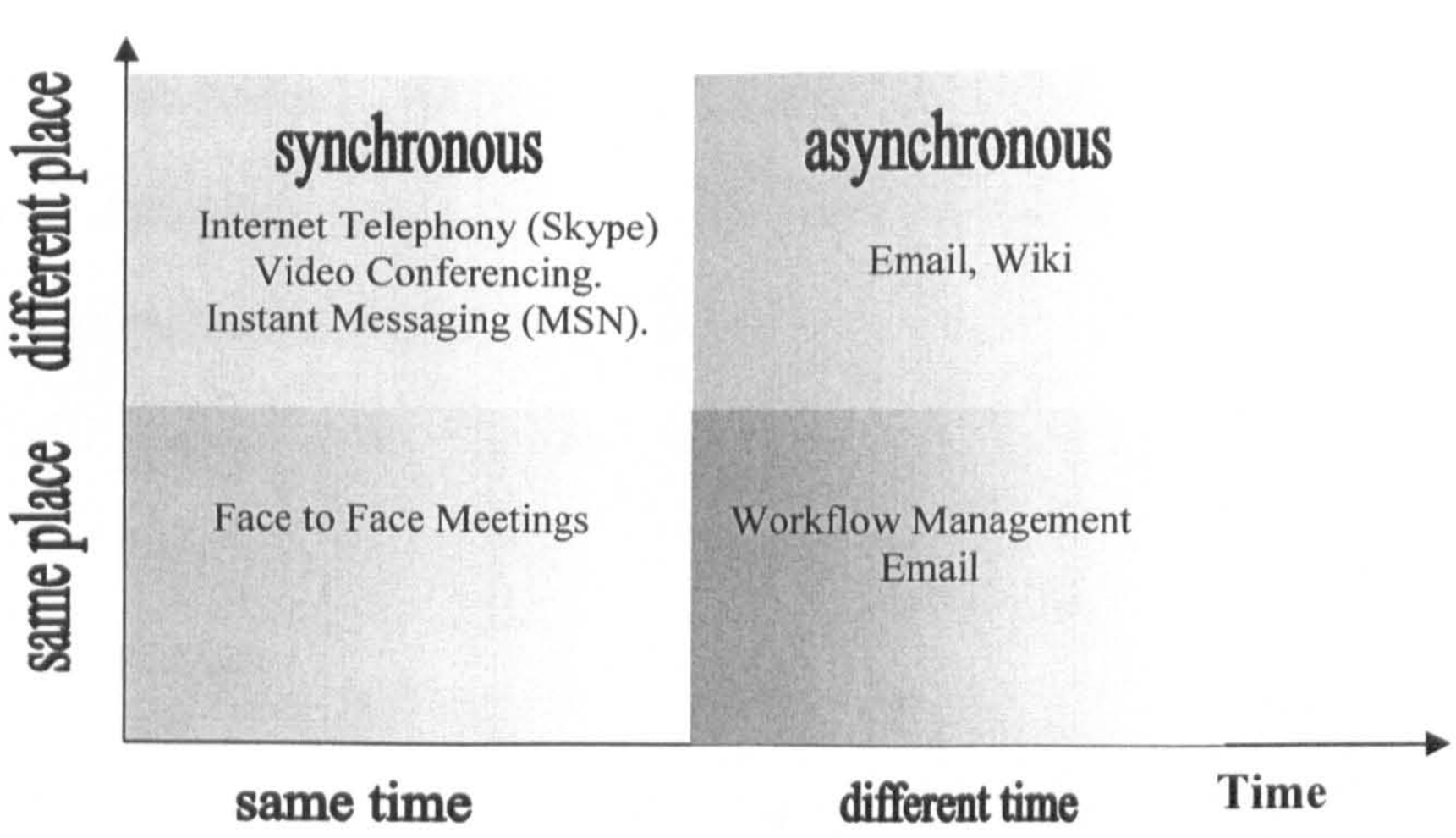


Figure 2-1: Classification of collaborative applications.

Different collaborative applications support one or more of these categories. As shown in Figure 2-1, examples are given for well-known collaborative applications that are commonly used at present for each category. When a collaborative application is used by different users at the same time this indicates that the collaboration is synchronous but if it is used at different times, the collaboration is then asynchronous.

2.2 Whiteboard Applications

The Whiteboard (or Blackboard) is the name used for any surface that can be written or drawn on using markers and can be easily erased. It is often used in school classes or workplaces. An Interactive Whiteboard [Glover, Miller et al., 2005] is an electronic board that can capture any drawings on its surface. It can be connected to a computer and used as its display screen whilst acting as a mouse-like input device. An interactive

Whiteboard can be used to annotate documents, images and other media formats.

Computer applications have been developed to emulate the function of a real-life Whiteboard. A typical graphics painting program such as Microsoft Paint [MSPaint] can be considered as a Whiteboard. A mouse can be used to sketch on the white background to express ideas and explain concepts, present data, etc. Users of a Whiteboard can create simple drawings in any chosen colour such as circles, rectangles, lines, freehand drawings and text, all of which can be easily erased. They can also fill an empty shape with paint of any colour, cut and copy part of the graphics to other places on the Whiteboard, and store and retrieve the content of the Whiteboard at any time for later reference. This simple type of Whiteboard has many uses in education, research and the workplace.

The Whiteboard can be used by one person as a tool to communicate with others similar to the way real-life Whiteboards are used. It can also be used by many users – each taking a turn – to collectively visualise an idea or a design. This approach is limited because of the need for those who want to use the Whiteboard to be in the same location. This has led to the introduction of the Shared Whiteboard, a shared surface among many users working in distributed locations. Each user has a view of the surface. When the shared surface is modified by one user creating or erasing a drawing, the view of the shared surface by the other users is updated to reflect the change. The challenge of this approach is to establish an efficient and reliable communication medium between the users of the Shared Whiteboard to maintain the consistency of the shared surface.

The two common architecture types used in building Whiteboard applications are: (1) centralised, and (2) distributed. The two Whiteboard applications this research chose as an example of each type are: WB (distributed) and NetMWB (centralised). Both applications belong to the 'same time' and 'different place' category of CSCW

applications as illustrated above. These two applications were chosen because they are the most challenging in that they represent extremes in the spectrum of architecture.

2.2.1 LBL Whiteboard (WB)

WB [Jacobson and McCanne, 1994] is one of the earliest Whiteboard applications. It is a distributed whiteboard tool based on the Multicast Backbone (MBone) [Eriksson, 1994]. MBone was introduced as an efficient approach to deliver data to multiple destinations simultaneously across the Internet (see Section 4.1.1 for further details).

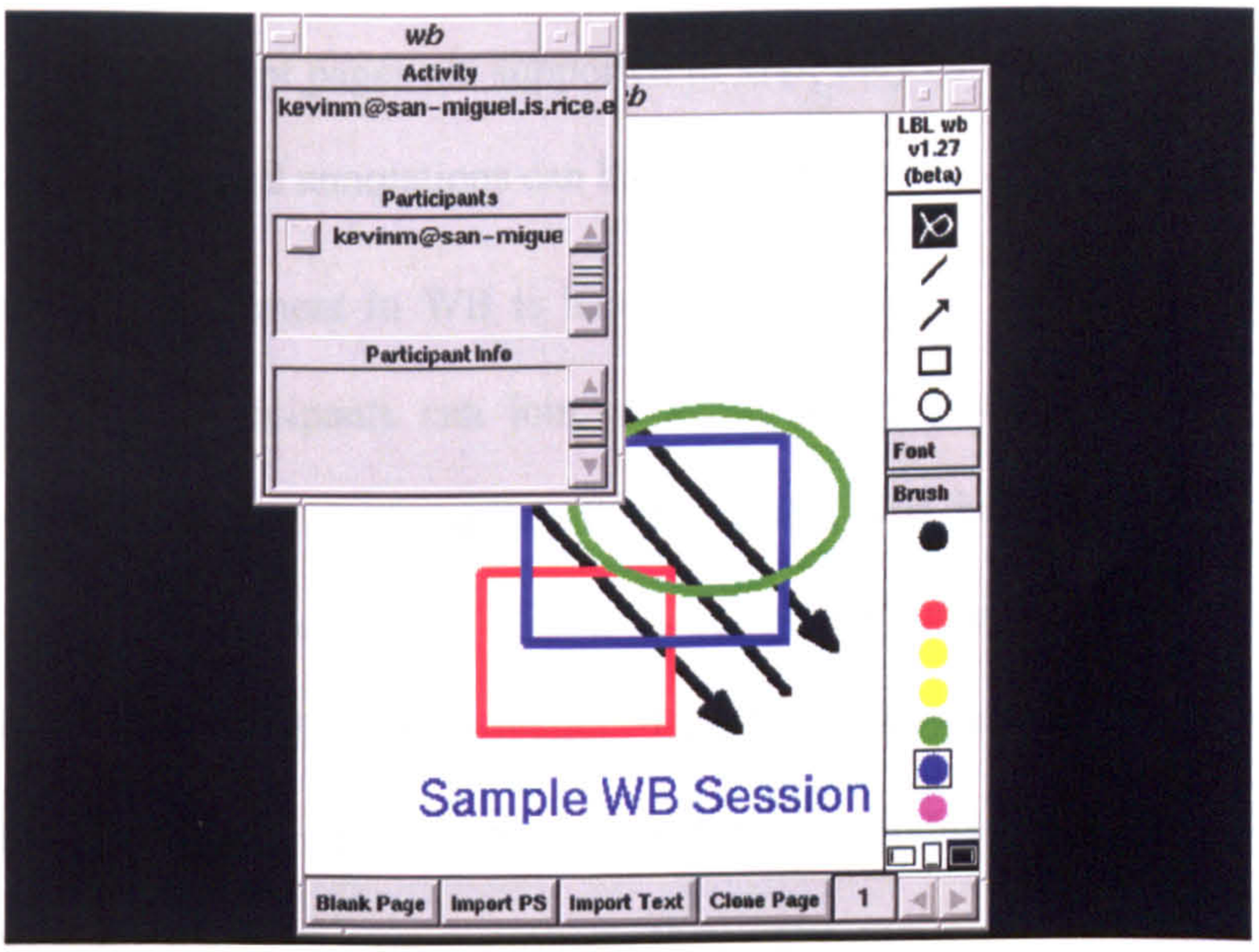


Figure 2-2: Screen shot of WB

WB supports multiple pages; those pages can be created and used by any member of the collaborative session. The mechanism for deciding who has control over creating or sketching on pages is handled by a session management tool (described below). Members and pages are identified by a unique id, Source-ID and Page-ID respectively. Members can draw on the surface of any arbitrary page created in the current session using different shapes and text. Many graphical objects are supported in WB such as freestyle lines, straight lines, arrows, rectangles, and ellipses. A palette of five colours is available to be used with the graphical objects. There is also an eraser tool and tools for

moving and copying objects on the whiteboard. Each action to draw, delete or move on the surface is called a drawing operation, drawop. Each drawop is assigned a sequence number relative to its type (e.g. circle, rectangle, etc.) and creator and then tagged with a timestamp. The drawing operations are sent over the network using a reliable multicast protocol over MBone. Members receive drawops in a queue ordered by their timestamps. However, in-time drawops are rendered instantly upon arrival. The tool does not support tele-pointers or arrows on the remote drawing surfaces so it is only possible to point at something on the Whiteboard by making a sketch with another drawing object. Postscript pages are supported in WB; they can be loaded and displayed on the surface of WB and annotations can be added on it easily.

Session management in WB is handled externally to WB through the session management tool. Participants can join or leave an on going collaborative session freely. Information about participants currently using WB is displayed in a separate window. Information about the most recently active participant as well as a list of all participants of the session is provided. Detailed information about a particular participant can be retrieved such as their IP address, the status of their display (updated or not), the number of drawing operations they have participated with, and the time spent in the idle state. It is also possible to selectively hide all the sketches that members have contributed to the collaborative session.

2.2.2 *Microsoft NetMeeting Whiteboard (NetMWB)*

NetMeeting [Summers, 1998] is a collaborative tool for video and audio conferencing. It provides electronic chat, shared application facilities and the Whiteboard tool, NetMWB. NetMeeting uses the H.320 [H.320] standard for communication. This protocol is used with a central server approach called a Multipoint Control Unit (MCU). H.320 consists of several components including the T.120 data protocol responsible for multimedia conferencing which is used by NetMWB. The H.320 standard defines an

MCU to enable group communication. Each member of the group connects directly to the MCU which controls the conference or communication using point-to-point connections. Figure 2-3 is a screenshot of NetMWB.

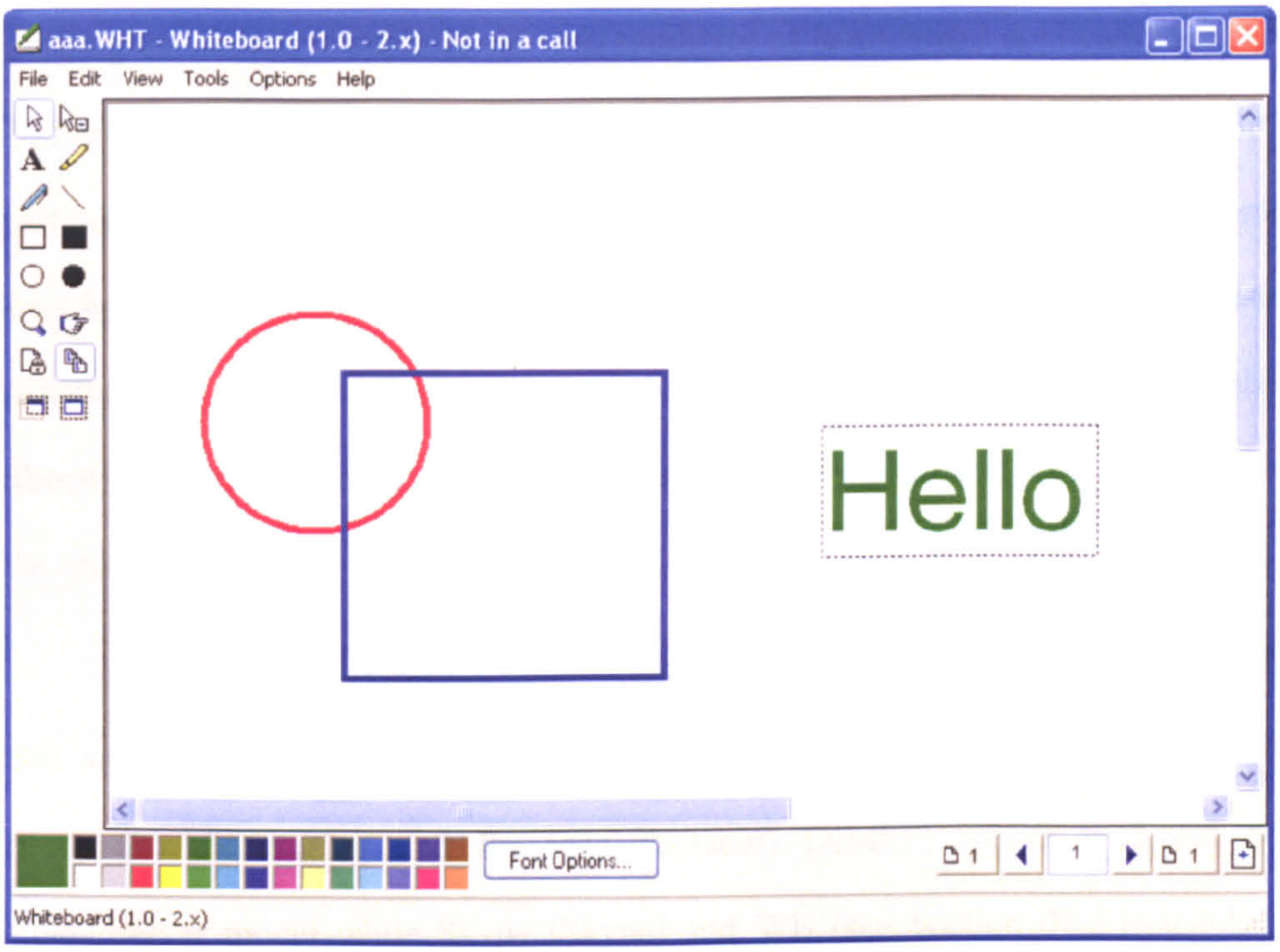


Figure 2-3: Screen shot of NetMWB

NetMWB provides multiple pages that can be used simultaneously. It is possible to invite other users to an online conference if their addresses are known. Also a user can login to an Internet Location Server (ILS) to meet with other people. NetMWB allows graphical objects to be drawn using several tools such as freehand pen, straight line, rectangle, filled rectangle, circle, filled circle and text objects. Graphical objects can be moved around and their colours can be changed. I believe the tool uses a description (i.e. graphical primitives and attributes rather than a bitmap) of the graphical surface and shares it among users in the same session. Images and screen capture are represented as uncompressed bitmaps. Changes to the graphics surface are duplicated across all other users of the tool to retain the consistency. Additional features such as

highlighting text and images, capturing the screen, importing and exporting graphics and the remote pointer, are also supported.

2.3 Limitations of Current Collaborative Systems

Existing collaborative applications (see examples of Whiteboard applications in Section 2.2) are developed to run on a specific platform and on a single device type. They are partitioned to run across multiple computers connected to a network in order to establish communication, exchange messages and data and ultimately facilitate collaboration and cooperation. Communication between these entities is established directly (i.e. distributed) or through a centralised server (Client/Server) depending on the architecture.

Chat programs, classical collaborative applications, use one or the other of these two architecture types. Google Talk [Talk] for example uses the XMPP protocol (Extensible Messaging and Presence Protocol) [XMPP] which is based on the Client/Server model while Skype [Skype] and WB (see Section 02.2.1) use the P2P model. This rigid choice of architecture can be a major limitation if the requirements change. For example, the requirements of an application could be scalability at one point but change to reliability at another. Furthermore, shortcomings such as single point of failure or communication bottleneck associated with central servers can be avoided by allowing a more flexible approach to architecture. Current applications built around one type of architecture can never change to the other once the decision has been made at the design phase. Collaborative applications that can change the underlying architecture when required are vital when the environment changes (i.e. change of the network infrastructure, from fixed to wireless network for example).

Various programming languages (C++, Pascal, C#, Java and many others) are being used to develop collaborative applications. These applications encompass many

distinctive qualities such as: they are interactive, are rich in graphics and support sophisticated graphical user interfaces. By their nature, these applications are closed systems because of the way they are built. Using a compiled programming language such as C++, Pascal or even Java to develop desktop applications will produce an executable code that is only machine-readable. Because of the closed nature of such applications the technologies used to build them are not open due to the following factors:

1. The technology used to display the graphical content is usually unknown or cannot be accessed (i.e. OpenGL, AWT, etc.).
2. The data structure and the design are unknown unless access to the design documents or source code is granted.
3. Access to the data of the application during or after run-time is not possible.

There is also the issue of sharing data from heterogeneous sources, where the application can only interoperate with applications developed with the same programming language and use the same type system. An alternative to this approach is the use of open standards and technologies such as Web technologies (see Sections 3.1).

Moreover, data used in computer applications is either stored locally or on a central shared server in the case of collaborative applications. This can pose limitations when the environment in which the collaborative application operates in is heterogeneous. For example, centralised data storage is often used for small scale collaborative applications. If more participants use the application simultaneously, the centralised server becomes a bottle neck for communication and a more scalable approach to storing the data is required; this is not possible with current systems.

2.4 Summary

This chapter introduced the area of CSCW and provided a description of Whiteboard

applications supported by two wide-spread example applications: NetMWB and WB. The following table describes the major differences between NetMWB and WB including some of their functionalities.

	WB	**NetMWB**
Architecture	Distributed	Centralised
Tools	Drawing, Erasing, Copying and Moving.	Drawing, Removing, Copying and Moving.
Graphical Objects	Freehand lines, Straight lines, Arrows, Rectangles, Ellipses, Text	Freehand pen, Straight lines, Rectangles, Filled rectangles, Circles, Filled circles, Text
Colour	Palette of 5 colours	Palette of 28 colours
Platform Heterogeneity	Microsoft Window, Linux, Silicon Graphics, Sun OS, DEC Alpha	Microsoft Windows
Network Heterogeneity	MBone	H.320, Packet Switching
Multicast Support	Yes	No
Other	Support multiple-pages, able to display postscript pages	Support multiple-pages, centralised server to meet people, highlighting, capture screen, import export graphics, remote pointers

As shown in the table above, NetMWB works on Microsoft Windows only, is built following a centralised architecture and works on networks that support H.320 protocol, originally designed for packet switching networks. On the other hand, WB has been implemented many times to run on different platforms, supports multicast and is built following a distributed architecture. Other minor differences have also been described in Section 2.3. One can observe how these applications have been developed with their features and facilities being an integral part of the software which makes it difficult to change later.

This chapter has highlighted the limitations of these systems in order to address

them in the next chapter. The major limitations of these systems are that they:

- Only run on a single platform.
- Are designed to suit just one type of device.
- Are extremely rigid in their choices of architecture (centralised or distributed).
- Cannot adapt to changes in the surrounding environment.
- Cannot handle new requirements.

In addition, WB uses a proprietary experimental protocol for multicast, MBone. This protocol is becoming increasingly redundant with the increase of commercial routers that support multicast.

The following chapter introduces a more flexible method to build collaborative applications following a more modular and flexible approach.

3

A Model for Adaptive Collaborative Applications

Based on the shortcomings of the current approaches to building collaborative applications discussed in the previous chapter, a four layer model for ACTs has been devised. This model provides the foundation for the engineering approach discussed in later chapters for overcoming these shortcomings.

The chapter starts by introducing the proposed four-layer model. The following sections give an overview of each of the layers of the model. Details on the development of these layers are provided in the subsequent chapters.

3.1 The Proposed Four-layer Model

This section proposes a generic model to develop collaborative applications that can adapt to changing environments, requirements and settings. The benefit of this model is that it separates out a range of concerns so that they can be addressed individually. The model shown in Figure 3-1 has four layers. Each concerned with a different aspect of building ACTs with the goal of meeting the Aim and Objectives presented in Chapter 1 (see Section 1.4).

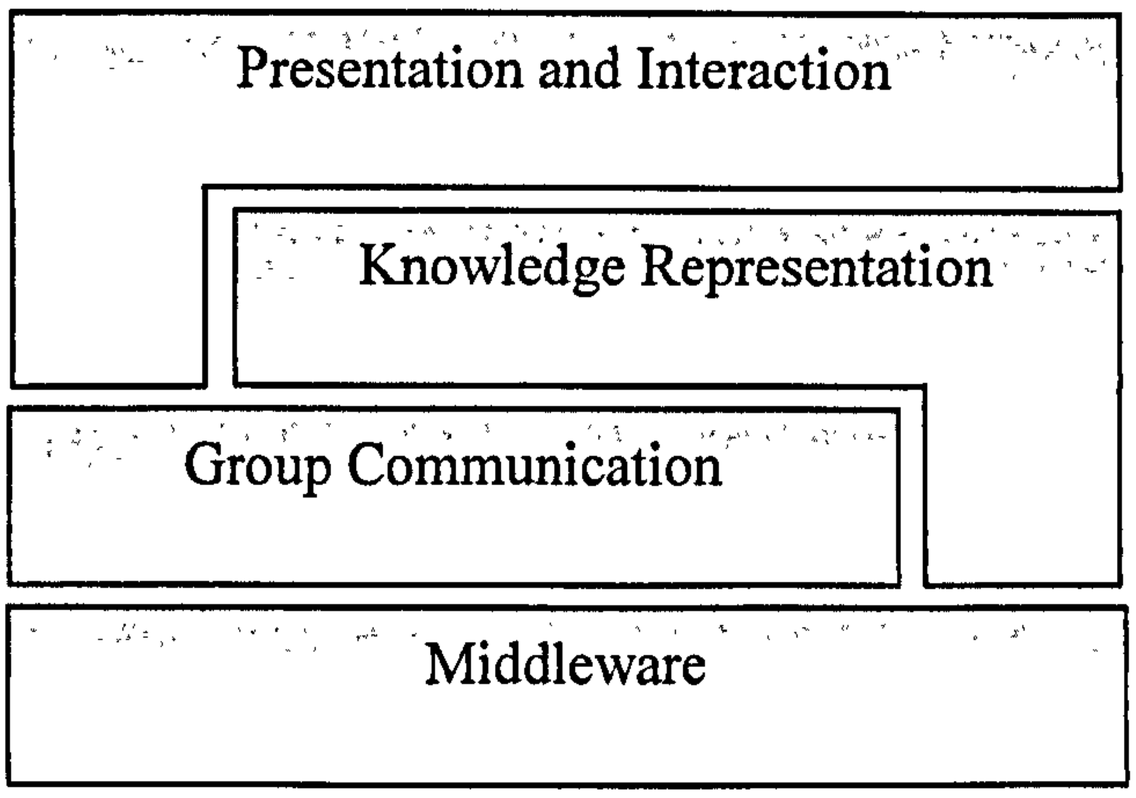


Figure 3-1: The four layers of the generic model for building ACTs.

The Middleware layer addresses network and operating system heterogeneity and provides low-level facilities, for example enabling communication in the layers above it.

The Group Communication layer is concerned with providing facilities to transfer data and to enable communication and collaboration for collaborative applications. This layer is also designed to hide much of the complexity of the middleware layer.

The Knowledge Representation layer addresses two concerns:

1. Developing a method that can be used to mix information, data and knowledge from various sources into a unified knowledge repository without prior knowledge or awareness of the structure or the model of the data, and
2. Establishing a suitable model and implementation for a flexible and adaptable architecture for a knowledge storage system that can operate in heterogeneous and changing environments.

Finally, the Presentation and Interaction layer addresses the concern of developing a method to construct adaptable user interfaces that works on different device types.

The Presentation and Interaction layer has access to the Knowledge Representation layer to access the data model of the collaborative application such as classes and variables, and to store data, knowledge and annotations. It also has access to the Group Communication layer to enable collaboration and data transfer.

The Knowledge Representation layer uses the Group Communication layer and facilities from the Middleware layer in order to fulfill its flexible approach towards architecture as described in the Aim and Objectives (see Section 1.4, Chapter 6).

We defined interfaces between each of these layers (or components of the solution) so that the work on each layer can be carried out separately. Furthermore, the gaps between the layers as shown in Figure 3-1 illustrates that each of these components or layers can be used independently in other contexts if need be. However, the fundamental principle is to use them together to build ACTs as will be demonstrated in Chapter 9.

3.2 Middleware Layer

ACTs need to be able to operate on a variety of fixed and wireless networking infrastructure. They also need to be adaptive to the changing environment and requirements. The Middleware layer provides support to resolve some of these concerns including dealing with heterogeneous networks. It connects the distributed components of the collaborative application and provides facilities for low-level communication. Chapter 4 provides further details.

3.3 Group Communication Layer

Communication support for collaborative applications is vital. This layer provides ACTs with administrative facilities to manage groups and users of a collaborative application. It also provides support for distributing data and information among users and groups. This layer hides the complexity of the lower middleware layer so that the

upper layers remain intact if the middleware layer is changed. This layer is generic and has been implemented using two different middleware platforms. Chapter 5 describes the Group Communication layer in further detail.

3.4 Knowledge Representation Layer

To clarify the meaning of data, information and knowledge used in this thesis, the following quotation is presented below from the Atlantic Canada Conservation Data Centre [ACCDC].

> *"Individual bits or "bytes" of "raw" biological data (e.g. the number of individual plants of a given species at a given location) do not by themselves inform the human mind. However, drawing various data together within an appropriate context yields information that may be useful (e.g. the distribution and abundance of the plant species at various points in space and time). In turn, this information helps foster the quality of knowing (e.g. whether the plant species is increasing or decreasing in distribution and abundance over space and time)."*

In keeping with this definition, our use of the term 'Knowledge Representation layer' refers to the support provided to knowledge which includes data and information.

HHEs challenge the way collaborative applications are built presently especially, in terms of knowledge representation (both in terms of data and knowledge modelling and storage). The Knowledge Representation layer of the proposed model addresses two major concerns in building ACTs: the data modelling aspect and the data storage facilities. It provides a new Web-based approach to express and structure data and knowledge for collaborative applications to allow for data from a range of sources to be mixed freely and queried by the application without previous knowledge of its model and structure. This layer also addresses the storage facilities that are required by

ACTs for storing knowledge. The novel storage facility, the KB resolves some of the issues posed by HHEs and changing requirements by adopting flexible architecture.

The KB does not commit itself to a particular architecture (centralised or distributed) but can be configured to operat in centralised, distributed and replicated structures. This is a powerful approach, for example in settings where a range of device types (some with limited storage capabilities) are used in a collaborative session. The knowledge storage can be configured to run only on the high-end devices but serve devices of different types. Another example is when the collaborative application requires a reliable storage facility with no initial concern about capacity. A centralised or replicated KB can fulfill the application requirements easily. But as the volume of data increases the application would require more storage capacity. In this case, scalability becomes significant and the KB architecture would need to be adjusted to a distributed mode, a more scalable architecture. These examples illustrate the advantages of adopting a flexible architecture to address changes of requirement and heterogeneous environments over the running life-time of the collaborative application. Chapter 6 describes further the work on the Knowledge Representation layer.

3.5 Presentation and Interaction Layer

The task of writing applications that target generic device types is challenging specifically from the user interface point of view. Different devices ranging from desktop computers, laptops, through to mobile devices have different specifications such as screen sizes and resolutions. There has not been a similar method to develop user interfaces that could produce a design which can fit different types of devices. This research recognises this issue from the Application Scenario described in Chapter 1 and proposes a solution. This approach offers a real alternative to the current methods of user interface construction. Chapter 7 explains our approach to developing adaptable

user interfaces for collaborative applications.

3.6 Summary

This chapter presented the proposed four-layer model that overcomes some of the shortcomings of current collaborative applications. The model has the potential to deal with issues such as inflexibility in architecture, working on single type of devices and the inability to incorporate heterogeneous data. The proposed model can be used to develop more flexible ACTs.

4

Middleware

This chapter describes the Middleware layer introduced in the previous chapter. There are various forms of networks the Middleware layer is required to support according to the Application Scenario including wireless ad hoc, fixed and wireless networks. The Middleware layer provides mechanisms to connect the distributed components of a collaborative application and provides facilities for low-level communication. It also handles operating systems and network infrastructures heterogeneity.

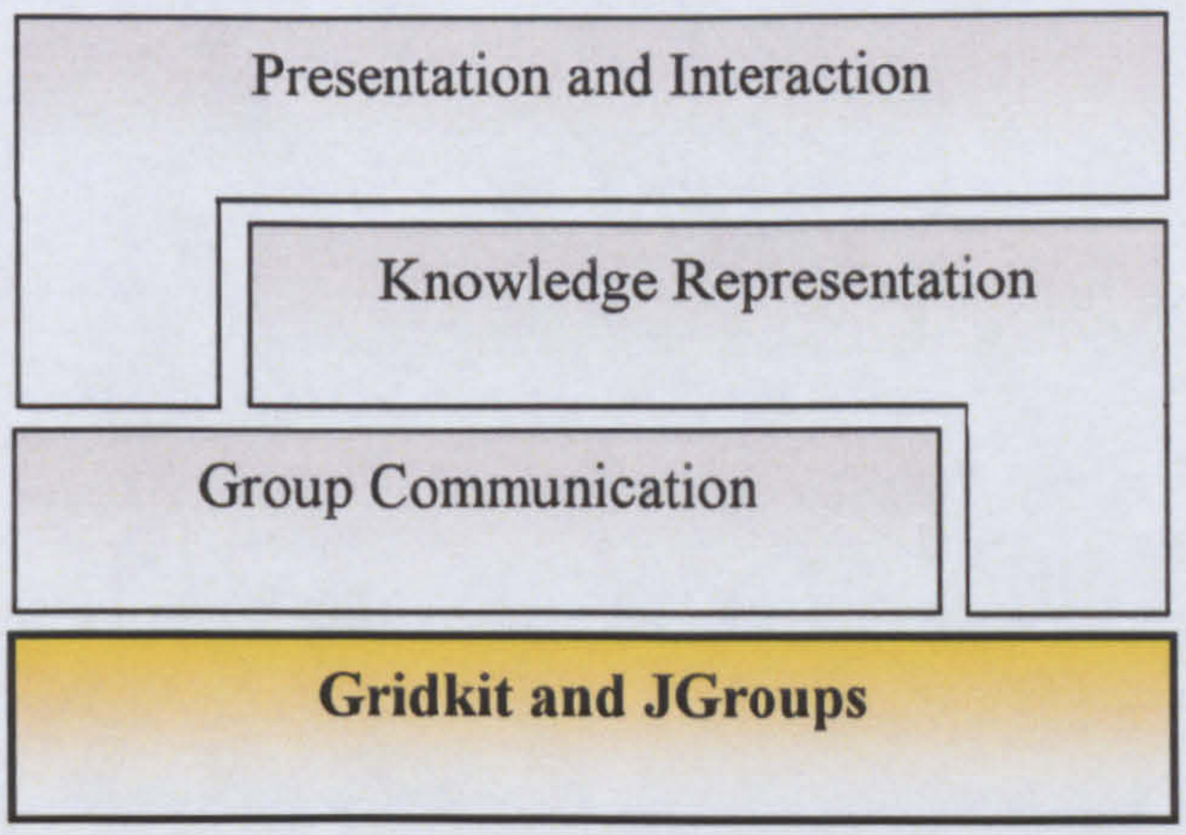


Figure 4-1: The four-layer model: Middleware layer (Gridkit and JGroups).

This chapter describes two middleware infrastructures that were used for this layer, JGroups and Gridkit (as shown in Figure 4-1).

4.1 Overview of Overlays Networks

"An overlay network consists of a collection of nodes placed at strategic locations in an existing network fabric. These nodes implement a network abstraction on top of the network provided by the underlying substrate network ".[Jannotti, Gifford et al., 2000].

The collections of nodes that construct an overlay network are connected with virtual links which may correspond to many physical links in the underlying network. Overlay Networks can be used to prevail over shortcomings of the underlying physical networks (for example, to deliver media streaming or IP multicast). The following subsections describe common examples of overlay networks.

4.1.1 MBone

Multicast is a term used to describe the most efficient strategy to deliver data to multiple destinations simultaneously. This involves ensuring that data is sent only once over each link of the network while creating copies of the data when the links to different destinations split. Multicast is difficult over the Internet for the following reasons:

1. Some of the Internet infrastructure networks support point-to-point communication only,
2. Internet routers either do not support multicasting or the multicast feature is switched off by default, and
3. Most multicast protocols are still experimental technologies.

The Multicast Backbone (MBone) [Eriksson, 1994] was introduced to resolve some of those issues. MBone is an experimental multicast protocol for the Internet which uses tunnelling to avoid point-to-point networks and routers that lack multicast support. It wraps multicast packets in traditional unicast packets so that unicast routers can handle

the information. To handle MBone multicast traffic an 'mrouter' is needed. Mrouters are either commercial routers that handle multicasting or are dedicated workstations running special software that works in conjunction with standard routers. As more and more commercial routers that handle multicasting are becoming available the MBone will eventually become outdated.

4.1.2 Peer-to-Peer

P2P networks have different architectures to the well-known Client/Server model (centralised architecture). The Client/Server model has two distinct entities: the Server, a high end computer capable of handling an increasing number of requests and the Client, a device that makes requests to the server.

A P2P network [Aberer and Hauswirth, 2002] is a collection of connected nodes. The structure of a P2P network can be a tree, a ring, or just random. The software that implements the necessary interfaces for a specific P2P network is usually called a 'node'. Each node of a P2P network acts as a Client and as a Server at the same time. They collaborate to accomplish various functions such as searching for information, transferring data and broadcasting media files.

The unique architecture of P2P networks resolves major shortcomings of the Client/Server model such as a single point of failure, the requirement of high specification servers (i.e. CPU speed, memory, etc.) and the consumption of high bandwidth between the server and its clients.

4.1.3 Distributed Hash Table (DHT)

P2P networks have a well-known advantage of distributing resources between their nodes (i.e. storage space, processing power, etc.). This has motivated researchers to investigate more in the field of sharing data (i.e. file sharing services). And as a result of this effort, DHT was introduced. DHT build on the P2P mechanisms to locate

information. DHT allows efficient storing and retrieving of (key, value) pairs on a P2P network. P2P nodes of a DHT collaborate to maintain the mapping of the keys and values on the network [Ratnasamy, Francis et al., 2001]. Fundamentally, there are two methods to locate information in P2P systems:

1. Using a centralised server which stores the addresses of all the resources available on the P2P network; while this is a reliable way to obtain information it makes the P2P network vulnerable because of the single point of failure.

2. Using a flooding search method to look for information; this technique is not reliable because the information might not be found even though it is available on the P2P network because the flooding search does not reach all nodes of the P2P network.

To avoid the single point of failure yet accomplish the reliable retrieval of data, some DHT systems were based on the flooding query model [Frankel and Pepper, 2000]. Search queries in these networks are broadcast to all possible nodes in the network. Key-based DHT systems followed, with search queries being routed efficiently (involves a minimum number of nodes) to a specific node which is likely to be hosting the data required [Stoica, Morris et al., 2001] [Druschel and Rowstron, 2001] by mapping file names or resources to locations. Applications built using a key-based DHT enjoy many advantages such as a decentralised structure, scalable architecture and fault tolerance. A broken node(s) would not have a big impact on the functioning of the DHT in this case.

4.2 JGroups

JGroups [Ban] is a Java package for reliable group communication (or reliable multicast communication) and this section briefly presents this package. JGroups models a rather

low-level Message Oriented Middleware (MOM) and it consists of three components:

1. A socket-like API for application development,
2. A protocol stack which implements reliable communication and
3. A set of building blocks which give the application/protocol programmer high-level abstractions.

The main features of JGroups are:

1. Group creation and deletion (across LANs or WANs),
2. Joining and leaving of groups,
3. Notification about joined/left/crashed members,
4. Detection and removal of crashed members,
5. Point-to-multipoint communication: Sending and receiving of member-to-group messages and
6. Point-to-point communication: Sending and receiving of member-to-member messages.

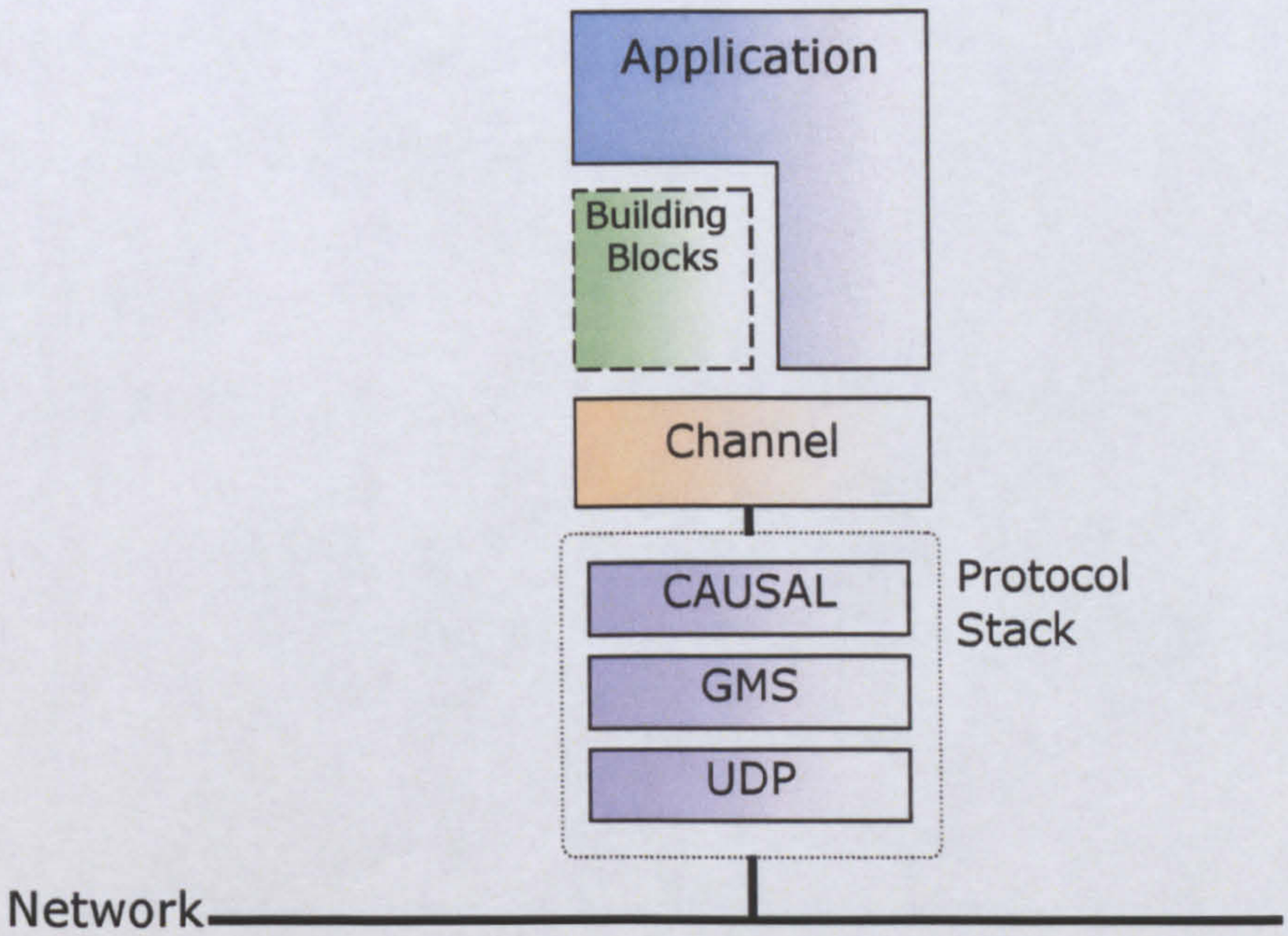


Figure 4-2: JGroups Architecture

Figure 4-2 illustrates the three main components of JGroups: the Protocol Stack, Channel and the Building Blocks. The following subsections briefly described the Channel and the Protocol Stack components of JGroups.

4.2.1 Channel

A channel looks like a socket and it represents a group. There are methods for joining and leaving groups, sending and receiving messages to/from members, getting the shared group state, and registering for notifications when a member joins, or an existing member leaves or crashes. The software connects to a channel by providing the name of the group it would like to join. At first JGroups looks for a channel with that name to see if that already exists and if not found, it creates a new one.

4.2.2 Protocol Stack

The protocol stack is a linked list of protocols, through which each message has to be passed. Each protocol implements an Up() and Down() method, and may modify, reorder, encrypt, fragment/unfragment, drop a message, or pass it up/down unchanged. The protocol stack is created according to a specification given when a channel is created. New protocols can be plugged into the stack easily. By mixing and matching protocols, various application requirements can be satisfied.

4.3 Gridkit

Gridkit [Grace, Coulson et al., 2004] is a middleware infrastructure that has been developed to realise the concept of Open Overlays described in Section 1.6 to support the building of Grid applications that can work on heterogeneous networks and operating systems.

4.3.1 The Grid

The Grid [Kesselman and Foster, 1998] is a paradigm that is used to harness scattered

resources such as supercomputers, storage resources, data sources, sensors and privileged devices. At present, applications written for the Grid employ Grid distributed resources to solve problems and provide services.

The Grid can utilise the power of the increasing number of computers connected to the World Wide Web (Web) since it was invented in 1990 by Tim Berners-Lee and Robert Cailliau. The Web has allowed us to share information, transfer data, and access services by using Web technologies such as HTML (Hyper Text Markup Language). While there are billions of computers connected to the Web, the Web has not been able to take advantage of their capacities and allow the sharing of their processing powers, storage resources and other facilities. The Grid is intended to address these concerns.

Grid applications are usually built on an underlying software infrastructure that provides vital Grid services, such as service discovery and data transfer; this software infrastructure is called Grid middleware. There has been a growing consensus among members of the Grid middleware community that current Grid middleware does not provide adequate services and facilities to Grid applications [Grace, Coulson et al., 2005] and that the methods used to construct Grid applications are not adequate to allow such applications to operate in HHEs. The Open Overlays project addresses the issues of heterogeneity in the area of Grid middleware. The original idea of the Grid was to link supercomputers available in spread out locations to work together for scientific research. The focus was to manage CPU time on supercomputers interconnected to local or wide area networks. This has been extended to include managing other resources in addition to the CPU time such as accessing memory, databases and data sources, storing devices, accessing services and electronic instruments.

Grid applications are a special class of distributed applications that employ Grid distributed resources to solve a problem. It can also be said that Grid applications are those applications built to operate on Grid middleware environments (i.e. Globus

[Foster, 2006]). Grid applications are commonly built to use the services provided by the Grid middleware (Grid Services). These services provide access to Grid resources, such as storage facilities, computational power, hardware instruments, etc. However, this approach has many shortcomings [Grace, Coulson et al., 2005].

Gridkit is the alternative Grid middleware proposed by the Open Overlays project to overcome the shortcomings of current Grid middleware mentioned earlier. Gridkit has been used in this research and will be described below following the OpenCOM platform section.

4.3.2 OpenCOM

OpenCOM v2 [Coulson, Blair et al., 2004] [Grace, Hughes et al., 2008] is a platform and programming language independent component model that can be used to construct component-based systems [Szyperski, 1997]. It can be used to build cross-platform and cross-languages applications. OpenCOM has reflective features to support dynamic runtime reconfiguration of applications to allow operations such as: load, unload, bind, and rebind components at runtime. It works on a wide range of environments such as operating systems, PDAs, embedded devices and network processors.

A component is a run-time entity that compliant applications and other components can utilise. The component model promotes a high level of abstraction in the design, implementation and deployment of software systems. It also enables configuration and third party reuse. There are two types of component models: standalone and distributed [Emmerich and Gruhn, 2004]. Examples of component models are: JavaBeans [JavaBeans] and CORBA Component Model [Orfali and Harkey, 1997].

OpenCOM component model has three key concepts; interfaces, receptacles and connections (see Figure 4-3). An interface conveys a point of service, while a receptacle

describes a service requirement. A connection binds an interface and a receptacle of the same type. An OpenCOM component can implement a set of interfaces and receptacles to interact with other components running in the same address space. In the OpenCOM model, each address space has a single OpenCOM runtime that manages the components. OpenCOM runtime creates, deletes, connects and disconnects components running within its address space. Moreover, the OpenCOM model also provides services for reflection and supports re-configuration by maintaining a system graph (a list of the components currently running). The reflection feature in OpenCOM allows inspection and adaptation of the current configuration of the components running in the runtime. In order to support reconfiguration, it allows inspection of the interfaces and methods provided by the component. Reconfiguration is achieved in the OpenCOM model by allowing a third-party agent to make or break the connections between components. To maintain independence from programming languages Object Management Groups (OMG) was used. OMG is Interface Definition Language (IDL) from CORBA [Orfali and Harkey, 1997].

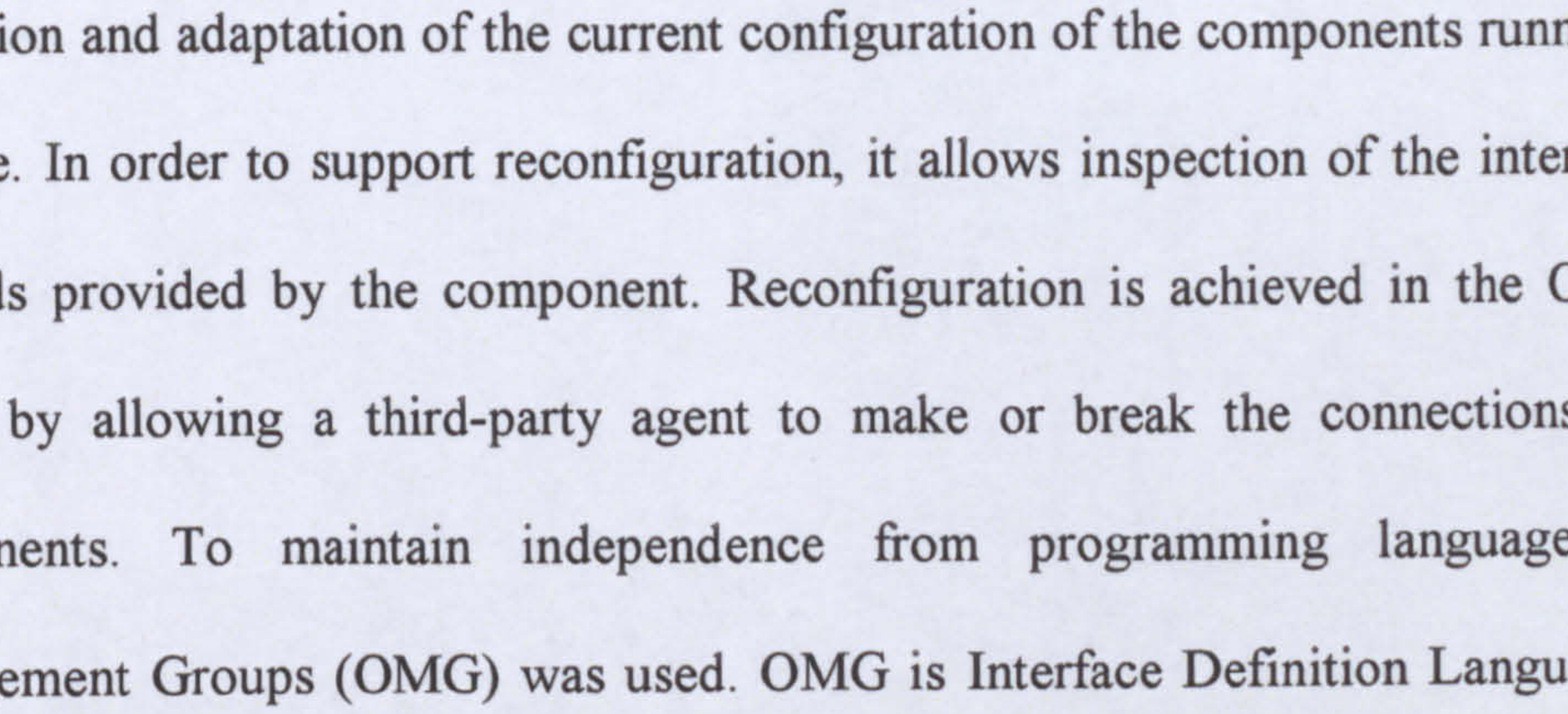


Figure 4-3: An address space that contains two OpenCOM components, one that has implemented an interface (right side) and the other one that has implemented a receptacle of the same type, bound together with a connection; an OpenCOM runtime (bottom right) holds the system graph.

OpenCOM can be deployed on a range of platforms and it can be built to work on many

devices (e.g. desktops, PDAs, routers, network processors, etc.). Different implementations are necessary for the OpenCOM model to work on platforms such as Windows or Linux. Implementations for smaller devices with minimal or no operating systems (e.g. sensor motes) can be achieved on top of their physical memory using machine code.

To further understand the OpenCOM platform, an Image Viewer was developed to demonstrate how a third-party can reconfigure an application at runtime to adapt to the surrounding environment changes. The application was written in C++ and used OpenCOM v1. This implementation of OpenCOM operates on devices running flavours of the Windows Operating system while OpenCOM v2 is an operating system independent version, which can be applied across diverse devices such as sensors and programmable routers.

Three components were implemented; ReadImage which implements an IImage interface, DisplayImage that implements an IImage receptacle and ResizeFilter which implements an IImage interface and an IImage receptacle. IImage interface/receptacle provides/consumes one operation, getImage (String imageName, float resizeRatio). Initially, ReadImage is connected to DisplayImage (see 'A' in Figure 4-4).

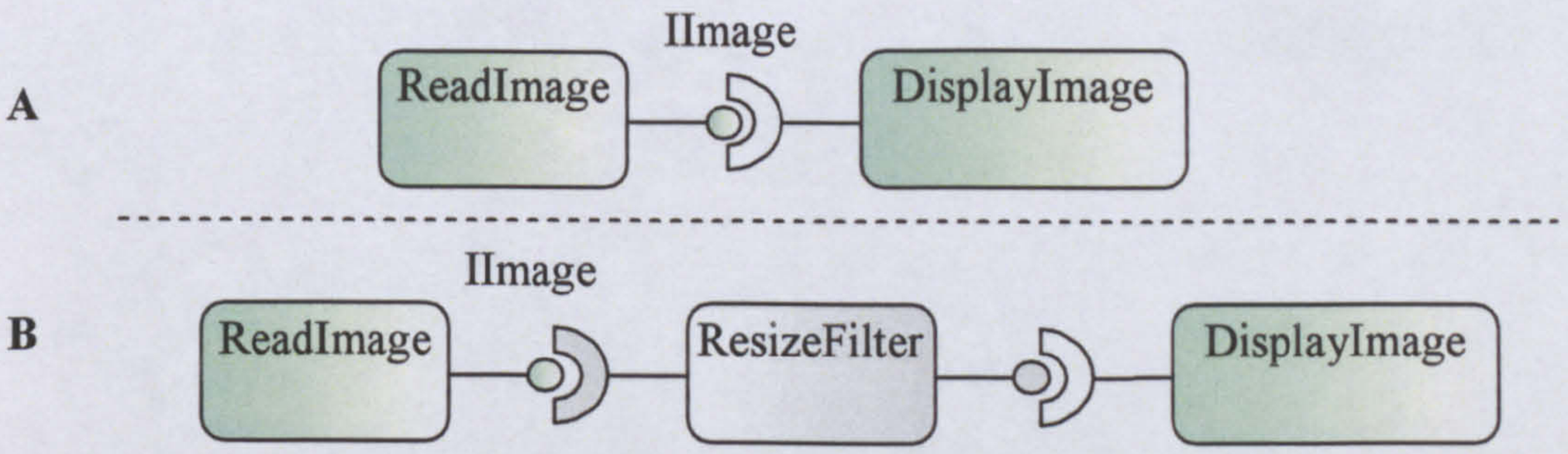


Figure 4-4: Image Viewer application that contains three OpenCOM components, ReadImage, ResizeFilter and DisplayImage.

The application can request an image from a list of images to be displayed on the client's screen. The first parameter of the getImage operation takes the image name and

the second parameter takes the percentage of the reduction to be applied to the image size (e.g. 50%, 75%, etc.). When the DisplayImage component is connected directly to the ReadImage component, there is no effect due to changes of the resizeRatio parameter. The application user interface (the third-party in this case) allows the insertion of a filter component (ResizeFilter) between the ReadImage and DisplayImage components (see 'B' in Figure 4-4). The application then gets a resized version of the original image depending on the value of the resizeRatio. This can save network bandwidth if the ReadImage and ResizeFilter components are hosted on a different machine from that hosting the DisplayImage component. Before ResizeFilter was inserted between ReadImage and DisplayImage component, the application was manually stopped to achieve the *quiescent* state. This is to maintain the integrity of the system for safe dynamic reconfiguration.

4.3.3 Architecture

Gridkit provides a highly configurable middleware framework based on the lightweight component model OpenCOM. Gridkit is designed to:

1. Support a diverse number of network infrastructures and end-systems.
2. Provide its applications with a range of communication styles.

These issues are addressed in Gridkit by providing a configurable set of middleware frameworks over a layer of overlay networks as illustrated in Figure 4-5.

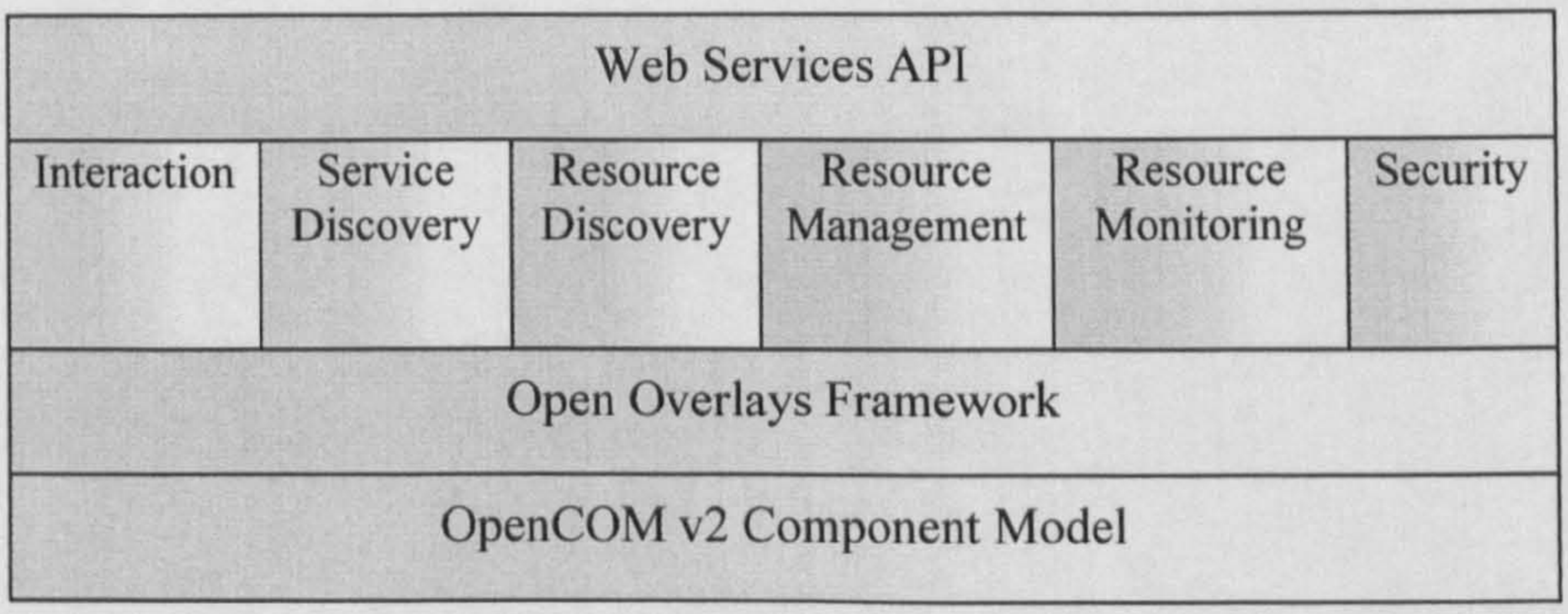


Figure 4-5: Gridkit Architecture

Gridkit supports building applications in components using the OpenCOM components model [Michael, Blair et al., 2001] which offers dynamic reconfiguration to its applications. Gridkit employs an extensible family of open and programmable Overlays Networks (Open Overlays Framework, see Figure 4-5 and Figure 4-6) whose role is to route packets through virtual networks to support various interaction types. Different device types usually use different types of networks enabling them to communicate with other devices. Supercomputers, clusters and desktop computers are often connected to fixed networks such as high-speed/low-speed local and wide area networks. Laptops, PDAs, and other mobile devices are increasingly using wireless networks, while sensor devices can be connected using wireless ad hoc networks. Communication between devices running on the same or different types of networks is essential for future Grid applications. Gridkit resolves the issues of network heterogeneity regarding communication and provides the interaction types for its applications. Classical overlay networks are fixed and cannot be reconfigured while open overlays are built from components and can be modified at runtime. They are rebuilt using Control, State and Forward component as explained below (see Figure 4-6).

Gridkit provides other services through a Web Service API such as Interaction, Service Discovery, Resource Discovery, Resource Management, Resource Monitoring and Security (see Figure 4-5).

The term overlay network can be used to refer to any computer network that is built on top of another network (see Section 4.1). The collections of nodes that construct the overlay network are connecting with virtual links which may correspond to many physical links in the underlying network.

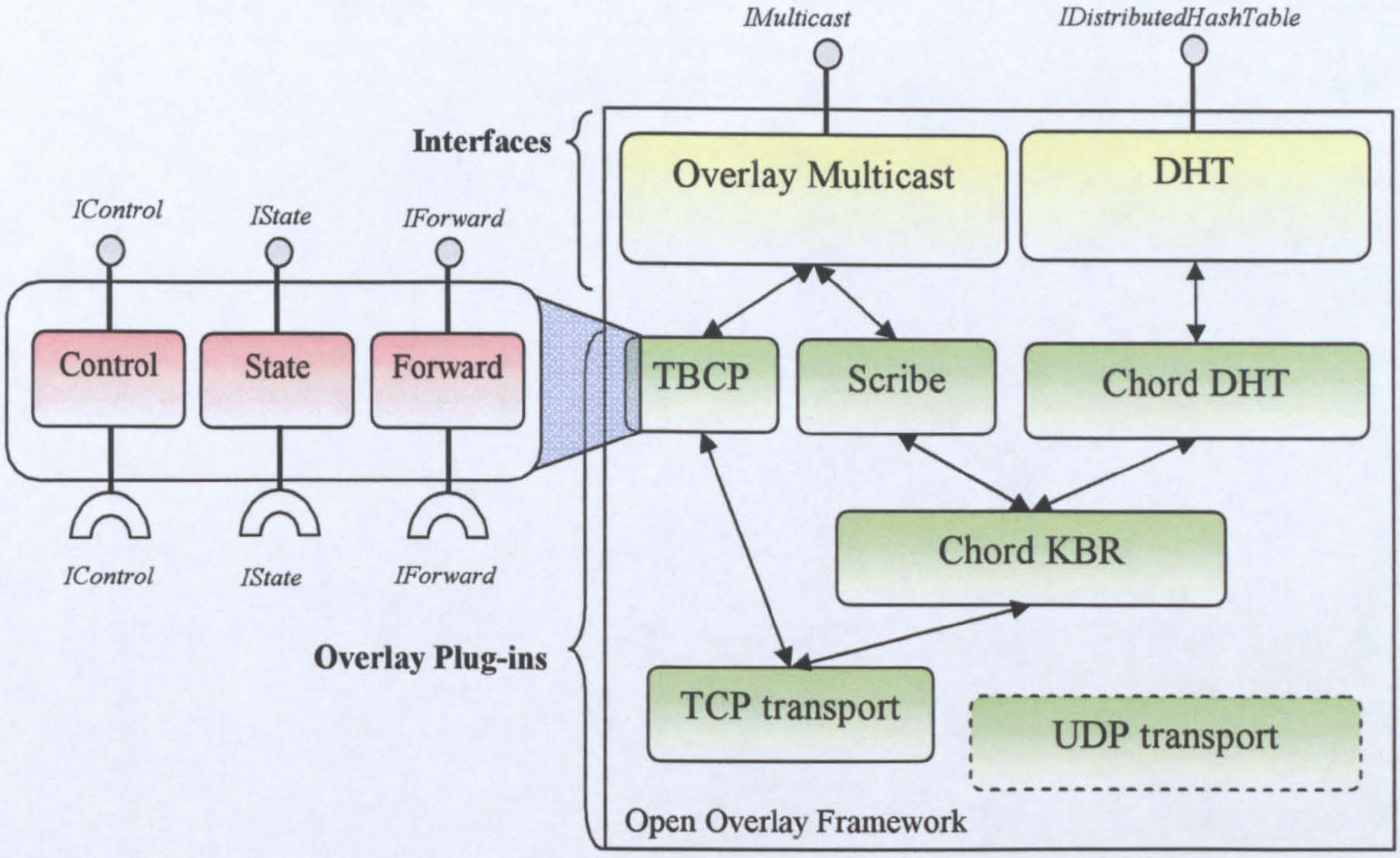


Figure 4-6: An example configuration of the Open Overlays Framework.

Using overlay networks enables Gridkit to span over a diverse set of networks (network heterogeneity) and it also allows for the provision of network services that are not supported by the underlying network infrastructures such as multicast. OpenCOM is used to build Gridkit and these overlay networks. Also, it guarantees that Gridkit can be implemented on a range of devices from high-end to very primitive devices.

Figure 4-6 shows an example of the Open Overlays Framework supporting many different overlays. There are two types of components that can be used in the Open Overlays Framework and these are:

1. Open Overlays plug-ins: implementations of overlays networks in the OpenCOM platform.

2. Interface plug-ins: capture common API patterns that can be shared among multiple overlays.

In Figure 4-6, there are 4 types of overlay plug-ins: TBCP (Tree Building Control Protocol) [Mathy, Canonico and Hutchison, 2001], Scribe [Castro, Druschel et al.,

2002], Chord DHT and Chord KBR (Key Based Routing). TBCP is a generic protocol for building overlay spanning trees to provide multicast without help from network routers. Scribe is also a scalable application-level multicast infrastructure. Chord is a protocol for building efficient P2P networks; Chord DHT is an implementation of DHT on Chord P2P network. Chord KBR is a more flexible protocol than DHT built on Chord network and used to create and use distributed services for P2P applications. Those overlays can operate in parallel either separately (such as TBCP and Scribe) or in a stacking relationship (such as Chord KBR and Scribe). TCP (Transmission Control Protocol) transport and UDP (User Data Protocol) transport are called null overlays because they implement the network transport behaviour but they do not perform any routing. Also, as shown in Figure 4-6, the Overlay Multicast interface is a common interface used by two overlay plug-ins (TBCP and Scribe).

The left-hand-side of Figure 4-6 depicts the decomposition of an Overlay plug-in (TBCP) into three OpenCOM components: Control, State and Forward. The Control component is responsible for coordinating with its peers in other stations to build and maintain an overlay-specific virtual network topology. The State component maintains information for the overlay such as nearest neighbours. And the Forward component determines how to route messages over the network topology. Each of these components has a receptacle and an interface to communicate with the peer nodes (see Section 4.3.2). This three-element architecture - which consists of control, state and forward components - is called the Overlay Pattern. By allowing the control and forward components of the current overlay network to be replaced with another type without loss of state, Gridkit achieves flexibility in configuration and reconfiguration.

4.3.4 Configuration and Reconfiguration

Gridkit uses profiles for configuration at deployment time to determine the Overlays Patterns, the overlays plug-ins and interfaces. The initial configuration of overlays can

later be reconfigured using the reflective features of the OpenCOM platform as described earlier. For safe dynamic reconfiguration the system should be in a *quiescent* state so that the changes do not affect the integrity of the system. The simplest way to achieve this state is to manually stop all activity in the system and trigger the reconfiguration process. Once the reconfiguration of the overlays has taken place, the system can be turned back on to run normally. Gridkit supports another way to achieve the quiescent state where a request for reconfiguration is made from a centralised node to each node in the framework requesting it to enter a quiescent state. Once a node is in a quiescent state it returns a notification to the configurator node. When all nodes are in a quiescent state the configurator node starts the reconfiguration process.

4.4 Summary

This chapter has provided background information about the middleware technologies used in this research including Gridkit and JGroups. JGroups and Gridkit are different in terms of their structure and target applications. JGroups provides a flexible approach to group communication for ACTs and is used to demonstrate the generic nature of the four-layer model. On the other hand, Gridkit can serve as an infrastructure to build collaborative applications while resolving any network and platform heterogeneity concerns (JGroups is not capable of doing this).

5

Group Communication

Collaborative applications require a reliable means for communication to enable collaboration to take place effectively between groups of people. This communication medium is known as group communication. This chapter introduces our approach to addressing group communication in HHEs (as described in Section 1.3) by focussing on the communication requirements of ACTs.

5.1 Introduction

Group communication is a term used here to describe a mechanism to establish communication among members of a group who are involved in a collaborative task. The collection of members of a particular group is dynamic where members are free to join or leave the group. Members of a group know one another and share a state which can be retrieved by newly joined members. This is different from the Publish/Subscribe approach, the asynchronous messaging model, where senders (publishers) of messages don't know the receivers (subscribers). Subscribers specifically register to receive the messages that they are interested in.

Access to the shared state in the group communication model cannot be obtained once a member of a group leaves the group. Recording the state of a collaborative application using the group communication model would enable new participants who

have joined an on-going collaborative session and retrieved its current state to make queries on the stored data, if any exists. Other advantages of such an approach are to use the data collected throughout a collaborative session for post analysis and to replay the collaborative session after it has ended. These issues are addressed in Chapter 6.

A classical approach to supporting group communication is Message Passing Interface (MPI) [Snirm and Otto, 1998]. It is a language-independent communication protocol used widely on parallel computers to allow many computers to communicate among themselves point-to-point or collectively. MPI defines a set of application programming interfaces (APIs) that can be used to facilitate group communication. These language-independent APIs provide a set of functions or classes to support group communication across applications written in various programming languages. There are many implementations for MPI as each focus on different concerns such as scalability, portability or performance. For example, Open MPI is an open source project that offers an implementation of MPI [Gabriel, E. et al., 2004].

5.2 Requirements

In collaborative applications, it should be possible to:

1. Create a group,
2. Delete a group,
3. Join a group and
4. Leave a group.

Those actions must be restricted to authorised users only. Naming and addressing are ultimately used to establish a connection between different entities in a collaborative application that can be used to transfer data between them. Connections can be controlled so the necessary data can be transferred to the whole group or to an

individual member. Here are the main four elements a collaborative applications design must address:

1. *Addresses*: There is a need in any collaborative environment to identify users and groups independent from their physical locations. Techniques to link names to physical locations are well established. Keeping the names of users and groups independent from their addresses is called *location-transparency*, so that if a user changes address that does not affect the organisation of the group as this has to be detected in the lower-level. In this approach we use URIs (Uniform Resource Identifiers, see Section 6.2.1) to identify users, members and groups. Addresses are identifiers used to locate entities which are handled by the support platform for the collaborative applications.

2. *Groups:* groups are used to represent a number of collaborative users and should have two attributes:

    a. Name (or group id, gid): which is used to refer to a group and

    b. Administrative information: to keep metadata and administration rights information such as: creation, modification and access dates, group's creator, textual description and access rights.

3. *Users:* which represent people who use the collaborative applications. Therefore it is necessary for each person to be registered as a user to enable each person in the real world to be represented by a user in the collaborative application. A user can be a member, a creator or a manager of a group. Users can have a number of properties in a collaborative application such as:

    a. Name (uid): each user is identified by a unique name. This is used internally by the collaborative layer or by the application. Name does not rely on the location (location-transparent),

b. Administrative information: contains data like real name, email, address, last login, etc.,

c. Access rights: to specify who is authorised to request information about the user. This could be 'open' to all or 'restricted' to group's members,

d. Identification information: the user has to identify him/herself to the system to gain access to the data,

e. Groups (gid): a list of group ids of which the user is a member.

4. *Data Storage*: Collaborative applications need to store related data and be able to retrieve it later. Each group should be associated a storage facility in order to satisfy this requirement.

5.3 Group Abstraction Interface (GAI)

Group communication is a vital facility for collaborative applications and for this research. In order to meet the requirements described in the previous section we propose the Group Abstraction Interface (GAI). GAI further enhances the model of group communication explained above (see Section 5.1). It is responsible for establishing connections between collaborative applications with the benefit of hiding the complexity of the middleware infrastructure from the application which allows alteration of the infrastructure without informing the application, see Figure 5-1.

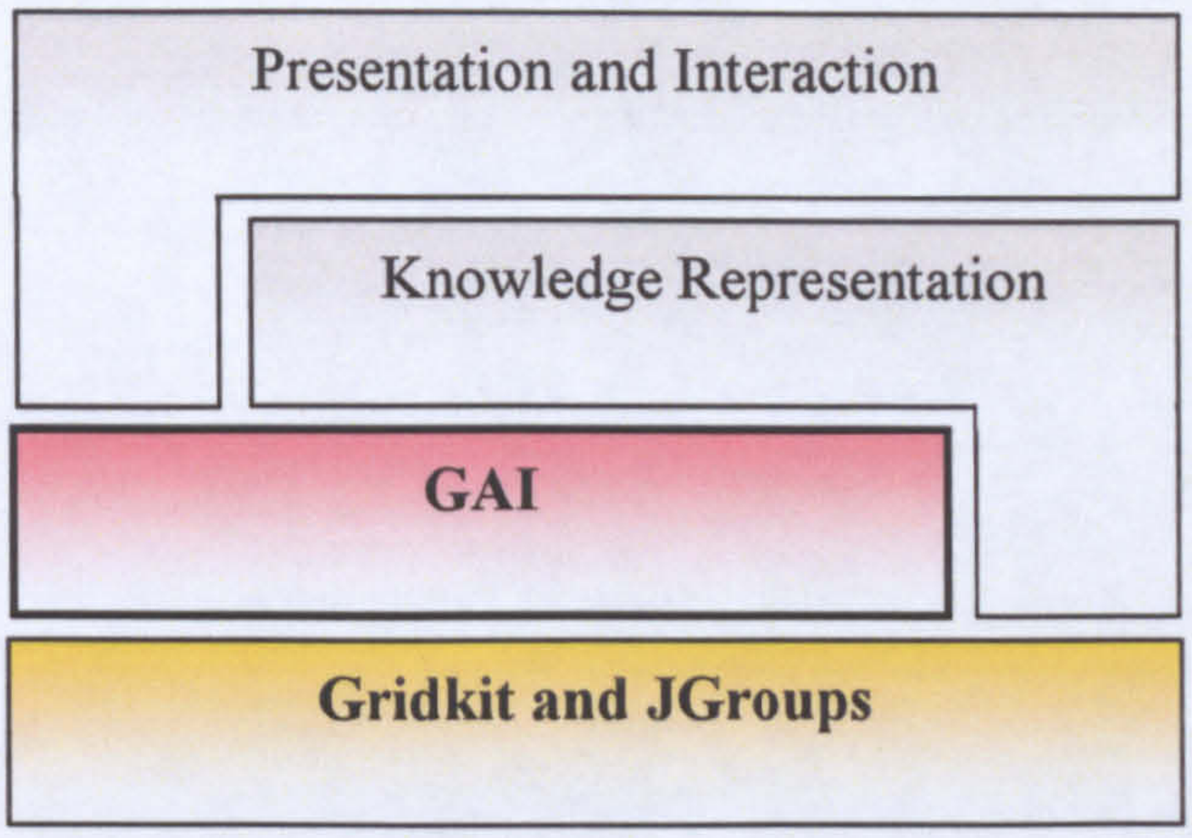


Figure 5-1: The four-layer model: Group Communication layer (GAI).

The benefit of using a GAI approach over, for example MPI (see Section 5.1), is that it hides the complexity of the underlying middleware infrastructure. This is achieved because collaborative applications only need to know about the programming interface of GAI to use regardless of the underlying infrastructure or the programming language.

Most collaborative applications are based on proprietary communication paradigms. This is due to the lack of general collaborative enabling architecture. The idea of GAI is to provide a collaborative enabling layer for collaborative applications that is not reliant on the low-level communication paradigm or the middleware layer. Applications adopting this approach are interoperable because they have a shared collaborative layer but can vary with the middleware layer.

GAI is not an anonymous form of communication and so every member of a group is visible to other members in the same group. This helps to enhance collaboration and build awareness of others within the group. In this paradigm, the notion of a *channel* is not used and groups are created explicitly. Multiple groups can be joined by the same user, and members of the same group are aware of each other.

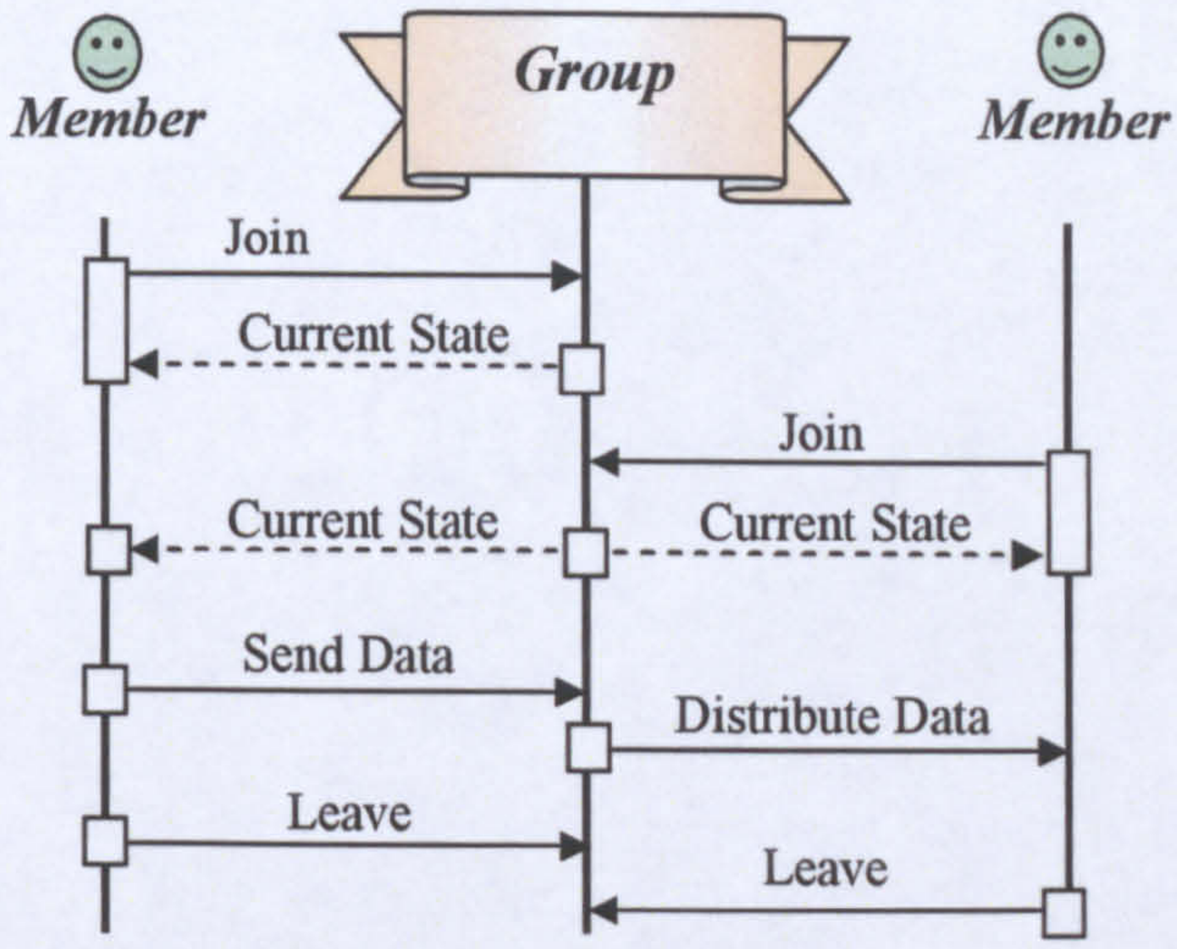


Figure 5-2: Interactions in the group communication model.

Figure 5-2 illustrates the interactions between users in GAI. When users join a group they become members of that group and all members of the group including the newly joined member get updated with the current state of the group (see Figure 5-2, arrowed dashed lines to indicate group state update), for example, a list of all other members of the group. A message sent by a member of the group is delivered to all other members. When a user leaves a group the member status between the user and the group expires and the user no longer receives updates from the group.

GAI defines a number of interfaces to support all aspects of group communication mentioned above. Concrete implementations of these interfaces can be realised using any communication paradigm. To demonstrate the generic and cross-platform nature of this approach two implementations of GAI have been realised using two types of middleware: JGroups and Gridkit (see Section 4.2 and Section 4.3).

The GAI model introduces a number of entities to support group communication such as: Members, Groups and Group Management. The following subsections introduce these interfaces.

5.3.1 Group Management Interface

The Group Management Interface (GMI) provides access to some aspects of group communication in GAI. It is used mainly to manage groups (creation, deletion, etc.). Some methods of the interface (createGroup, removeGroup) are restricted to certain users with necessary privileges. The interface methods with descriptions are shown below:

```
Public interface GroupManagement {

// Only users with necessary privileges can execute this method
Public Group createGroup (String groupId);

// Returns a list of ids of all available groups
Public Vector getGroupsIds();

// Return a group reference of the given group id
Public Group getGroupById(String groupId);

// Stop all communications of a group, disconnect all
// members of that group and finally, dispose it.
/* 1 */ public boolean removeGroup(String groupId);
/* 2 */ public boolean removeGroup(Group group);
}
```

A reference to the GroupManagement interface can be obtained using a public method called *createGroupManagement* (see Section 5.4). And members can get a reference to the group they want to join using the *getGroupById* method. Each newly created group is associated a KB automatically. This is used to store the group's data and other related information.

5.3.2 Group Interface

Instances of this interface can only be created via the GMI. This interface allows users to retrieve references to group members and broadcast messages to individual members or to the whole group. The following is the description of the Group interface:

```
Public interface Group{
```

```
// Get the id of this group (e.g. "South Brigade").
Public String getGroupId();

// Get a list of ids of all members.
Public Vector getMembersIds();

// Return a member reference of the given member id
Public Member getMemberById(String memberId);

// Send data to all members
Public void send(String msg);

// Send data to some members
Public void send(Vector memsIds, String msg);

// Get the knowledge store
Public KnowledgeStore getKnowledgeStore();
}
```

This interface also provides a reference to the group KB using *getKnowledgeStore* method.

5.3.3 Member Interface

A member is created once a group is joined. A member can send a message to the group, listen to incoming messages or leave the group. Members of a group can be local or remote (running on different machines). The following is the description of the Member interface:

```
Public interface Member{

// Get the id of this member
Public String getMemberId();

// Get the id of the group which this is a member of
Public String getGroupId();

// Send a message to the group
```

```
Public void send(String msg);

// Set a listener to listen to all messages delivered to this member
Public void setListener (MessageListener ml);

// Leave group
Public void leaveGroup();}
```

The method *setListener* accepts an object which implements the *MessageListener* interface (see below)

```
public interface MessageListener {

// Called when a message is received.
void receive(Message msg);

// The group state
byte[] getState();
}
```

5.4 Implementation

In order to demonstrate the generic nature of GAI, two different implementations have been created. The first implementation was created using JGroups (see Section 4.2) whilst the second implementation uses Gridkit (see Section 4.3).

JGroups is used as a form of reliable group communication by several projects. Processes (or members) can join a group to send or receive messages to/from other members of the group. The communication between members of a group is managed by a channel. Groups do not have to be created explicitly, hence, when a process wants to join a group that does not exist, the channel which the process is using to gain access to the group creates that group automatically and the process becomes its first member (coordinator). Groups - in JGroups - keep track of all of their members. The implementations of the above interfaces including GroupManagement, Group and Member have been completed using JGroups.

Figure 5-3 illustrates an implementation of the GAI on Gridkit, showing the OpenCOM components and the Gridkit framework used.

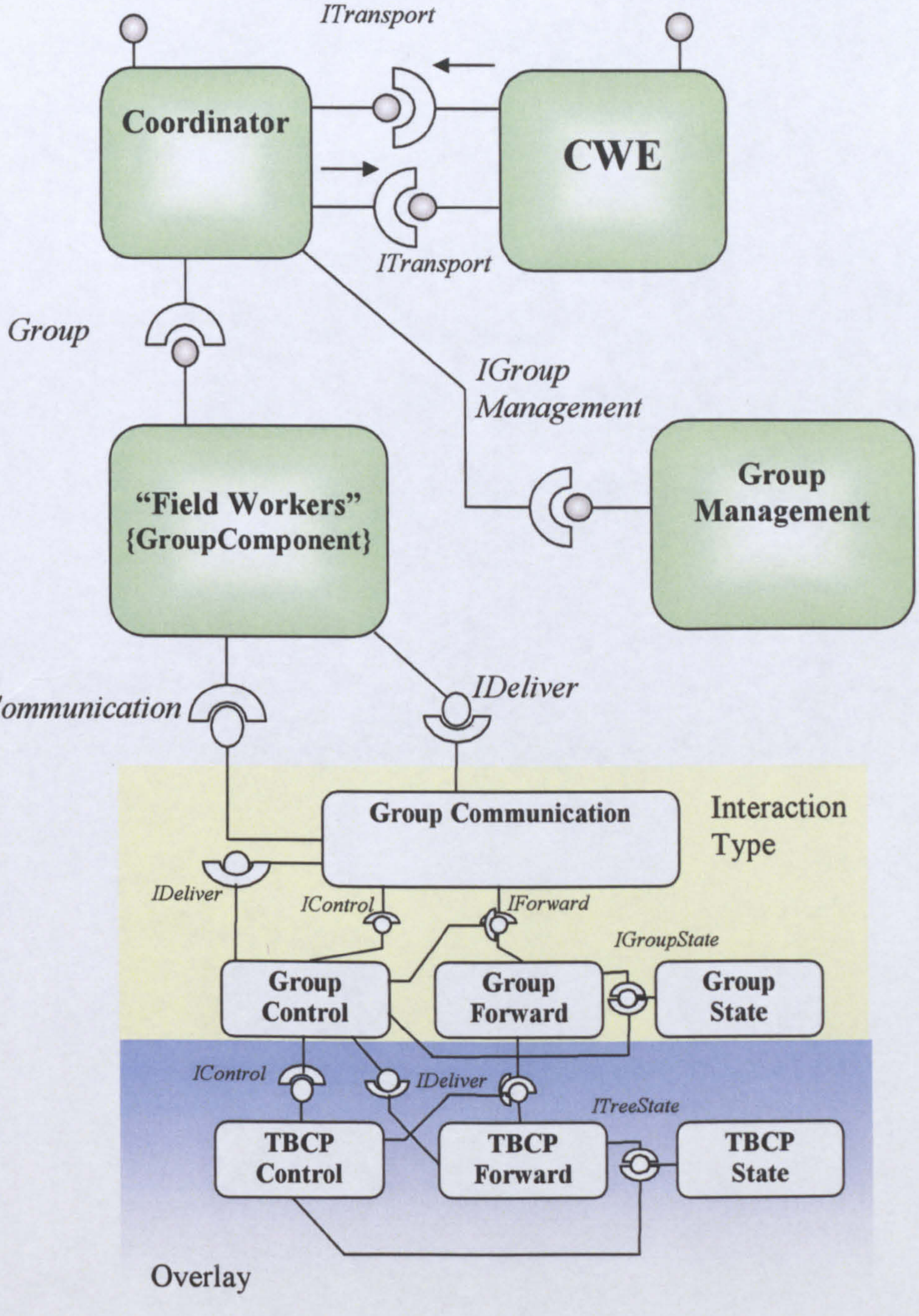


Figure 5-3: Implementation of GAI using Gridkit, ITransport supports send and receives methods

Figure 5-3 exposes the internals of Gridkit middleware, showing the components used to construct the overlay network used by Gridkit to support group communication and known as wide area multicast overlay (known as Tree Building Control Protocol,

TBCP). TBCP is a generic protocol for building overlay spanning trees to provide multicast without help from network routers (see Section 4.3.3). CWE is wrapped in a component which uses the ITransport interface for communication to send and receive messages. The 'Coordinator' component is used as a proxy between the CWE and the IGroupManagment interface (see Section 5.3.1). The group "Field Workers" was created via the IGroupManagment interface as an OpenCOM component. This component fulfills the group communication functions using the Group Communication component. The Group Communication component implements the Group Interface described in Section 5.3.2 by building on a TBCP multicast overlays network.

Below is a working example which works on both implementations, JGroups and Gridkit.

```
Public static void main(String[] args) {
GroupManagement gm = createGroupManagement();
Group myGroup = gm.createGroup("Field Workers");
Member myMember1  = myGroup.joinGroup();
Member myMember2  = myGroup.joinGroup();
                myMember1.setListener(this);
                myGroup.send("Hello World");
                myMember2.send("Brookes University"); }

public void receive(Message msg) {
 System.out.println("Received message: " + msg.getObject());
}
```

Concrete classes were developed to implement a GAI. From the code above, the 'main' method creates a GMI instance and uses it to create a 'Field Workers' group. Two members join this group (myMember1 and myMember2) and the method 'receive' of myMember1 is registered to listen to the group messages. The group interface instance 'myGroup' is used to send a message to all members then another message is sent to the group using the 'send' method on the member 'myMember2'. Both messages will be

received by myMember1 (delivered to the receive method) and printed out on the screen.

5.5 Summary

This chapter has described our approach to supporting group communication in building ACTs. Here is a summary of some of the advantages offered by this approach:

- GAI decouples the implementation of any collaborative application from the underlying communication facilities (middleware infrastructure). This feature has been very beneficial in this research by allowing implementation of GAI on JGroups and Gridkit.
- GAI hides the complexity of the underlying middleware infrastructure.
- Interoperability between collaborative applications can be made possible using GAI while sharing the same name space (Groups, Users) using RDF (this will be explained in Section 9.4.1).
- GAI does not impose the use of any particular communication infrastructure (middleware) and at the same time, different application may choose to use different underlying communication paradigms (Gridkit, JGroups, etc.) yet they can easily intercommunicate between one another.

In summary, the advantage of using this enabling layer of GAI is to reduce the implementation costs, provide a shared namespace of users and groups, and provide a simple interface to different collaborative applications regardless of the transport infrastructure.

6

Collaborative Data and Knowledge Representation

This chapter describes the Knowledge Representation layer of the four-layer model introduced in Chapter 3. The Knowledge Representation layer addresses the requirements imposed by HHEs on knowledge modelling and storage following a Web-based approach.

6.1 CoRDF

A key contribution of this research is CoRDF, an engineering approach addressing the data and knowledge heterogeneity aspect of building ACTs, based on RDF technologies and the component-based Open Overlays approach. It addresses the following two points in this research's Aim and Objectives (see Section 1.4):

1. To develop a method that can be used to mix data, information and knowledge from various sources into a unified repository without prior knowledge or awareness of the structure or the model of the content of the repository.
2. To establish a suitable model for a flexible and adaptable architecture for a knowledge storage system (knowledge repository or knowledge base) that can operate in heterogeneous and changing environments.

Figure 6-1 shows where CoRDF fits into the four-layer model described in Chapter 3.

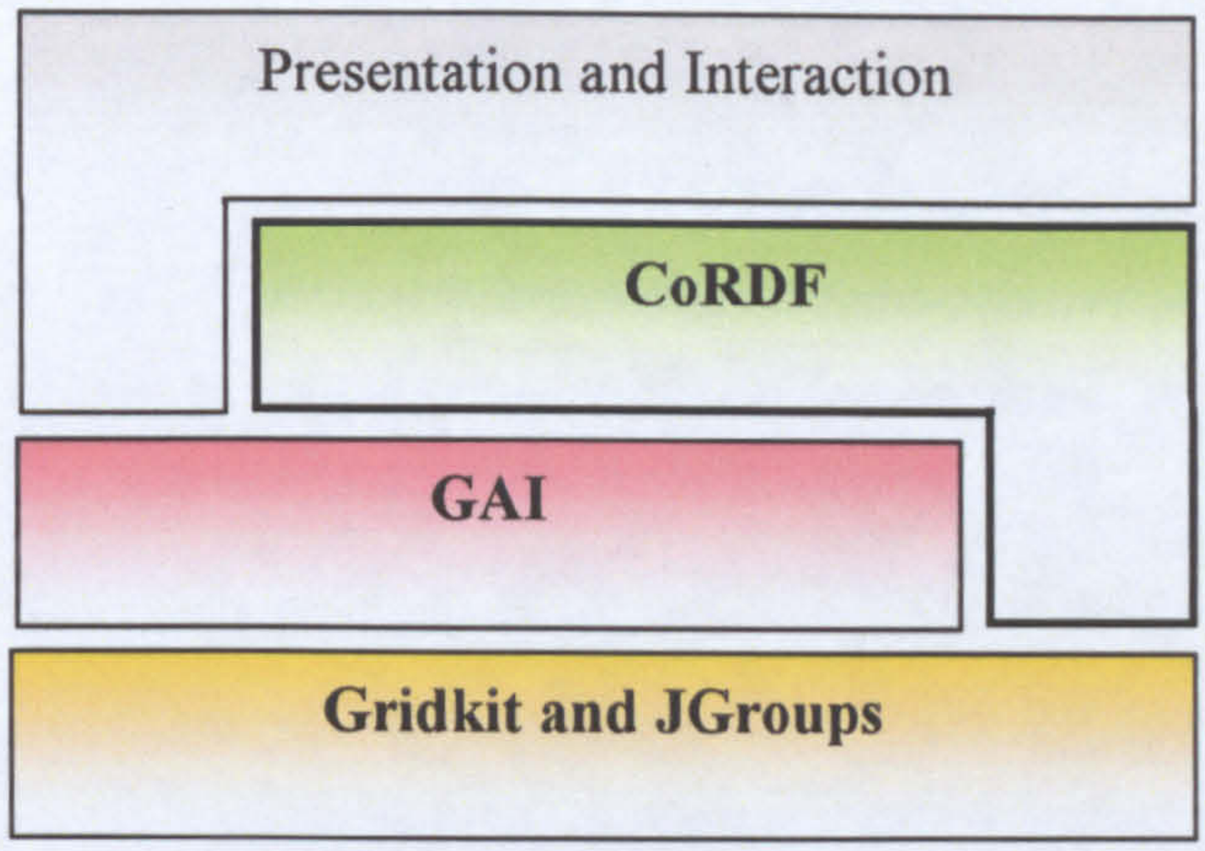


Figure 6-1: The four-layer model: Knowledge Representation layer (CoRDF).

There are two elements to CoRDF:

1. The RDF Platform-Independent Data Model (RDFPIDM).

2. The Knowledge Base (KB).

This chapter will start by introducing the Web technologies used in this part of the research. RDFPIDM will be introduced next, explaining our new perspective of exploring novel uses for RDF technologies as a common data type system. The following section introduces the KB, our flexible facility to store information and knowledge.

6.2 Semantic Web Technologies

The information available on the Web today is largely for human consumption and needs a great deal of effort to be understood by computers (i.e. Artificial Intelligence, AI). A vision of the future Web – known as the Semantic Web [Antoniou and van Harmelen, 2004] [Berners-Lee, Hendler and Lassila, 2001] – was put forward to make locating and understanding information on the Web easy for machines as well as humans. The Semantic Web relies on machine-readable metadata and information expressed in the Resource Description Framework (RDF). Information published in the

Semantic Web has well-defined meaning which can be examined by both computers and humans. In the world of RDF anything in the universe (an entity) is considered as a resource, whether that resource is on the Web: XHTML page, video clip, mp3 song, etc. or in the real world: TV, washing machine, house, car, etc. Resources can be identified using URIs. The technologies used to realise the vision of the Semantic Web include: RDF, Resource Description Framework Schema (RDFS) [Brickley and Epinions], Web Ontology Language (OWL) [van Harmelen and McGuinness, 2004], and SPARQL, an RDF query language [Prud'hommeaux and Seaborne, 2006].

6.2.1 Resource Description Framework

RDF is the prime technology that the Semantic Web builds upon to achieve its goals set by the World Wide Web Consortium (W3C) [Berners-Lee, 1994] Semantic Web working group. The first published recommendation of the RDF data model specification and its XML syntax was made public in 1999. Primarily, RDF is a data model and it has various syntactic formats including XML.

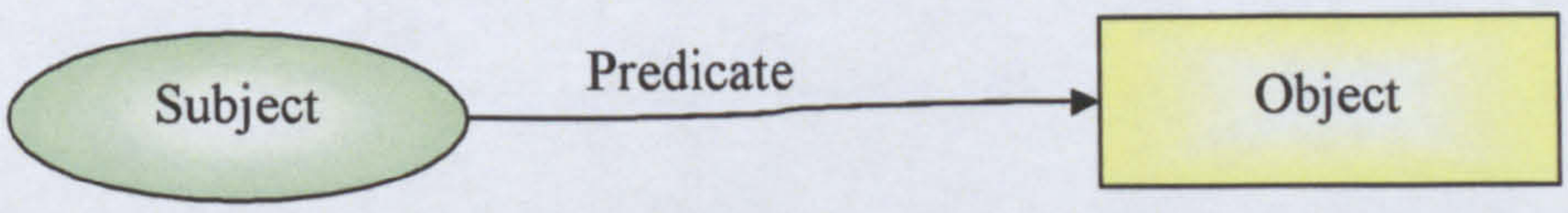


Figure 6-2: RDF triple graph, Subject – Predicate – Object.

XML does not provide any means of talking about meanings (semantics) of data. On the other hand, RDF is able to express semantics and meanings of data and can be used to talk and make statements about resources identified by URIs. The RDF statement (also called a triple) is the basic building block of RDF. It has three structural parts: the subject, the predicate and the object. The author's name, language, keywords, subject are an example of the sort of metadata that can be attached to a resource given its URI [Berners-Lee, Fielding and Masinter, 2005]. Directed graphs can be used to express

RDF as shown in Figure 6-2. Below is an example that presents four RDF triples that give information about two people (Tim and Ben). The XML syntax of RDF has been used.

```
<foaf:Person rdf:about=" http://www.url.com/ppl#Tim">
  <foaf:name>Tim James</foaf:name>
  <foaf:nick>tee</foaf:nick>
</foaf:Person>
```

```
<foaf:Person rdf:about="http://www.url.com/ppl#Ben">
  <foaf:name>Ben Adam</foaf:name>
  <foaf:nick>bee</foaf:nick>
</foaf:Person>
```

The Friend Of A Friend (FOAF) ontology has been used. FOAF is used to describe persons, their relationships and activities with others (see Section 9.4.1.1).

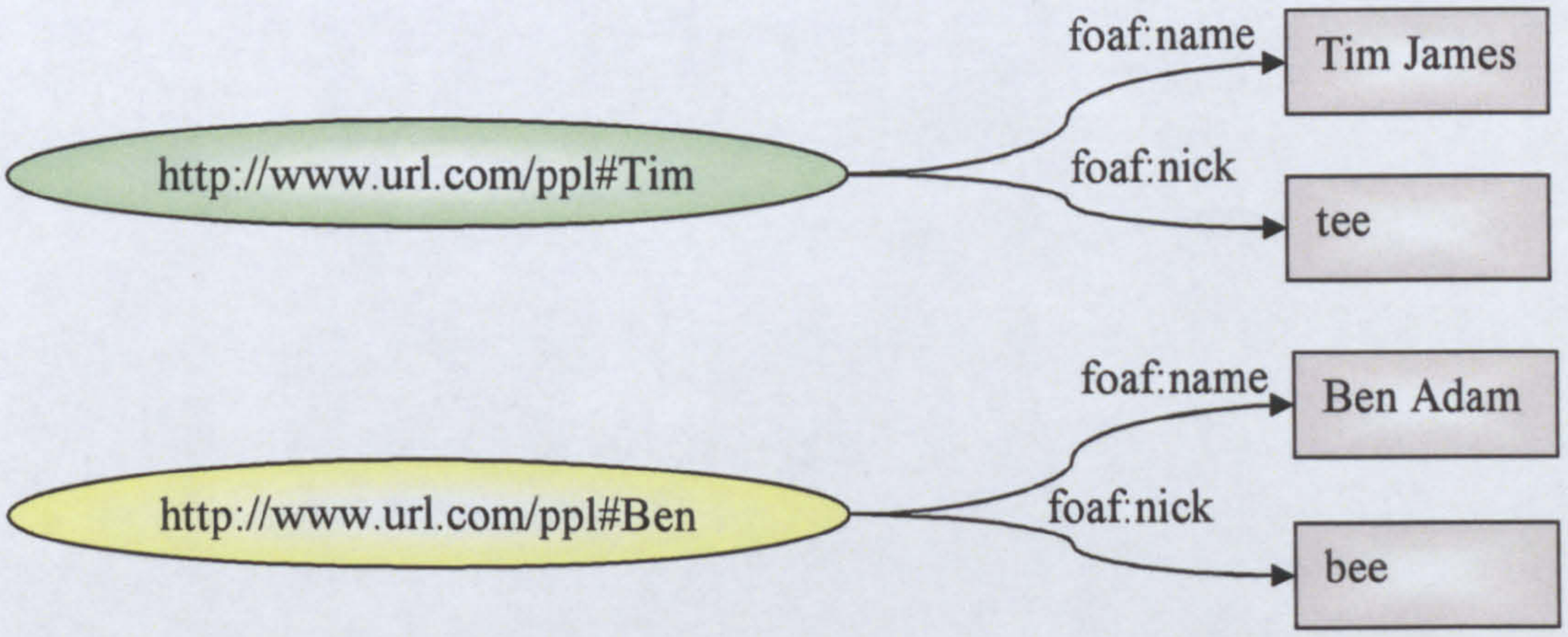


Figure 6-3: RDF Triples graph using Tim and Ben example

The example gives two pieces of information about each person, their names using *foaf:name* property and their nick names using *foaf:nick* property. Figure 6-3 shows the triples of the above RDF example as a graph.

An alternative language to the RDF's XML syntax used in the example above, is N3 (Notation 3 language). N3 is compact and readable. Below is the above example

expressed in N3.

```
@prefix foaf:http://xmlns.com/foaf/0.1/.

<http://www.url.com/ppl#Tim> a <foaf:Person>.
<http://www.url.com/ppl#Tim> <foaf:name> "Tim James".
<http://www.url.com/ppl#Tim> <foaf:nick> "tee".

<http://www.url.com/ppl#Ben> a <foaf:Person>.
<http://www.url.com/ppl#Ben> <foaf:name> "Ben Adam".
<http://www.url.com/ppl#Ben> <foaf:nick> "bee".
```

In N3, the RDF property *rdf:type* can be replaced by 'a' for clarity. The directive *@prefix* can be used to abbreviate repeated URIs by declaring short prefix names (as above, foaf). The Semantic Web Primer [Antoniou and van Harmelen, 2004] is a good source of information regarding RDF and its potential applications in the world of the Semantic Web.

6.2.2 Resource Description Framework Schema

RDFS is a representation language used to structure RDF resources. RDF users can define their own terminology which they can use in their RDF documents. RDFS provides basic elements (classes, properties, values) to create RDF vocabularies and ontologies. RDFS describes the relationships between classes of objects and restricts the properties used, the classes they can describe and apply to, and the kind of values they accept. The restrictions of properties in RDF are optional. If RDFS does not specifiy any restrictions on properties, then properties accept any value, otherwise the RDF environment should report a violation. For example, for (a p b) where *a* is the subject, *p* is the predicate and *b* is the object, the environment should report a violation if the *rdfs:range* property of the predicate *p* is defined and that *a* is not in its range. According to W3C, RDFS can be used to create lightweight ontologies, but with less expressive power than OWL.

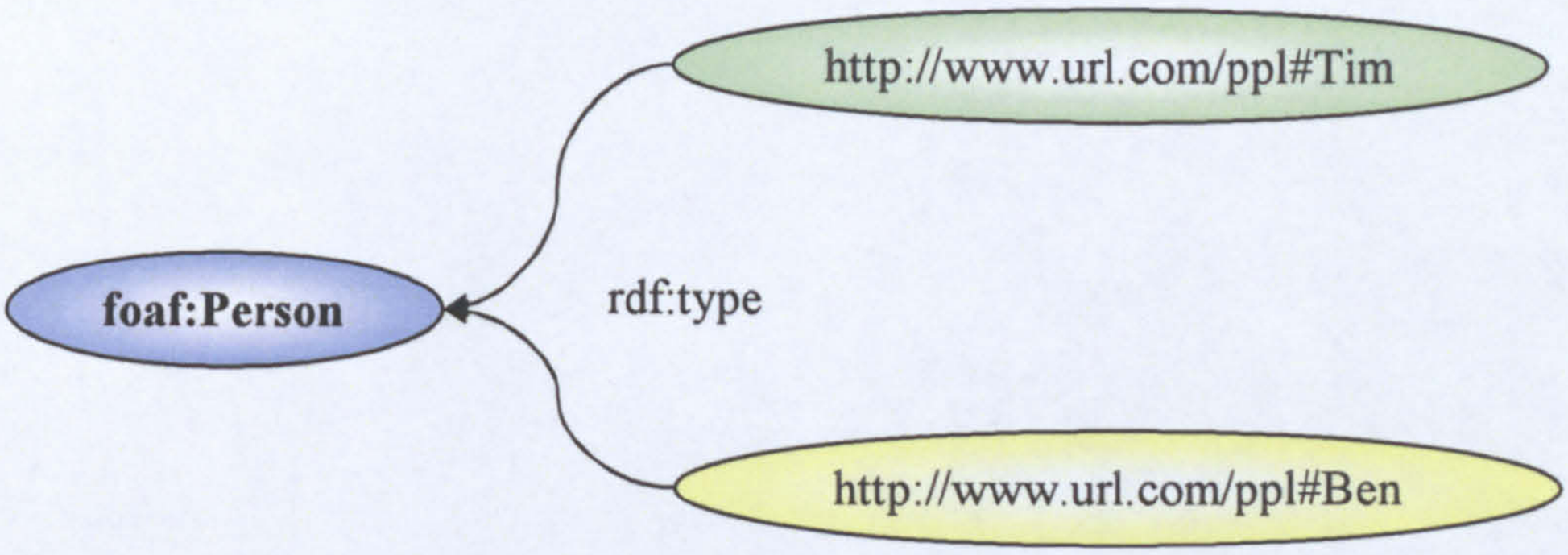


Figure 6-4: RDFS model for Tim and Ben example

In the RDF example presented in the previous section, Tim and Ben are of type Person (foaf:Person). Figure 6-4 shows the graph of the RDFS model of Tim and Ben example. Please refer to the Semantic Web Primer [Antoniou and van Harmelen, 2004] for more details.

6.2.3 SPARQL Query Language for RDF

SPARQL is an RDF query language and access protocol. It is a major part of the Semantic Web initiative. SPARQL stands for SPARQL Protocol And RDF Query Language. SPARQL is a very effective tool for querying RDF data models. SPARQL is a good pattern matching language to answer relationships-based queries, unlike XQuery [XQuery] used by other XML languages to find data in tree representations.

The name Queries can be expressed in SPARQL and applied to an RDF data source. The result of a query is returned as sets of variables or as an RDF graph. For clarity, SPARQL allows declaring prefixes and base URIs using the keyword PREFIX. These prefixes can be used later in the query. The names of query variables start with '$' or '?'. The query returns the match of all the variables listed after the keyword SELECT. Here is an example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?nick.
```

```
WHERE { ?person foaf:name ?name;
                foaf:nick ?nick.}
```

From the example above, one can see that SPARQL queries contain a set of triple patterns called a 'basic graph pattern'. These patterns are similar to RDF triples except that a variable might constitute the subject, the predicate or the object. This example will return tabular results with columns for 'name' and 'nick' variables (?name, ?nick). The result of the above SPARQL query if applied to the RDF example in Section 6.2.1 is presented here:

?name	**?nick**
Tim James	Tee
Ben Adam	bee

SPARQL borrows many keywords from the database query language SQL, such as SELECT, FROM, DISTINCT, WHERE, OFFSET and LIMIT, with slightly different usage. Our work was based on the first version of SPARQL which was designed to query RDF data (but cannot be used to modify or delete). This version of SPARQL was implemented in Jena version 2.3. SPARQL was extended in this research to allow insert, update and delete functionalities. Recent related work [Schich and Cyganiak, 2008] has addressed similar concerns although the SPARQL standard has not changed at the time of writing this thesis.

6.3 Platform Independent Data Model, Universal RDF Model

This research proposed to use RDF and RDFS technologies as the data model for the design and the implementation of ACTs. This means that these technologies will be used as design tools replacing traditional methods such as UML and to express interoperable data, information and knowledge for collaborative applications.

Concepts and notations defined by these technologies such as classes, properties,

relationships between classes will be used to express the design ideas. Using RDFS (and potentially OWL or other knowledge representation languages in the future) in the design stage greatly reduces the time spent to develop a design into an implementation following this approach. In fact, all the designs to be written in RDFS or OWL can be used in the implementation stage too as will be shown in Chapter 9. Furthermore, this approach uses RDFS to express the types and semantics of the application data instead of using the type system of the programming language used for the implementation. This decouples the logic layer of a programming language from the data and data type layer. This allows the use of several programming languages to write the application without worrying too much about data interoperability and inter-communication. It allows the mixing of data and information from different sources such as other collaborative applications, data sources, sensor information, simulations, etc. into a data repository (the KB) for retrieval later. For example, JavaScript and Java were the two programming languages used to implement the exemplar application CWE (see Chapter 9), the type systems of neither was used. Microsoft has followed a similar approach with its .NET platform by providing a common type system which can be used with all its programming languages (C++, C#, Visual Basic, etc.).

We have adopted the use of RDF and RDFS (as explained above) in the development of ACTs. Recent work from W3C has shown a broad recognition of similar ideas and their suitability for building computer software which reinforces our earlier views. For more information please refer to the Semantic Web Primer for Object Oriented Software Developers (http://www.w3.org/TR/sw-oosd-primer/). Our work and published papers (see Page ix) predates the work described in the W3C working group note on "A Semantic Web Primer for Object Oriented Software Developers" [Knublauch, Oberle et al., 2006].

6.3.1 Type Systems

Programming languages can be thought of as a tool that provides programmers with the following:

1. A set of concepts used when thinking about what can be done to solve a problem.
2. The means to specify actions (instructions) to be executed.

One of the fundamental concepts is that of type (a type system). The type system sets the rules for how data values and expressions are categorised into types and how types should interact. A type is the meaning of the collection of bits that represent a value in memory. There are two kinds of programming languages, typed and untyped languages. Some may argue that there is no such thing as an untyped (type-less) programming language (i.e. Lisp, Assembly, etc.) but if there were, then they would allow all operations (addition, multiplication, etc.) to be applied to any data of any type including text, integer, float, etc. Typed programming languages can also be classified into two categories; weakly-typed and strongly-typed languages [Bell, 2005]. In weakly-typed languages, the type of the data depends on its content, for instance '10' could be considered as an integer type or as a string type depending on the context. Strongly-typed languages restrict the value of data to its declared type. They also allow conversion between different data types (i.e. primitive arithmetic data type such as int, float, double etc).

Programming languages such as Java and C++ support the Object Oriented model of programming. This model fundamentally relies on the concept of *Class*, a user-defined type. In a programming language such as Java, most primitive data types such as *int*, *float*, *double*, etc. have equivalent class types: Integer, Float, and Double for convenience and to establish a unified programming approach.

6.3.2 Common Type System

Different programming languages have different data models and type systems. This variation is an obstacle facing programmers wanting to migrate their software (or designs) from one programming language to another. Programmers may also want to use several programming languages to develop an application. This makes the concepts used in the design of the system as well as the data produced by the application throughout its running lifecycle available to navigate, observe, investigate, analyse or query later on. The reward for using a single data model and type system that is shared across several programming languages is to make the ideas and data structures involved in the design and the information and data an application possesses over its running lifespan open and accessible.

RDF can be interpreted in different ways because of its flexible nature. Our proposed approach considers RDF as a generic data-model which can be used in any programming language to express the design of an application and realise its data structure. Mapping between RDF and the type system of the programming language at hand can be done systematically. For example, Jena (version 2.3) [McBride, 2002] provides a utility called schemagen [schemagen] (that converts an RDFS vocabulary into a Java class file containing static constants for the terms in the vocabulary). Jenabean is another very recent RDF-Java binding project [Cowan, 2008] that uses Jena and which aims to convert RDF properties to and from Java types. Jenabean bridges the gap between Java objects and RDF. However, in this research simpler facilities were developed to do the same job because Jenabean was developed only recently after our work was completed. Obviously, this approach works only on modern programming languages which have similar types to these offered by RDFS (or OWL).

The following sections introduce the RDF data model and describe the process of software design and the tools programming languages offered to developers to help

them to design a solution with the focus on our alternative approach using RDF.

6.3.3 RDF Data Model

The basic building block of RDF is a statement; an assertion about a resource. A resource can be anything (an object) and is identified by a URI. An RDF statement asserts that a resource has a named property with a given value. As described earlier, an RDF statement or triple has three structural parts: a subject (represents the resource) a predicate (represents the property) and an object (represents the value).

6.3.4 Software Design

The first stage a programmer goes through in order to solve a particular problem is to:

1. Gain a clear understanding of the problem,
2. Develop a consensus of the key concepts that seem to be part of the solution and
3. Choose a programming language to implement these concepts into a computer program.

Concepts are usually expressed as primitive data and functions in procedural/functional programming languages, or as related-classes in modern OOP languages.

In OOP languages, a class is a user-defined type with associated properties and methods. Usually, a good design results in a clean program with each concept represented by a single class. Related concepts are expressed by related classes, therefore, the more expressive the language is the more complex the concepts and relations between concepts that can be expressed. It helps a great deal for the programmer to be able to express the exact relationships between different concepts involved in the design. However, common programming languages at present have limited expressive powers which make the implementation of complex concepts and

relations between concepts difficult. At present, programmers would need to realise and enforce such complex concepts and relationships such as mutually exclusive, disjoints and other relationships in code. This is not an ideal situation and there is clearly room for improvement as will be shown in the following section.

6.3.5 RDF as Common Type System

In order to develop applications that use an RDF model to realise designs and as a data model, we are going to evaluate the RDF data model in this section against some of the facilities programming languages provide through their type systems to express ideas and concepts gathered in the analysis and requirement gathering phases. As explained earlier, RDF is a universal language used to describe 'anything' (resources), hence, our aim is to use it to express design concepts and describe the data and data types in the process of developing software. The default data type in RDF is *string* or *literal*. RDF has a very generic yet strong and extensible data model. By default, there are three basic data types that can be used in RDF: *Resource*, *Property* and *Literal*, here is an example in XML format and N3 :

```
<rdf:Description rdf:about="http://www.brookes.ac.uk/MusbahSagar">
    <ex:has-email>03186787@brookes.ac.uk</ex:has-email>
</rdf:Description>
```

N3 notation:

```
@prefix ex:http://www.brookes.ac.uk/
ex:MusbahSagar ex:has-email "03186787@brookes.ac.uk"
```

The statement above associates the property 'has-email' to the resource 'http://www.brookes.ac.uk/MusbahSagar' with the value '03186787@brookes.ac.uk' of type literal. In plain English it states: Musbah Sagar has an email address, 03186787@brookes.ac.uk. This example demonstrates the simple model which RDF is based on, however, to replace the type system of programming languages with RDF

technology we need more expressive powers to describe the concepts and relations between concepts from what RDF offers by default.

RDF is generic and does not make assumptions about the application or the semantics of a specific domain. However, the user can use RDFS to express these semantics. RDFS is considered as a basic ontology language that contains a subset of what is offered by its superior ontology language, OWL. We use RDFS here for simplicity as the following section will explain but OWL can be used in the future if needed.

6.3.6 *Data Modelling with RDFS*

RDFS presents a Class type to describe concepts. The definition of a class in RDFS - is a set of objects, where objects are instances of that class. There is also a Property type used in RDFS to describe attributes of/relationships between classes or concepts. RDFS supports multiple inheritance between classes and properties so that one class or property may have multiple superclasses/superproperties.

Classes and properties can be restricted in RDFS. The RDF parser ensures that the restrictions are imposed onto classes and properties in a similar fashion to type-checking in programming languages to prevent unintended use of the data model. Unlike other programming languages, properties of a particular class do not have to be defined locally; more properties can be added to an already existing class without the need to change that class. This is a powerful feature in RDFS and its data model.

As mentioned earlier, RDFS imposes optional restrictions on properties (using *rdfs:domain* and *rdfs:rage* properties). For example, it restricts the class type of those resources which could be used in the value section of an RDF statement. The value of a property can be a resource, a class or a literal in RDF. In case of a literal, the program reading the value cannot determine the actual type of that string if for example the value

is a number (1000456). However, RDFS supports a flexible typing allowing the use of any externally defined data typing schema which is sufficient for providing types for any literal value. The most widely used typing system in RDF is the XML Schema which defines a large range of data types such as *integer*, *float*, *boolean*, *string*, *time*, etc. For example, to define three variables following this approach: V1, V2 and V3 which are of types *integer*, *float* and *Boolean* respectively, three RDF statements are required to declare those variables and assign values to them, as follows:

```
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd:<http://www.w3.org/2001/XMLSchema#>.

ex:V1 ex:equals "10"^^ xsd:int.
ex:V2 ex:equals "10.5"^^xsd:float.
ex:V3 ex:equals "false"^^xsd:boolean.
```

Notice the property 'equals' connects the variable with its value. This is explained further in the XML Schema data types in RDF and OWL [XMLsdtRDFOWL].

Figure 6-5 illustrates the class diagram of an example. *Shape* is the superclass while *Circle* and *Rectangle* are its subclasses.

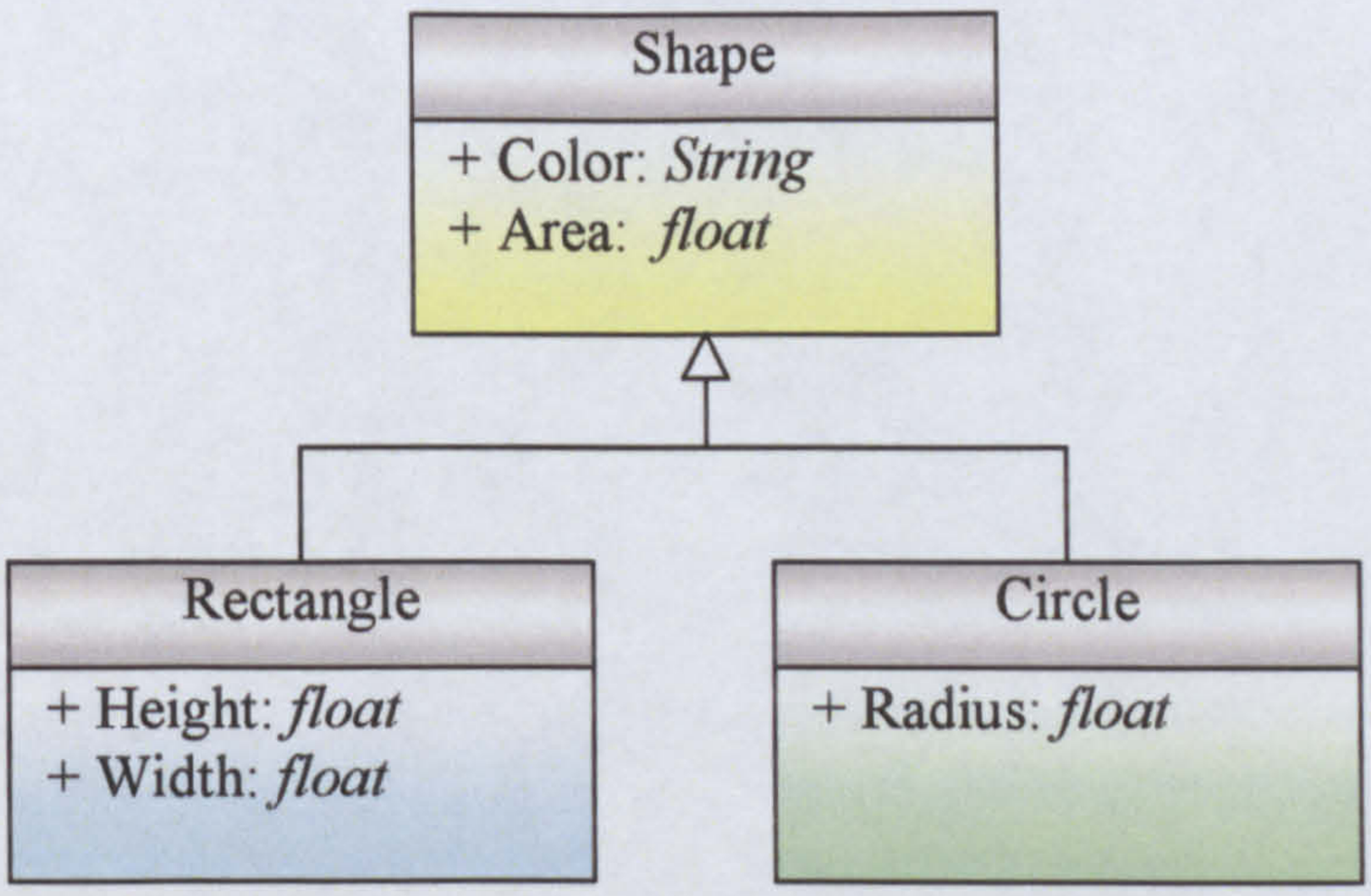


Figure 6-5: Class diagram of an example.

The shape class has two attributes, Colour (*string* type) and Area (*float* type). Circle has

one attribute, Radius (*float* type) and the Rectangle two attributes, Height and Width (both *float* type). This can be expressed in RDFS as follows:

```
ex:Shape rdf:type rdfs:Class.

ex:Circle rdf:type rdfs:Class;
rdfs:subclass ex:Shape.

ex:Rectangle rdf:type rdfs:Class;
rdfs:subclass ex:Shape.

ex:Color rdf:type rdf:Property;
rdfs:domain ex:Shape;
rdfs:range xsd:string.

ex:Area rdf:type rdf:Property;
rdfs:domain ex:Shape;
rdfs:range xsd:float.

ex:Radius rdf:type rdf:Property;
rdfs:domain ex:Circle;
rdfs:range xsd:float.

ex:Width rdf:type rdf:Property;
rdfs:domain ex:Rectangle;
rdfs:range xsd:float.

ex:Height rdf:type rdf:Property;
rdfs:domain ex:Rectangle;
rdfs:range xsd:float.
```

In this work we used Jena (version 2.3) [Carroll and Reynolds, 2004], a Java framework for building Semantic Web applications, to store and manipulate the RDF models created using this method. This method was further validated and used in the design of the CWE (see chapter 9).

6.3.7 *Related Technologies*

Another comparable system which might be used similarly to our approach described

above is a relational database system [Kim, 1979] [King, 1980]. In relational database systems, data is stored explicitly in predefined tables. The structure of these tables must be defined prior to use and once defined, it cannot be changed. Data is obtained from the relational database using a query language such as Structured Query Language (SQL). [Date and Darwen, 1997]. Relational database systems have a type system they use to ensure data is consistent independent from the programming language and they support mapping between the database and the programming language type system. Our approach to use RDF and RDFS to model data for developing collaborative applications (RDFPIDM) does not require us to define tables prior to use and it is more flexible in terms of defining new relationships and structures during the life-time of the application without affecting its operation. RDFPIDM also supports an Object Oriented approach which is not the case with relational database systems, although, a relatively newer system of database called Object Oriented databases [Kim, 1993] does support the Object Oriented approach. However, this is very different to a relational database. Object Oriented databases provide persistent storage for objects while objects are not loaded into memory until they are used. The system writes the changes into permanent storage when objects change. They also provide a mechanism to retrieve data/objects using a query language or simply by means of navigation which is more efficient because of the use of pointers. A major drawback of many Object Oriented databases is that they work in the context of the target programming language such as Java, C++, etc. and therefore are not interoperable. Another database management system is Object Relational databases; a half way system between Object Oriented databases and conventional relational databases. This database management system is very powerful, providing support to objects, classes and inheritance, in addition to custom data-types and methods. This research has not considered Object Relational databases because it is not a standard Web technology.

6.4 Knowledge Base (KB)

The KB is intended to store data, information and knowledge (or data for short, see Section 3.4) shared in a collaborative application. The data can be of different types such as annotations, commands, application data or external data from several sources like simulations, sensors, etc. ACTs require a reliable medium to enable collaboration and store data.

CoRDF proposes a flexible data storage architecture (the KB) aiming to accommodate the storage requirements of ACTs especially when run on devices with different capabilities. The KB can work in three modes: (1) Centralised, (2) Distributed and (3) Replicated. Figure 6-6 illustrates these three modes of operation.

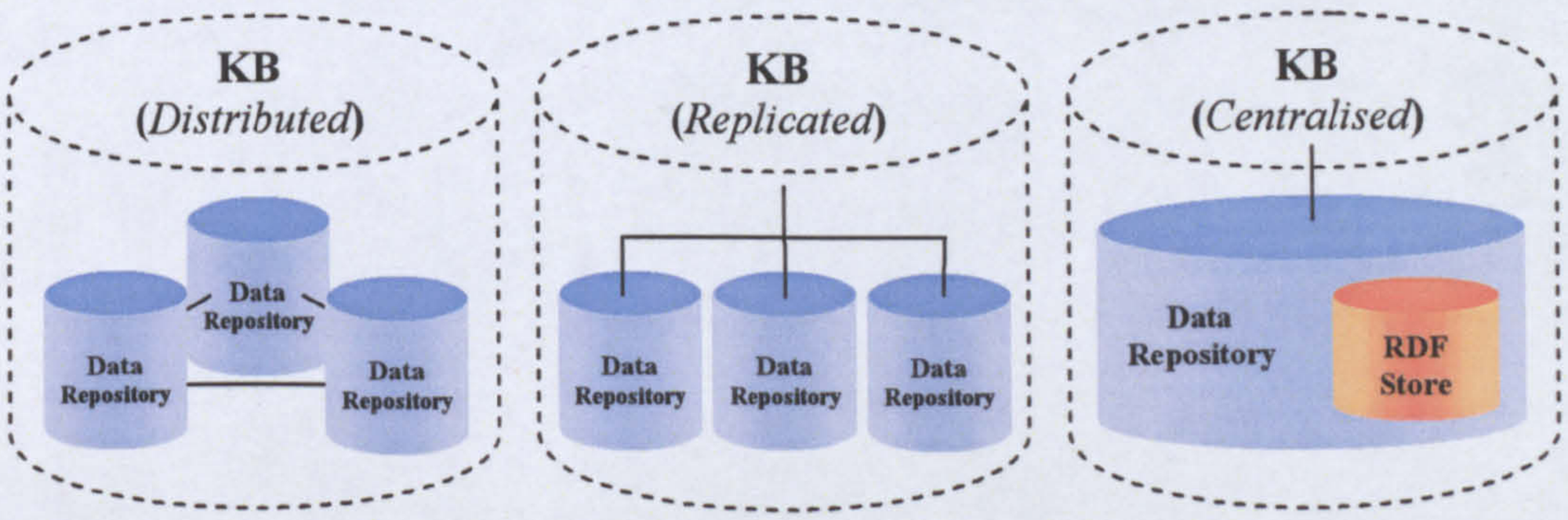


Figure 6-6: KB modes of operation.

The centralised mode is a special case where only one instance of the DataRepository (described in Section 6.4.2.2) is used to serve the application.

The KB is used to store data in RDF and RDFS. The data stored in the KB can be queried using SPARQL and the result of the query returned in XML format. SPARQL query results can take advantage of logic engines (if available, as with Jena). Entities of the application can also register with the KB to receive updates of a query that they are interested in following the Publish/Subscribe communication style.

The KB is designed to take into account the capabilities of the devices used to

run the collaborative application in terms of available processing power, memory, etc.

6.4.1 Design

The KB stores data from mixed sources so it can be shared by collaborative applications. Collaborative applications access the KB through GAI (see Section 5.3, Figure 6-7). They are granted access to group communication facilities via the Group Interface (GI, see Section 5.3.2) after successfully creating a group via the Group Management Interface (GMI, see Section 5.3.1). Each newly created group is associated to a KB. A reference to the KB can be obtained via the method *getKnowledgeStore* (see Section 5.3.2). The binding to the group can be achieved in several ways. The approach taken here is by creating an extra member in the group used as a proxy. The proxy routes the data exchanged in the group to the KB. Once the application has a reference to the KB, it can insert new data or update, delete or query all or part of the existing data.

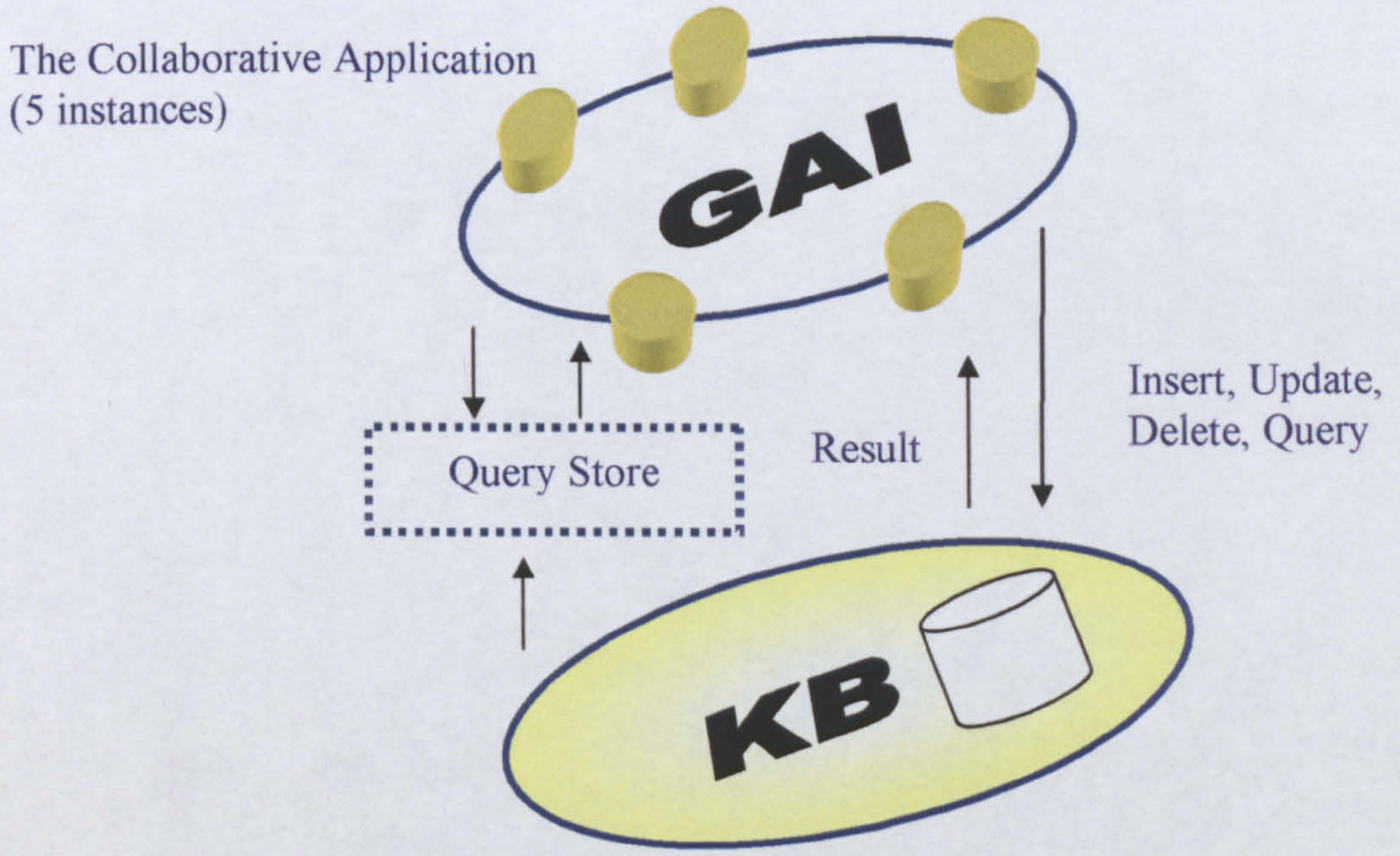


Figure 6-7: The use of GAI with the KB

The KB consists of a collection of one or more physical Data Repositories (DataRepository, see Section 6.4.2.2) that are hosted locally or remotely on other nodes of the collaborative application. Each Data Repository (DataRepository) has an RDF Store (RDFStore) to save the RDF content (see Figure 6-6). The KB is a logical entity so that if it is attached to a physical entity (Data Repository) it can store all communication between members of a group. However, if KB has not been attached to a physical entity then the data is discarded.

The RDFStore realises the mechanism that handles RDF data and queries. The RDFStore holds the RDFS of the collaborative application which describes the data types and relationships between them (see Section 6.2) as well as its RDF data.

There are two types of RDFStore, both built as OpenCOM components. One is used to implement the centralised and replicated modes of the KB (called Self-sufficient RDFStore) while the other one is used to implement the distributed mode of the KB (called Distributed RDFStore).

6.4.2 Implementation

This section explains the realisation of the KB. There are three components to the implementation of the KB: the RDFStore, the DataRepository and the KnowledgeStore. This implementation does not consider handling failures and does not provide acknowledgment for completing a certain action (i.e. insert, update, delete, etc.).

6.4.2.1 RDFStore

The RDFStore realises the mechanism that handles RDF data and queries by implementing the IRDFStore interface (see below) to allow data to be inserted, updated, removed and queried; as shown in the following code:

```
Interface IRDFStore {
   void insert(Model rdf);
   void update(Model rdf);
```

```
    void delete(Model rdf);
    String query(Query query);
}
```

The implementation uses Jena version 2.3 [McBride, 2002], a Java framework for building Semantic Web applications. RDFStore is built as an OpenCOM component within the Gridkit architecture.

For the Self-sufficient RDFStore, the GAI is used in the replicated mode to synchronise the data among RDFStores. Each node of the collaborative application will have a dedicated Self-sufficient RDFStore. The first RDFStore node created starts a common group using the GAI. Each time a new RDFStore is added to the KB it is assigned a name and then joins a common group. Transactions (inserts, updates, deletes) are replicated in all RDFStores by updating members of the group, while queries are done locally.

For the centralised mode of the KB, only one Self-sufficient RDFStore is used to satisfy the storage requirements of a collaborative application.

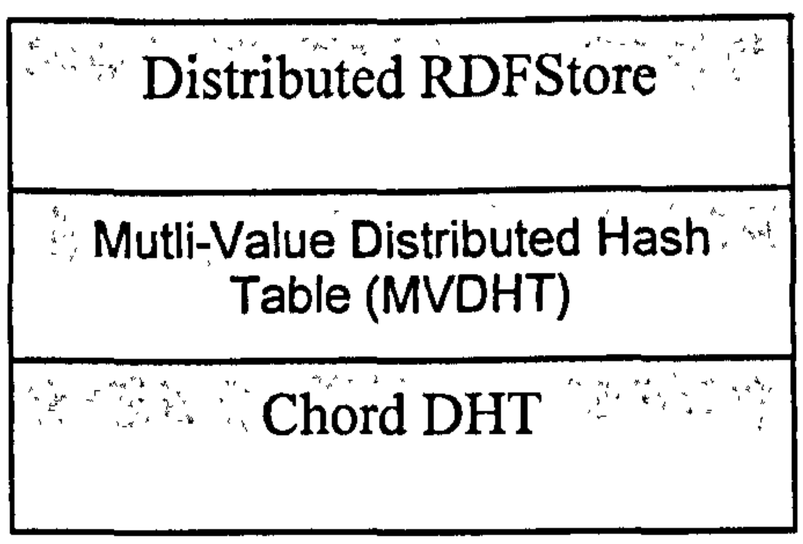


Figure 6-8: The architecture of the Distributed RDFStore.

Figure 6-8 shows the architecture of the Distributed RDFStore. The Distributed RDF Store is built on the idea of RDFPeers [Cai and Frank, 2004]. RDFPeers is interesting because it causes reconfiguration of the underlying overlay network. As this research is interested in reconfigurable applications, RDFPeers was a natural form of

reconfiguration to explore.

The Distributed RDFStore distributes RDF data over a number of nodes using the Multi-value Distributed Hash Table (MVDHT). MVDHT is built on top of DHT (see Section 4.1.3) and stores more than one value for each key (key → {Value(1), Value(2), ... Value(x)}). MVDHT allows the Distributed RDFStore to associate multiple RDF triples to the same key (i.e. object, predicate and subject). The Distributed RDFStore saves each RDF triple three times using the 'object', 'predicate' and 'subject' as keys, and the entire triple is saved as the value for each.

6.4.2.2 DataRepository

The DataRepository is simply a container for the RDFStore described above with the added functionality of registering queries. It implements the IRDFStore interface and allows listeners to receive the results of posted queries. It supports a form of the Publish/Subscribe interaction by allowing objects/processes to register as listeners to a particular SPARQL query. Every time the KB is updated, queries are executed and the result is delivered to registered listeners.

```
public interface IDataRepository extends IRDFStore {

//  listen to the query result
public void addListener(NodeInfo nInfo, String query);

//  stop listening to the query result
public void removeListener(NodeInfo nInfo, String query);
```

The definition above presents two methods on the IDataRepository interface; one to add a query listener to the KB and the other one to remove a query listener from the KB. Both methods accept the details of the node interested in the query and a string containing the RDF query written in SPARQL. Information about the interested node such as its address, name, etc. are passed on in an instance of the NodeInfo class.

6.4.2.3 KnowledgeStore

The KnowledgeStore is the software implementation of the KB. It holds the data, information and knowledge of the collaborative application. When the KnowledgeStore has only one DataRepository (either local or remote) this is considered as a *centralised* mode. The KnowledgeStore implements the IRDFStore interface and has only one method that controls the operational mode. The following describes the KnowledgeStore interface:

```
Interface KnowledgeStore extends IRDFStore {

// The KnowledgeStore works in three modes:
// centralised, distributed or replicated.
Void setMode(int mode); // 0:centralised, 1:distributed and
2:replicated
DataRepository createDataRepository();
}
```

The method 'setMode' changes the mode of operation (to centralised, distributed or replicated) causing a trigger to the reconfiguration process (see below). 'createDataRepository' explicitly creates a DataRepository.

6.4.3 Reconfiguration:

Reconfiguration happens when the KB changes modes, which results in a change to the architecture. This can be due to changes in the environment or the requirements (i.e. reliability, scalability, etc.). The centralised mode requires the centralised architecture type, while the distributed and the replicated modes require different settings of the distributed architecture type.

The collaborative application has to be in a quiescent state before the reconfiguration from one architecture type to the other takes place. This is handled by the Open Overlays middleware Gridkit (see Section 4.3.4). In this implementation the

quiescent state is realised manually. This means that, before the reconfiguration takes place data sharing/storing and other activities of the collaborative application must cease. This should prevent any conflict while the reconfiguration is taking place.

When the decision to change the architecture type of the KB is made, the running of the KB is stopped and reconfiguration begins. The collaborative application starts to run again right after the reconfiguration of the KB is complete.

Below are the algorithms which were used to maintain the consistency of the data after the changes of the overlay network supporting each mode was complete. Gridkit takes care of changing the overlays networks. The following steps describe the sequence of events that take place when the KB changes mode from centralised to distributed or replicated:

- A new overlay network is created. If the target mode is distributed, the type of the overlay network is Chord DHT; or group communication if the target mode is replicated.
- Self-sufficient RDFStore is used in the overlay network nodes if the mode is replicated or Distributed RDFStore for the distributed mode.
- The content of the centralised RDFStore is inserted into the new overlay network.

The following steps describe the process of changing the mode of the KB from replicated to distributed:

- A node is selected from the current group overlay network and its data is copied into a temporary RDFStore.
- The reconfiguration process takes place changing the overlay network type from group communication to Chord DHT.

- The content of the temporary RDFStore is inserted into the new DHT network.
- The resources of the old KB freed.

The following steps take place when the mode of the KB changes from distributed to replicated or centralised:

- A list of all the keys of the MVDHT is obtained.
- The data from the KB is retrieved using the keys obtained from the previous step and stored in a temporary RDFStore. Redundant RDF triples are eliminated.
- The reconfiguration process starts and changes the overlay network type from Chord DHT to group communication. In centralised mode, there will be only one member of the new group overlay network.
- The data stored in the temporary RDFStore is broadcast to all node(s)/member(s) of the group (the actual RDFStores, as described in Section 6.4.2).

6.5 Summary

CoRDF provides a novel approach that uses RDF technology to model collaborative applications and store data. This method provides two tools; an RDF-based data model that can be used to design and model collaborative applications and the KB, a flexible structure which can work in different modes depending on the context of the surrounds or the changing requirements. Chapter 9 will use CoRDF and the GAI described in Chapter 5 to build CWE.

7

Web-based User Interfaces: The Oea[1] Framework

There are hundred of millions of mobile phones that have SVG as their presentation layer in use today [Ferraiolo, 2008]. Also, an increasing number of Web browsers for desktop computers, laptops and hand-held devices provide support for SVG. To deploy collaborative applications onto a broad spectrum of devices SVG was a serious candidate technology. Also, SVG satisfies the precondition of using Web technologies in this research as described in Section 1.4, and therefore was chosen for this purpose. This chapter will present our approach - which is called the Oea framework - to developing Web-based user interfaces for ACTs using SVG.

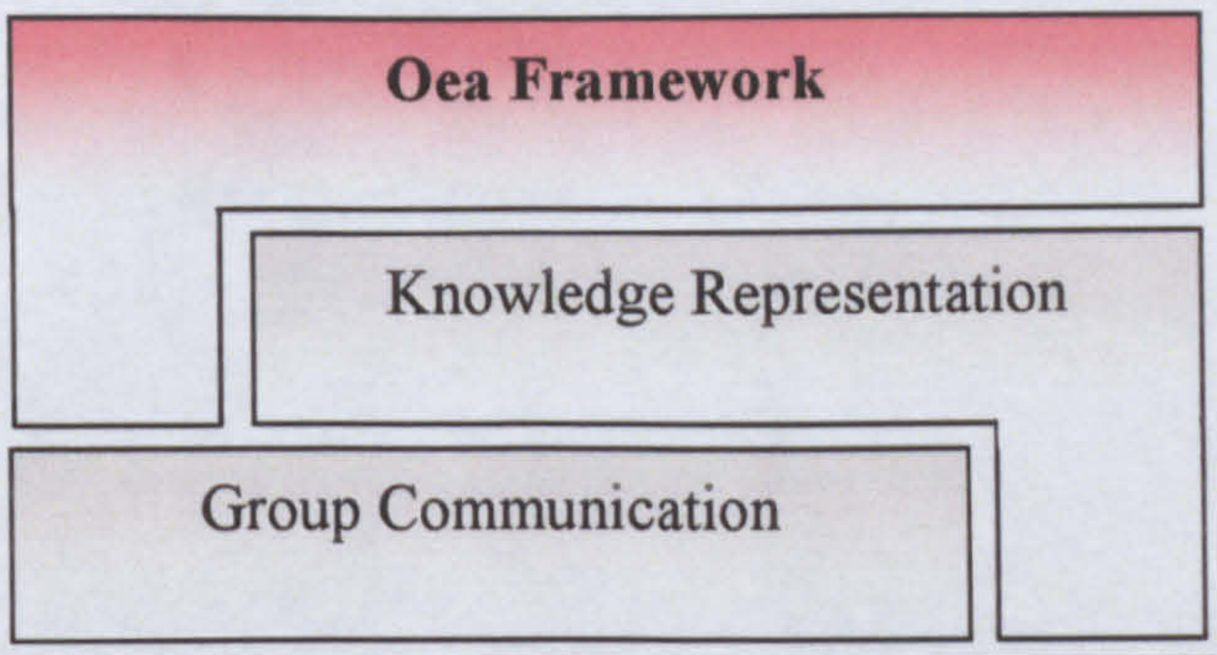


Figure 7-1: The four-layer model: Presentation and Interaction layer (Oea).

[1] Oea is the ancient name of Tripoli (Libya), a city founded in the 7th century BC by the Phoenicians.

This work constitutes the Presentation and Interaction layer of the four-layer model described in Chapter 3 (see Figure 7-1). The Oea framework goes beyond what is normally expected from a presentation layer (i.e. display of text, graphics, etc.). It provides support for Class-based programming in JavaScript, 2D graphics, GUI widgets and more (see Section 7.4).

This chapter will begin by introducing SVG and JavaScript followed by a discussion of the current widely used methods for developing Web-based user interfaces/Web applications, highlighting their strengths and weaknesses. The following section will introduce the Oea framework. The next two sections will describe two practical extensions to current technology of developing user interfaces using Web technology. These are: (1) the Class-based model for JavaScript, ClassBJS, and (2) a new mouse event model for DOM, domMouse.

7.1 Scalable Vector Graphics (SVG)

Scalable Vector Graphics (SVG) is a cross-platform, device-independent and open standard language based on XML and developed by W3C. It is based on a vector-based 2D graphics paradigm initially designed to provide an alternative to bitmap images (i.e. jpeg, png, gif, etc.) on the Web. SVG images are compact, scalable (zoom-in/out), can be animated and their embedded text and metadata can be searched. In addition, SVG allows simple vector shape drawing primitives such as circles, rectangles, polygons, raster images, path and simple text. SVG graphics can be styled using CSS (Cascading Style Sheet) to change their visual appearances by changing the colour, border width (by changing the stroke boundary), opacity, etc.

One of the distinctive features of SVG is that it has a declarative syntax. However, a scripting language such as JavaScript can be used to create SVG content dynamically. Vector graphics created in SVG can be static or animated using in-built

declarative animation elements. They can also be made dynamic and interactive using JavaScript the DOM APIs [Le-Hors, Wood et al., 2004]. It could be argued that using JavaScript to create/modify SVG content invalidates the advantage of the SVG declarative syntax. The declarative syntax of SVG is only helpful if it is written by users to be reused and understood by others. However, if the SVG content is highly interactive, computer-generated and it is not intended to be read by users, then using JavaScript exclusively to generate the SVG content can be justifiable.

Another Vector Graphics Language is VML (Vector Markup Language) [VML]. It was proposed by Microsoft in 1998 to W3C as standard vector graphics language for the Web. VML is an XML markup language for vector graphic. It defines a set of XML elements as basic vector shapes. The vital difference between SVG and VML is that, SVG separates presentation from content. Attributes to change positions, dimension etc are specified with the particular SVG element (using XML attributes), where presentation (such as colour, transparency etc) are set using CSS. In VML, CSS is used for both, presentation and content (change location, size etc). Microsoft still uses VML to date with its applications. VML is an outdated proprietary technology which did not become a Web standard, and therefore, it was not considered for this research.

In the early days of SVG, Web browsers failed to provide SVG support; instead SVG content was viewed via the use of plug-ins. The Adobe SVG Viewer [ASV3] was the leading plug-in for SVG. Nowadays, other SVG viewers have been developed by other vendors and organisations. Early versions of Firefox and Opera offered limited support for SVG while newer versions have considerable support for SVG. Other ways to view SVG are through stand-alone applications/viewers such as the Apache Batik SVG Toolkit (version 1.6) [Batik], Bitflash [Bitflash] and Opera Mobile [OperaMobile] for mobile phones & PDAs.

During the first years of this research, Adobe SVG Viewer 3 was the most

popular SVG viewer (currently not supported by Adobe). It supports Scalable Vector Graphics (SVG) 1.0 Specification, W3C Recommendation 4 September 2001. Batik was the second popular SVG viewer with support to the static features of Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation 14 January 2003. For the above reasons, this research adopts SVG 1.0 Specification mainly for implementations and testing, using Adobe SVG Viewer 3 and Batik only. However, at the time of writing this thesis Scalable Vector Graphics (SVG) Tiny 1.2 Specification has become W3C Recommendation in 22 December 2008 (see Open Research Issues and Future Work, Section 11.4).

7.2 JavaScript

JavaScript [Flanagan, 2001] is a prototype-based scripting language similar to the ActionScript language used with Adobe Flash. A version of JavaScript known as ECMAScript has been standardized by Ecma International (ECMA-262 specification) [Ecma262]. The JavaScript language is widespread with a copy of the JavaScript interpreter in all Web browsers used today. JavaScript is mostly used as a client-side scripting language for Web pages. It is embedded into HTML pages and uses DOM interfaces to make Web pages more interactive. The use of JavaScript has increased tremendously especially with the advent of other XML-based applications (such as SVG), and techniques for building highly interactive asynchronous Web applications such as Ajax (Asynchronous JavaScript and XML) [Garrett, 2005].

7.3 Web applications

A Web application is defined as a software application that uses a Web-based user interface to interact with the user usually via a Web browser. The benefit in developing (and using) Web applications is that they are pervasive and platform independent. Nearly all current computers have the capacity to run and interact with Web

applications ranging from very small computers in the form of mobile devices, to the very large. Web applications are based on the Client/Server software architecture. There are two parts to a Web application, the client-side (also known as the front-end) and the server-side (or the back-end). Typically, the client-side is a light-weight program which makes service requests to the server-side program. The server-side is a heavy-weight program and generally runs on a high-end computer. The server-side program fulfills the requests made by the client-side program by returning the data requested. The client-side program responds to the server-side reply by making changes to the user interface.

There are many standard and well-established methods for developing Web applications on both sides: client and server side. For example, HTML is widely used with CSS, DOM and JavaScript as a front-end interface for Web applications. Java has been used to develop Web applications also on the client-side and server-side. Perl, PHP, Java Servlets and Active Server Pages (ASP) are familiar programming languages to write server-side programs. Figure 7-2 shows an illustration of a few commonly used programming languages and technologies to develop Web applications.

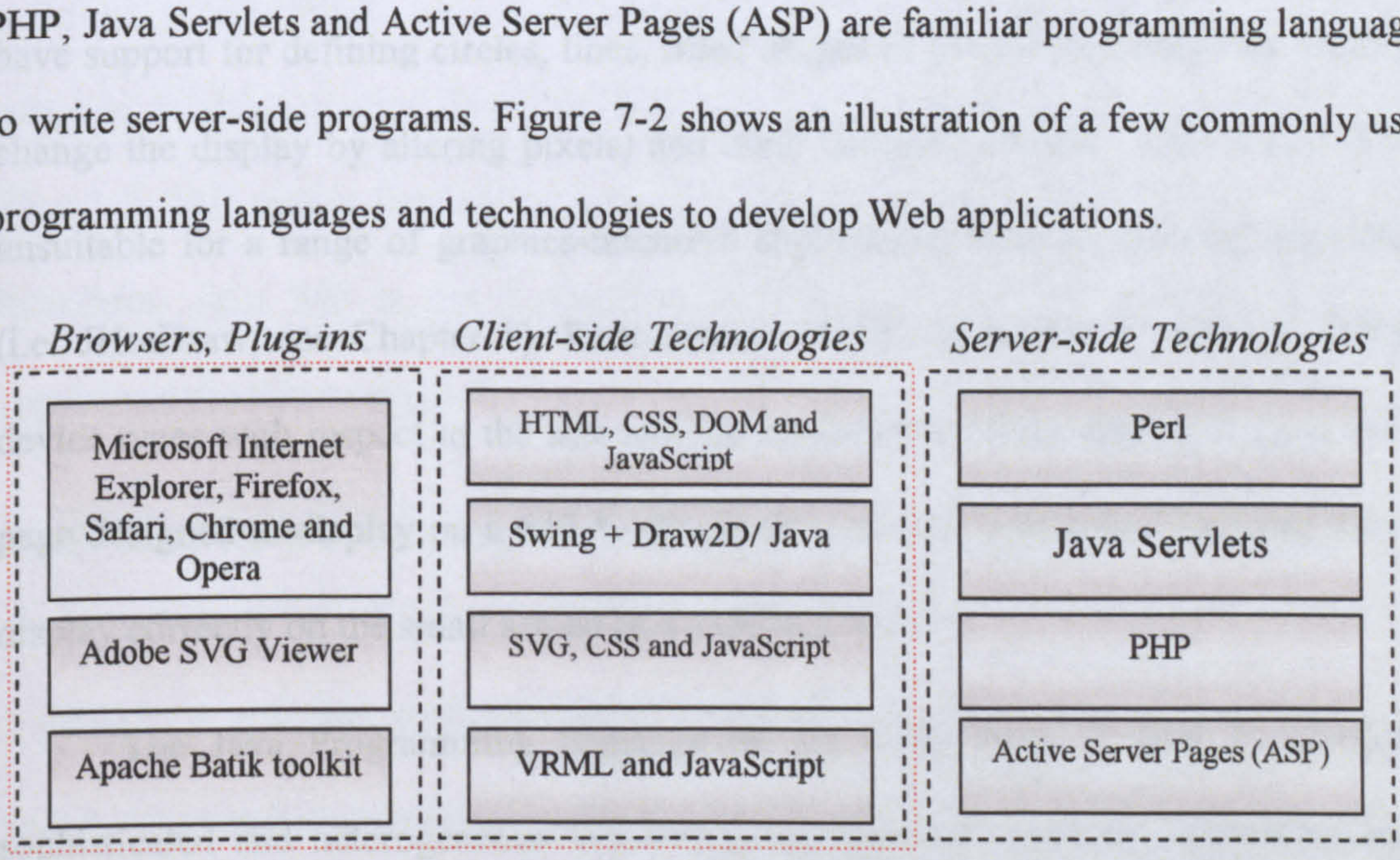


Figure 7-2: The server-side (right) and the client-side (left) components of a Web application.

It also shows some of the browsers available through which to view them. Microsoft Internet Explorer, Firefox, Safari, Chrome and Opera are used to display HTML and run

JavaScript applications. Adobe SVG Viewer and Apache Batik toolkit can render SVG documents. The Adobe SVG viewer runs as a plug-in while the Apache Batik toolkit also can run as a stand alone SVG browser known as Squiggle.

7.3.1 Traditional Methods to Develop Client-side Web applications

Client-side Web applications have two components: the presentation technology and the programming language. Virtual Reality Modelling Language (VRML, see Figure 7-2) and Draw2D/Swing (for Java) are examples of technologies used to present information and media to the user. JavaScript is used to add logic and interactivity to many Web technologies (i.e. HTML, VRML and SVG).

As mentioned earlier, the vast majority of Web applications have been developed using HTML with CSS, DOM and JavaScript as the front-end interface. Nevertheless, browsers displaying HTML are lacking in speed and HTML has not been designed as a markup language for complex graphics. For example, HTML does not have support for defining circles, lines, filled shapes or pixel-based drawings (ability to change the display by altering pixels) and other complex objects. This makes HTML unsuitable for a range of graphics-intensive applications such as drawing applications (i.e. JHotDraw, see Chapter 8). Furthermore, HTML is unable to adapt to different device types with respect to the size and the resolution of their display (e.g. an HTML page designed to display on a 640 * 480 pixels screen of a desktop computer does not display correctly on the small screen of a mobile phone.

The Java Programming language on the other hand is used in developing sophisticated and heterogeneous Web-based applications while not compromising on performance, graphics capabilities and interactivity. Any Java application can be wrapped in an HTML document and run in a Web browser. Java programs are usually compiled into a platform-independent byte-code format before they can be downloaded

via a browser to the client and interpreted by the Java Virtual Machine (JVM). A fixed sized segment of the browser window (e.g. 300 pixels width * 300 pixels height) is often assigned to the Java application for display. The size of this area cannot be changed once set. Similarly to HTML, Java also lacks the adaptability and scalability factors (being able to adjust to the size of the device display and being able to zoom in/out the display and pan in different direction for a suitable view). This makes Java an inadequate programming language for the task of building scalable and adaptable Web-based user interfaces.

However, Java ME [JME] is a different platform used to build applications for small devices such as mobile phones and PDAs. It was first introduced in 1999 by Sun Microsystems, and it was called Java 2 Micro Edition (J2ME). Java ME provides a modular and scalable architecture to support application to run on different types of devices with various capabilities and resources. It includes security and network support and flexible user interfaces. Java ME provides Scalable 2D Vector Graphics API which supports Scalable Vector Graphics (SVG). A recent release of this API supports SVG Tiny 1.2. This makes Java ME potentially suitable to build scalable and adaptable user interfaces. Java ME is not a standard Web technology and therefore, it was not considered for this research.

7.3.2 SVG for Developing Web Applications

In recent years, SVG has been perceived as a possible host environment for Web applications, powered with JavaScript and DOM. However, developing applications with SVG is still in its infancy. Applications written using SVG are accessible, scalable and adaptable. The recent SVG 1.2 Full (working draft) [Ferraiolo, Duce et al., 2005] shows a tendency for moving towards facilitating SVG for building Web applications. Moreover, the commercial sector has larger corporations investing in developing faster SVG viewers that incorporate compiled languages such as C# (C-sharp) [C-sharp] to

support potentially large and complex Web applications using SVG [Emia]. JavaScript has also been made many times faster with the recently launched Chrome Web browser from Google. This will contribute significantly to increasing the performance of future SVG viewers.

SPARK [Fettes and Mansfield, 2004] is an SVG project to define standards for developing Web applications and a GUI framework for SVG. SPARK uses XML, SVG and Java to establish the development of flexible SVG applications. A set of rules and techniques for creating SVG user interface widgets are offered by SPARK to assist SVG developers. In this model, SVG represents the *data* and the *view* while JavaScript represents the *control* in the well-known Model View Control (MVC) model (see Section 8.2.2). Although SPARK provides a framework in which developers can build SVG widgets, it does not however resolve the issue the Oea framework has addressed (i.e. adaptability and scalability, see Section 7.4).

Another recent approach that uses SVG is the Lively Kernel [Taivalsaari, Mikkonen et al., 2008] developed by Sun Microsystems. Lively Kernel provides a Web environment to build applications using the JavaScript language and the graphics available on ordinary Web browsers. This system is based on a graphical framework called Morphic which was originally developed for the Self [Maloney, 1995] in 1995 and Squeak systems [Ingalls, Kaehler et al., 1997]. A JavaScript implementation of Morphic has been written specifically for the Lively Kernel. The Lively Kernel provides a set of GUI widgets and utilities that can be used to write applications for the platform. On the negative side, Morphic is an outdated graphics system which dates back to 1995.

7.4 Oea Framework

As explained earlier, current methods for building Web applications lack either graphics

capabilities or the adaptability to accommodate various device types. The Oea framework has been developed to build adaptable and scalable Web-based interfaces that can be used to create Web applications and ACTs. It addresses many of the challenges developers experience when building SVG applications.

The Oea framework overcomes the weaknesses of using HTML, CSS, DOM and JavaScript for developing Web applications and the adaptation/scalability inadequacy of Java. It proposes the use of SVG as the technology to develop future Web-based interfaces. It is envisaged that applications' interfaces (including Web applications and ACTs) developed with the Oea framework are as powerful in graphics and interactivity as those developed in Java while still maintaining the edge through being adaptable and scalable (see Section 8.6). Furthermore, the Oea framework can be used to port fully fledged applications written in other programming languages (e.g. Java) into SVG and JavaScript making them adaptable and scalable in the process. We have successfully demonstrated the generic nature of the Oea framework by porting JHotDraw (see Chapter 8). This is very useful and its implications can be profound in the Web community as the usage of SVG gains wide acceptance. The Oea framework can bring applications from other programming languages into SVG.

The Oea framework provides libraries with reusable classes and utilities packaged as:

- *ClassBJS*: a new Class-based approach to supporting Class-based OOP in JavaScript allowing programmers to add functionalities and support interactivity,
- *svgDraw2D*: a lightweight 2D graphics package to generate graphics and drawings,
- *domMouse*: an advanced mouse event model for DOM (to develop SVG applications) that resolves an out-of-sync problem (see Section 7.6), and supports an elegant set of mouse events that makes the mouse event handling process

straightforward,

- *svgSwing*: a GUI framework that provides Oea framework applications with reliable, stable and interactive user interface widgets (i.e. textbox, button, window, etc.) and finally,

- *Ajar*: a package for RDF that supports interactions with external knowledge storage facilities (i.e. KB, see Chapter 6) using RDF and SPARQL as a query language.

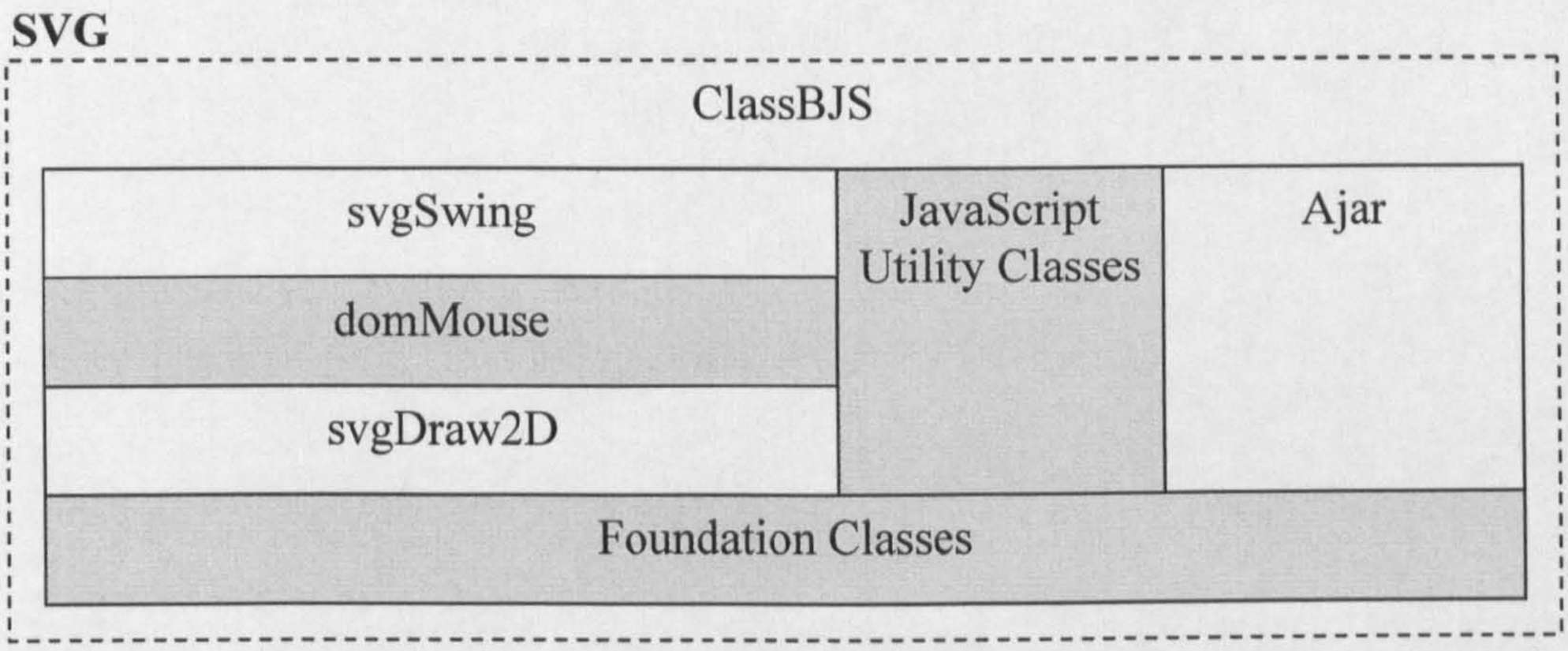


Figure 7-3: The architecture of the Oea framework.

Figure 7-3 depicts the Oea framework architecture. The Oea framework is written in JavaScript following the ClassBJS. The work on the ClassBJS has been published [Sagar, Duce et al., 2008] and is described in Section 7.5 (or see Paper A for more detail). The bottom layer of the Oea framework is a package called the Foundation Classes. The Foundation Classes provide JavaScript encapsulations of SVG elements. This enables programmers to easily access the DOM interfaces related to a particular SVG element through JavaScript code.

7.4.1 2D Graphics for SVG (svgDraw2D)

The svgDraw2D package provides a higher level of abstraction for JavaScript developers in an SVG environment to manipulate graphics independent from the DOM

interfaces. It supports capabilities for drawing sophisticated 2D shapes, working with fonts, text and text layout, controlling colours; and in addition it features layering management, styled tooltips and a desktop object.

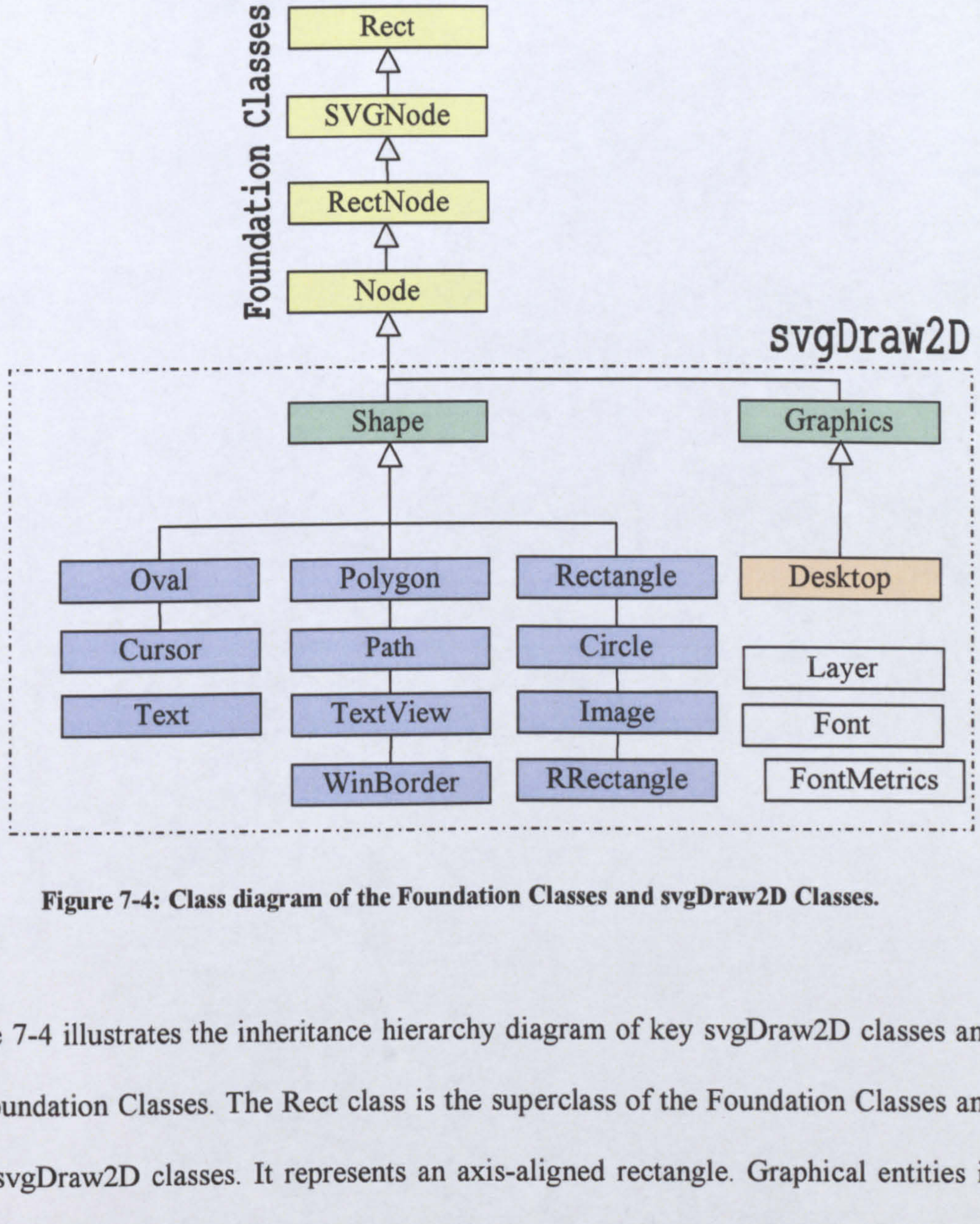


Figure 7-4: Class diagram of the Foundation Classes and svgDraw2D Classes.

Figure 7-4 illustrates the inheritance hierarchy diagram of key svgDraw2D classes and the Foundation Classes. The Rect class is the superclass of the Foundation Classes and most svgDraw2D classes. It represents an axis-aligned rectangle. Graphical entities in svgDraw2D (i.e. shapes such as rectangles, circles, etc. and graphics objects) are bounded by a rectangular area. The Rect class also provides interfaces to change the width/ height, rotate, scale and to translate the graphical content.

The SVGNode class is used as a wrapper around an SVG node (its DOM tree element). This class provides access to the SVG node corresponding to any Shape or

Graphics instances in svgDraw2D. SVGNode provides convenient methods to return the actual SVG node, set/get/remove the SVG node attributes, set/get the Id, add/remove DOM event listeners, set visibility and opacity, set/get the cursor, set/get the tooltip text and finally to dispose of the SVG node from the SVG document permanently.

The RectNode class is responsible for applying all changes and transformations - made on the rectangular area defined by the Rect class - on the SVG content. For example, when the user invokes any method on the Rect class, for instance: *obj.rotate(45)*, RectNode will change the properties of the SVG node referenced by the SVGNode class accordingly.

Finally, the Node class is used to maintain a list of internal (local within the inheritance hierarchy of the class) and external listeners for DOM Level 3 Mouse Event Model mouse events. When a DOM Level 3 Mouse Event Model mouse event is received the class notifies all listeners of that particular event type. Subclasses of this class could have separate event handlers for any DOM event.

The main two classes for svgDraw2D are Graphics and Shape. The Graphics class acts as a graphical container that can be used to generate lines, images, rectangles, ovals and other drawing primitives (Shapes) to be drawn on its space. All drawing methods of the Graphics class return Shape objects (Rectangle, Image, Text, TextView, Oval, Path; for other shapes, see Figure 7-4). The Shape class provides an interface to help manipulate the corresponding SVG primitive. The Graphics class is represented by the group element 'g' in SVG and the SVG drawing primitives that correspond to Shape instances are contained within the SVG group element 'g' of the Graphics object that generated them.

Figure 7-5: The use of Graphics class to generate Shapes of different types.

Figure 7-5 shows a Graphics object represented by the white background with a few Shapes (text, lines, etc.) drawn on it.

The Graphics class contains methods for drawing, colouring and font manipulation. Additional methods are supported by the Graphics class such as clipping, tooltip, cursor manipulation, DOM events handling, and coordinate transformation (scale, rotate and translate). The Graphics class permits external listeners to handle DOM events that originated from within the Graphics content; and for internal handling of events that originated elsewhere in the SVG document.

The svgDraw2D package provides a layering feature (Layer class, see Figure 7-4). All Graphics objects have to be associated with a layer. A default layer is used if the user has not specified one. Layers have a z-order property that is used for controlling the display order. The SVG representation of a Layer object is an SVG group element 'g'. The group elements that represent any Graphics object associated with a Layer object are contained within the group element of that later object.

The Cursor object provides an interface to the system cursor. The cursor glyph can be changed to any entry in the cursor default list supported by SVG/DOM (i.e. crosshair, move, e-resize, etc) or can be set to a Shape or Graphics object. Font and Font Metrics classes provide methods and constants for font control.

The Desktop class is a key class in svgDraw2D. There is one instance (object) of the Desktop class that svgDraw2D uses. The Desktop object continuously listens to all DOM events that occur within the SVG document (mousedown, mouseover, mouseup, mouseout, mousemove and click). It maintains a list of listeners that it notifies whenever an event is received. JavaScript objects can act as event listeners by registering themselves with the Desktop object. Listeners should provide a callback method that the Desktop invokes to notify the object of an event.

7.4.2 Graphical User Interface for SVG (svgSwing)

A Graphical User Interface (GUI) allows users to interact with computer applications. Java has an advanced Graphical Interface provided by Sun Microsystems called Java Swing [Elliott, Eckstein et al., 2002]. The Java Swing package is extensible and flexible with a rich set of widgets. An equivalent library to Java Swing (called svgSwing) was developed in this research to provide a similar level of richness and flexibility to SVG applications. svgSwing was built on top of svgDraw2D and domMouse.

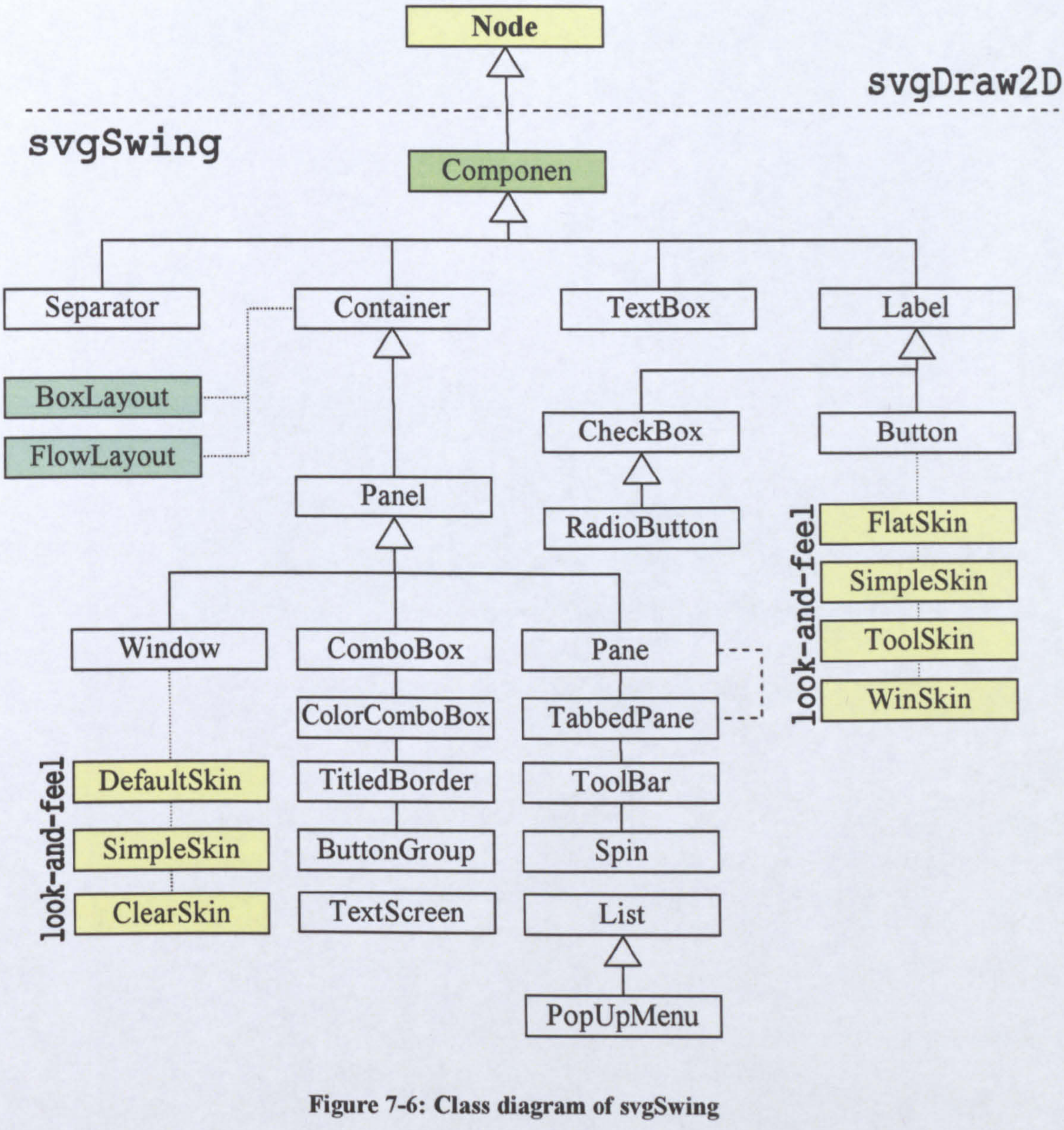


Figure 7-6: Class diagram of svgSwing

svgSwing features a collection of classical GUI widgets such as Windows, Buttons, RadioButtons, CheckBoxes, Lists, TextBoxes, ComboBoxes, Icons, Menus and PopUpMenus, Panels, TabbedPanes, etc. (see Figure 7-6 for the class diagram) in addition to various layout managers such as BoxLayout and FlowLayout. Also, svgSwing supports a pluggable look-and-feel style (see Figure 7-7) similar to those in the Java Swing toolkit. The default look-and-feel theme is similar to that of Microsoft Windows.

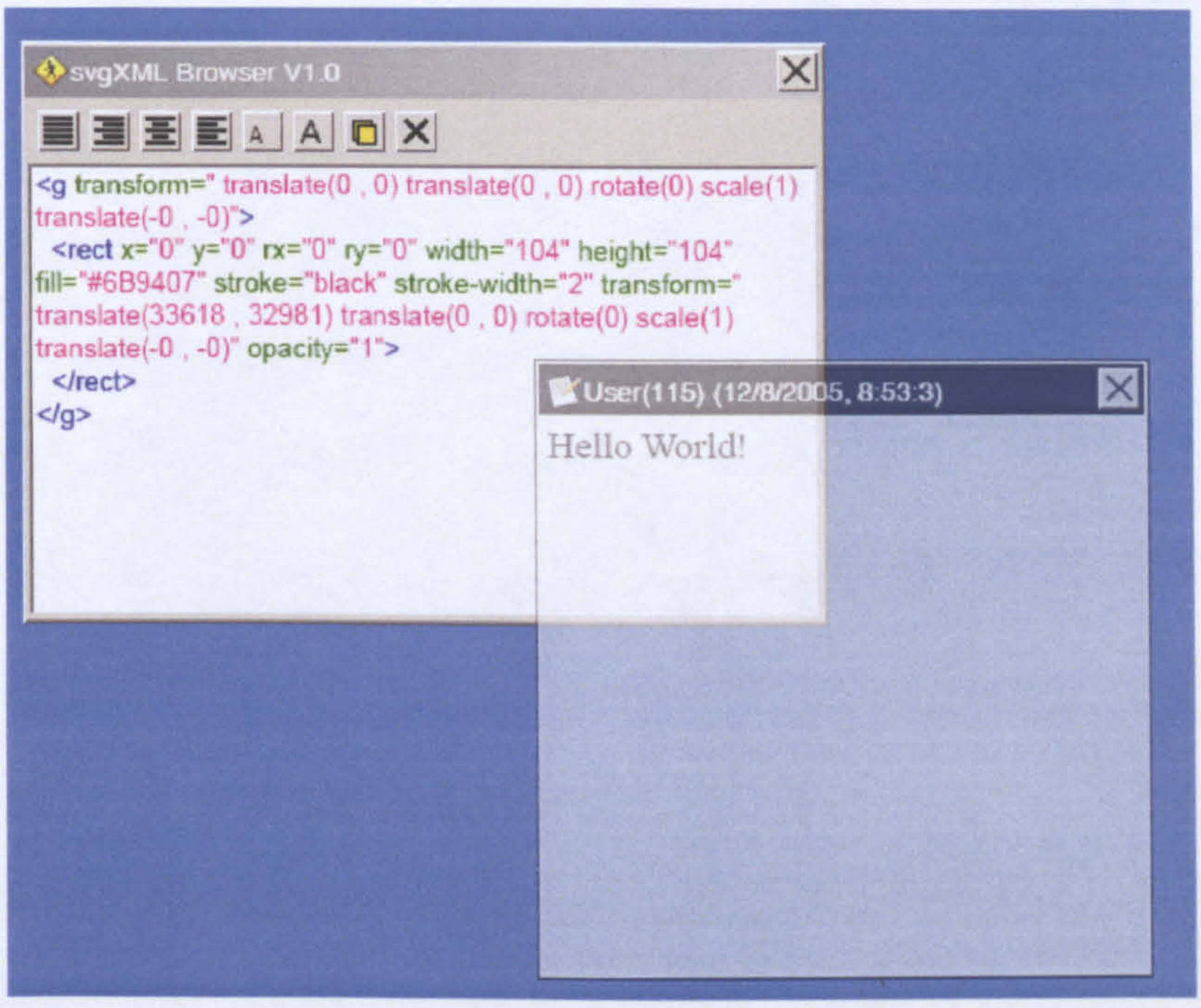


Figure 7-7: Two windows with different look-and-feel.

svgSwing is extensible and Web developers can easily use its components and design patterns to develop more complex widgets. The difference between svgSwing and the many other SVG-based GUI toolkits provided in systems such as SPARK and Lively Kernel mentioned earlier (see Section 7.3.2) and other SVG GUI packages such as KevLinDev [Lindsey, 2000] is that svgSwing has been built independently from the DOM interfaces, using svgDraw2D and domMouse. The result has been a well-designed, reliable, extensible and robust set of GUI widgets.

Particular attention, time and effort were paid to the development of svgSwing to ensure a satisfactory user experience. The following subsection describes the TextBox widget to highlight this point (also, see Appendix I for a gallery of svgSwing GUI widgets).

7.4.2.1 TextBox

Figure 7-8 shows 5 components of type TextBox. The TextBox is used to aid the user to enter text. This component features a cursor. The cursor is a thin rectangle – which was

implemented as an SVG rectangle shape - that can be moved across the text to indicate the position to perform an insert or delete operations. The TextBox has two modes of operation; (1) type mode, and (2) insert mode. To switch between the two modes, the 'insert' key is used. The cursor in the type mode is opaque with a fixed thin width, while in the insert mode it is transparent and takes the width of the character beneath it. The mouse cursor takes the 'text' shape when the mouse hovers over the TextBox editing surface (SVG 1.2 Full working draft supported by Adobe SVG viewer version 6). The widget cursor moves to the left, right, up, down, 'home' and 'end'. Also it aligns itself to the letter nearest to the mouse click spot. Text selection is supported; segments of the text are selected by moving the cursor using the keyboard arrows, 'home' and 'end' while pressing on the shift key, or by dragging the mouse cursor across the required fragment of text. A mouse double-click on a single word marks it selected; all the text is selected by pressing Ctrl-A.

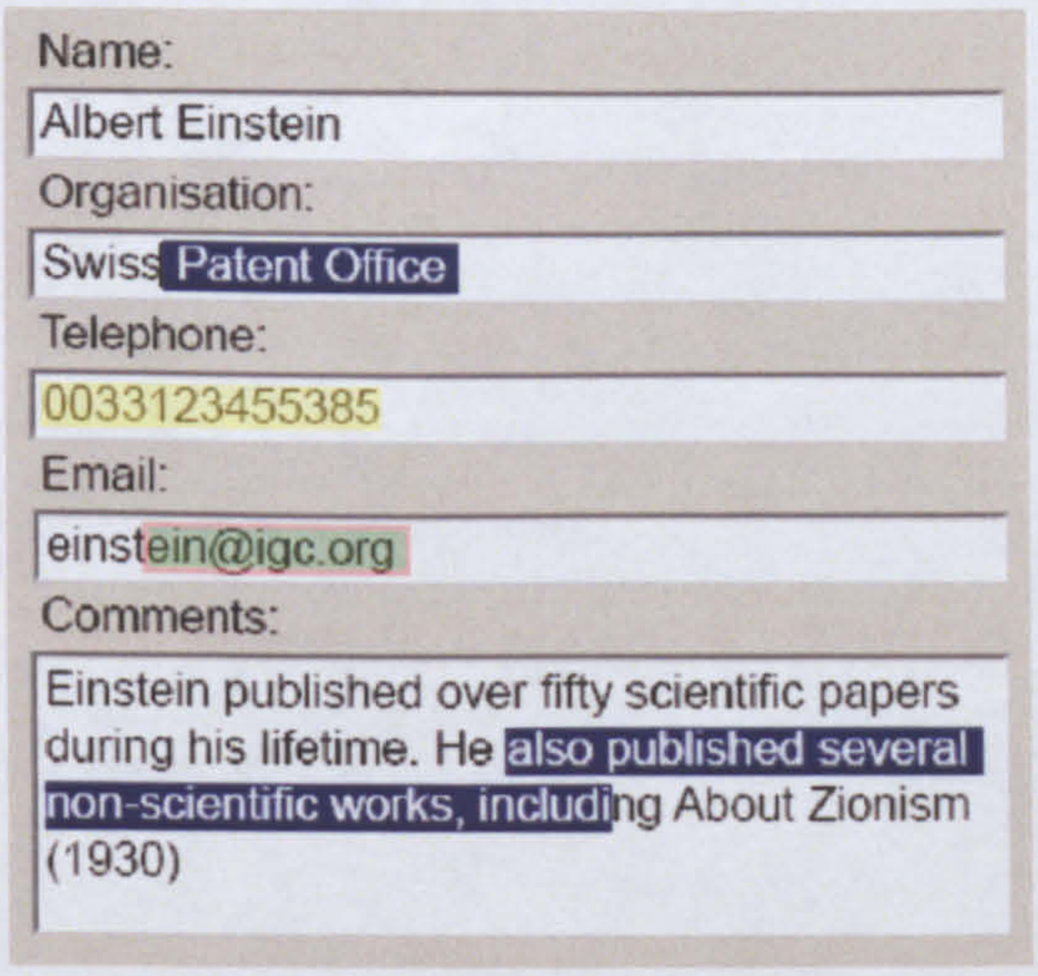


Figure 7-8: TextBox in different formats and styled selection rectangle

The TextBox widget can be configured to work in single or multi-line forms. The multi-line form of the component uses new features suggested in the proposed SVG 1.2 Full. The TextBox widget can be used to input a single line of text, where it uses a simple SVG text element that can be used in the Batik and ASV viewers; the component resizes itself to fit only a single line of text. This component cannot deal with an amount

of text that does not fit into its working space so it cuts the extra text out. A warning message is provided to the user when this happens. TextBox supports a styled selection rectangle in the single line mode. The colour, stroke colour and stroke width attributes of the selection rectangle can be changed. TextBox supports a simple implementation of a clipboard; Ctrl-C is used to copy the selected text, Ctrl-X/Shift-Delete to copy and cut and Ctrl-V/Shift-Insert is used to paste. The copied text can be either used locally, with other TextBox objects, or with text editors external to SVG (only with Adobe SVG plug-ins). If the widget is in the 'insert' mode, the newly typed characters are inserted into the text from the cursor position otherwise they replace the selected text or the following characters in the text. This component supports some sort of 'focus' mechanism independently. The technique to support focus (gaining control and receiving keyboard events) is a simple one and easy to implement. At first, the component does not have focus and therefore the cursor is hidden and the widget does not listen to the keyboard events. If the user clicks on the component the component gains focus. When the user clicks anywhere else or on the desktop the widget loses focus. Therefore, for several TextBox objects, they all start with no focus, but when the user clicks on one TextBox all the TextBox objects receive click/press events from the desktop which causes them to lose focus (including the one that the user has clicked), but after the same component receives another mouse click event this one originates from the component itself and that causes it to gain focus and start to listen to keyboard events. The widget supports password mode where the component hides what is being entered and displays the star '*' character.

7.4.3 Asynchronous JavaScript and RDF (Ajar)

The purpose of Ajar is to allow Web-based interfaces developed following the Oea framework to interact with RDF services (i.e. KB) following a simple interaction model. It also allows for some basic representation of RDF triples in JavaScript. Ajar

mediates between Web-based interfaces and the RDF services to insert, update, delete or query all or part of the data stored. Ajar's interaction model is inspired from the well-known Ajax. The additional elements added here to the Ajar's interaction model are RDF and the use of SPARQL to interact with RDF services. There is a similar library from the Decentralized Information Group at Massachusetts Institute of Technology also called AJAR (http://dig.csail.mit.edu/2005/ajar/ajaw/js/rdf/).

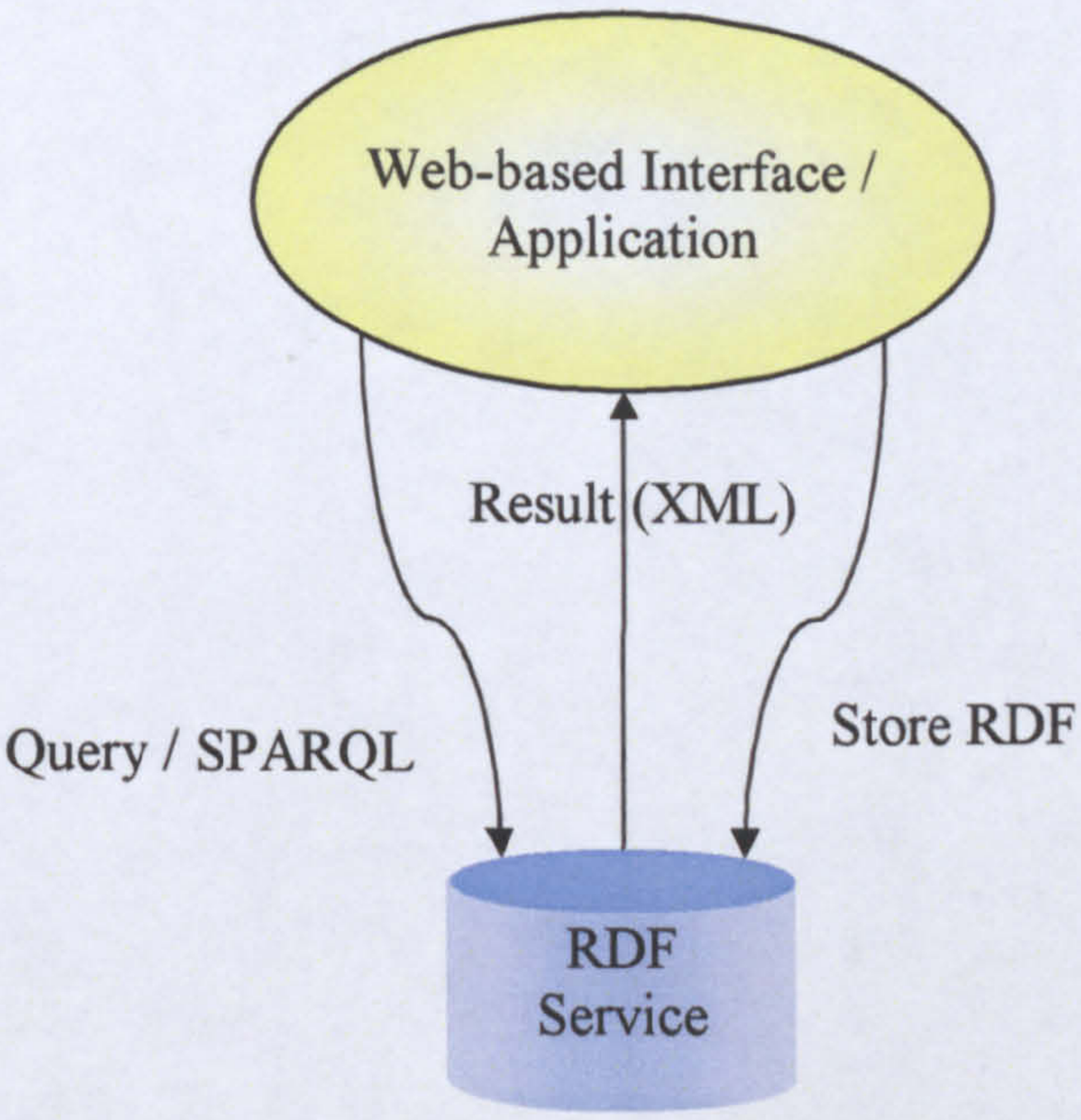


Figure 7-9: The Ajar interaction model.

As shown in the Figure 7-9, the communication starts from the application side. The application stores data in the RDF Service and queries it using SPARQL. The RDF service queries the underlying data storage and sends the results to the application in XML format. The Ajar interaction model consists of the following elements:

1. Standards-based presentation using XML-based languages (e.g. SVG/CSS, XHTML/CSS, etc).

2. Dynamic display and interaction using DOM.

3. Data interchange and manipulation using RDF and XML.

4. Asynchronous data retrieval using pull methods (i.e. XMLHttpRequest).

5. JavaScript binding everything together.

The Ajar interaction model provides two ways to interact with an RDF service:

1. By submitting queries and getting back results,

2. By means of registered queries.

With registered queries the application is notified each time a match occurs after a successful update of an RDF Service.

Ajar consists of a few basic classes. There are three types of RDF nodes in Ajar; a blank node, a URI node (a resource) and a string literal node. The class RemoteRdfStore is used to interact with a remote RDF Service. It implements the IRDFStore interface which supports insert, update, remove and query interactions. The IRDFStoreClient interface has to be implemented for those wanting to use RemoteRdfStore in order to receive callback replays from a remote RDF Service. The functions provided by this interface allow applications to communicate with RDF services (such as the KB, see Chapter 6.4) by submitting queries and receiving a reply. ResultBinding and ResultSet classes map variable names to values of a SPARQL query which are returned in XML format. This is used in this project in the development of CWE, see Chapter 9.

7.5 Class-based Object Oriented JavaScript (ClassBJS)

This section is based on Paper A, entitled: The Oea framework for Class-based Object Oriented style JavaScript for Web Programming [Sagar, Duce et al., 2008]. ClassBJS is an approach to supporting the Class-based object oriented model in JavaScript.

The OOP methodology comes in two flavours:

1. The Class-based programming model, which is very well-known and

adopted by popular programming languages such as C++ and Java, and

2. The Prototype-based programming model (used in JavaScript) [De-Meuter, D'hondt et al., 2003].

The prime focus in Class-based and Prototype-based languages is objects. However, the way objects are created by the two models is fundamentally different (see Section 7.5.1).

In general, the Prototype-based model is less familiar than the Class-based model. Due to the broad adoption of the Class-based approach by wide-spread programming languages such as Java and C++, developers believe the Prototype-based object oriented approach is inferior to the Class-based one. This has had implications on the ECMAScript standard itself. The fourth edition of ECMAScript (under development within ECMA), for example, suggests changing JavaScript from a Prototype-based to a Class-based language [ECMAScript4]. ActionScript version 2 (and higher) is already an example of an ECMAScript-based language following the Class-based model. This confirms the strong trend among developers to pursue a Class-based model and avoid the much misunderstood Prototype-based model.

However, the use of the Prototype-based method has become widespread due to the extensive usage of Web browsers. Web browsers come equipped with JavaScript interpreters which uses the Prototype-based method. This research recognises the significance of the concerns described above. It aims to provide JavaScript developers - and especially newcomers - with a familiar programming setting to boost their confidence in programming in JavaScript and this could lead to increases in productivity, efficiency and the quality of their work. It will also make it easy to borrow ideas and solutions from other Class-based OOP languages (i.e. to port programs from other Class-based languages to JavaScript). The new approach may also help

programmers to discover the potential of the Prototype-based model after they have gained familiarity with the JavaScript environment.

7.5.1 Class-based vs. Prototype-based

In the Class-based OOP model, a class is the encapsulation of the attributes and functions required to define the behaviour of its instances (objects). In most programming languages these behaviours are defined before compilation time and cannot be changed at run-time. Attributes of a class are used to store the state/data and its methods (functions) are used to change the state/data, perform tasks and to communicate with other classes' instances (objects). The behaviour of a class can be reused through inheritance. An object is created by instantiating the class that defines the attributes and methods it must have and which cannot be changed.

On the other hand, the Prototype-based Object Oriented model is class-less. Objects can be created (1) from scratch, (2) by cloning other objects and (3) from a template called an object prototype. Creating objects from scratch is tedious; using an object prototype is more common where objects are initially created to match their prototypes. Objects can be modified at run-time. Methods and attributes can be added or removed. Reuse of code in the form of inheritance in the Prototype-based model is achieved through delegation (also called prototype inheritance). The delegation mechanism allows an object to pass messages to another object (the delegator) in case it does not know how to handle them itself.

Inheritance is a powerful mechanism in the Class-based model that promotes reuse of code. The class that is being inherited (reused) is called a superclass. The derived class (called a subclass) inherits the attributes and methods of the superclass. New methods and attributes can be added to the subclass to change its behaviour and functionality.

JavaScript offers Prototype-based inheritance which has many differences to Class-based inheritance. As mentioned before, the fundamental difference between the Class-based and Prototype-based models is that in the Prototype-based model, the methods of the superclass can not be accessed explicitly/directly from its subclasses (this will be explained further below).

In Prototype-based languages, new objects can inherit attributes and methods from other objects. The inherited object (parent object or delegator) in JavaScript is associated to the child object by a link (attribute) called 'prototype' coupled with the name of the object constructor. An 'object constructor' is the function used to create an object in JavaScript (this will be referred to as a class in the context of JavaScript). When a method is invoked or an attempt to access an attribute is being made to an object, JavaScript looks for that method or attribute within the object itself; if the method or attribute is not found the link to the delegator is followed and the availability of the attribute or the method is examined. If it is found, the appropriate action is performed, otherwise the link to the delegator of the parent object is followed again and the processes are repeated until the top object in the object hierarchy is reached. If the attribute or the method being sought is not found, the 'undefined' value is returned.

The major concern with Prototype-based inheritance is that if the child object has a method or an attribute with the same name (identifier), the method or attribute of the parent object can not be reached. For example, if the parent object has a method that is called 'toString' and the child object has a method with the same name 'toString', the child object can only access its own 'toString' method. This is because JavaScript does not have the notion of 'super' found in Class-based OOP languages such as Java.

7.5.2 Requirements

The previous section has shown the difference between Class-based and Prototype-

based models. Inheritance in the Class-based model allows the calling of overridden methods and access to overridden attributes of the superclass including the class constructor. This feature is required to be able to follow the Class-based model. This research has identified further requirements to be satisfied by ClassBJS:

1. To have the class notion (class definition) and be able to initialise instances of classes (objects) at the time of creation (class constructor).
2. To be able to have public, private and static methods/attributes of a class.
3. To obtain information about the class type (e.g. class name).
4. Having an easy and clean syntax.

7.5.3 Implementation

Originally, two classes were implemented to support the Class-based Object Oriented model for JavaScript (ClassBJS): 'Object' and 'Class'. The Object class provides extra services such as support for serialisation, cloning and version control. The class Class is used to support classes in a running JavaScript program. As shown in the source code in Appendix II, and for simplicity, class Class was eliminated and its methods were added to the class Object as static methods (Appendix II, see lines 166-207).

To achieve Class-based properties in JavaScript, classes (or object constructors) need to call the static method of the class Object 'initClass' (Appendix II, see line 174). The rest of the approach is a set of conventions that are followed in order to support different aspects of the Class-based model and the requirements identified in Section 7.5.2. See below for an example.

```
1 /****************** Class: Point2D ***************************/
2 function Point2D(x,y){
3  /* public */ this.x = 0;
4  /* public */ this.y = 0;
5  var _super = Object.initClass(this,"Point2D");
```

```
 6
 7  this._constructor = function(x,y){
 8          this.x = x;
 9          this.y = y;
10   }
11
12   if(arguments[0] != "inherit")  this._constructor(x,y);
13
14 }
15 /************************ Class: Point3D ***************************/
16 Point3D.prototype = new Point2D("inherit");
17
18 function Point3D(x,y,z){
19  this.z = 0;
21  var _super = Object.initClass(this,"Point3D");
22
23  this._constructor = function(x,y,z){
24        _super._constructor.call(this,x,y);
25   this.z = z;
26  }
27
28   if(arguments[0] != "inherit")  this._constructor(x,y,z);
29
30 }
```

To write a class in JavaScript with ClassBJS all the attributes supported by the class are placed at the start of the class definition (lines 3:4). It is a convention in ClassBJS, to have a method that is called _constructor to initialise the class properties (lines 7:10). To avoid calling the class _constructor when the class is being used as a delegator for inheritance (line 12) the class constructor is only called if the first parameter passed to the class does not match the string 'inherit' (lines 12, 28). This also means that whenever an object is being created for delegation (line16) the message "inherit" must be passed as a parameter. To access overridden methods of the superclass the special operator _super must be set when the object is being created (lines 5, 21) by calling Object.initClass and passing the object itself as a parameter (this) and the name of the class as a string. The _super operator will then refer to the prototype attribute of the

object constructor (class definition) of the object type itself which holds a reference to the delegator. Calls can be made to overridden methods, for example as shown in line 24. The call to Object.initClass method will also add an attribute called _className that holds the name of the object type. This information can be obtained by invoking the 'getClassName' method on the Object class. As from the example above, one can notice that JavaScript code written following ClassBJS is elegant and has a strong resemblance to Java syntax.

As discussed in Section 7.5.1, the object constructor has a special attribute called prototype that JavaScript uses to implement prototype inheritance. Generally, other attributes and methods can be attached to the object constructor to do different things. When thinking about the Class-based model, the object constructor can be regarded as the class definition. Other attributes and methods associated with an object constructor can also be regarded as attributes and methods relating to the class. Thus, from Object Oriented methodology it is known that the static attributes and methods are members of the class and not members of the instance; therefore, the attributes and methods associated with the object constructors are indeed static members of the class (Appendix II, see line 174). Private methods and attributes can also be used in ClassBJS as described by Douglas Crockford [Crockford]. Methods defined inside the object constructor can be treated as private, hence they cannot be seen from outside the object but are accessible from its methods (see example, lines 7, 23).

7.5.4 Performance Evaluation

There are a number of related approaches to Class-based Object Oriented and classical inheritance for JavaScript, many of which use built-in features of JavaScript to simulate the Class-based model inheritance. Most of these methods are mainly focused on achieving support for classical inheritance. A number of these methods were considered

(Kevin [Lindsey], ThinWire [Gertzen], Base [Edwards], Sugar [Crockford]) in a performance evaluation with ClassBJS. These methods have been thoroughly tested in the majority of browsers that support JavaScript, such as Microsoft Internet Explorer, Firefox, and the Adobe SVG viewer. Each of these browsers has its own JavaScript engine.

We have attempted to measure the performance of ClassBJS, against the other methods in order to assess which approach performs best running in different browsers. Two major browsers were used: Microsoft Internet Explorer 6, and Firefox 1.5 with the Adobe SVG viewer version 6. The specification of the machine used to carry-out the tests was: Intel Pentium CPU 3.46GHz with 2GB of RAM. The aim was to measure the time required by each approach to make a call to an overridden method. Times were averaged over 10,000 runs. Also the test was designed to make calls to two overridden methods; again for less biased results. The test measures the time of execution of an overridden method of class ColorPoint3D. The results are shown in Figure 7-10.

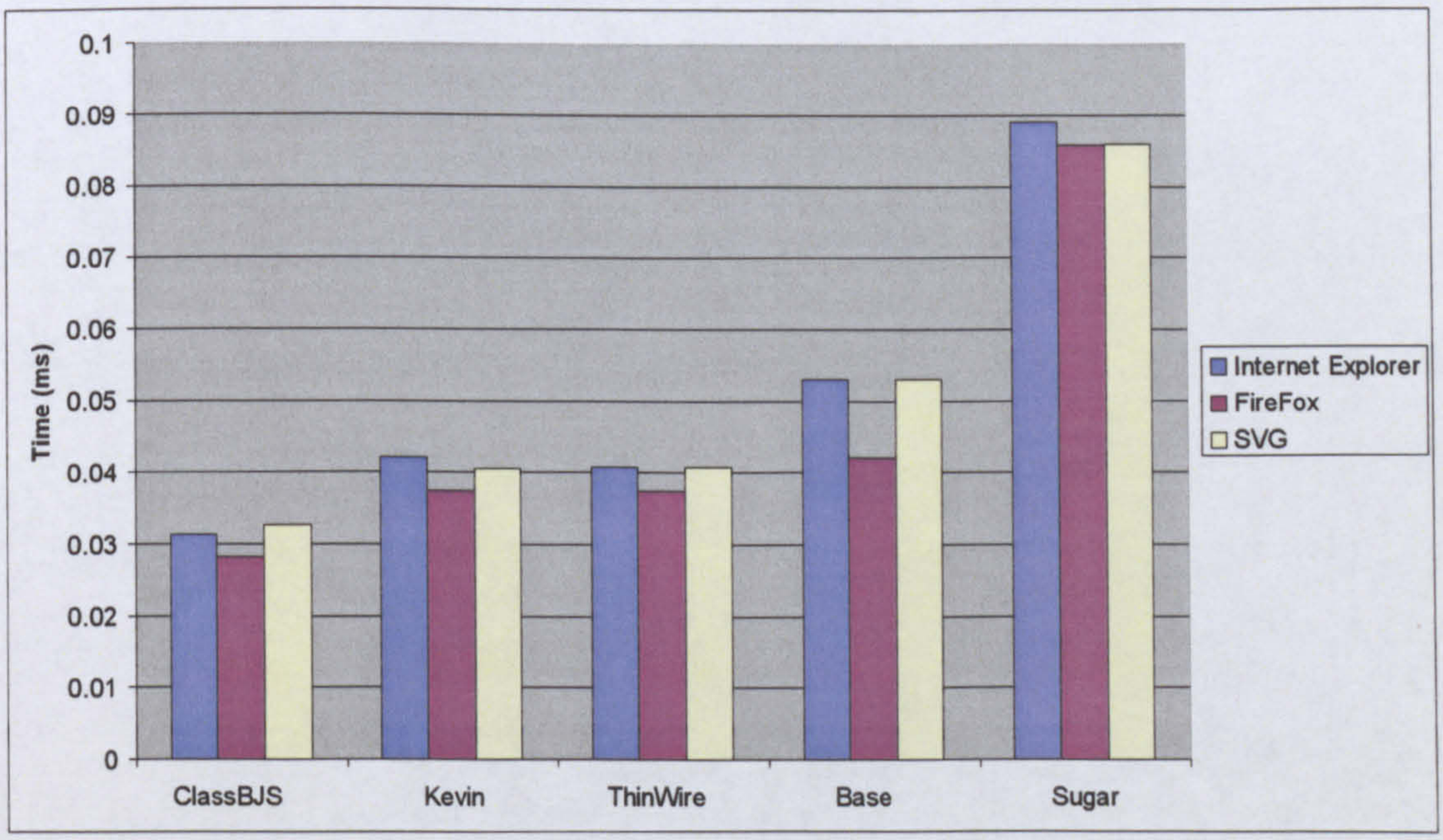


Figure 7-10: A results diagram of the performance test.

ClassBJS implementation out performed all the other implementations. Kevin and Thinwire performed about the same and Sugar was the worst. The tests are presented in further detail in Paper A.

7.6 Advanced Mouse Event Model for DOM (domMouse)

The work described here has been published in Paper B, entitled: Advanced Mouse Event Model for SVG [Sagar, Duce and Cooper, 2005]. Initially, this model was designed for SVG. However, this section provides a generic outlook to our approach and shows it to be suitable for any environment that uses DOM (hence the name domMouse).

Event processing code is at the heart of the majority of interactive graphical environments. Applications with Graphical User Interfaces (GUI) are usually event-driven such as those written for Java (e.g. JHotDraw, see Chapter 8). However, it has an out-of-sync problem that makes it difficult to develop stable GUI-based XML applications. The following section discusses the problem in the context of SVG but the proposed solutions are generally applicable.

7.6.1 Out-of-sync

The out-of-sync problem occurs when the software view of the mouse state is not the real mouse state. This can occur when the mouse attention is lost once the mouse pointer leaves the painted area of the target XML element - where a particular mouse event is being listened to (captured). It also occurs when the mouse pointer goes completely out of the XML document canvas (the document painted area).

The following SVG hypothetical scenario describes a typical situation when the out-of-sync problem would occur. Imagine that a user wants to drag the scrollbar of a TextList widget written for SVG. The user clicks the mouse button on the scrollbar moving box and starts dragging it. But before the mouse button is released the mouse

pointer - accidentally - goes out of the boundary of the scrollbar (the reason can be a slow machine or intensive drawing) and the user loses the mouse focus (mouse events stop being delivered to the scrollbar mouse events handler). The user releases the mouse button while the mouse pointer is out of the scrollbar boundary and then moves the mouse pointer back to be inside the scrollbar region. Because the mouse events handler of the scrollbar stops receiving mouse events once the mouse pointer is out of its boundary (or if the mouse events are being captured on the background it will stop receiving mouse events once the mouse pointer is out of the SVG canvas/painted area), the mouse state of the widget becomes out-of-sync with the real mouse state which can cause all kinds of confusion.

7.6.2 Problem Analysis: Handling Mouse Events

Three approaches for handling mouse events have been considered:

1. Microsoft Windows Operating System (MS Window OS),
2. Java Environment and
3. DOM Level 3 Mouse Event Model.

MS Windows OS (and other Window-based operating systems) supports several interactive applications running simultaneously. Each application displays its content in a rectangular area, a window. Applications receive messages from the operating system for a variety of reasons. Messages could have originated from the operating system (i.e. window-create, window-size), an input device (i.e. mouse-down, mouse-click, key-press) or from the application itself. The operating system dispatches messages to the target application from a message queue. In the case of the mouse input device, events occur when the user moves the mouse, presses or releases a mouse button. The operating system converts mouse events into messages; and messages are delivered to the active application - which has the focus - and whose window is positioned under the

mouse pointer. The operating system also allows applications to 'capture the mouse'; in that case, the mouse events are delivered to the target application regardless of the position of the mouse pointer or whether the application has the focus or not. Only one application at any particular time can 'capture the mouse'. The mouse input can be discarded after the application has finished with it. The MS Windows OS supports four types of mouse messages: BUTTONUP, BUTTONDOWN, BUTTONDBLCLK (for the left, middle and the right mouse buttons) and MOUSEMOVE. Additional messages, MOUSEHOVER and MOUSELEAVE are sent to the specified application upon request (Windows does not voluntarily send these messages by default). Applications use mouse events generated by the operating system with the ability to 'capture the mouse' input to achieve any desired behaviour.

The mouse event model for Java version 1.1 is called the Abstract Window Toolkit (AWT) Mouse Event Model. It is based on the concept of "event listener". Objects (or handlers) can register themselves or remove themselves as listeners of any mouse event type. The Java environment made the process of handling mouse events easy for Java developers by introducing a wider set of mouse events including: mousePressed, mouseReleased, mouseEntered, mouseExited, mouseClicked, mouseMoved and most importantly mouseDragged. All Java mouse events are generated from the windows (e.g. MS Windows OS) simple mouse messages described earlier. Java hides the complexity of having to 'capture the mouse' once the mouse is out of the application allocated area on the screen (or window) to generate the mouse drag events (i.e. mouse start dragging, mouse end dragging, mouse dragged). This extra layer of abstraction in handling mouse events has the great advantage of making the development of Java applications relatively straightforward.

In Windows, mouse events are sent to a specified application, whereas in Java, mouse events are sent to a target component. In the DOM Level 3 Mouse Event Model,

mouse events are despatched to a specified element in the DOM tree. This is a key concept in the DOM Level 3 Mouse Event Model but can also be a limitation when it comes to handling mouse events effectively as will be explained later.

The event model of DOM Level 3 is based on the 'event listener' model and offers a generic event system that defines an event flow architecture and an event handling mechanism. When an event occurs, it propagates from the top-level element of the DOM tree (the root) down to the target element. Event listeners are notified when the matching event types are received. The event then bubbles back up to the root element of the DOM tree. It is allowed in DOM Level 3 Mouse Event Model to stop an event from propagating or bubbling at any stage of the event despatching process. The event model of DOM Level 3 Mouse Event Model supports a number of mouse event types which includes: click, mousedown, mouseup, mouseover, mousemove and mouseout.

Mouse Events	Microsoft Window	Java Environment	DOM Level 3 Mouse Event Model
Mouse Button Down	✓	✓	✓
Mouse Button Up	✓	✓	✓
Mouse Click	✗	✓	✓
Mouse Double Click	✓	✗	✗
Mouse Enter	✗	✓	✓
Mouse Leave	✗	✓	✓
Mouse Over	✗	✗	✓
Mouse Move	✓	✓	✓
Mouse Drag	✗	✓	✗

The above table gives a list of all the supported types of mouse events provided by: MS Windows OS, Java and DOM Level 3 Mouse Event Model. Java and DOM do not provide an event for mouse double-click. The information about the number of clicks

that occurs when the mouse button is pressed and released several times consecutively is provided with the mouse click event. MS Windows OS does not support mouse click, enter, leave, over or drag events while Java does not support mouse over event.

On the other hand, the set of mouse event types provided by the DOM Level 3 Mouse Event Model lacks support for mouse drag events. The Windows OS also lacks drag events but applications can easily emulate them using the 'capture the mouse' facility. This crucial facility however is left out of the DOM Level 3 Mouse Event Model. The ability to 'capture the mouse' is vital to maintain the state of the mouse when the mouse pointer gets outside the painted area of an XML element or the XML document canvas. This is the primary cause of the out-of-sync problem.

7.6.3 Case Study

The Java AWT Mouse Event Model described in Section 7.6.2 is an elegant, simple and effective way to handle mouse events. The number of mouse event types provided by the AWT Mouse Event Model is close to those provided by the DOM Level 3 Mouse Event Model. The AWT Mouse Event Model also provides support to drag events, fundamental to overcome the out-of-sync problem. For these reasons, this research advocates the AWT Model Event Model to be used in DOM.

The new AWT-like mouse event model for DOM (referred to as domMouse events hereafter) will be able to process raw low-level DOM Level 3 Mouse Event Model mouse events and produce Java AWT-like mouse events (mousePressed, mouseReleased, mouseEntered and mouseExited, mouseMoved, mouseStartDragging, mouseEndDragging, mouseDragged). This new set of mouse events will prevent the out-of-sync problem from occurring because the state of the real mouse will always be in-sync with the mouse state of the software.

We have used SVG for the implementation which has helped us discover the

way to overcome the out-of-sync problem and to outline some recommendations for a future DOM Mouse Event Model showing how to avoid out-of-sync (see Section 7.6.4).

7.6.3.1 Implementation for SVG

The new mouse event model domMouse makes use of the propagation flow phase to handle mouse events in the DOM Level 3 Mouse Event Model.

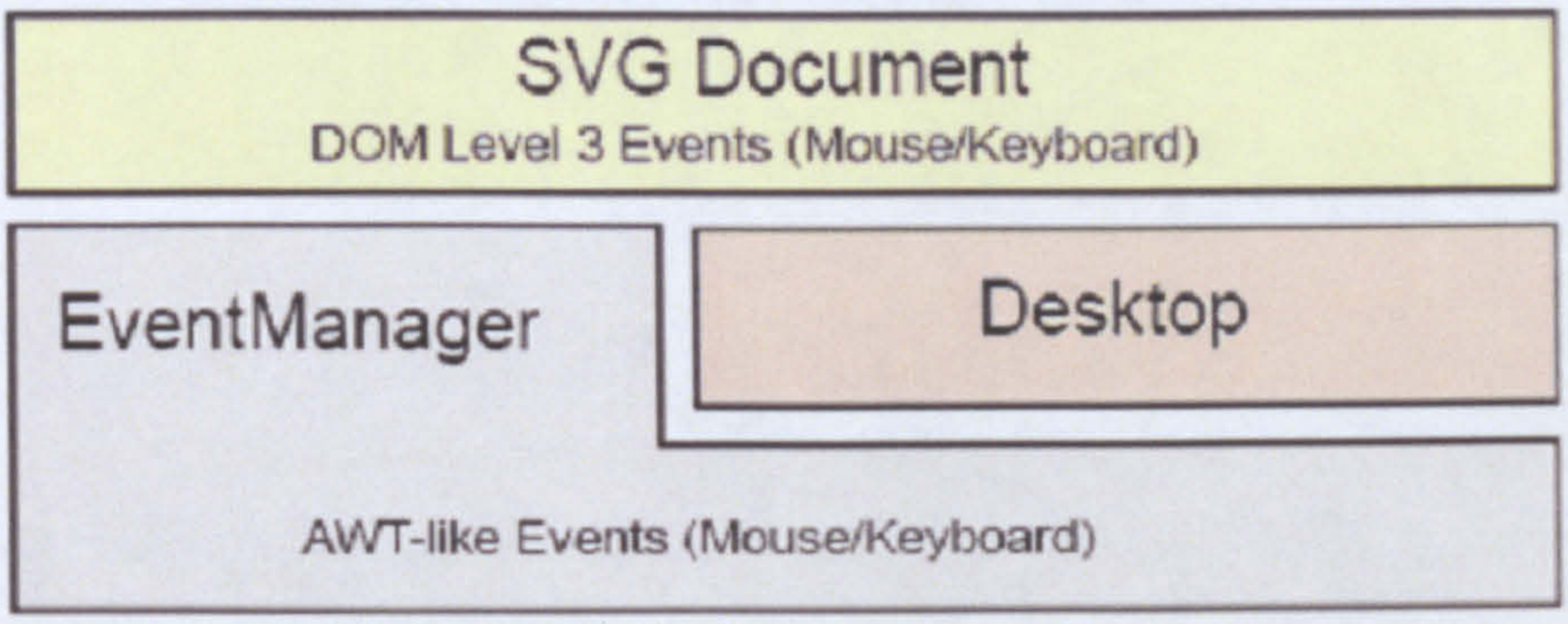


Figure 7-11: The architecture of domMouse

Figure 7-11 shows a basic diagram of the architecture of the implementation of domMouse. The main class in this new method is the EventManager. The EventManager converts raw low-level DOM Level 3 Mouse Event Model mouse events that originate from the SVG document into domMouse events. The Desktop class from the svgDraw2D package (see Section 7.4.1) is used to ensure delivery of the necessary SVG document mouse events to the EventManager when the mouse cursor goes outside of the painted area of its corresponding SVG content. In other words, the Desktop is used to simulate 'capture the mouse' mode in the Windows operating system as explained earlier (see Section 7.6.3.3).

The implementation of domMouse makes use of the svgDraw2D package. The implementation involved re-implementing some of the Java AWT classes and interfaces in JavaScript (for example, MouseMotionListener, MouseListener, MouseEvent).

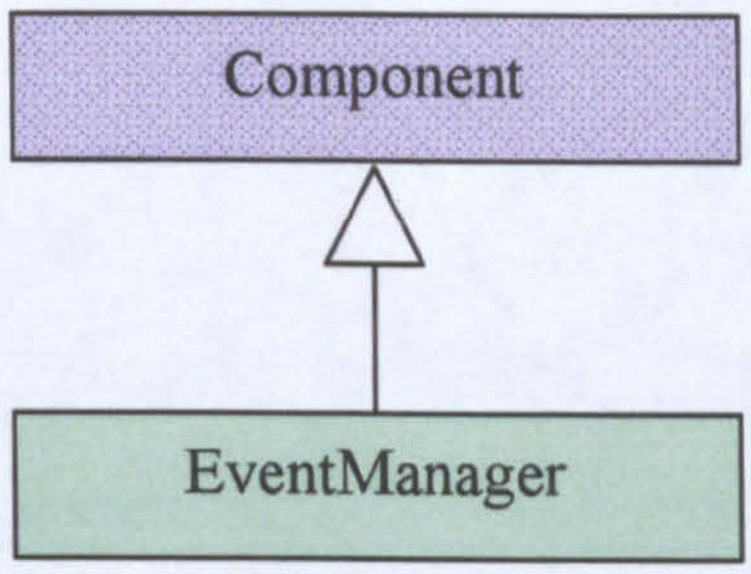


Figure 7-12: Using EventManager in svgSwing.

Figure 7-12 shows the inheritance hierarchy of the model design. All classes of the svgSwing must inherit from the EventManager in order to receive domMouse events.

The Node class (superclass of the Component class, see Figure 7-6) uses the handleEvent method to distribute any DOM Level 3 Mouse Event Model mouse events received to its internal and external event listeners. EventManager deals with DOM Level 3 Mouse Event Model mouse events in a totally different way. Subclasses of EventManager will not be able to handle DOM events directly as EventManager overrides the handleEvent method. But instead, they will be able to handle domMouse events. An object of type MouseEvent - that contains the event context information (i.e. x, y, number of mouse clicks, etc) - is created and passed to the target component when a domMouse event is generated. The ListenerManager class is used to maintain a list of internal and external listeners of the domMouse events. When a new event is received the class notifies all listeners of that particular event type. Listeners have to implement either the MouseMotionListener interface or the MouseListener interface or both.

7.6.3.2 Mouse Events Process Diagram

The code of domMouse runs in two separate modes; Drag mode and Default mode. Figure 7-13 illustrates the state diagram of the two modes. It shows how the domMouse code switches between Drag mode and Default mode.

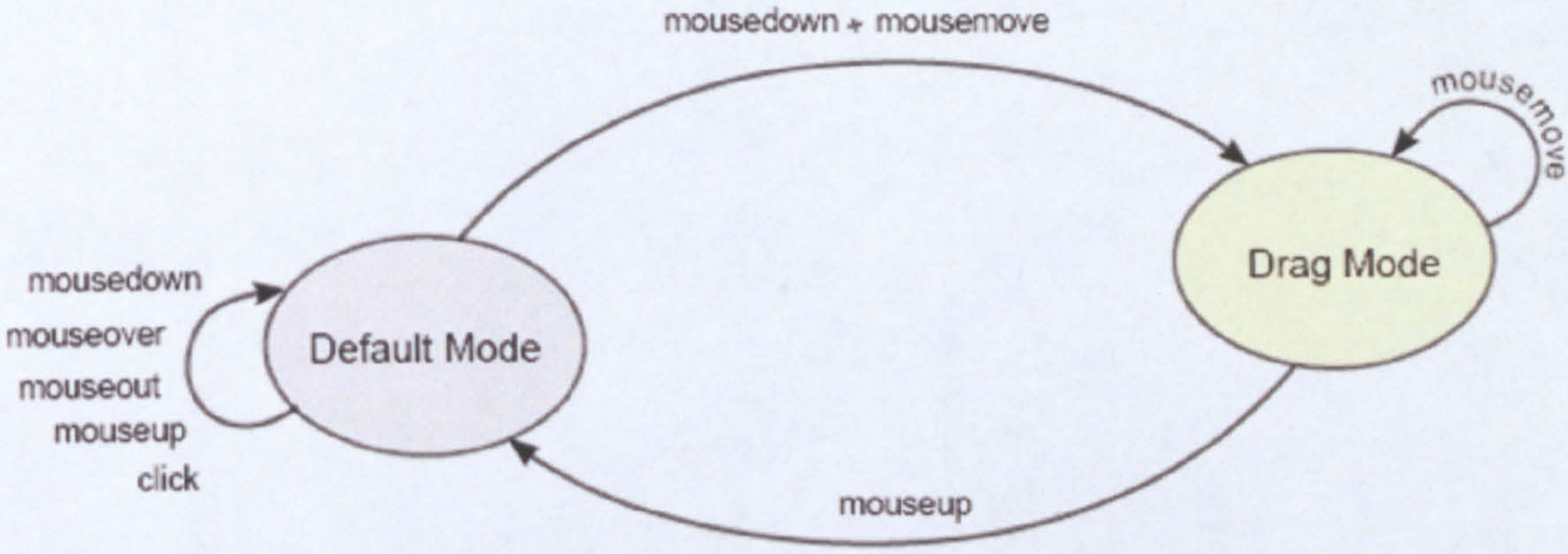


Figure 7-13: Mode state diagram (left to right).

Initially, the mode is set to Default mode. The mode changes from Default to drag once the mousedown event followed by the mousemove event are received. The mode changes back from Drag to Default if the mouseup event is received. For simplicity, assume that EventManager registers itself as a listener to all available DOM mouse events of a particular SVG element. If mousedown, mouseover, mouseup, mouseout or click events are received, EventManager generates mousePressed, mouseEntered, mouseReleased, mouseExited or mouseClicked respectively, and then sets a special variable called mouseState to the type of the received event. The purpose of the mouseState variable is to keep track of the last mouse event received. If mousemove is received and the mouseState was previously set to mousedown then, EventManager generates a mouseStartDragging event and triggers the Desktop to enter the Drag mode. The Desktop takes control of DOM mouse events by stopping the propagation phase of all DOM mouse events. The EventManager will no longer receive any events. The Desktop also stops delivering global mouse events to registered listeners. In Drag mode, the Desktop listens only to mouseup and mousemove events. It then routes those events to EventManager. While the Desktop is in Drag mode, EventManager generates the mouseDragged event for each mousemove event received and generates the mouseEndDragging event when it receives a mouseup event. The EventManager will

trigger the Desktop to switch to Default mode after it generates the mouseEndDragging event.

7.6.3.3 Simulate 'Capture the Mouse'

In the previous section it was shown how the Desktop was used to simulate the 'capture the mouse' feature. Mouse events continue being delivered to the EventManager even after the mouse cursor leaves the painted area of the target SVG element. In order to be able to capture mouse events, the Desktop listens to all mouse events occurring in the SVG document by registering itself as an event listener of the DOM tree root element. In addition, for mouse events to occur continuously regardless of whether the mouse cursor is on an SVG element or on an empty space, the Desktop has to create an invisible (or visible) SVG content that encompasses the whole client area of the host (i.e. a window or a section of the screen). Hence the Desktop will be able to intercept all mouse events that occur inside the client area (see Figure 7-14).

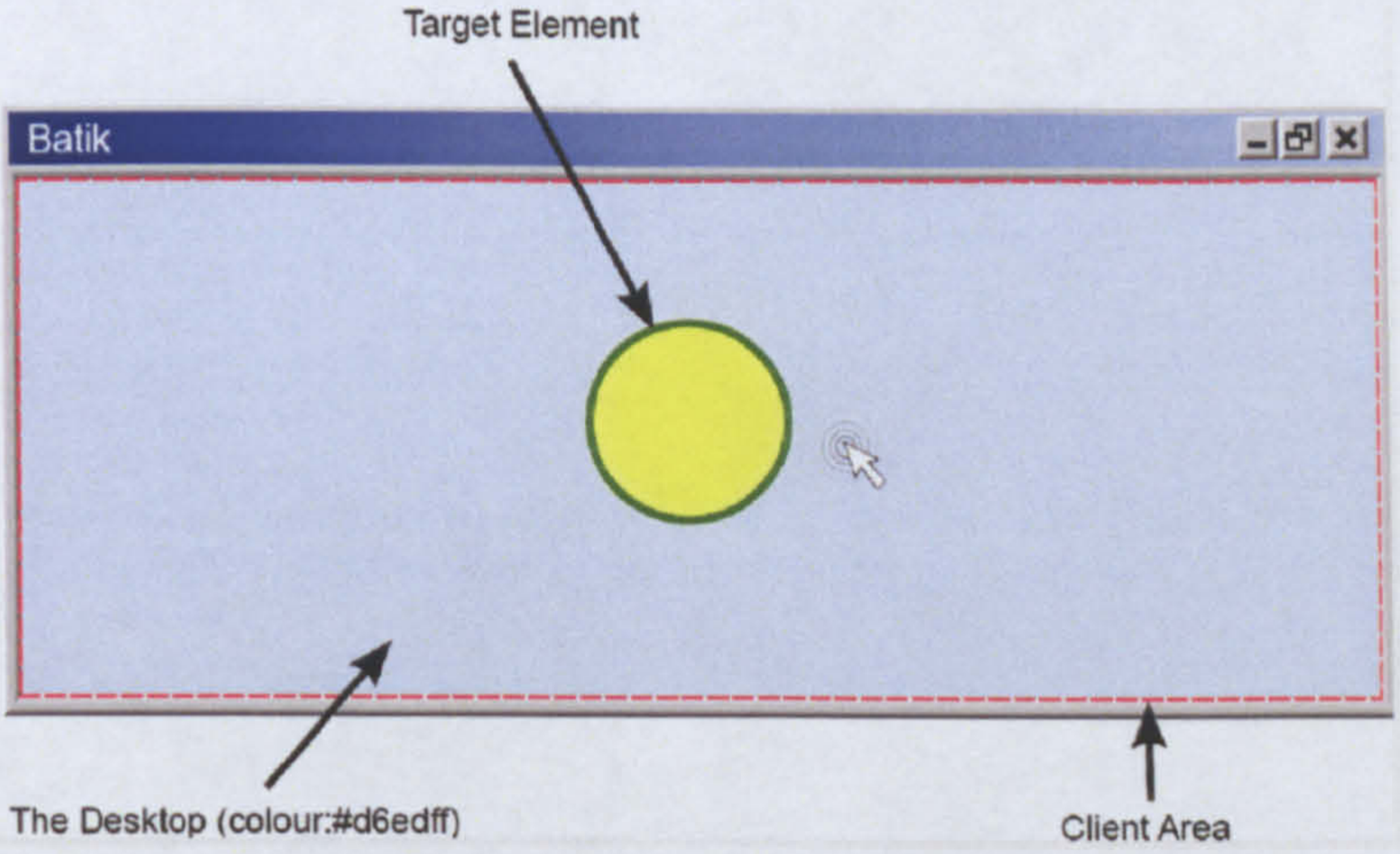


Figure 7-14: The application client area filled with the Desktop content

The mouse attention will be lost if the mouse cursor leaves the SVG canvas completely and goes out of the client area because neither SVG nor DOM provide any means to 'capture the mouse'.

SVG viewers (i.e. Adobe SVG plug in version 3, 6 beta and the Batik 1.6)

ensure that event listeners of a target element continue to receive events so long as the mouse is over the element's painted area even if that area is not visible from within the client area view (clipped out, See Figure 7-15). This is achieved because these SVG viewers do 'capture the mouse' automatically if the content of an SVG element -where mouse events are being captured - is larger than the client area.

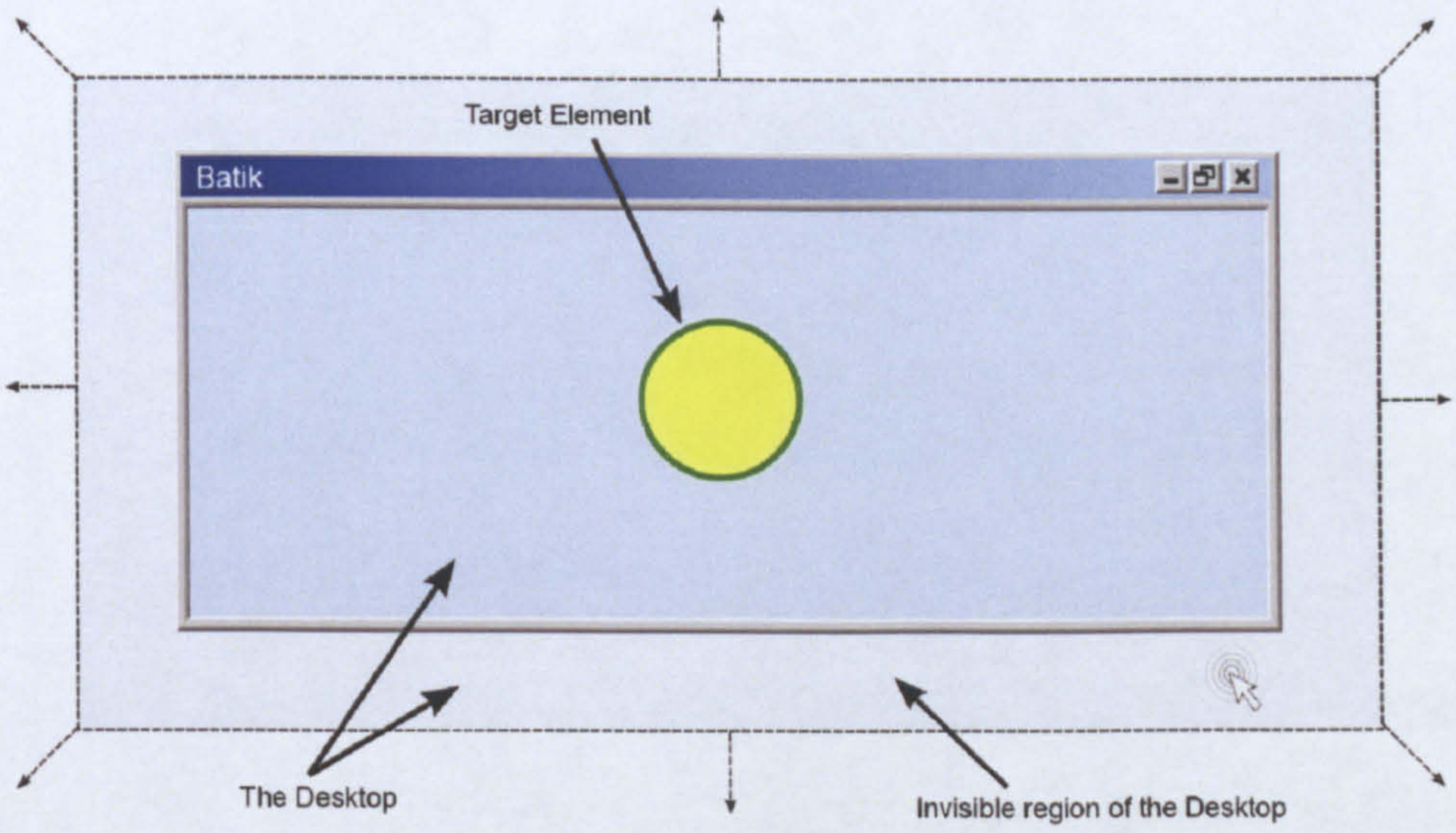


Figure 7-15: The Desktop content spans beyond the application window client area

This feature has been utilised to resolve the out-of-sync problem. The Desktop class automatically generates content (i.e. an SVG rectangle element) to cover the SVG document in all directions as shown in Figure 7-15 to guarantee that the mouse focus can never be lost.

7.6.4 Recommendations

XML applications that are currently using the DOM Level 3 Mouse Event Model suffer from the out-of-sync problem described earlier. This is because the DOM Mouse Event Model captures mouse events only on XML elements. To overcome this problem, we recommend that future DOM Mouse Event Models should provide a global facility to 'capture the mouse' so that mouse events are captured regardless of the location of the

mouse pointer on the screen and independent of any target XML elements. We also recommend that mouse events must be captured on the visible or partly visible XML elements, including the painted area outside the view area of the SVG browser in all SVG implementations (i.e. Apache Batik, Adobe SVG viewer, Firefox, Safari, etc.).

7.7 Summary

This chapter has introduced a novel approach to developing user interfaces that are adaptive towards different device types. This approach has addressed the major issues in developing an SVG-based user interface including: (1) a Class-based model to write applications in JavaScript (ClassBJS), (2) a flexible 2D graphics package which decouples the manipulation of DOM/SVG interfaces from writing graphics applications (svgDraw2D), (3) a generic, easy to use, sophisticated approach to handle mouse events in DOM (similarly to that of Java AWT) which avoids using the DOM Mouse Event Model directly and therefore avoids the out-of-sync problem (domMouse) and finally, (4) a reliable, extensible, robust set of GUI widgets based on a well-designed and strong framework (svgSwing).

The next chapter will demonstrate the use of the Oea framework as a way of constructing adaptable user interfaces that work on different device types.

8

Use Case 1: Porting JHotDraw Via The Oea Framework

8.1 Introduction

In Chapter 3 we introduced our four-layer model to develop ACTs followed by chapters describing the software that populates the model in each layer. This chapter will use the four-layer model in the Application Scenario context (see Section 1.2). The first step to achieving this is to run a complex application in the Web environment. This will be accomplished by porting code of a generic application called Java HotDraw (JHotDraw) [Gamma and Eggenschwiler].

JHotDraw can be used to build various graphical applications such as painting programs, UML and CAD tools, and chart and schematic diagrams [Brant, 2006]. JHotDraw is the Java implementation of HotDraw, a framework for developing 2D structured drawing editors. The HotDraw framework [Johnson, 1992] was originally developed as a "design exercise" - extensively using design patterns [Alexander, Jshikawa et al., 1977]. It was first implemented in the VisualWorks Smalltalk language [Tomek, 1999] and later in Java. This chapter described how JHotDraw was ported from Java to SVG and JavaScript using the Oea framework (see Figure 8-1).

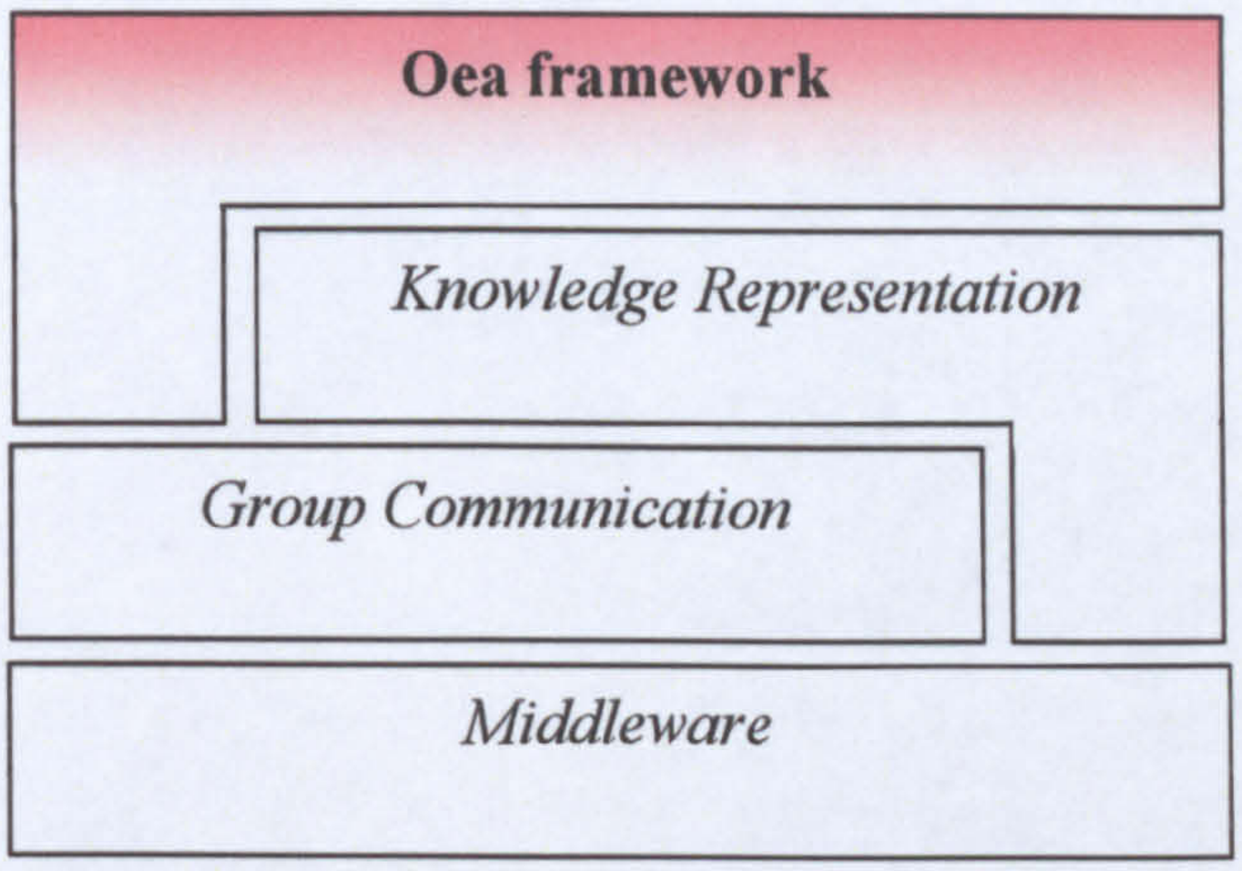


Figure 8-1: The four-layer model: Presentation and Interaction layer (Oea framework) used to implement JHotDraw.

This chapter starts with a description of the JHotDraw user interface. The JHotDraw architecture is presented next, followed by the challenges and the requirements identified in order to re-implement JHotDraw in the SVG environment via Oea framework. The following section describes the implementation of JHotDraw in SVG and JavaScript (called Oea HotDraw). From this experience, a step-by-step guide to porting Java applications to Oea framework will be presented. The final section will demonstrate how the Oea framework has successfully resolved the shortcomings of more traditional methods previously used to develop client-side Web applications.

8.2 JHotDraw User Interface

Applications built from JHotDraw edit drawings that are made up of figures. Figures are the main graphical elements in JHotDraw such as rectangles, ellipses, lines and text. Applications built in JHotDraw often require new application specific figures (for example a *class* figure in a UML editor application). Figure 8-2 shows a screenshot of JHotDraw version 5.1.

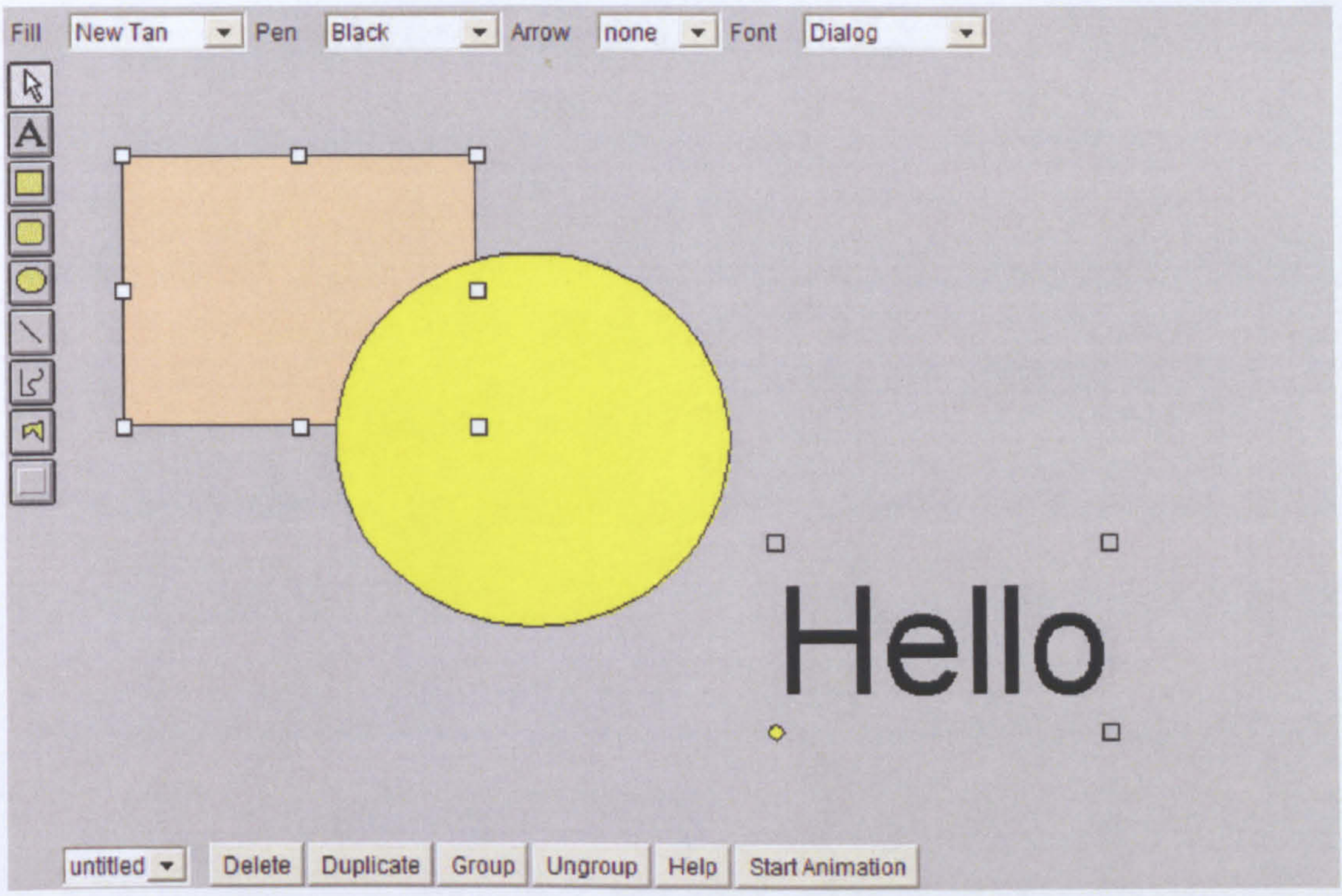


Figure 8-2: JHotDraw 5.1 Applet

On the left side, there is a set of tools (called the tools palette) that is used to manipulate the drawing. Each tool is represented by a button and they are from top to bottom: the Selection tool, Text tool, Rectangle tool, Round rectangle tool, Ellipse tool, Line tool, Scribble tool, Polygon tool and Border tool. Only one tool can be active at any particular time; some are used to create figures, others (i.e. the Selection tool and Border tool) are used to manipulate existing ones. JHotDraw enables applications to easily create new tools. Properties of figures, such as fill colour, border colour and font type for text figures, can also be changed using the drop-down menus at the top of the screen. The Selection tool is essential; it is used to select a figure or several figures to move, delete, group, ungroup or animate (see the buttons at the bottom of the screen, Figure 8-2). When a figure is selected, JHotDraw presents appropriate handles to modify some of the figure's properties. In Figure 8-2, the selected rectangle - on the left - has eight handles (small filled boxes) used to change its size. The text figure - on the bottom right (Hello) - has only one active handle (filled circle) that is used to change the

font size; the other three handles (empty boxes) are inactive.

8.2.1 JHotDraw Architecture

The main architectural components of the JHotDraw framework are shown in Figure 8-3. At the heart of JHotDraw is the DrawingEditor interface which is implemented as the application window (DrawApplication). DrawApplication maintains a list of tools which are part of the DrawingController (not featured in Figure 8-3) and represents the application window by inheriting from the JFrame class (from Java Swing package). Also, DrawApplication contains one or more DrawingView(s). The DrawingView is an area that can display a Drawing and listen to user input. The Drawing contains a collection of Figure(s) and it informs the DrawingView of any changes (to update the view). The Figure is a visible component of a drawing but can also be a container of other Figure(s).

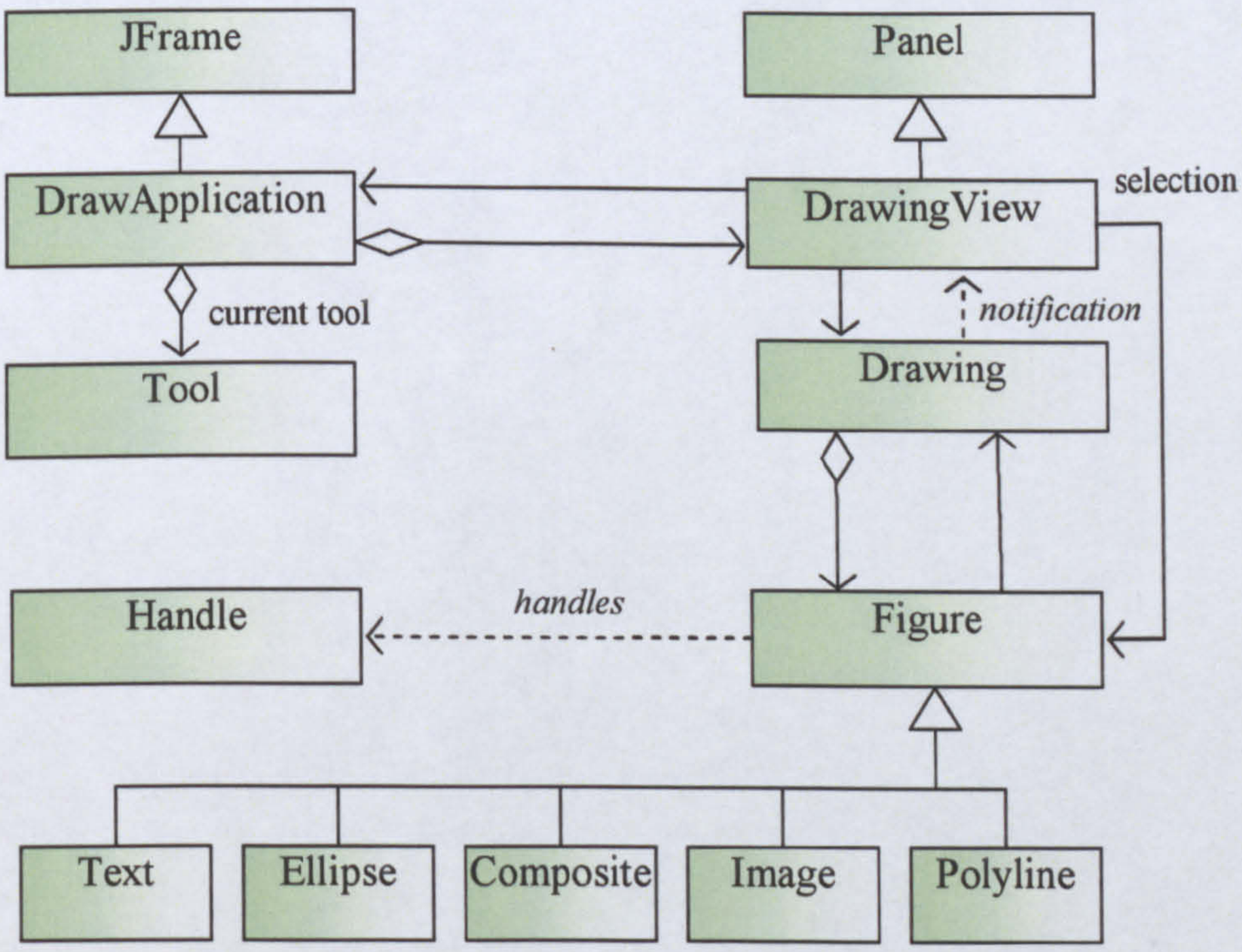


Figure 8-3: Class diagram of JHotDraw Architecture.

JHotDraw supports different types of Figure(s). Each Figure has one or more Handle(s)

(see Section 8.2). The Handle has a specific graphical representation and determines how to interact with a figure (e.g. change its size, colour, etc.). The DrawingEditor has one active Tool which can be chosen from the tools' palette. JHotDraw has many Tool(s) to allow the user to interact with the application. Each Tool represents a user interface mode (select, create, etc.) and operates on the Figure(s) contained in the Drawing that is associated with the current DrawingView.

The DrawingController delegates all operations to the current tool; therefore, changing the current tool changes the editor's behaviour (DrawApplication). DrawingView inherits from Panel to capture user interaction (mouse or keyboard events) and display the drawing. The Drawing consists of figures; each figure has an associated set of handles.

JHotDraw uses the Java AWT toolkit [Zukowski, 1997] for graphical user interface support. AWT includes sophisticated graphical user interface widgets such as windows, buttons, text boxes, labels, tables, etc.

8.2.2 Model-View-Controller

The design of JHotDraw is based on the Model-View-Controller (MVC) paradigm [E.Krasner and Pope, 1988]. The MVC paradigm separates the application's logic from the user interface dependencies by decoupling the data from the view and the user interface. The model contains the application data, the view represents the data on the screen and the controller defines how the user interface reacts to user input (i.e. mouse, keyboard, etc.).

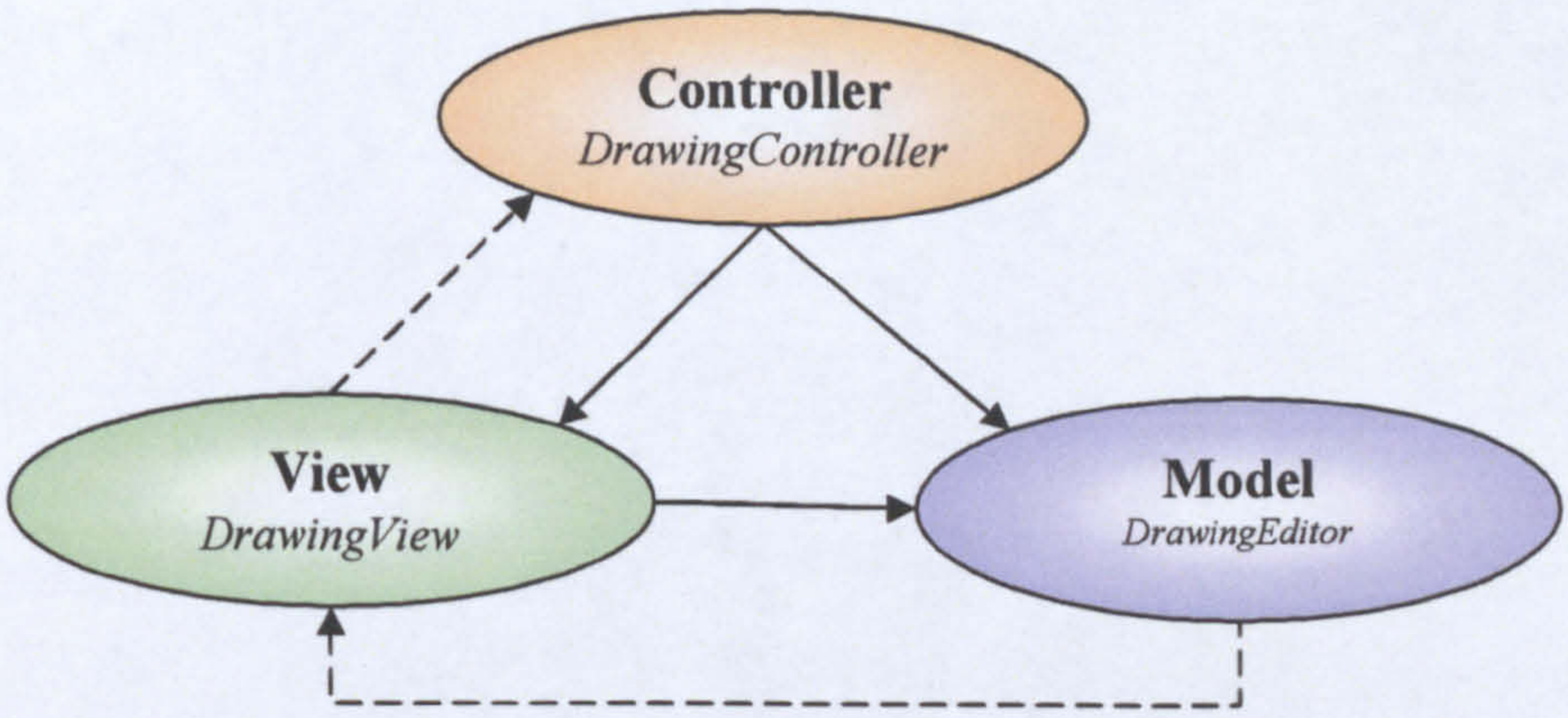


Figure 8-4: Model View Controller

Figure 8-4 illustrates the MVC model and the relationships between its components. The Model is kept isolated from the View and the Controller so that it can be reused by any presentation and input technology. Hence, the solid lines represent a direct association and the dashed lines represent an indirect association (the Model does not have a direct association with the other components of the model). The Model can have more than one View. The Model notifies its View(s) indirectly when its value changes. The View reacts by reading the Model's value to update the screen. The Controller responds to the user actions (e.g. press a key) by directly notifying the Model which might result in a change to the model's value.

Three interfaces are defined to represent the MVC model in JHotDraw: DrawingEditor (Model), DrawingView (View) and DrawingController (Controller). The user interacts with JHotDraw using the mouse or the keyboard. The DrawingController receives input events indirectly from the user interface (this relationship is represented by the dashed line between the View and the Controller in Figure 8-4). The DrawingController collaborates with Tools (see the tools palette in Figure 8-2) to make changes. The user's actions are passed to the currently active tool. The tool updates the model (DrawingEditor) according to the tool type and the user actions on behalf of the DrawingController. For example, when the ellipse tool is

selected, the action click and drag creates a new ellipse. The command pattern is used to encapsulate actions and to enable undo operations. The DrawingView gets the data from DrawingEditor to display the drawing. Normally, DrawingEditor has no direct knowledge of the DrawingView; although the DrawingView needs to be acknowledged when a certain change happens in the DrawingEditor (this relationship is represented by the dashed line between the model and the view in Figure 8-3). For instance when a new Figure has been added to the DrawingEditor by the DrawingController, the DrawingView is acknowledged by the DrawingEditor to update the display with the new change.

8.3 Challenges and Requirements

JHotDraw has a particularly rich and complex architecture and it uses very rich collections of graphics primitives and user interface widgets. The graphics primitives that JHotDraw uses include the following: Rectangles, Rounded-Rectangles, Ellipses, Lines, Polylines, Text, Images and Borders. JHotDraw also uses an extended set of user interface widgets including: Windows, Buttons, Labels, Drop-down Menus, Edit Boxes, Pop-up Menus and Tool-tips.

JHotDraw makes use of several design patterns [Gamma, Helm et al., 1994]. Each design pattern describes a reoccurring problem in the particular domain. By learning about all the design patterns of JHotDraw one can build other applications following the same approach. Some of the design patterns used by JHotDraw [Kaiser, 2001] are: Composite pattern (e.g. CompositeFigure, DrawingView), Strategy pattern (e.g. DrawingController), Prototype pattern, Decorator pattern (e.g. DecoratorFigure), Factory pattern (e.g. DrawApplication), State pattern (e.g. Tool), Observer pattern (e.g. DrawingEditor).

8.4 Implementation

The work on the Oea framework has prepared JavaScript/SVG for the implementation of JHotDraw. Extensive implementation and hundreds of abstract classes, interfaces and classes from the original JHotDraw were ported to JavaScript through the Oea framework manually. There were more than 170 classes that have been used to implement JHotDraw.

Appendix III lists the SVG document for Oea HotDraw. It shows the default structure of the SVG document used to build applications via the Oea framework. The SVG document does not have any content initially. All the classes of the Oea framework as well as the other classes required to develop applications for the Oea framework such as JHotDraw classes have to be included. In Appendix III the classes included in the SVG documents are:

1. Lines 18-30: System and Foundation Classes, such as: svgNode, RectNode, etc. (see Section 7.4.1).
2. Lines 20-64: svgDraw2D classes, such as: Shapes, Graphics, Layers, Desktop, etc. (see Section 7.4.1).
3. Lines 66-72: Utility Classes, such as: Vector, Hashtable, Enumerator, etc. (see Section 8.3).
4. Lines 74-89: AWT classes, such as: MouseEvent, MouseListener, Colour, etc. (see Section 7.6).
5. Lines 91-99: Swing Look and Feel classes, such as: Button Skin, Window Skin, BoxButtonSkin, etc. (see Section 7.4.2).
6. Lines 101-126: Swing classes, such as: Component, Container, Button, Window, List, etc. (see Section 7.4.2).

7. Lines 128-130: Helping tools classes, such as: DebugWindow, svgXMLBrowser, etc.

8. Lines 133-217: JHotDraw classes, such as: Figure, Drawing, View, Handle, Command, Tool, etc.

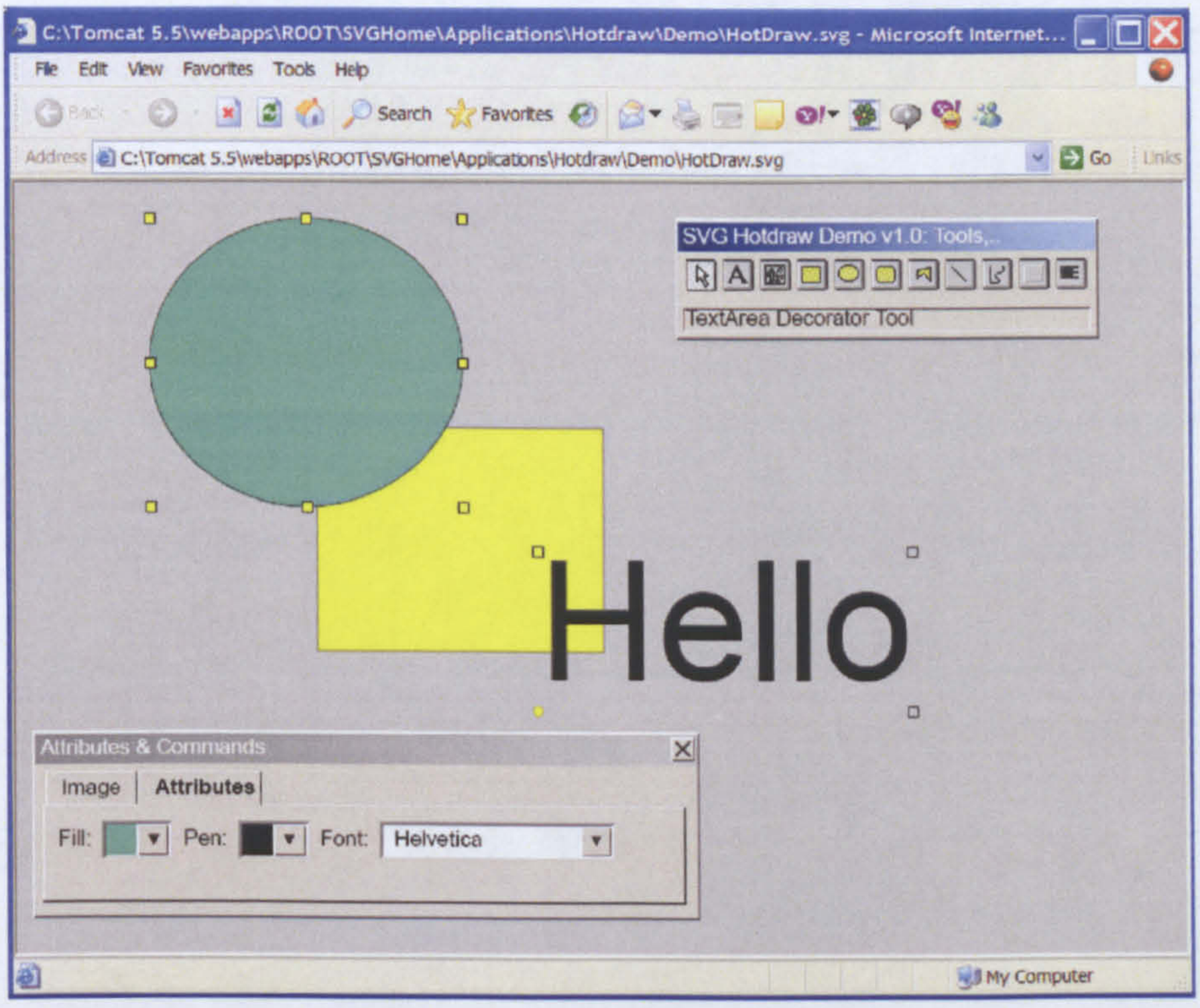


Figure 8-5: Screenshot of Oea HotDraw running in Microsoft Internet Explorer with Adobe SVG Plug-in version 6 beta.

Figure 8-5 shows Oea HotDraw running in the Microsoft Internet Explorer Browser with the Adobe SVG plug-in version 6 beta. The user experience (i.e. interaction with the user interface) and the functionalities of Oea HotDraw match those of JHotDraw. Oea HotDraw also has been tested successfully on the Apache Batik SVG browser. This work did not test Oea HotDraw on other SVG viewers.

8.4.1 New Features

To demonstrate the effectiveness and extensibility of our approach to writing SVG applications, a new tool (TextAreaDecoratorTool) and figure (TextAreaDecorator) have been implemented in Oea HotDraw. The features of these new tools were not provided by the original JHotDraw. The TextAreaDecoratorTool allows the user to insert and edit

text inside figures (circles, rectangle, etc.). The inserted text is laid out to fit within the shape of the figure used. This relies on features that are available in the SVG 1.2 Full working draft. As shown in Figure 8-6, the TextAreaDecorator tool is represented by the button on the far right end of the Tools Window  .

Figure 8-6: The new TextDecorator Figure and Tool.

The TextAreaDecorator figure has been built to conform to the Decorator pattern used in JHotDraw. The Decorator pattern allows new behaviours to be added dynamically to existing classes. By selecting the TextAreaDecorator tool and clicking on the required figure, the TextAreaDecorator figure wraps up the required figure and fits a portion of text inside its display area width and height. The text is laid-out every time the original figure changes size. Also TextAreaDecorator has three control handles as shown in Figure 8-6. The one on the left changes the opacity of the original figure from totally opaque to totally transparent. The middle handle changes the font size, and the one on the left changes the alignment of the text (left, centre, right). The text can be changed by clicking on the figure with the TextAreaDecorator tool; a floating TextBox appears to

allow modifications of the text.

8.5 How to Port Java Applications into the Oea Framework

Generalising from the steps taken to port JHotDraw, the following methodology emerges for porting Class-based applications written in OOP languages (i.e. Java) into the Oea framework (i.e. JavaScript/SVG):

1. Create an SVG document.

2. Set the SVG version to 1.2 (Appendix III, line 9) to take advantage of the SVG 1.2 Full facilities supported by the Oea framework (for Batik and Adobe SVG viewer, see Sections 7.3.2 and 8.4.1). At the time of writing, SVG 1.2 Full is still a working draft.

3. Include all the necessary Oea framework and JHotDraw classes (see Section 8.4)

4. Allow the SVG document to receive all pointer events (i.e. mouse events) whenever the pointer is over either the interior (i.e. filled area) or the perimeter (i.e. stroke) by setting the property 'pointer-event' to 'all' (Appendix III, line 9). This is important for domMouse to work correctly.

5. Set the size of the SVG document (inside the browser) by setting the 'width' and 'height' properties (100% for full screen/window or other values for fixed sized view, Appendix III, line 9).

6. Specify the required resolution of the SVG document (not the device) by setting the property viewBox (Appendix III, line 10), (see Section 8.6 for more detail). The Oea framework organises itself to handle different sizes and resolutions.

7. If desired, disable the browser's Zoom and Pan feature by setting the property 'zoomAndPan' to 'disable'. This should apply to any SVG browser. The Oea framework provides its own generic mechanism to zoom and pan which does not interfere with its other functions.

8. Specify the name of the main function of the application (e.g. main()).

9. In the main function, initialise all the Oea framework packages including SvgDraw2D, SvgSwing. Detect the current browser (Adobe SVG Viewer or Batik) by invoking the public method initialise().

10. Following the ClassBJS model, manually port the source code of the required application into JavaScript. Use the available Foundation Classes, the svgDraw2D package and the svgSwing package as needed or extend to new classes if necessary. There is scope for developing tools to automate this process.

11. Test the newly created JavaScript classes of the application in Adobe SVG and Apache Batik viewers.

Repeat point 10 and 11 until the application is completely ported into JavaScript/SVG.

8.6 Test and Demonstrate

This chapter has demonstrated how the Oea framework has been successfully used to develop adaptable Web-based user interfaces (i.e. re-implementation of JHotDraw in SVG/JavaScript). These interfaces are scalable and can adapt to different screen sizes and resolutions of various device types. This is providing that these devices support SVG and have a reasonable screen size for the desired application.

This section will test how Oea HotDraw adapts when the resolution or the

screen size of the device that it is running on changes. We will compare Oea HotDraw with JHotDraw.

These tests use a physical device: a Dell XPS M1710 Laptop with 17" screen and Windows XP installed. Java 2 (build 1.3. 0_01), Microsoft Windows Internet Explorer 6 and the Adobe SVG plug-in (version 6 beta) were used to run Oea HotDraw.

Devices of different sizes and resolutions were emulated by the physical device. The resolution will be changed using the display properties setting of Windows XP, while the screen size will simply be emulated by changing the window size of the running application (Internet Explorer Web browser, JHotDraw). The internal window size represents the screen size of the device running Oea HotDraw. The smaller the application window size the smaller the device screen and the bigger the application window size the bigger the device screen. Screenshots are provided to illustrate this.

SVG has a sophisticated and flexible coordinate system which was exploited to achieve the adaptability and scalability of the Oea framework. SVG images are drawn on a 2D plane called the SVG canvas. Conceptually, the SVG canvas is infinite in size. Graphics and drawings can be painted on this vast canvas but only a finite rectangle of this canvas can be shown at any point in time. This rectangle is called the viewport. There are three attributes present on the SVG document element that control the viewport:

(1) *width:* to set the width of the viewport on the screen,

(2) *height:* to set the height of the viewport on the screen and

(3) *viewBox(x y width height):* to set the *x*, *y* coordinates of the top left corner and the *width* and *height* of the visible part of the SVG coordinate space.

As a plug-in running in a Web browser (Microsoft Internet Explorer), Java applications are required to specify the size they need for display. In this test, JHotDraw

was set to occupy an area of 560 * 370 pixels. In order to carry out this evaluation, the viewBox of Oea HotDraw was also set to (0,0,560,370) while the width and height of the viewport were set to 100% (maximum width and height with regard to the window size of the browser). All screenshots presented in the subsections below show some Windows XP desktop icons (My Documents, My Computer, Recycle Bin, Desktop, Safari) to the left side which serve as a reference point to the changes in size and resolution which will take place. Three figures (circle, rectangle and text figure) have been drawn on the workspace of both applications Oea HotDraw and JHotDraw, to help illustrate the effect of changing the two parameters (screen size and resolution). There will also be a subsection to demonstrate the scalability feature of Oea HotDraw (ability to zoom in/out) provided by the Oea framework.

8.6.1 Screen Size

This section will contrast the effect of changing the screen size in Oea HotDraw in comparison to JHotDraw. The resolution will be fixed for these tests to the default 1920 * 1200 pixels while the screen size will be altered. Below is a series of screenshots for the two applications side-by-side, Oea HotDraw (to the left) and JHotDraw (to the right).

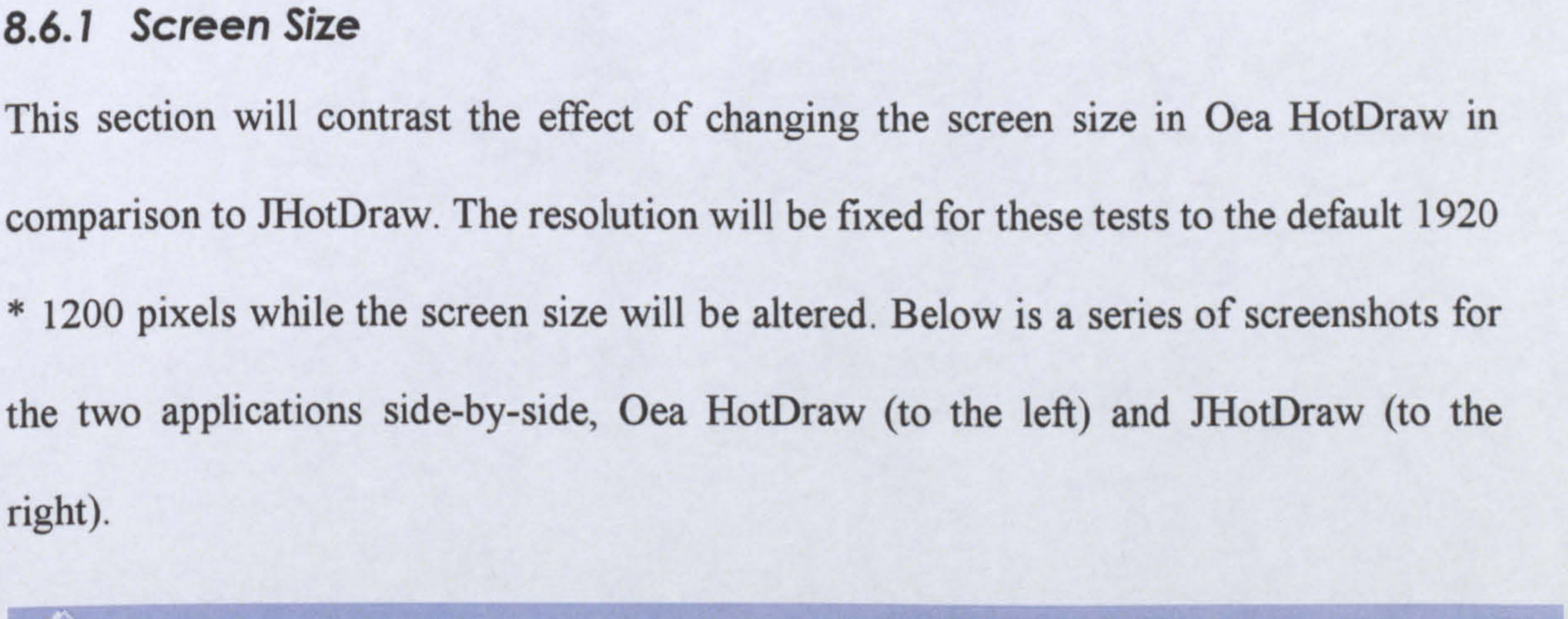


Figure 8-7: Oea HotDraw and JHotDraw with a screen size of 700 * 465 pixels.

Figure 8-7 shows two equal sized windows of Oea HotDraw and JHotDraw. The internal size of the window (presumably the screen size of the device) is 700 * 465 pixels. The content of both applications is perfectly visible.

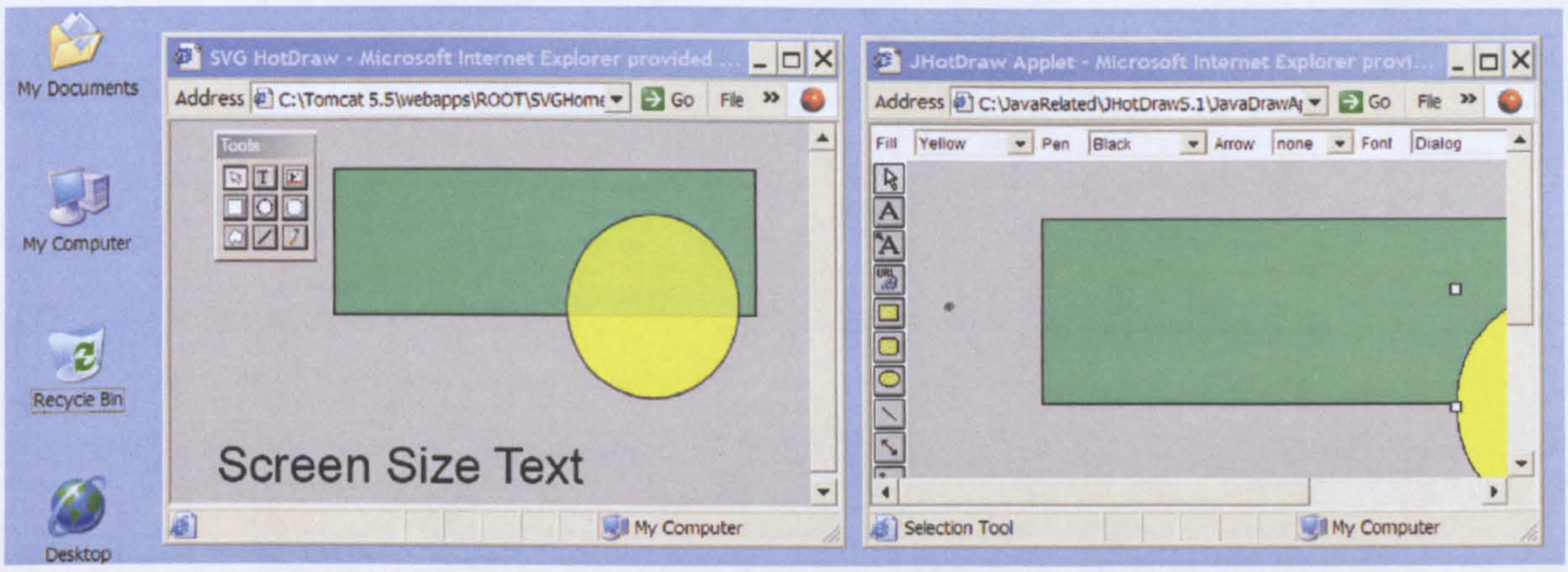


Figure 8-8: Oea HotDraw and JHotDraw with a screen size of 495 * 295 pixels.

Figure 8-8 shows that when the size of the screen for the two applications changes to 495 * 295 pixels, while Oea HotDraw adapts to the new changes, the workspace of JHotDraw becomes partially invisible (the text figure and part of the rectangle and circles cannot be seen).

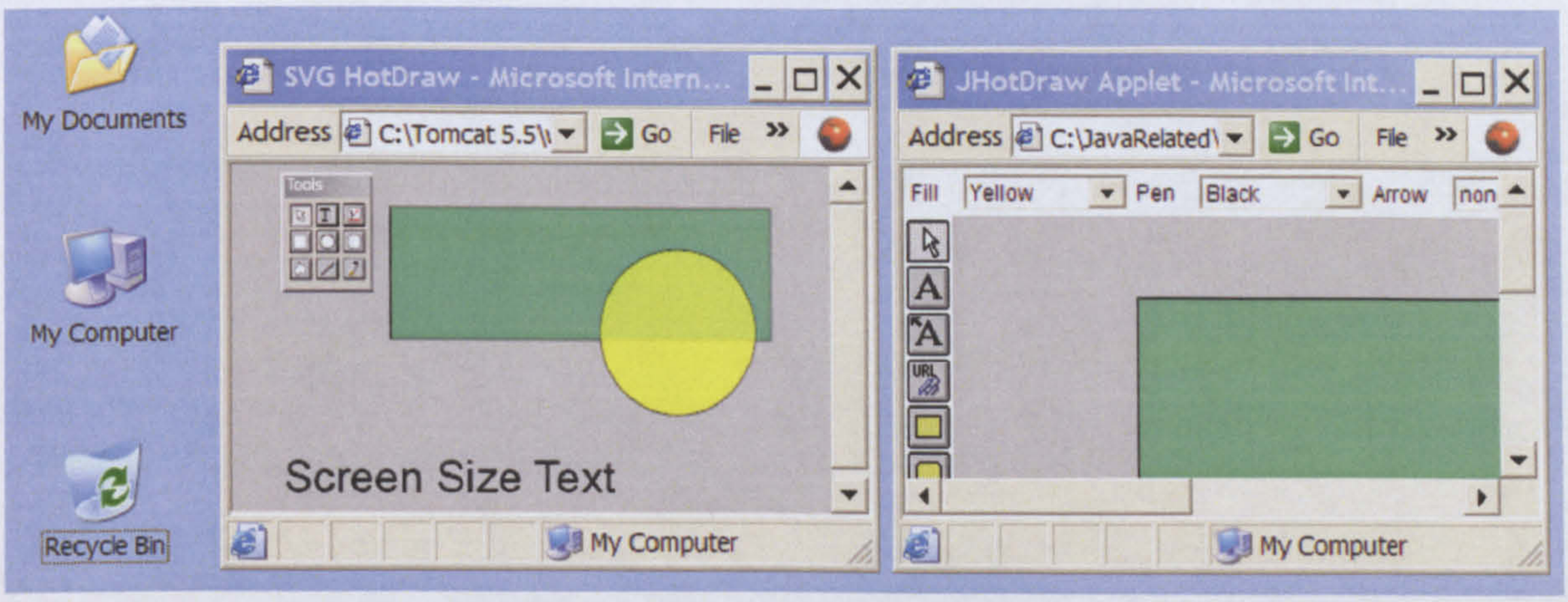


Figure 8-9: Oea HotDraw and JHotDraw with a screen size of 340 * 195 pixels.

Invisibility of JHotDraw workspace worsens as the screen size drops to 340 * 195 pixels (see Figure 8-9), and it becomes mostly invisible with a screen size of 154 * 88 pixels, while Oea HotDraw adapts every time.

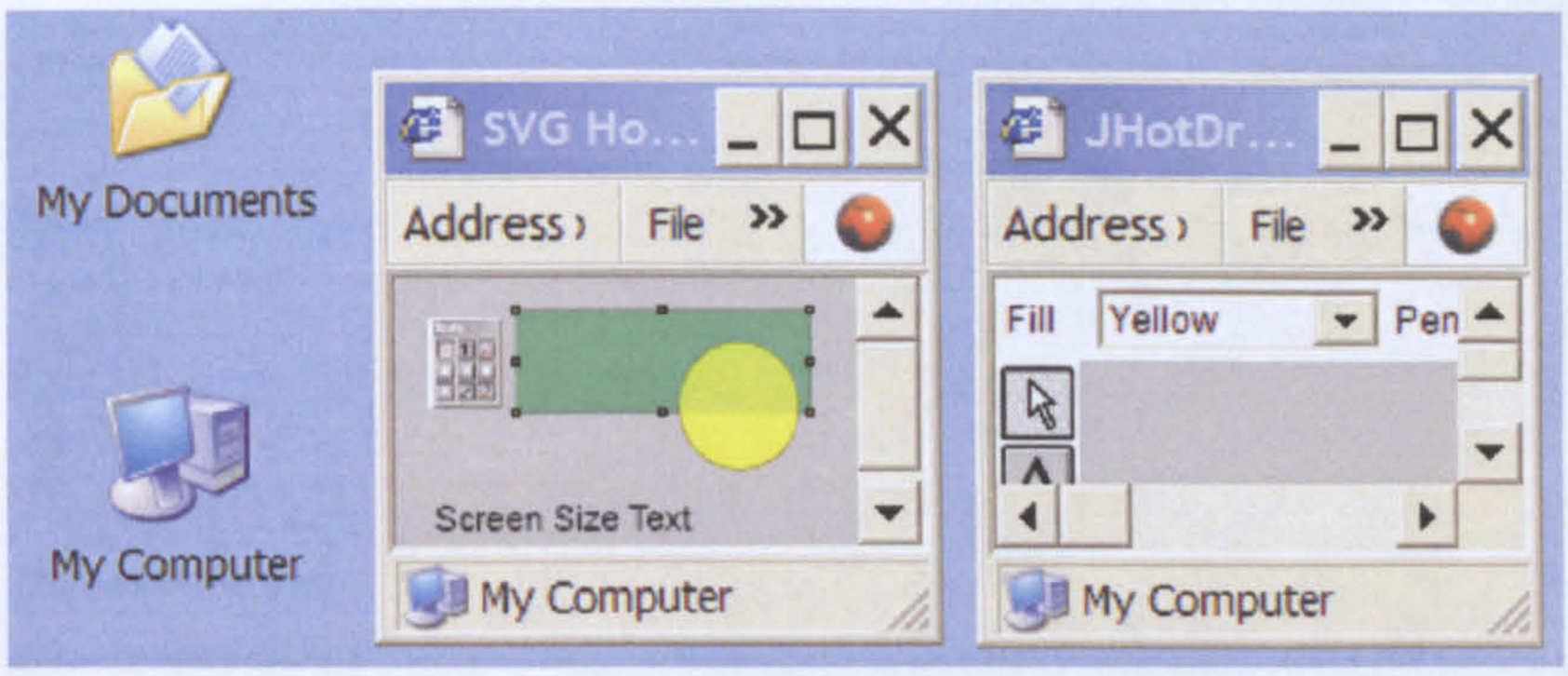


Figure 8-10: Oea HotDraw and JHotDraw with a screen size of 154 * 88 pixels.

The user might find it difficult to interact with an application such as Oea HotDraw running on such a small screen size.

8.6.2 Resolution

This section will test the adaptability of Oea HotDraw in comparison to JHotDraw running in two different screen resolutions, 1920 * 1200 pixels and 640 * 480 pixels. Both work nearly in full screen size but allowing space to show some desktop icons as a point of reference as explained above.

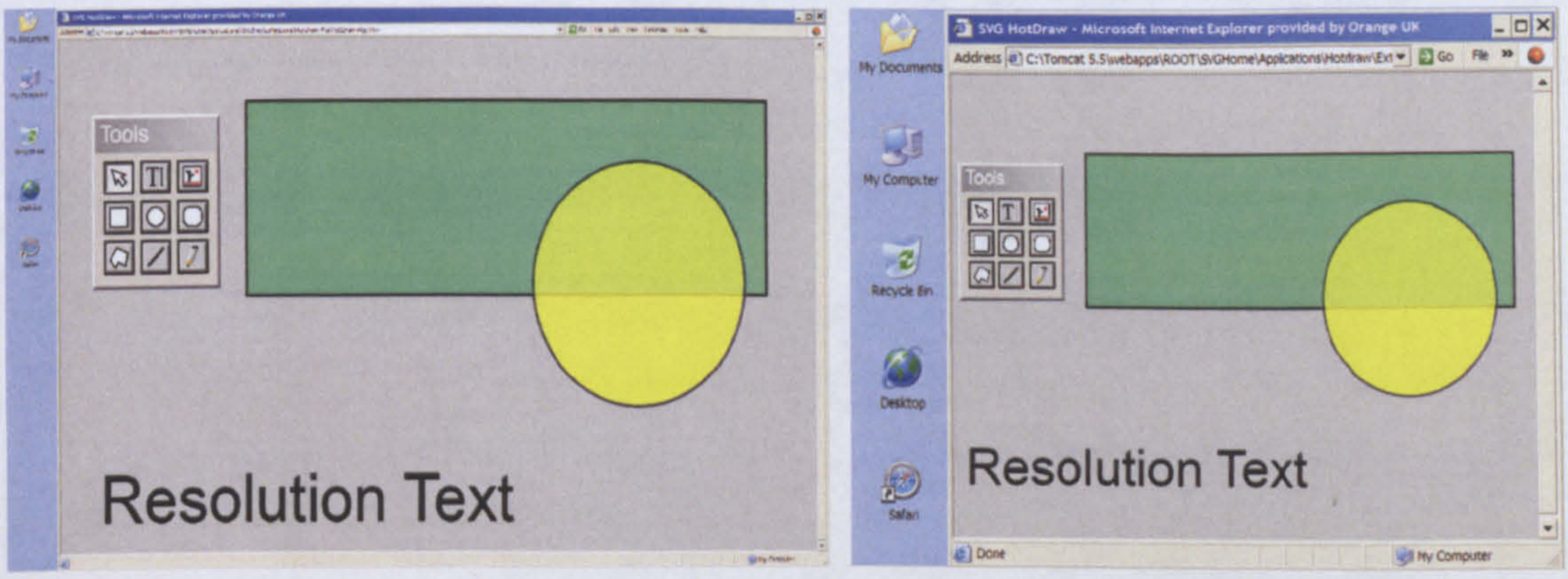


Figure 8-11: Oea HotDraw with screen resolution of 1920 * 1200 pixels (left) and 640 * 480 pixels (right).

Figure 8-11 shows Oea HotDraw running in two different screen resolutions, in both, the toolbar (the tool menu on the left side of the window) and the graphics of Oea HotDraw adapt to the changes in the resolution while maintaining consistent visual appearance.

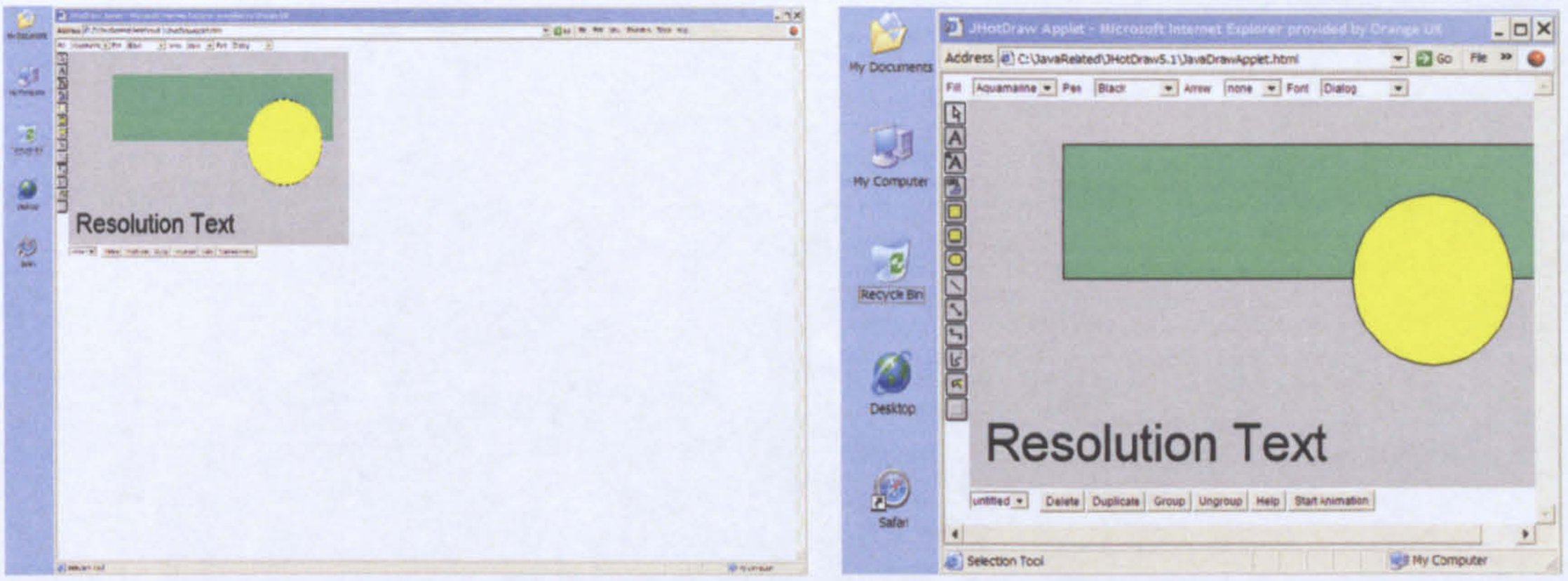


Figure 8-12: JHotDraw with screen resolution of 1920 * 1200 pixels to the left and 640 * 480 pixels to the right.

Figure 8-12 shows JHotDraw running in the same two different screen resolutions as Oea HotDraw above; on the right the view of JHotDraw is acceptable and can be used easily as the resolution of the screen (640 * 480 pixels) is close to the dimensions of the application's original size (560 * 370 pixels). While on the left, JHotDraw occupies a small portion of the screen with a smaller toolbar and figures making it quite difficult to use.

8.6.3 Scalability

This section demonstrates the scalability feature of Oea HotDraw compared to JHotDraw. The screen resolution is fixed to 1920 * 1200 pixels while the screen size (window size) is changed and the scalability of the workspace of Oea HotDraw is altered. It is noticeable that the toolbar on the left hand side of Oea HotDraw is always kept in proportion the screen size and does not change when the scale of the workspace is altered.

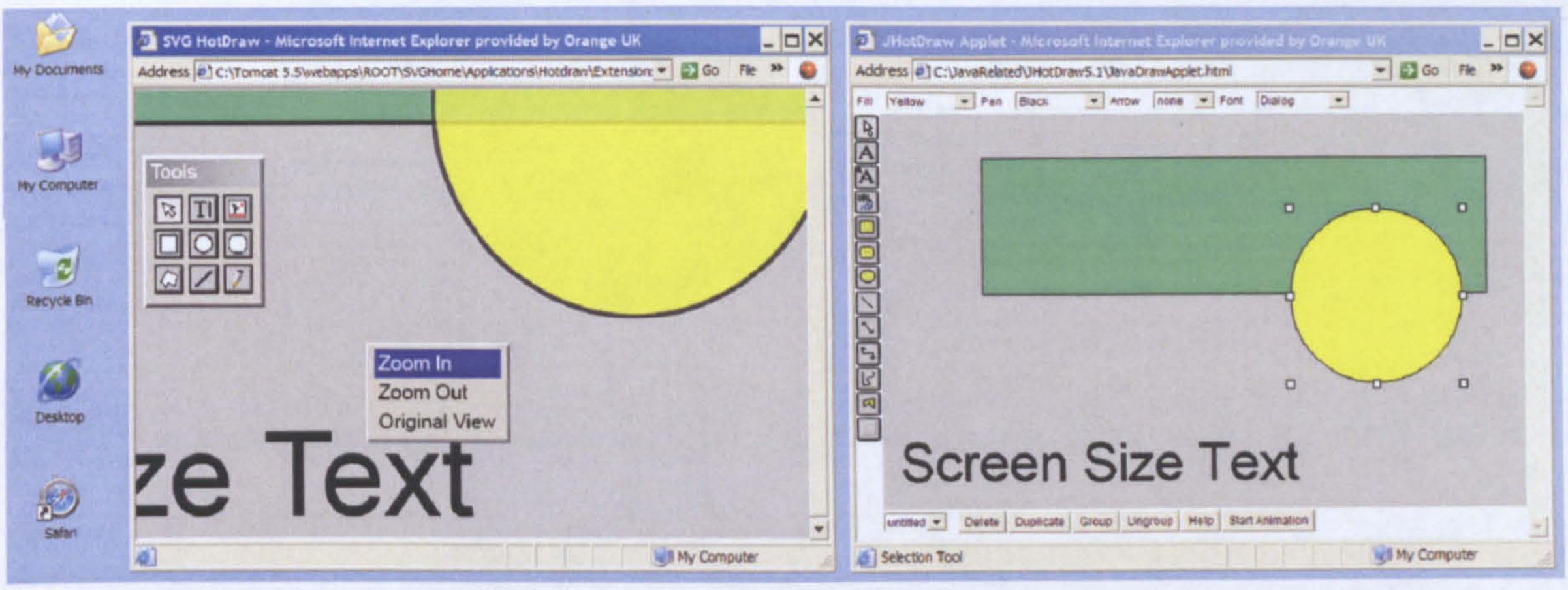


Figure 8-13: Zoom-in, Oea HotDraw (left) and JHotDraw (right) with screen size of 700 * 465 pixels.

Figure 8-13 shows Oea HotDraw after a zoom in action, making the figures on the workspace appear bigger (easier to handle), while Figure 8-14 showing Oea HotDraw after a zoom out actions makes the figures on the workspace appear smaller.

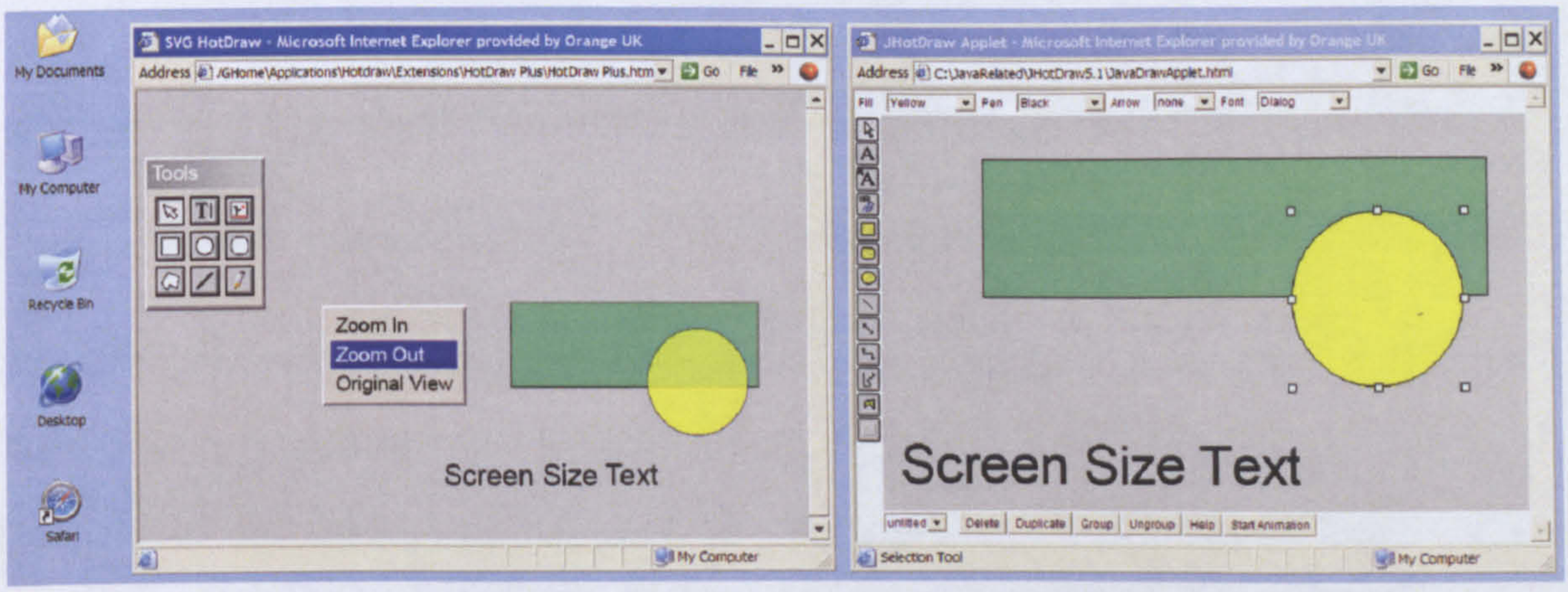


Figure 8-14: Zoom-out, Oea HotDraw (left) and JHotDraw (right) with screen size of 700 * 465 pixels.

Zoom in/out are two actions which are not supported in JHotDraw. Figure 8-15 shows Oea HotDraw running in a smaller screen size (407 * 264 pixels) while the workspace was scaled up.

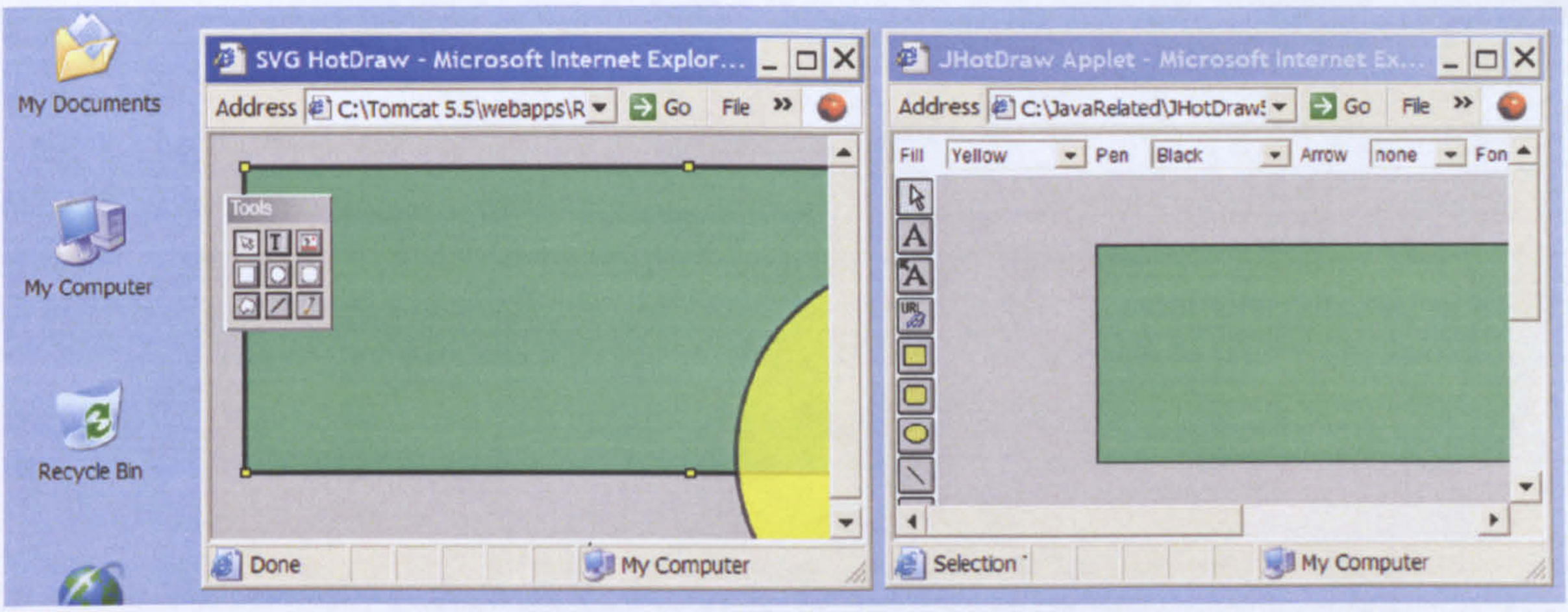


Figure 8-15: Zoom-in, Oea HotDraw (left) and JHotDraw (right) with screen size of 407* 264 pixels.

The zoom in/out facility in adaptable user interfaces is vital as it allows to have an overview of the drawings over the entire workspace (zoom out) or to do more specific detailed work (zoom in).

8.7 Summary

This chapter has demonstrated the use of the Oea framework as a new generic approach to constructing adaptable user interfaces that work on different device types. The Oea framework has been used to port JHotDraw – a rich and complex application - from a Java to a JavaScript/SVG environment. The resulting Oea HotDraw has the functionality of its predecessor JHotDraw, and is fully scalable and adaptable to the specifications of different device types (screen sizes and resolutions) unlike JHotDraw. This validates the principal aim of constructing adaptable user interfaces (see Section 1.4). Oea HotDraw runs quite fast on the computer used for the demonstration (Section 8.6). However, performance can be an issue with less powerful devices. This is because JavaScript implementations tend to be interpreters which make them slower to run. Nevertheless, attempts to develop faster JavaScript engines are underway (see Section 7.3.2) and soon SVG browsers will integrate the new engines to achieve better performance.

9

Use Case 2: SVG Annotator and the Wildfire Management Scenario Via RDF Data Modeling, the Oea Framework and Knowledge Base

The goal of this chapter is to address the final point of the Aim and Objectives presented in Section 1.4. It will describe the implementation of two systems (i.e. SVG Annotator and Wildfmt) in order to test and demonstrate the four-layer model described in Chapter 3.

9.1 Demonstration of the Four-layer Model

Figure 9-1 depicts the four-layer model, illustrating the technologies used in each layer. For the Presentation and Interaction layer (see Chapter 7) the Oea framework is used. CoRDF (see Chapter 6) has been used for the Knowledge Representation layer, while GAI (see Chapter 5) is used for the Group Communication layer. Gridkit and JGroups (see Chapter 4) have been used for the Middleware layer.

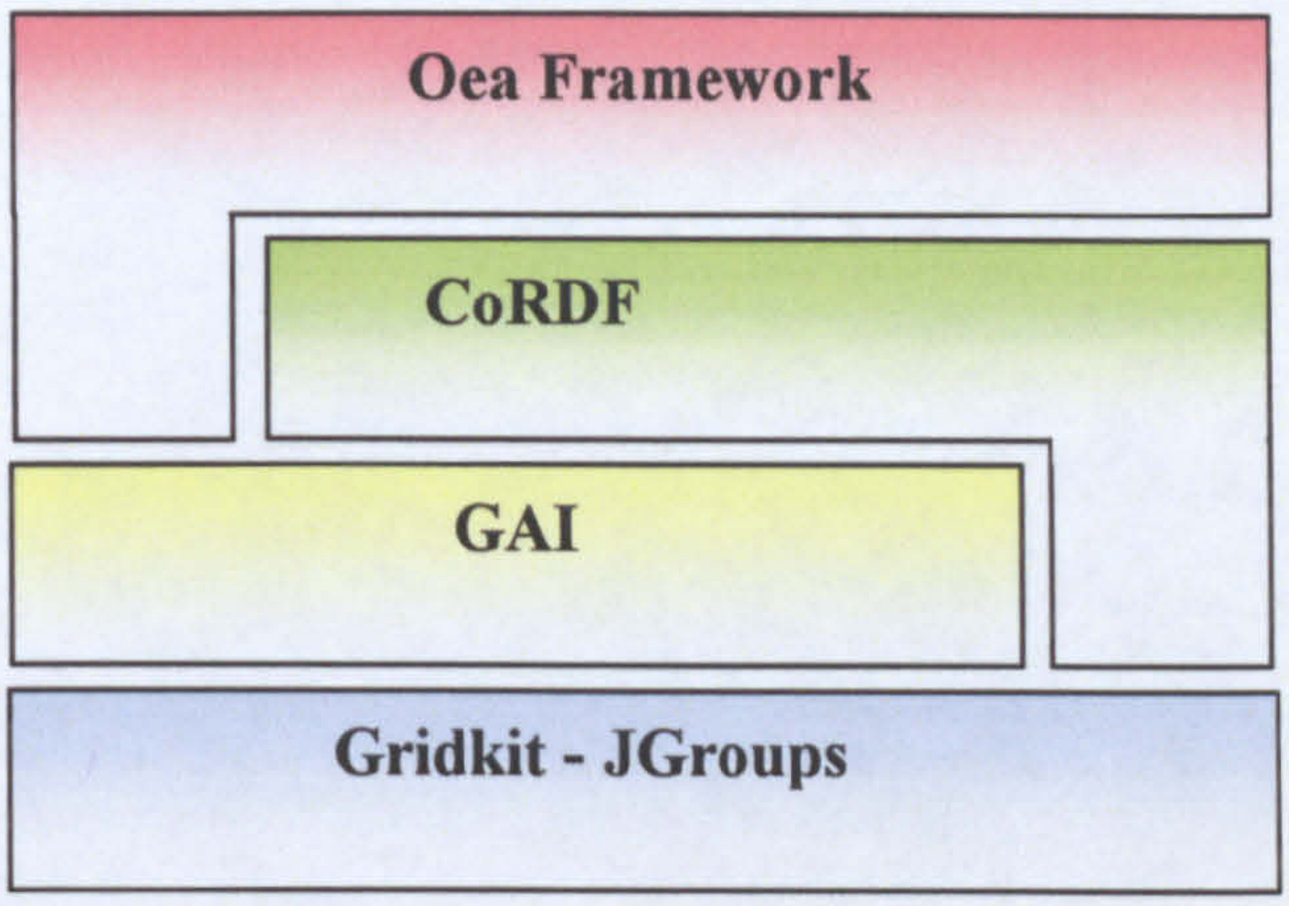


Figure 9-1: The four-layer model with the technologies used in each layer.

The SVG Annotator is a simple application used to annotate SVG images. It uses the Presentation and Interaction layer (i.e. Oea framework) and some aspects of the Knowledge Representation layer (i.e. CoRDF). The goal of including the SVG Annotator in this chapter was to exercise the user interface building capabilities of the Oea framework and to illustrate the use of Oea HotDraw to build an application with embedded RDF. This is followed by a more sophisticated application, Wildfmt. Wildfmt is built on top of the generic application CWE which was introduced in Section 1.3.1 and described in Section 9.3. Wildfmt demonstrates the Wildfire Management Scenario (see Section 1.2) and validates the entire four-layer model.

9.2 SVG Annotator

SVG Annotator allows the user to draw shapes and hand scribbles, import images and print text on the application workspace. These shapes, images and text are represented as SVG primitives and can be saved in an SVG document. The application allows annotations to be attached to these SVG primitives using RDF. The annotations are simple text added through the user interface (see Figure 9-2). The required shape is selected first by hovering the mouse cursor over the shape and clicking the left mouse button. The user clicks the right mouse button to display the context menu and chooses

Open Note Window. A new window opens allowing the user to type in text (see Figure 9-3). The window title displays the name of the current user, the date and time. This information alone with the text entered by the user will constitute the annotation.

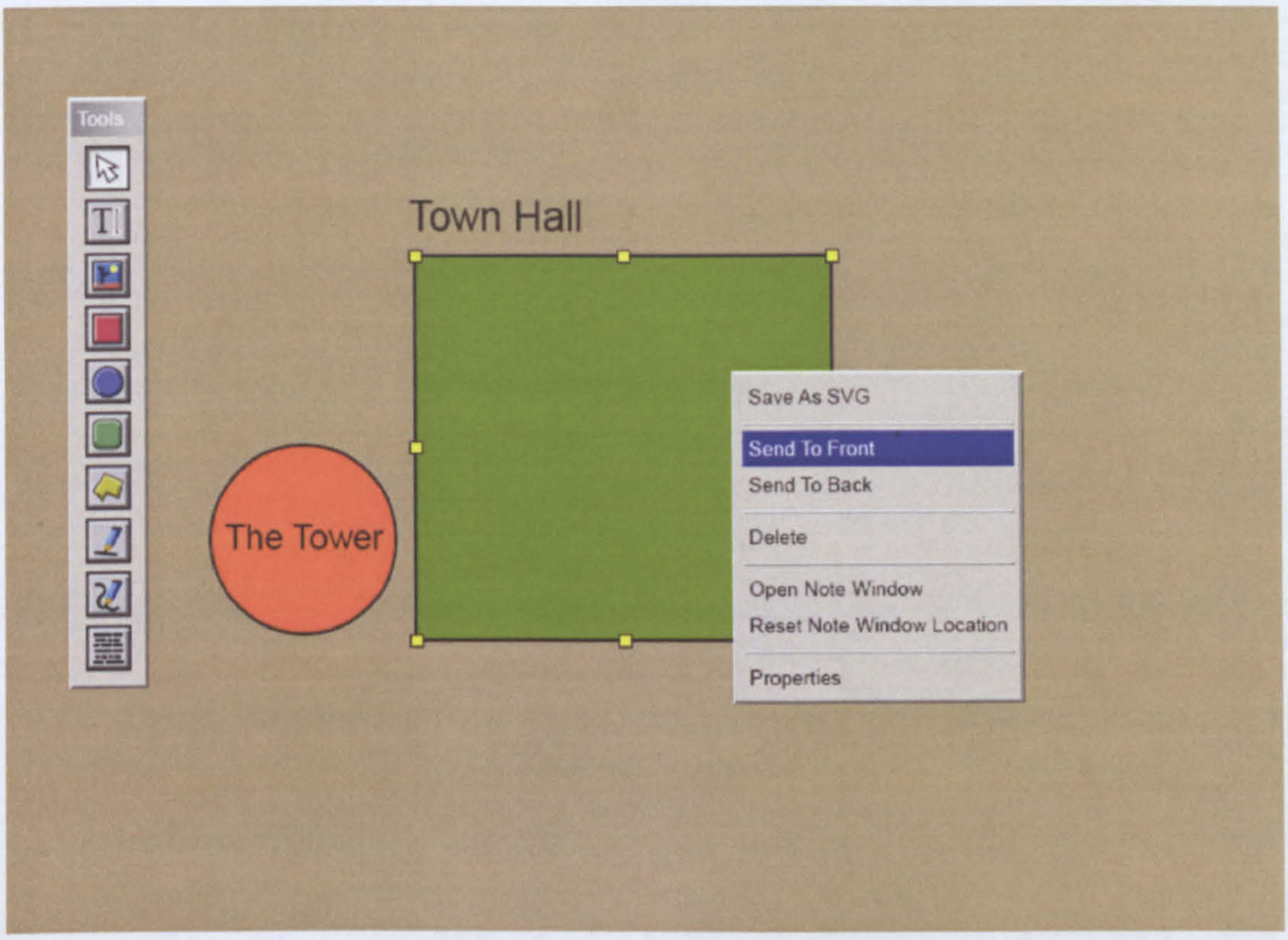


Figure 9-2: SVG Annotator User Interface.

Figure 9-2 shows a screenshot of the SVG Annotator, the toolbar (to the left) and two drawings (a circle and a rectangle, each with a text title) in different colours. Figure 9-3 shows a popup window used to allow users to add their text to the drawings.

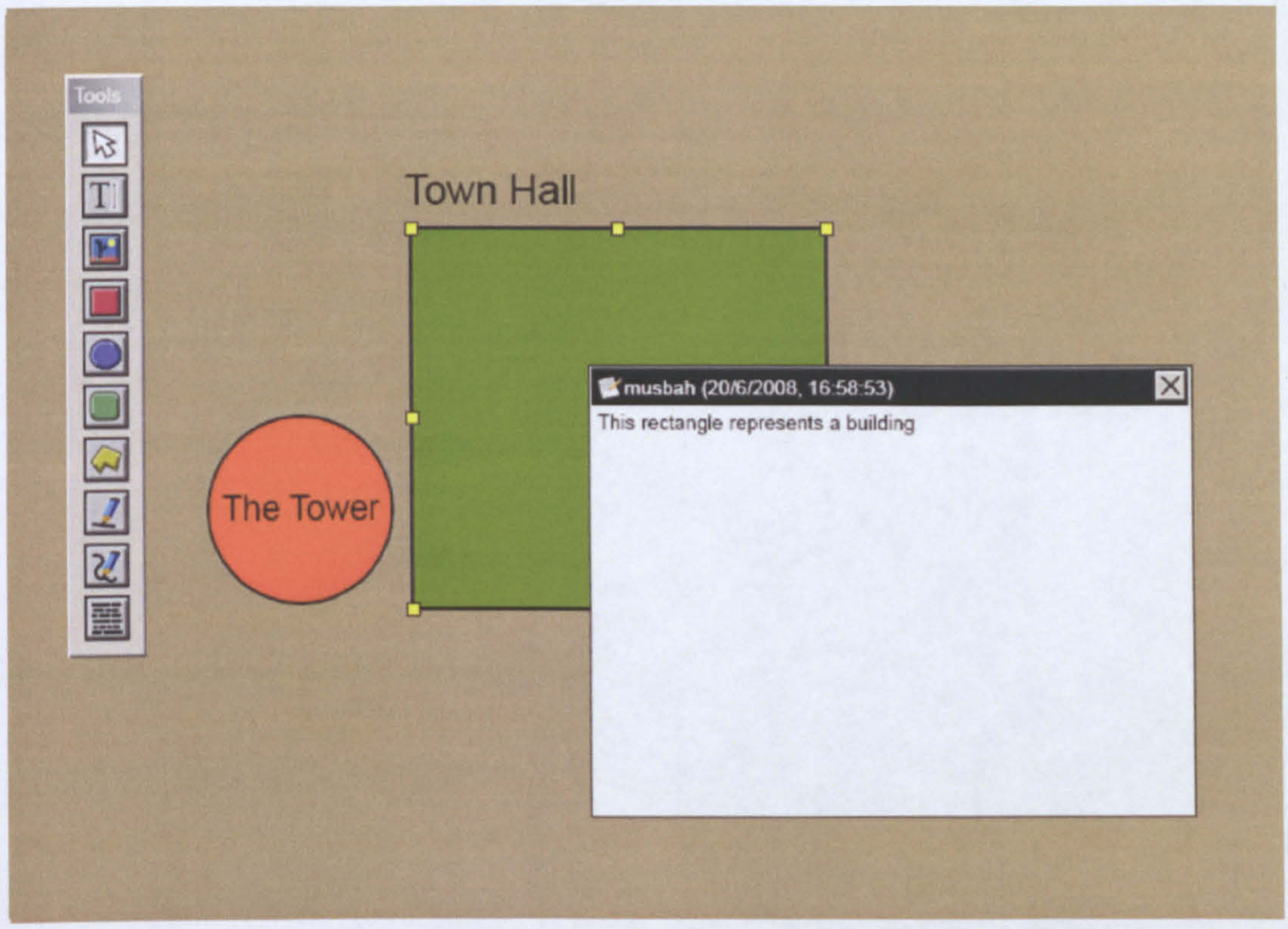


Figure 9-3: How RDF annotations are introduced into the SVG document.

Figure 9-4 shows another screenshot of the SVG Annotator with the view of the SVG source code of the SVG document being annotated to highlight the embedded RDF annotations.

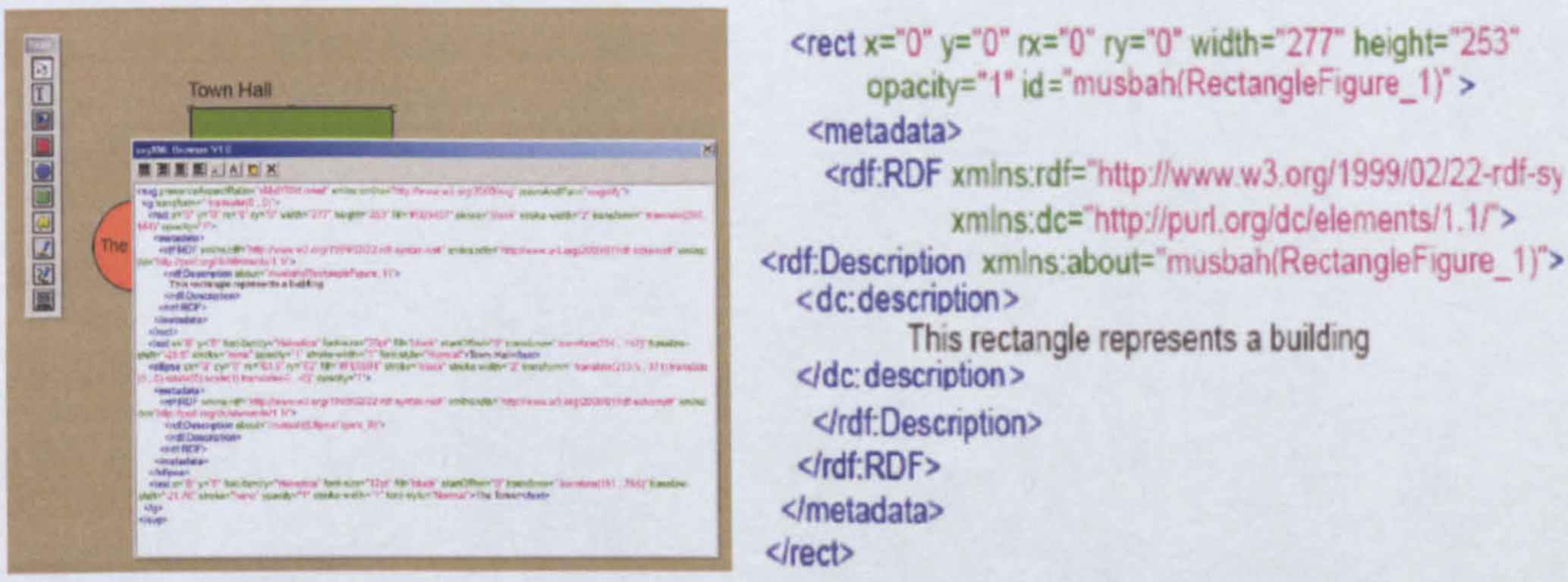


Figure 9-4: The XML code of the SVG document with the RDF annotations.

From Figure 9-2, when the user selects Save As SVG from the context menu, the SVG source code of the current SVG document will be displayed. Figure 9-4 shows the source code of the SVG document being edited by the SVG Annotator. Each annotation

is embedded within the SVG primitive that it belongs to as recommended by W3C. In Figure 9-4 (right), the size of the SVG element 'rect' is 277 * 253. It has an embedded RDF triple inside its 'metadata' element. The subject of the RDF triple is the rectangle (id: Musbah(RectangleFigure_1) and the predicate is dc:description, while the object is literal "This rectangle represent a building".

The SVG document can be saved as an SVG file and viewed in any SVG-enabled browser and the annotation can be retrieved and viewed or changed by anyone. More sophisticated annotations can be easily added such as adding semantics for example, the types of the shapes, and what they represent. This is discussed further in Section 9.4.3.

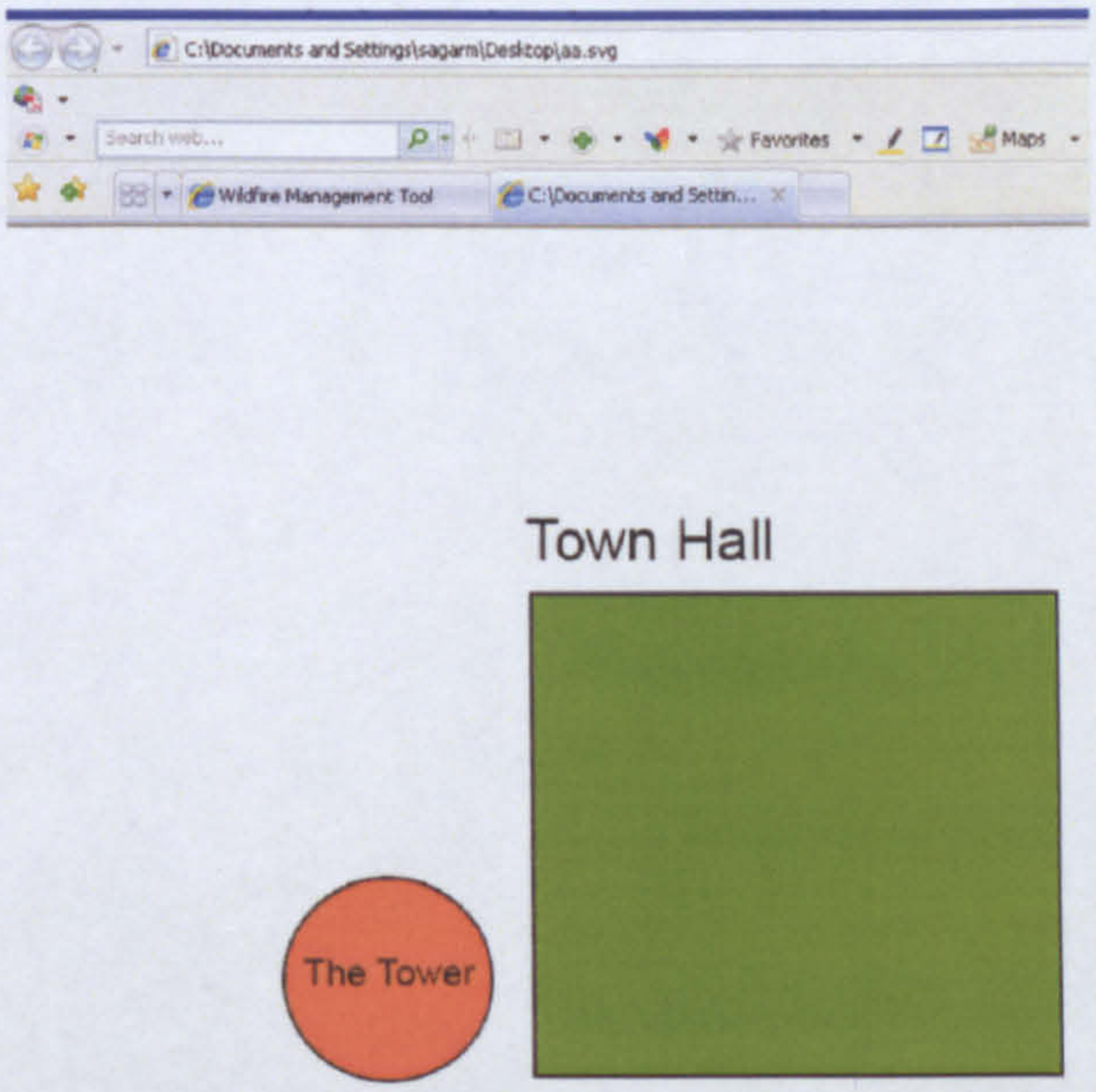


Figure 9-5: The SVG document generated by the SVG Annotator viewed in Internet Explorer using Adobe SVG viewer.

As shown in Figure 9-5, any SVG browser can view the saved SVG document. Because the SVG Annotator is actually written for the SVG environment it can run on any device type that supports SVG regardless of its specification, such as screen size and resolution.

The SVG Annotator has demonstrated the use of Oea HotDraw to build simple

Web-based applications with embedded RDF support.

9.3 CWE Application

This section presents a much larger case study. CWE (see Section 1.3.1) is a generic tool used as the basis for other more specific applications. Wildfmt for example, is a descendant application of CWE (see Section 9.4). The CWE architecture is sufficiently flexible to define a hierarchy of layered knowledge that can be used to express information on different aspects of CWE applications (e.g. structure, users, custom annotations, other data, etc.). CWE is designed to be used for sharing ideas, issuing commands and helping in decision making and post event analysis.

Additional features are easily built on the CWE to demonstrate advantages of our new model as in the case of Wildfmt application where support to present maps is overlaid with the users' locations, sensor information, fire prediction information, commands and annotations made by the participants of the scenario (controllers and fire fighters) are provided.

The development method of CWE provides advanced features to users over the classic bitmap approach such as superior visual presentation using vector graphics, collaborative sessions recording and replaying, data and information querying. This is due to the use of an RDF data model which enables the application to hold information about the people making annotations, the annotations themselves, and the history of annotations in one consistent structure.

The roadmap towards CWE, starts with Oea HotDraw (see Chapter 8) and evolves into a sophisticated exemplar for ACTs designed and developed using Web-based technologies. The screenshot in Figure 9-6 shows a screenshot of the early stages in evolving Oea HotDraw to CWE.

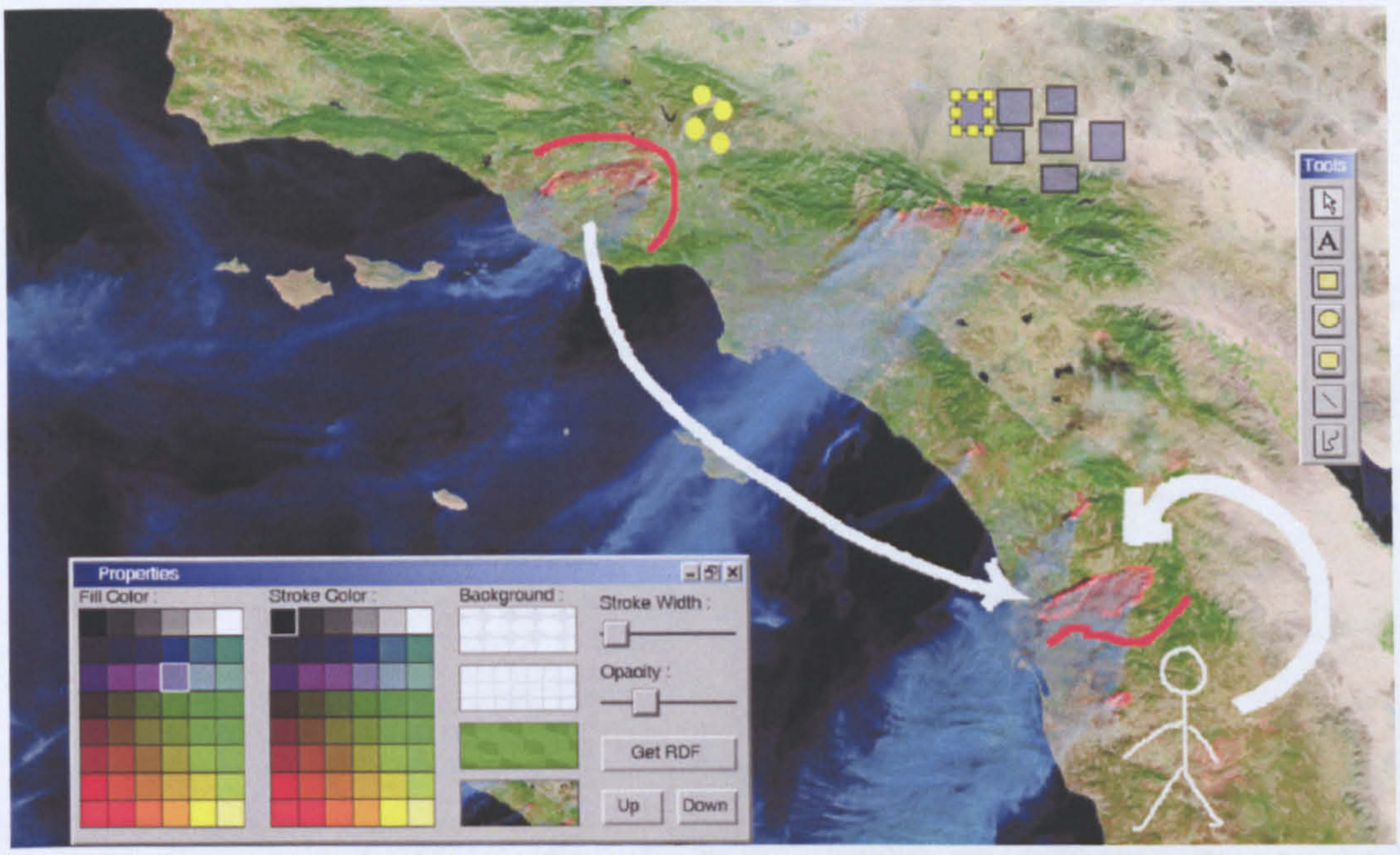


Figure 9-6: CWE beta

The toolbar in the beta version of CWE (shown in Figure 9-6, on the right-hand-side of the screen) allows the user to draw simple shapes. The property of those shapes such as the fill colour, stroke colour, stroke width and shape opacity can be changed. The background images can also be altered, where in the later stages of developing CWE, real map of the Wildfire site in an SVG format can be used.

9.3.1 Data Model

This section will introduce the construction of the CWE data model, developed following CoRDF described in Chapter 6. RDFPIDM is used to define the basic data types which are the foundation for the work on CWE. These types are used to describe the workspace of CWE, and the annotations (drawings) attached to it. Initially, there are no specific meanings to these annotations. They will simply describe the desktop and the drawings attached to it in RDF terms. Additional semantics can be added at the application level as we will see later in this chapter (see Section 9.4.3).

The XML fragment below shows the RDFS used to establish the basic drawings

of CWE. The application desktop of CWE is considered as the workspace (type Workspace) which has several contexts (type Context), where annotations – which represent SVG primitive shapes can be attached. The following is the RDFS for CWE. Some details of this schema were left out for the sake of clarity (not key ideas):

```
<?xml version="1.0" standalone="no"?>
<rdf:RDF
  xmlns:rdf     = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs    = "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:svgcwe  = "http://www.openoverlays.com/svgcwe#"
  xml:base = "http://www.openoverlays.com/svgcwe#">

<!-- ******* CWE/Group Level ************************** -->
<rdfs:Class rdf:ID="Workspace"/>
<rdfs:Class rdf:ID="Context"/>
<rdfs:Class rdf:ID="HistoryNode"/>

<rdf:Property rdf:ID="owned-by"/>

<rdf:Property rdf:ID="command">
 <rdfs:domain rdf:resource="#HistoryNode"/>
 <rdfs:range rdf:resource=" http://www.w3.org/2000/01/rdf-
schema#Literal"/>
</rdf:Property>

<!-- === SVG Classes == -->
<rdfs:Class rdf:about="http://www.w3.org/2000/svg/node"/>
<rdfs:Class rdf:about="http://www.w3.org/2000/svg/rect"/>
<rdfs:Class rdf:about="http://www.w3.org/2000/svg/ellipse"/>
<rdfs:Class rdf:about="http://www.w3.org/2000/svg/path"/>
<rdfs:Class rdf:about="http://www.w3.org/2000/svg/line"/>
<rdfs:Class rdf:about="http://www.w3.org/2000/svg/polygon"/>
<rdfs:Class rdf:about="http://www.w3.org/2000/svg/image"/>

<rdfs:Class rdf:about="http://www.w3.org/2000/svg/text"/>
 <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/svg/node"/>
</rdfs:Class>

<!-- == SVG Properties == -->
<rdf:Property rdf:about="http://www.w3.org/2000/svg/attribute"/>
```

```
<rdf:Property rdf:about="http://www.w3.org/2000/svg/id"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/x"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/y"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/width"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/height"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/fill"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/stroke"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/stroke-width"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/opacity"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/points"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/x1"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/y1"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/x2"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/y2"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/font-family"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/font-style"/>
<rdf:Property rdf:about="http://www.w3.org/2000/svg/content"/>
<rdf:Property rdf:about=" http://www.w3.org/1999/xlink#href"/>

<rdf:Property rdf:about="http://www.w3.org/2000/svg/font-size">
 <rdfs:subPropertyOf
rdf:resource="http://www.w3.org/2000/svg/attribute"/>
 <rdfs:range rdf:resource=" http://www.w3.org/2001/XMLSchema#float"/>
 <rdfs:domain rdf:resource="http://www.w3.org/2000/svg/text"/>
</rdf:Property>
</rdf:RDF>
```

Annotations can be created, modified or deleted. The trace of changes is attached to the history node (class HistoryNode). The property 'owned-by' is used in different ways for linking RDF statements. For example, a Context instance (instance of type Context) is owned-by a Workspace instance or a HistoryNode instance is owned-by a Context instance. The 'command' property is used with HistoryNode to indicate its function (i.e. Create, Update, Delete, etc.) with respect to the SVG content it is attached to. The remaining schema is the SVG representation in RDF. The first class is the 'node', a superclass of all other SVG annotations and the first property is 'attribute', the super-property of all SVG related attributes. The property 'font-size' applies to the resource of svg:text type only and its value is of type float.

9.3.2 Architecture and Implementation.

The previous section has shown the use of the RDFPIDM method to develop the CWE data model. This section introduces the use of the KB and GAI. The KB (see Section 6.4) is used here to enable CWE to store and share data. A number of SVG viewers were available to run CWE including Batik [Batik] (see Section 7.1), to enable the Oea HotDraw to run in the Java environment. As explained earlier, the code of CWE (originally Oea HotDraw) is written mainly in JavaScript, while Java was only used as a wrapper and to link with other parts of the application. Such a complex mix of the two languages to develop CWE has produced the ideal situation in which to apply the RDFPIDM method.

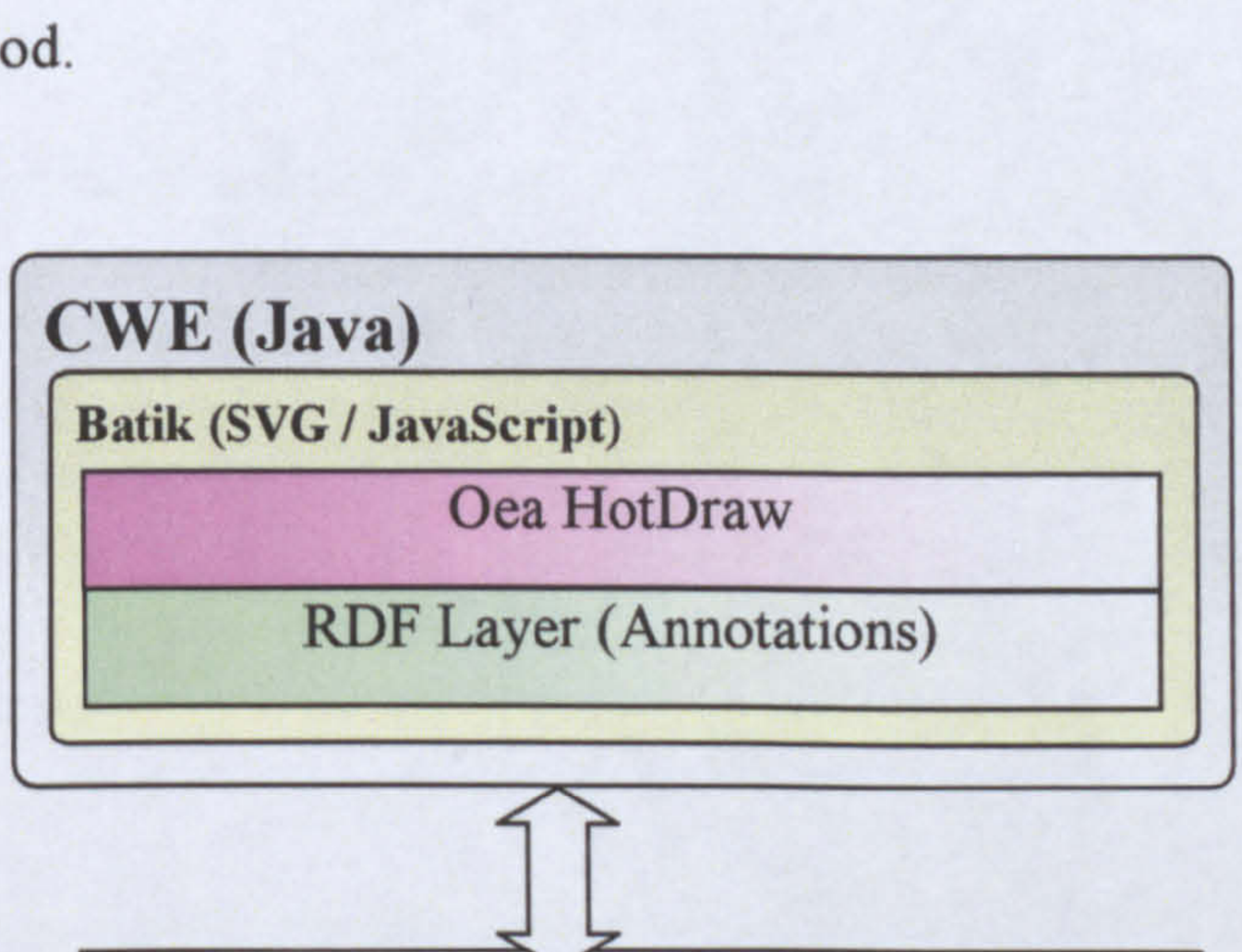


Figure 9-7: CWE Architecture.

As shown in Figure 9-7, CWE is primarily defined in SVG and JavaScript which runs in a Java environment. CWE uses GAI to communicate with other instances of CWE and to access the KB.

The communication between the JavaScript code running in Batik and Java is transparent. Methods in the main class of CWE in Java (SVGCWE) can be called from the JavaScript environment and vice versa. For example, to send a message to the group, the method *sent* is in scope anywhere in the JavaScript environment and can be called directly. Public attributes of the main class are also accessible from JavaScript. For example, *simulationProxy* and *dataRepositoryProxy* objects which are proxy nodes to the fire simulation (an external application which runs as a fire simulator, see Chapter10) and the data repository services respectively can also be accessed from JavaScript. JSVGCanvas is the name of the class provided by Batik which is responsible for displaying and running SVG applications. This class provides direct access to its local JavaScript interpreter. Communication with JavaScript code from Java is achieved using the *evaluation* method. This method takes a string which contains the JavaScript code that will make the appropriate calls to internal functions or objects in SVGCWE.

GAI is used for managing group communication such as join groups, leave groups, sending and receiving messages. CWE uses the KB to store the RDF data and to register SPARQL queries for the data that it is interested in. Updates from the KB for the registered queries are then passed on to SVGCWE whenever they become available. Direct queries can also be made.

9.3.3 Enabling Collaboration

At this stage CWE is ready for collaboration. CWE establishes collaboration by using the GAI. The GAI will enable instances of CWE to communicate and collaborate together so that when a new annotation is added to the workspace of a CWE instance, this annotation will be stored in the KB and sent to all other instances of CWE. Figure 9-8 illustrates this.

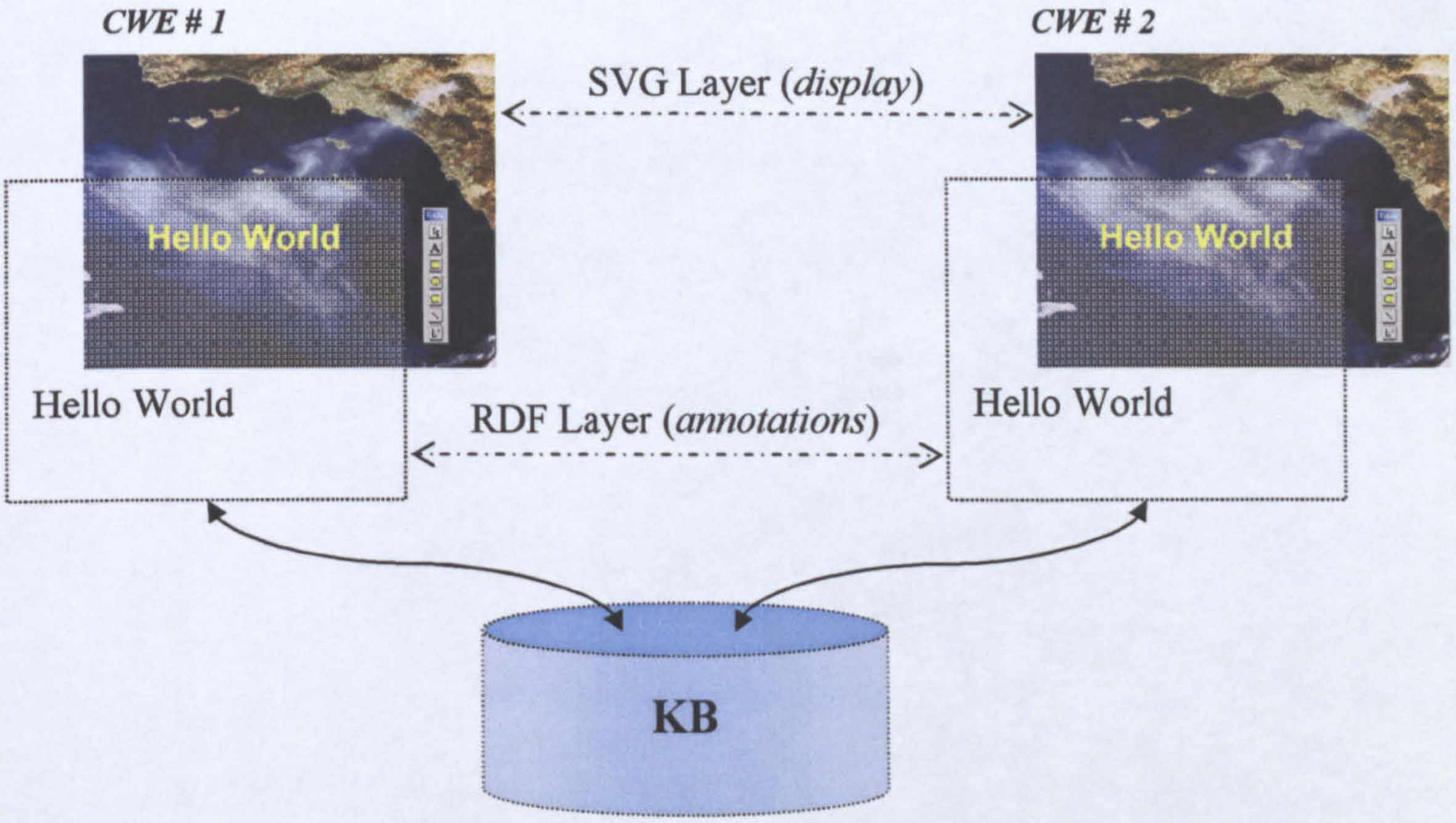


Figure 9-8: Two instances of CWE in collaboration (other details such as GAI is not shown here) .

Figure 9-8 shows two instances of CWE with a single text annotation. The annotation is part of the RDF layer which is visualised in CWE using SVG. The annotation is stored in the KB to be used later for querying or other activities.

Figure 9-9 shows the RDF diagram for the workspace stored in the KB. The blue ellipses are resources and each represents part of the annotations created in the CWE. For example, http://svgcwe#workspace_1/ is the URI of the CWE Workspace resource. This resource is of type svgcwe:Workspace. The Workspace has one context instance which has one history node attached to it. The history node records the action 'create' of an SVG text element.

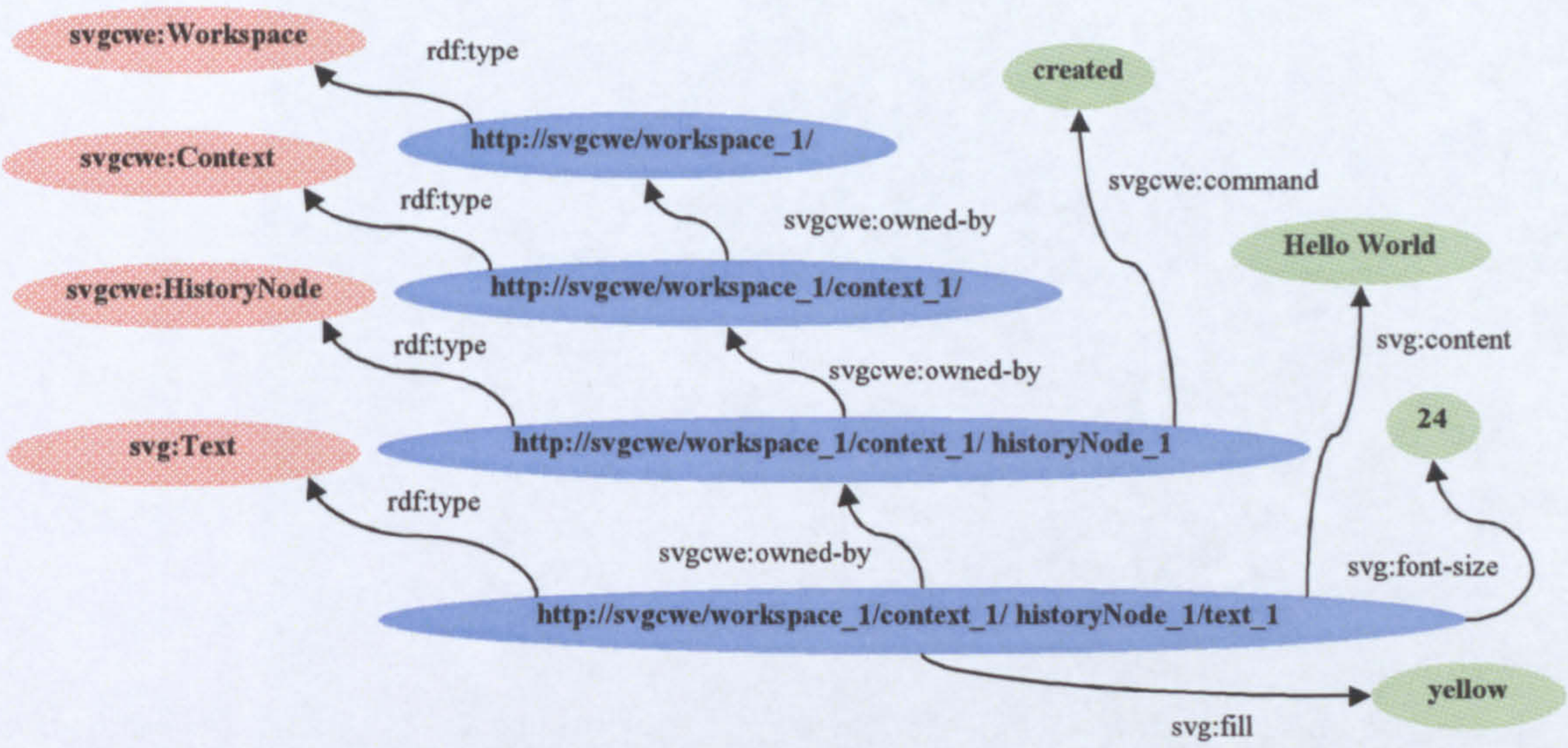


Figure 9-9: The representation of Text annotation on the workspace

The text element reads "Hello World" and has the attribute 'font-size' set to '24'. The value '24' is interpreted as of type float using the XML Schema. The fill colour of the text element is set to 'yellow'

9.3.4 Data Querying

All the annotations associated with the workspace of CWE are stored in the KB. This data can be queried using SPARQL. Figure 9-10, shows an interface to CWE that allows the user to write SPARQL queries freely. This interface is presented for demonstration purpose only. The query in the figure returns only the SVG rectangle annotations from the KB and displays them on the workspace. Information about the creator of the annotation, time and other style properties can also be obtained if required.

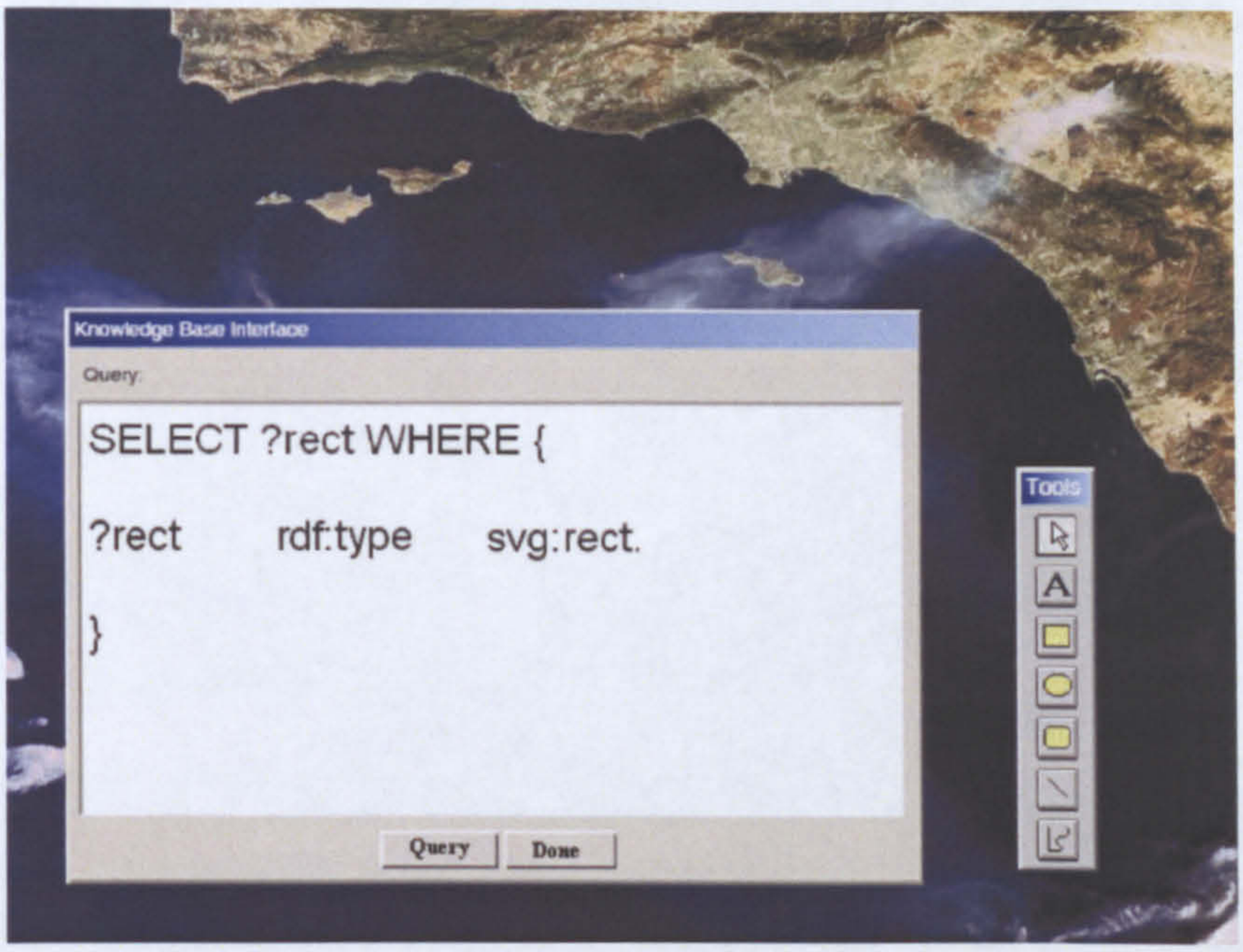


Figure 9-10: Query form in CWE.

Below is an example of a query which will return all types of annotations supported by CWE from the KB:

```
SELECT ?type ?command ?id ?x ?y ?w ?h ?arcWidth ?arcHeight ?fillColor
       ?frameColor ?frameWidth ?opacity ?points ?date ?x1 ?y1 ?x2 ?y2
       ?url ?fontName ?fontSize ?fontStyle ?text

WHERE {
   ?annotation wildfmt:OwnedBy <http://svgcwe/workspace_1/context_1/>.
   ?annotation rdf:type ?type.?hnode wildfmt:OwnedBy
   ?annotation; wildfmt:Command ?command;
                svg:id ?id;
                svg:x ?x;
                svg:y ?y;
                svg:width ?w;
                svg:height ?h;
                svg:opacity ?opacity;
                dc:Date ?date.

OPTIONAL { ?hnode svg:fill ?fillColor; svg:stroke ?frameColor;
                svg:stroke-width ?frameWidth.}
OPTIONAL { ?hnode svg:rx ?arcWidth; svg:ry ?arcHeight.}
OPTIONAL { ?hnode svg:points ?points.}
OPTIONAL { ?hnode svg:url ?url.}
```

```
OPTIONAL { ?hnode svg:font-family ?fontName; svg:font-size ?fontSize;
            svg:font-style ?fontStyle.}
OPTIONAL { ?hnode svg:content ?text.}
OPTIONAL { ?hnode svg:x1 ?x1; svg:x2 ?x2; svg:y1 ?y1; svg:y2 ?y2; .}
}
OrderBy ?date
```

The parameters returned from this query include: the type of the annotation, the history node command (create, delete or update) the annotation's location, content and style properties. Properties which are not common to all annotations such as 'fill color', 'font-size' or 'url' are returned optionally using the OPTIONAL statement. The queries are returned in order of date by default (older annotations returned first) so that the state of the workspace can be reconstructed accurately. For example, if the user initially created a rectangle annotation and later on modified its size, the order this is stored within the KB is as a history node with the 'create' command first, and second using the 'modify' command. The query will also return the 'create' annotation first followed by the 'modify'.

9.4 Wildfire Management Tool

CWE provides the foundation on which other more complex applications can be built. Wildfmt is an application built on the principal ideas of CWE. It serves to aid fire fighters and controllers to fight fires as explained in the applications scenario (see Section 1.2). In addition to the generic annotations provided by CWE, Wildfmt provides annotations with specific meanings (application level semantics) and this section illustrates this. Wildfmt also provides examples of how to incorporate other sources of data such as location information (GPS) and fire simulation using RDFPIDM.

Wildfmt uses login information to allow access, provide information about groups and members of groups, associations between annotations, groups and users. All of these add-ons demonstrate the extensibility and flexibility of CoRDF used here.

Figure 9-11 shows the architecture of Wildfmt.

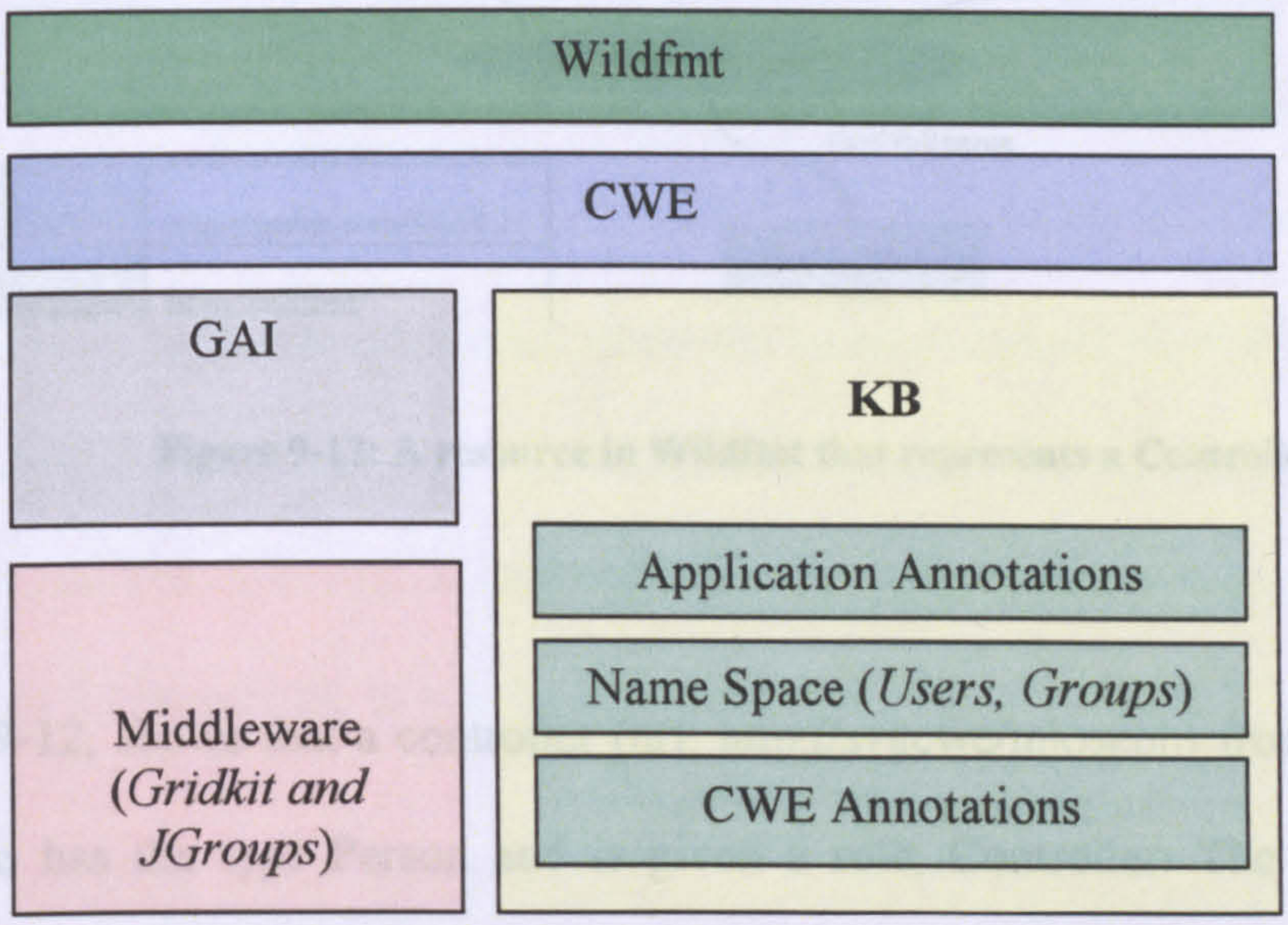


Figure 9-11: Wildfmt Architecture.

The KB is more structured in Wildfmt than in CWE. Two more layers of annotations are added: the application annotations and the name space. The subsequent sections will describe these new RDF layers in more detail.

9.4.1 Name Space

As was mentioned in the previous section, CWE facilitates group work; however, it lacks a data model to support this added dimension. This section will use RDFPIDM to introduce the notions of users, persons and groups to support collaborative activities within Wildfmt.

9.4.1.1 Real World Level

Personnel who are involved in the Application Scenario (controllers and fire fighters) are identified by URIs in the RDF model of Wildfmt. For example, http://svgcwe/simon, http://cms.brookes.ac.uk/staff/MusbahSagar and http://www.example.com/john are the kind of URIs accepted in this model to identify a person.

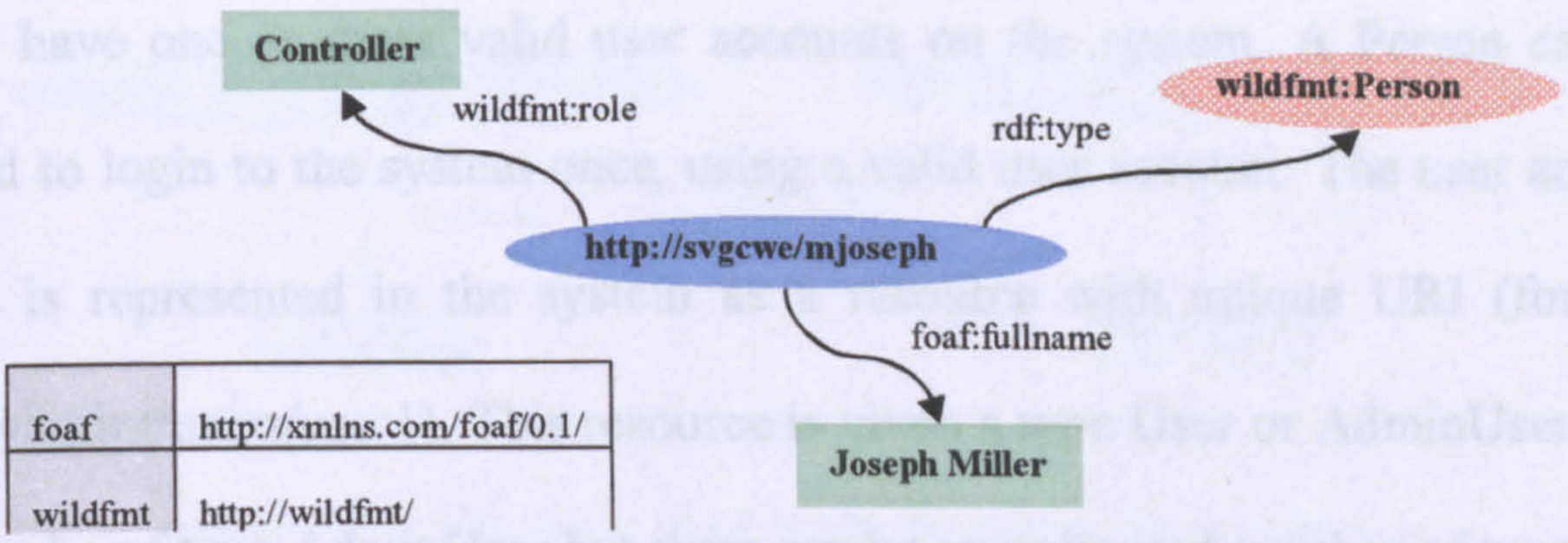


Figure 9-12: A resource in Wildfmt that represents a Controller.

Figure 9-12, shows that a controller (url, http://svgcwe/mjoseph) from the Application Scenario has the type Person and is given a role, Controller. The term Person is a generic term used to refer to fire fighters and controllers in the sections below. The FOAF ontology (see Section 6.2) has been used to associate a name to the person using the foaf:fullname attribute.

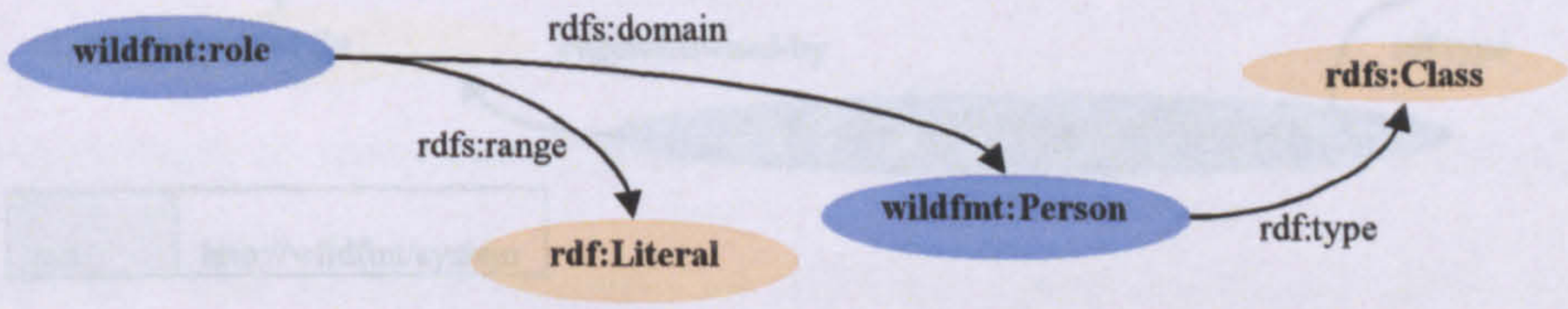


Figure 9-13: RDFS for the real world representation in Wildfmt

The RDFS for describing persons in the Wildfmt namespace is illustrated in Figure 9-13. Person is a class in RDFS, the role is a property of Person (takes Person for the domain); and it takes a string for the range (set to 'Controller' or 'Fire fighter').

9.4.1.2 System Level

In the Application Scenario, controllers and fire fighters of type Person are expected to use Wildfmt for communication and collaboration. Therefore they need to gain access to Wildfmt through their PCs, laptops or PDAs. Persons can access the system as users

if they have one or more valid user accounts on the system. A Person can only be allowed to login to the system once, using a valid user account. The user account of a Person is represented in the system as a resource with unique URI (for example: http://wildfmt/users/user1). This resource is given a type User or AdminUser. Only one user can be of type AdminUser but there can be an unlimited number of type User. The AdminUser's main duty is to create user accounts for the system and to give various permissions. For instance, a user that is given GroupAdmin permission can create, modify or delete groups, as will be explained later.

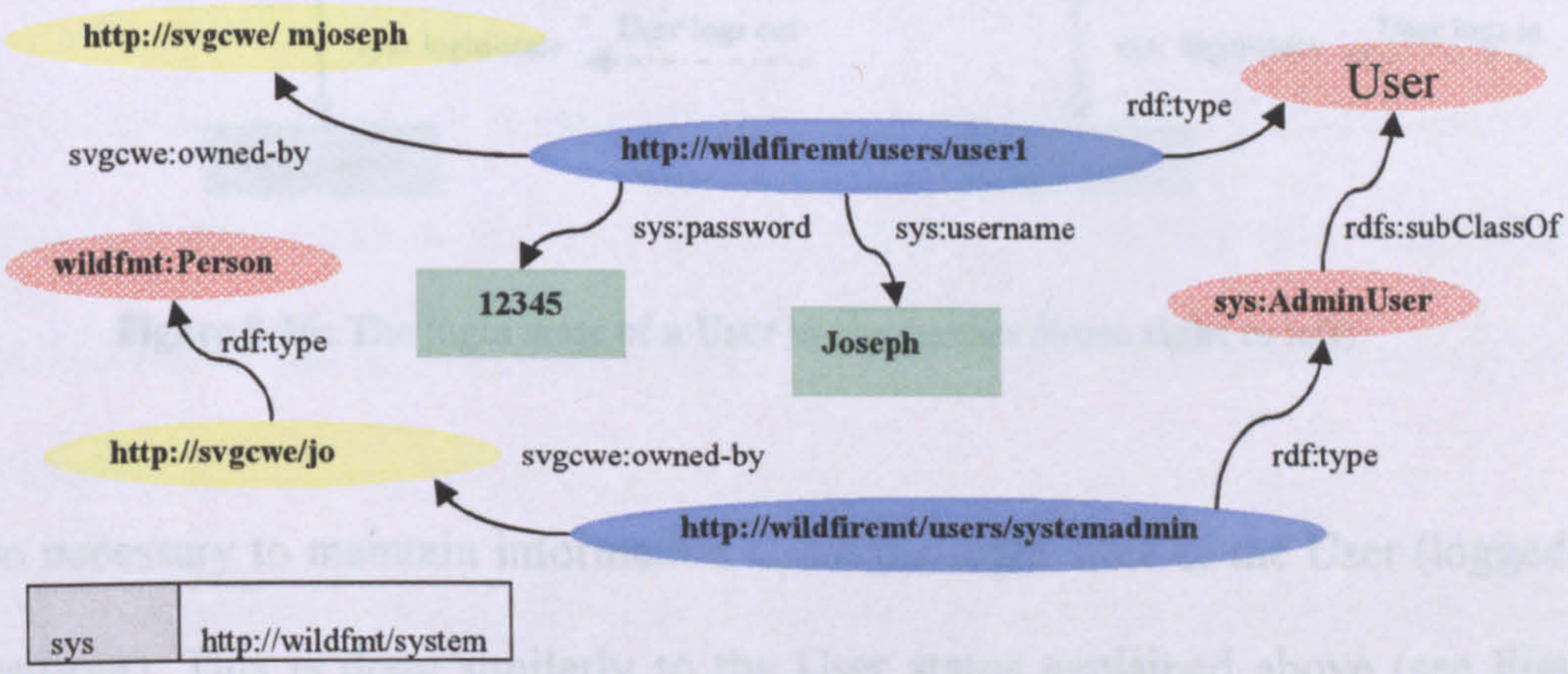


Figure 9-14: Example of the representation of a User and an AdminUser in the system

Figure 9-14 shows the relationship between a User and a Person in the RDF model. Users once created by the AdminUser, cannot be deleted. This is because there will be other assertions and statements in the RDF model that refer to the users of the system (as Subject or Object of a triple). Figure 9-15, shows an alternative course to delete a user.

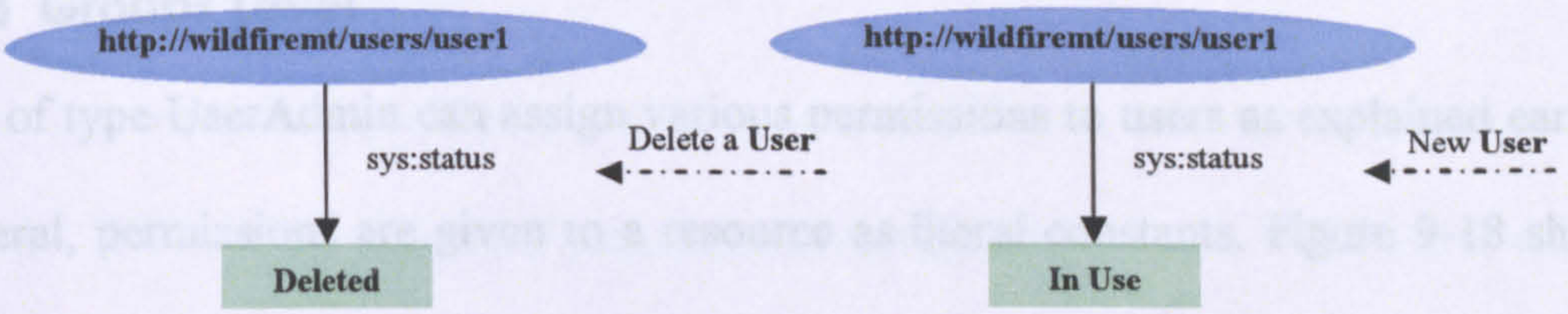


Figure 9-15: The status of a User in the system (from right to left)

When a new User is added to the system an annotation is added to the RDF model to assert that the User is 'In Use'. When the User is deleted, the status assertion changes the value to 'Deleted'.

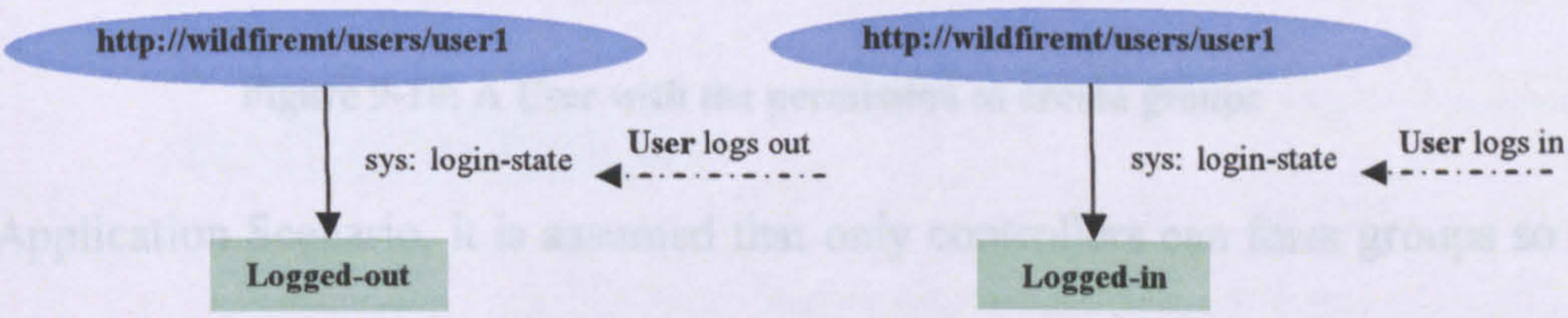


Figure 9-16: The login state of a User in the system (from right to left)

It is also necessary to maintain information about the login state of the User (logged-in or logged-out). This is done similarly to the User status explained above (see Figure 9-16).

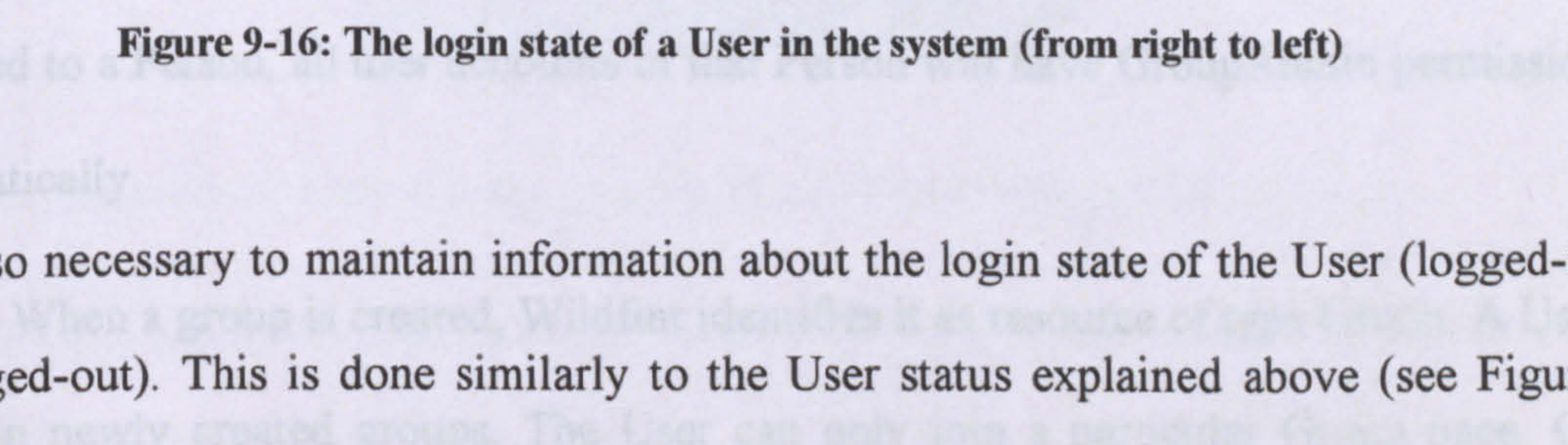


Figure 9-17: RDFS for the system level

Figure 9-17 shows the RDFS for the system level data model.

9.4.1.3 Groups Level

A user of type UserAdmin can assign various permissions to users as explained earlier. In general, permissions are given to a resource as literal constants. Figure 9-18 shows the relationship between a User and GroupAdmin permission. When a User is granted GroupAdmin permission by UserAdmin, the user gains the ability to create and modify groups.

Figure 9-18: A User with the permission to create groups

In the Application Scenario, it is assumed that only controllers can form groups so that controllers and fire fighters can join them. Therefore, when the role Controller is assigned to a Person, all user accounts of that Person will have GroupAdmin permission automatically.

When a group is created, Wildfmt identifies it as resource of type Group. A User can join newly created groups. The User can only join a particular Group once. On joining, a User becomes a member of that group. The relationship between a User and a Group can be expressed in RDF as shown in Figure 9-19.

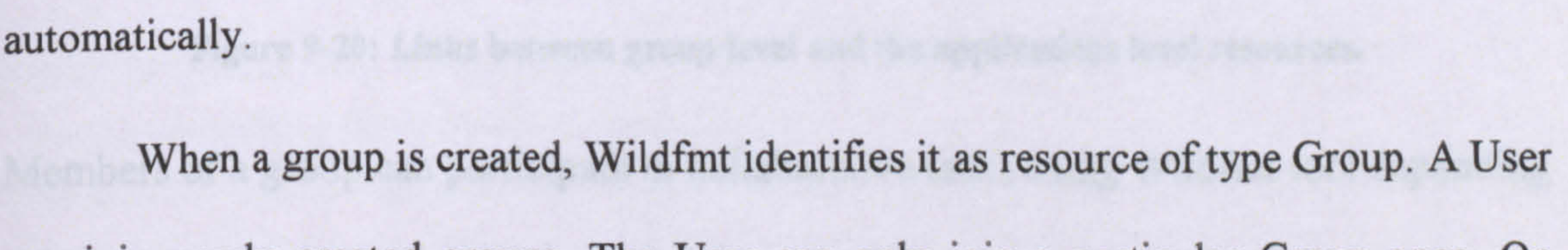


Figure 9-19: The relationship between a User and a Group

Figure 9-19 shows that the User http://wildfiremt/users/user1 is a member of two Groups: http://wildfiremt/group/g1 and http://wildfiremt/group/g2.

9.4.1.4 Application Level

When a User with GroupAdmin permission creates a Group, the system automatically creates a CWE workspace (http://svgcwe/workspace_1, see Figure 9-20) and a default context (http://svgcwe/workspace_1/context_1/ , see Figure 9-20) which the system then allocates to the newly created Group.

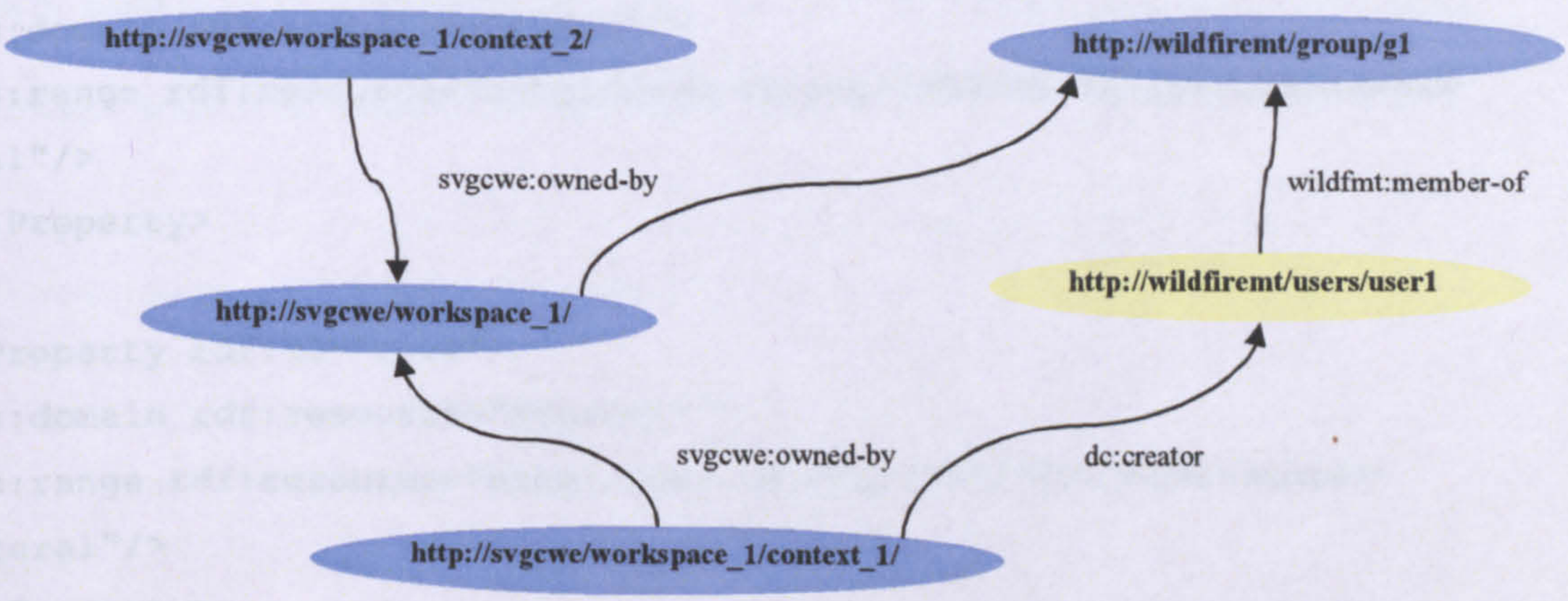


Figure 9-20: Links between group level and the applications level resources.

Members of a group can participate in collaborative tasks using Wildfmt tool depending on the permissions that are given to them.

9.4.1.5 RDFS

The complete RDFS of the Name Space data model is presented here:

```
<?xml version="1.0" standalone="no"?>
<rdf:RDF
  xmlns:svgcwe  = "http://www.openoverlays.com/svgcwe#"
  xmlns:wildfmt = "http://www.openoverlays.com/wildfmt#"
  xml:base = "http://www.openoverlays.com/wildfmt#">

<rdfs:Class rdf:ID="Person"/>
<rdfs:Class rdf:ID="User"/>

<rdfs:Class rdf:ID="AdminUser">
 <rdfs:subClassOf rdf:resource="#User"/>
```

```
</rdfs:Class>

<rdfs:Class rdf:ID="Group"/>

<rdf:Property rdf:ID="username">
 <rdfs:domain rdf:resource="#User"/>
 <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="password">
 <rdfs:domain rdf:resource="#User"/>
 <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#
Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="role">
 <rdfs:domain rdf:resource="#Person"/>
 <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="status"> <!-- i.e.In Use,Deleted,Permanent  -->
 <rdfs:domain rdf:resource="#User"/>
 <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="login-state"> <!--  Logged In, Logged Out -->
 <rdfs:domain rdf:resource="#User"/>
 <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="permission"><!-- GroupAdmin -->
 <rdfs:comment>Give a user extra permissions (i.e. GroupAdmin). Only
AdminUser can assign new permissions</rdfs:comment>
 <rdfs:domain rdf:resource="#User"/>
 <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Literal"/>
</rdf:Property>
```

```
<rdf:Property rdf:ID="member-of">
 <rdfs:domain rdf:resource="#User"/>
 <rdfs:range rdf:resource="#Group"/>
</rdf:Property>
</rdf:RDF>
```

9.4.2 Authentication and Authorization

Wildfmt follows a simple application-level model for authentication and authorization.

Information about users and groups are stored as RDF assertions as explained earlier. Each participant/person identified by a URI in the RDF repository can have one or more user accounts, each with a username and password.

```
<svgcwe:Person rdf:about="http://www.wildfmt.com/staff#p00123568">
    <vcard:FN>Musbah Sagar</vcard:FN>
</svgcwe:Person>

<svgcwe:Person rdf:about="http://www. wildfmt.com/staff#p00123565">
    <vcard:FN>David Duce</vcard:FN>
</svgcwe:Person>

<svgcwe:User rdf:ID="user0771">
 <svgcwe:owned-by
rdf:resource="http://www.wildfmt.com/staff#p00123565"/>
 <svgcwe:username>david</svgcwe:username>
 <svgcwe:password>david</svgcwe:password>
</svgcwe:User>

<svgcwe:AdminUser rdf:ID="user5151">
 <svgcwe:owned-by
rdf:resource="http://www.wildfmt.com/staff#p00123568"/>
 <svgcwe:username>musbah</svgcwe:username>
 <svgcwe:password>musbah</svgcwe:password>
</svgcwe:AdminUser>
```

The above RDF fragment creates two user accounts for each member of staff, both accounts with a username and password.

As explained earlier, there are two types of user accounts, User and AdminUser.

An AdminUser is a unique user who has special permissions by default to create, delete or modify user accounts or give them special permissions such as GroupAdmin permission. The following RDF fragment below states that 'user0771' has GroupAdmin permission.

```
<svgcwe:User rdf:about="#user0771">
 <svgcwe:permission>GroupAdmin</svgcwe:permission>
</svgcwe:User>
```

Each user account has authorization information associated with it depending on the role that the user plays in the collaborative session. In the context of the Application Scenario, there are two roles one can play, Controller or Fire fighter. This can be easily changed and extended as needed. The following RDF fragment states that the 'user0771' has a fire fighter role.

```
<svgcwe:Person rdf:about="#user0771">
 <wildfmt:role>FireFighter</wildfmt:role>
</svgcwe:Person>
```

When a user attempts to login to Wildfmt (see Figure 9-21), a query is sent to the KB (we assume a secure channel is used). The query is expressed as the following (username/password: jo/jo):

```
SELECT      ?user_uri ?user_full_name ?user_type ?permission ?group
WHERE {     ?user_uri svgcwe:username "jo";
                      svgcwe:password  "jo";
                      rdf:type ?user_type;
                      svgcwe:owned-by ?person_uri.
OPTIONAL{   ?user_uri svgcwe:permission ?permission.
            ?person_uri vcard:FN ?user_full_name.}
            LIMIT 1
```

The implementation abstracted away security concerns. In a 'real' application, communication will be secure as appropriate for the login stage and subsequent communications. The query is expecting the KB to return the full name of the user, the user type (User of AdminUser), the permission given (GroupAdmin, etc.), and the name of the group that the user belongs to. If the answer to the query is an empty set, it means that the user has failed to enter the correct login information or does not have an account. Therefore the request to gain access to Wildfmt has been denied. If the correct information has been entered, the URI of the person who owns the user account, the user's full name and privileges are returned.

This information is then stored locally – as an authentication token -in Wildfmt until the user logs off (authentication token gets removed) or switches to a different user account. Hence, the user identity is maintained locally in the Wildfire run-time. However, the KB is always kept up to date as any change happens to users, groups or the relations between them by the application.

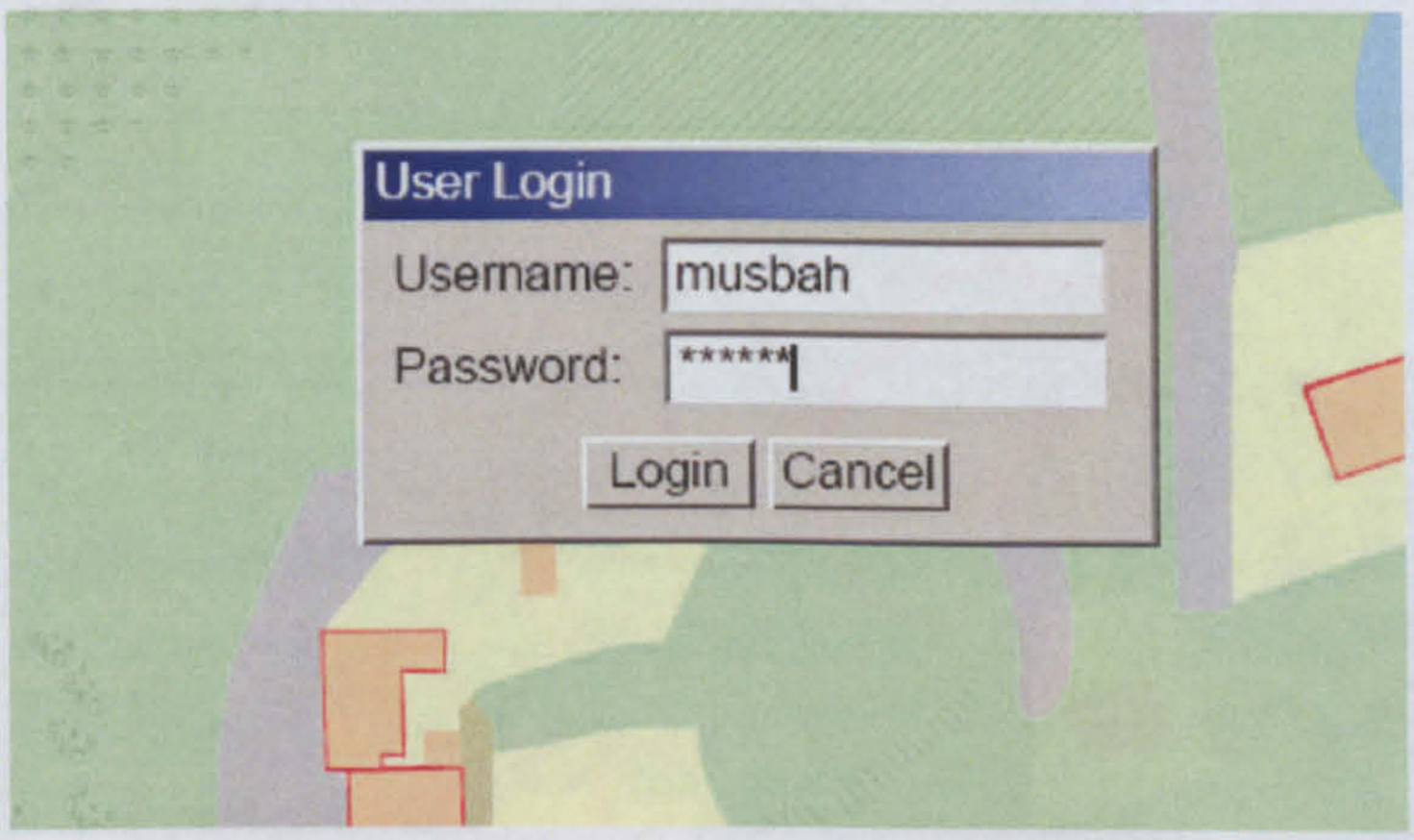


Figure 9-21: Wildfmt login screen

After a successful login (see Figure 9-21), Wildfmt presents the user with the appropriate user interfaces according to the role and privileges the user has. For instance, a user with GroupAdmin permissions will be able to create and delete groups.

9.4.3 Application Level Annotations

Annotations supported by CWE are of a generic nature. CWE allows annotations to be made on the workspace using various shapes and text. The meaning of those annotations is not predetermined by CWE, but rather left to the application (i.e. Wildfmt). To add a new application annotation, follow these three simple steps:

1. Choose an SVG primitive shape (circle, rectangle, etc.) with a distinctive style (colour, thickness, etc.) to represent the annotation (using the RDFS described in Section 6.3),

2. Create a type for it using RDFPIDM to the annotation meaning and

3. Add a button to the Wildfmt specific toolbar (see Figure 9-22) to create the new annotation.

To demonstrate this, three meaningful-annotations have been added to the annotation set of Wildfmt.

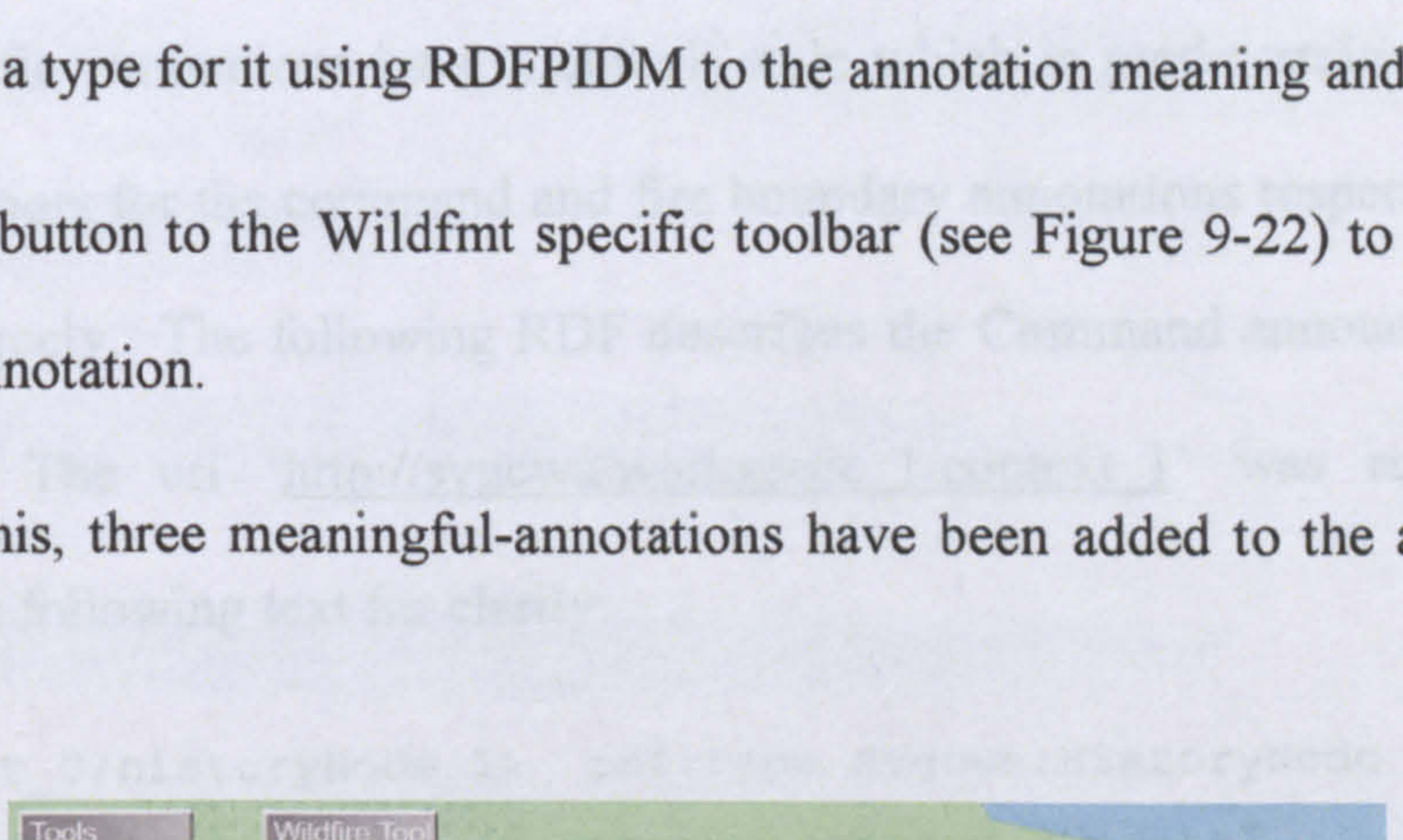


Figure 9-22: Wildfmt toolbar, Command, Fire boundary and Pointer annotations.

As shown in Figure 9-22, the new toolbar window (top left) has three buttons (from left to right: Command tool, Fire Boundary tool and Pointer tool). Each of these tools creates an application-level annotation that has a specific meaning. These annotations are described here (left to right):

1. The Command annotation: inherits from the text annotations with added meaning that serves as a command to the group.
2. The Fire Boundary annotation: to determine the boundary of the real fire on the map (can be used by fire fighters to draw what they see on the ground).
3. The Pointer annotation (arrow): for drawing the attention of the group to specific features on the map or used with the command annotation, for example, to order a fire fighter to change location.

Application specific annotations have a default style which is predetermined, such as green and red colours for the command and fire boundary annotations respectively; this can be changed freely. The following RDF describes the Command annotation shown in Figure 9-22. The url 'http://svgcwe/workspace_1/context_1' was replaced by 'context_1' in the following text for clarity:

```
<context_1/text_0/historyNode_1>  rdf:type svgcwe:HistoryNode .
<context_1/text_0/historyNode_1>  svgcwe:command "Update" .
<context_1/text_0/historyNode_1>  dc:Date "2008-05-14T16:30:12.828Z".
<context_1/text_0/historyNode_1>  dc:Contributor "Musbah Sagar".
<context_1/text_0/historyNode_1>  svgcwe:owned-by <context_1/text_0> .
<context_1/text_0/historyNode_1>  svg:id "id1212324" .
<context_1/text_0/historyNode_1>  svg:x 335586.1733203505 .
<context_1/text_0/historyNode_1>  svg:y -491914.3135345667 .
<context_1/text_0/historyNode_1>  svg:width 50.024417877197266.
<context_1/text_0/historyNode_1>  svg:height 16 .
<context_1/text_0/historyNode_1>  svg:fill "green" .
<context_1/text_0/historyNode_1>  svg:stroke "black" .
<context_1/text_0/historyNode_1>  svg:stroke-width 0 .
<context_1/text_0/historyNode_1>  svg:opacity 1 .
<context_1/text_0/historyNode_1>  svg:font-family "Helvetica" .
```

```
<context_1/text_0/historyNode_1>  svg:font-style "Normal" .
<context_1/text_0/historyNode_1>  svg:font-size "10pt" .
<context_1/text_0/historyNode_1>  svg:content "Move 10 feet right" .

<context_1/Command_0> rdf:type wildfmt:Command .
<context_1/Command_0> svgcwe:owned-by <context_1/> .
<context_1/Command_0> wildfmt:represented-by <context_1/text_0> .
```

In the RDF fragment above, the last three lines describe the actual Command annotation, which is owned by context_1 and is represented by context_1/text_0 (a CWE generic text annotation). The remaining RDF describes the text annotation itself and the specific style given to it to make up the application-level Command annotation. Other information associated with it includes creator, time, etc.

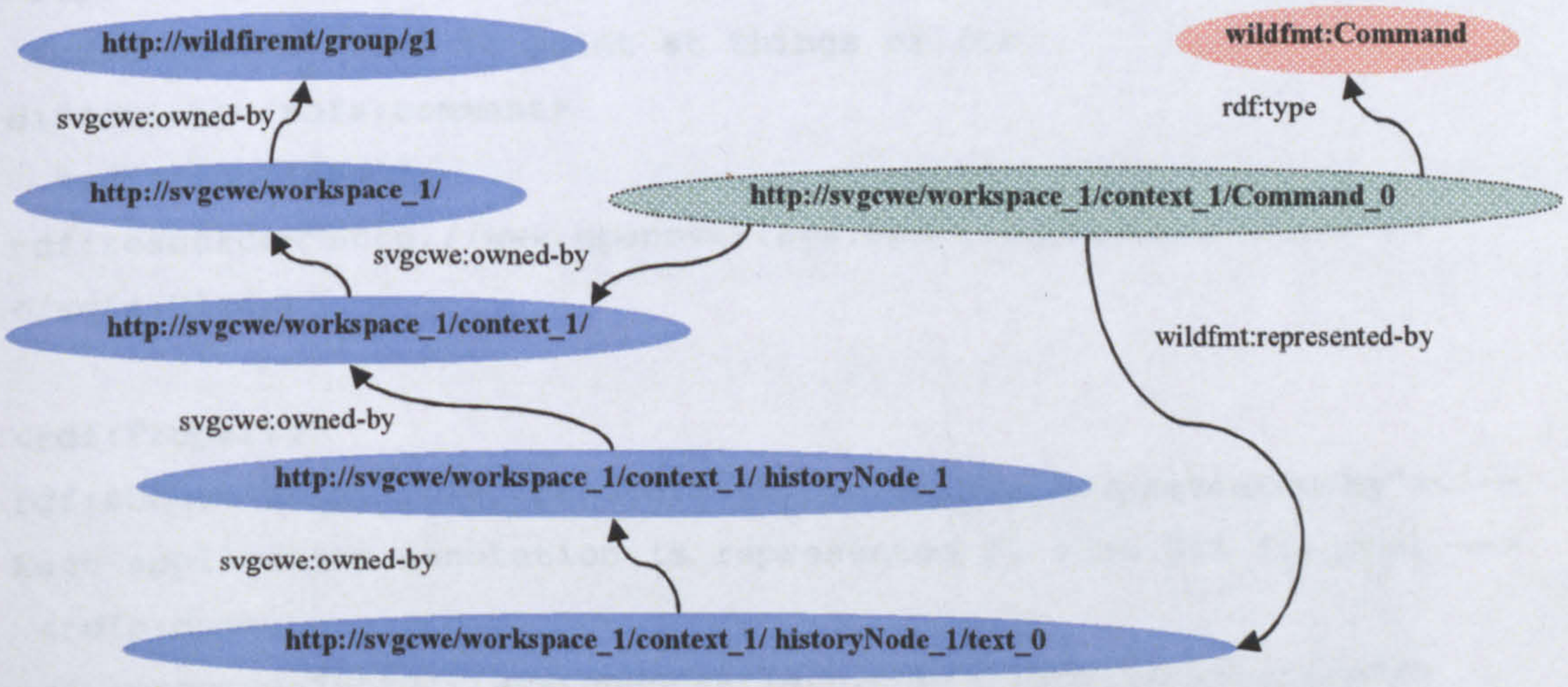


Figure 9-23: The Command annotation as an application-level annotation.

Notice from Figure 9-23, that the command annotation has a direct link with the context resource. The same principle applies to the other two remaining Wildfmt annotations: the fire boundary and the pointer annotations.

9.4.3.1 RDFS

There are three application specific annotations in Wildfmt, the Fire Boundary, the Command and the Pointer annotations. All inherit from the Annotation superclass. More application annotations can be added as required. The following RDFS describes

Wildfmt annotations.

```
<rdfs:Class
rdf:about="http://www.openoverlays.com/wildfmt#FireBoundary">
 <rdfs:comment>represents the fire boundaries</rdfs:comment>
  <rdfs:subClassOf
rdf:resource="http://www.openoverlays.com/wildfmt#Annotation"/>
</rdfs:Class>

<rdfs:Class rdf:about="http://www.openoverlays.com/wildfmt#Command">
 <rdfs:comment>used for commands issued by
controller(s)</rdfs:comment>
  <rdfs:subClassOf
rdf:resource="http://www.openoverlays.com/wildfmt#Annotation"/>
</rdfs:Class>

<rdfs:Class rdf:about="http://www.openoverlays.com/wildfmt#Pointer">
 <rdfs:comment>used to point at things or for
directions</rdfs:comment>
  <rdfs:subClassOf
rdf:resource="http://www.openoverlays.com/wildfmt#Annotation"/>
</rdfs:Class>

<rdf:Property
rdf:about="http://www.openoverlays.com/wildfmt#represented-by"><!--
Each application annotation is represented by some SVG fragment -->
 <rdfs:domain
rdf:resource="http://www.openoverlays.com/wildfmt#Annotation"/>
 <rdfs:range rdf:resource="http://www.w3.org/2000/svg/node"/>
</rdf:Property>
```

The property 'represented-by' links the application specific annotation with one of the CWE generic annotations.

9.4.4 Advanced Annotations

In addition to annotations made by users of Wildfmt (Controllers, Fire fighters), external applications can also make annotations to the workspace of CWE by directly posting annotations to the KB.

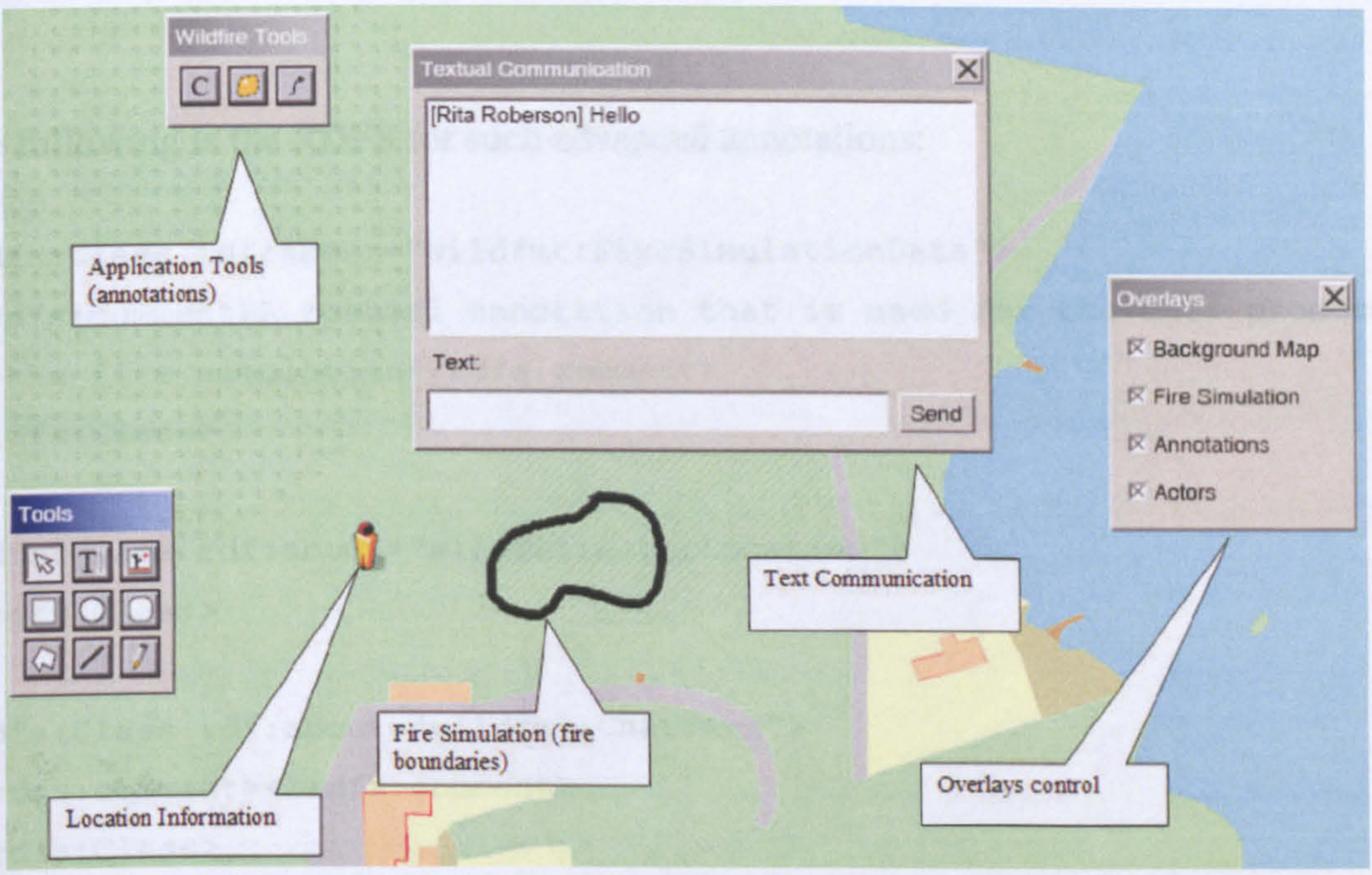


Figure 9-24: Overview of other types of annotations.

A Controller using Wildfmt can control the fire simulation using the fire simulation interface (see Figure 9-25). Other annotations to the workspace are for example: fire fighter location information and text chat (see Figure 9-24).

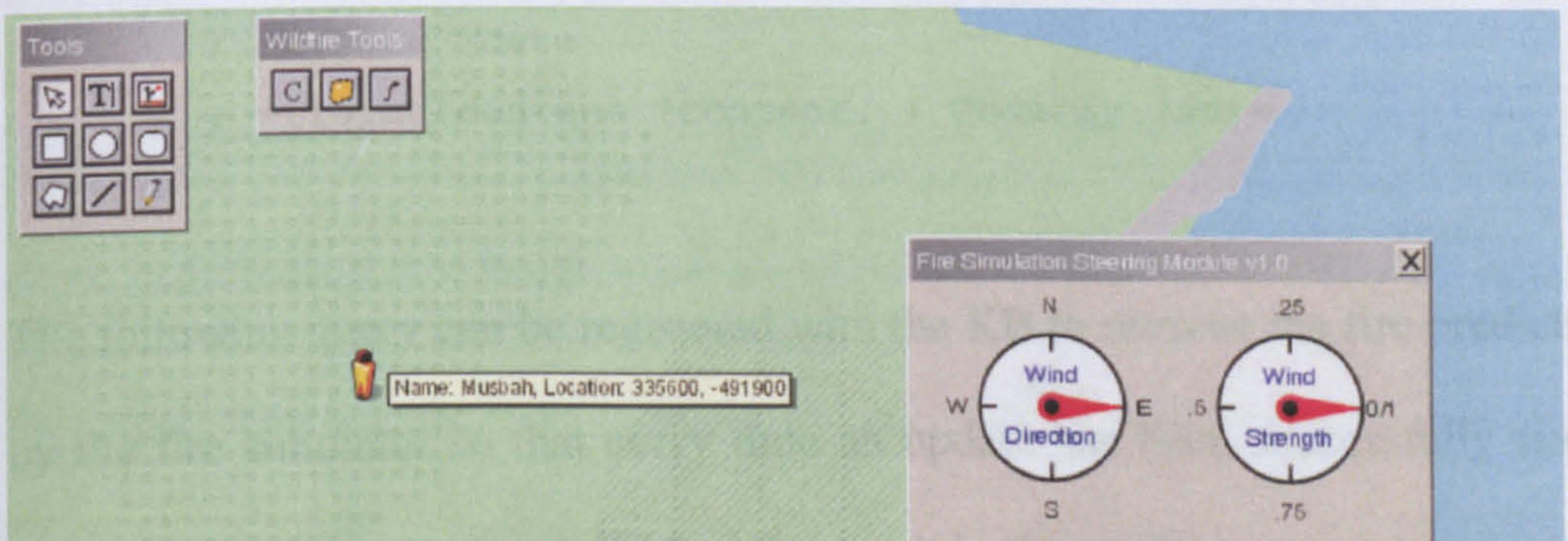


Figure 9-25: User interface to control fire simulation, the black annotation in the middle of the screen is the fire simulation result while the red borders are the application-level annotation of the fire boundaries as drawn by a fire fighter on the ground.

The following is the RDFS for such advanced annotations:

```
<rdfs:Class rdf:about="wildfmt:FireSimulationData">
 <rdfs:comment>A special annotation that is used for the data produced
by the fire simulation</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="wildfmt:ActorLocation">
</rdfs:Class>

<rdfs:Class rdf:about="wildfmt:ChatText">
 <rdfs:comment></rdfs:comment>
</rdfs:Class>
```

Wildfmt uses SPARQL to retrieve annotations from the KB stored by other applications (i.e. fire simulator, GPS sensors, etc.). The following query for instance will retrieve the chat annotations of a session:

```
SELECT DISTINCT ?content ?date
WHERE {?chatText rdf:type wildfmt:ChatText
        ; dc:Date ?date
        ; wildfmt:content ?content. } OrderBy ?date
```

The following query can be registered with the KB to retrieve the fire predictions made by the fire simulator so that every time an update has been successfully stored in the KB, Wildfmt is notified and displays the result in the workspace.

```
SELECT ?actorLocation ?location ?name ?color ?period

WHERE {?actorLocation svgcwe:owned-by <+this.contextURI+>.
       ?actorLocation rdf:type wildfmt:ActorLocation .
       ?hnode svgcwe:owned-by          ?actorLocation
            ; dc:Date                  ?date
            ; wildfmt:actor-name       ?name
            ; wildfmt:actor-color      ?color
            ; wildfmt:actor-location ?location
```

```
        ; wildfmt:actor-period   ?period .
    }

ORDER BY DESC(?date);
```

This query will return the location information of a fire fighter and other information, which will be used to annotate the workspace with an icon to represent the fire fighter on the workspace. The icon that represents a fire fighter is moved each time new information is stored in the KB.

9.5 Summary

This chapter has demonstrated the use of Web technologies to build adaptive applications (i.e. SVG Annotator) and collaborative applications (i.e. CWE and Wildfmt) following the four-layer model described in Chapter 3 and using RDF technologies in the design, data modelling and storage (using CoRDF), GAI for group communication and SVG for user interfaces via Oea framework. The table below gives an overview of the amount of work carried out to build CWE and Wildfmt in comparison to the development of Oea Framework and Oea HotDraw.

Packages	Number of Files	Comments	Code
Oea Framework	110	7209	8828
Oea HotDraw	80	4853	5271
CWE & Wildfmt	60	2954	5185
Total	***250***	***15016***	***19284***

As shown in the table above, nearly half of the development (46%) has been devoted to the Oea Framework while HotDraw has taken 27% of the work. The development of CWE and Wildfmt has taken considerably less work in comparison with Oea Framework and Oea HotDraw (only 26%). This gives an idea of the time saving the Oea Framework and Oea HotDraw provides to developers.

The following chapter will put the product of our approach (Wildfmt) into an emulated environment to test how it can cope with various types of data (mixed knowledge) and to demonstrate the working of the KB in that environment.

10

The Killer App – the Wildfire Management Scenario Demonstration

This chapter presents a demonstration of the complete approach described in this thesis. It describes a successful attempt to deploy and use the KB and Wildfmt which demonstrates that Web technologies can address the challenges expressed in the Application Scenario to build ACTs.

10.1 Detailed Application Scenario

To evaluate the research, a demonstration of Wildfmt and the KB was conducted by five people each with a Dell laptop (XPS M1710, 2GHz, 1GBytes). The objective of this demonstration was to act out the Application Scenario (see Section 1.2) using the applications developed in the project including Wildfmt and the KB.

A populated version of the Application Scenario was devised with details including the number of people involved and the activities that they would carry out. This was an essential step in scripting the demonstration. The table below illustrates the stages of a possible fire fighting activity for the Application Scenario:

Real World	**Demonstration World**
STAGE 1	
Controller receives report of a fire.	Determine location on map displayed in collaborative workspace of which controller is sole member.
Controller makes an initial assessment and decides 4 fire fighters should form a group.	Invites 4 fire fighters from the pool (group 0) to join his group to communicate with each other.
Decide what sensor resources required and issues command to fire fighter group.	Issue command through Wildfmt.
STAGE 2	
Fire fighters assess situation at specified location and report to controller.	Sketch in Wildfmt workspace; draw estimated fire boundary, direction of spread of fire.
Controller decides how to deploy resources and orders fire fighters to deploy sensors at selected locations.	Issue commands to the group to deploy sensors at specified location using Wildfmt.
Controller decides how to attack fire with hand-beaters/fire-breaks.	Issue commands on Wildfmt workspace using the Command annotation to the group to change location and start fire fighting.
STAGE 3	
Controller monitors sensors and decides to simulate fire; initialises simulation with data from sensors.	Start simulation and receive data into Wildfmt workspace and store in the KB.
Fire fighting groups report status of fire, in some areas fire is extinguished, in others fire is still spreading.	Sketching on Wildfmt workspaces the boundary and direction of fire using the Fire Boundary and the Pointer annotations.
STAGE 4	
Report back to controller; fire extinguished.	Sketching in Wildfmt.

The table above was used as a guideline to carry out the demonstration. The first column in the above table (Real World) describes the actions that are happening in the real life situation while the second column (Demonstration World) describes how the participants/fire fighters (called actors hereafter) would utilise Wildfmt to aid fighting

the fire. The aim of this scenario is to highlight the advantages Wildfmt brings to these harsh circumstances by providing means for communication (textual and graphical) to help overcome the fire.

10.2 Emulating Reality

The Application Scenario (called the scenario hereafter) is happening in the real world and it has two aspects. The first includes the hardware and the network infrastructure (i.e. laptops, mobile devices, computers, wired/wireless network, etc.). The second is the software (the middleware and the application) which will run on this infrastructure.

The scenario is taking place in a specific location (scenario location) and over a specific period of time (scenario time). The Wildfmt will operate in scenario time and location. It was not within the scope of this research to demonstrate the scenario in a real life situation. As a substitute, a decision was made to 'emulate' the real world scenario. Instead of actors being placed in the field to fight a real fire we have introduced the concept 'stage' which allows these actors to perform the scenario in an emulated world. The stage has specific time (stage time) and specific location (stage location). The stage location can be anywhere (for example, inside a research lab) and the stage time is in general shorter than the scenario time. This allows the actors to perform the scenario which could take days, in an hour or less for example.

As explained in Section 1.2, the scenario uses a fire simulator to predict the fire spread. This fire simulation takes place in scenario location and scenario time. On the stage, the real fire was replaced by another fire simulation which was called a 'fire emulation' to distinguish it from the scenario fire simulation. The fire simulation code was also used to develop the fire simulator. Actors on the stage will be able to interact with the emulated fire with the help of two Web-based applications built specifically for the stage. These applications were developed to emulate many aspects of the

Application Scenario such as the GPS devices, the fire and wind sensors (for wind speed and direction and the construction of fire-breaks). These two applications are:

(1) User Stage Control and Monitoring (USCM):

(2) Loge

Actors on the stage will use USCM to emulate acting the real life fire fighter while using Wildfmt for the scenario. They will be represented with Icons by the USCM interface on the map of the fire location, and they will be able to observe the fire progress (generated by the fire emulation), move around and fight the fire. The Loge application is used by the Controller to drive the fire emulator. The word 'Loge' is derived from Wagner's version of the Norse god Loki, a tricky character who is in charge of fire. The Controller is able to start the fire anywhere on the map and control the wind direction and speed at any point in time. Both of these applications (USCM and Loge) communicate with the Stage Service, a state-full Active Service running as the backend of both applications.

10.2.1 User Stage Control and Monitoring

This is an SVG application developed with the Oea framework. An instance of the application represents an individual actor in the scenario. Each actor in the Wildfmt will use USCM tool to represent himself in the stage. An actor is represented by an icon on the workspace of the tool with the site map as the background. Actors can move around and fight the fire; this is done by creating fire-breaks or clearing old ones (see details and screenshots of how USCM is used in Section 10.5). Actors are also able to view the fire emulation produced by Loge (see Section 10.2.2) and so avoid running into the fire. They are also able to create fire-breaks in the right locations before the fire reaches them.

The USCM tool produces location information from the actor and stores it in the

Stage Service (see Section 10.3). This information is shared among all actors and visualised on their USCM tools (so they can see each other, see Figure 10-13). These location information details are also passed on to the KB so that the Wildfmt can query, retrieve and present these on the shared workspace of the application (this will be explained later).

10.2.2 Loge

Loge is an SVG application developed using the Oea framework. Loge has a fire emulator engine built-in (see Section 10.4) to control the spread of the fire in the scenario storyboard.

Loge responds to the actions of actors on the stage (using USCM) such as clearing fire-breaks, making new fire-breaks. The settings of Loge's fire emulator can only be changed by controllers. Loge produces the fire boundaries in a shared data repository on the Stage Service. This data is then visualised by all actors using the USCM tool.

10.3 Stage Service

The reality emulation part of the demonstration (the stage) relies on the Stage Service which was developed as a special Web server to facilitate communication and storage to Loge and the USCM tool.

10.3.1 Architecture

Figure 10-1 shows how the Stage Service is used to allow the USCM tool and Loge to work. Loge generates the fire boundaries and stores them on the Stage Service. Fire fighters use the USCM tool to view the fire and create fire-breaks which are also stored in the Stage Service local storage.

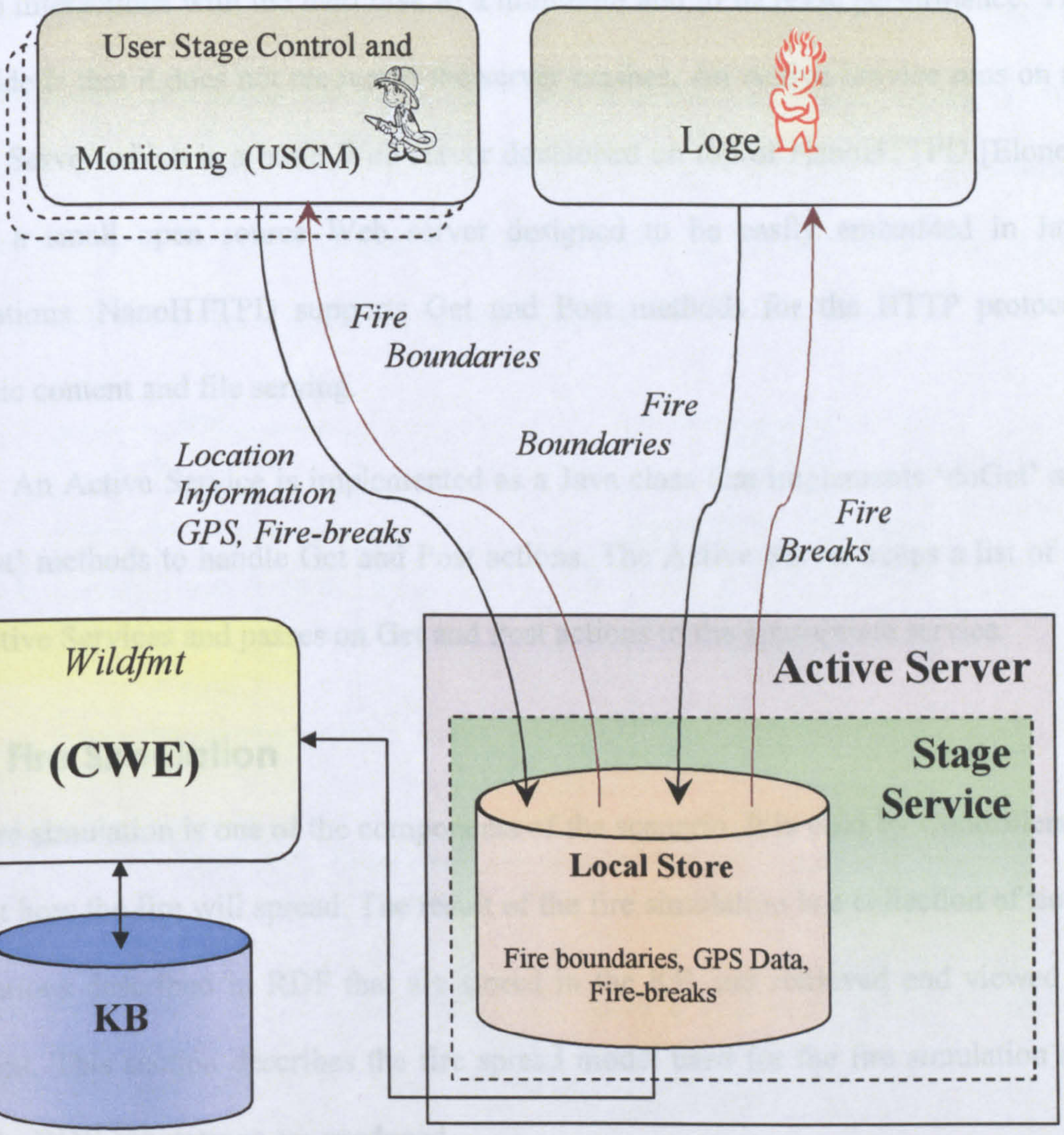


Figure 10-1: Loge and User Stage Contol and Monitoring tool used to emulate the Real World environement in relation to Wildfmt and the KB.

The fire-breaks data produced by fire fighters are used to control the fire emulator (to stop the spread of fire). The USCM tool also stores the location information of fire fighters on the local storage of the Stage Services so all fire fighters can see each other's locations. The Wildfmt retrieves the GPS data live from the Stage Service and stores them in the KB.

10.3.2 Implementation

The Stage Service was developed as an Active Service, a state-full Web service that

does not lose the data between different requests. An Active Service is always running to keep interactions with the hard disk to a minimum and to increase performance. The downside is that it does not recover if the server crashes. An Active Service runs on an Active Server which is a basic Web server developed on top of NanoHTTPD [Elonen, 2001], a small open source Web server designed to be easily embedded in Java applications. NanoHTTPD supports Get and Post methods for the HTTP protocol, dynamic content and file serving.

An Active Service is implemented as a Java class that implements 'doGet' and 'doPost' methods to handle Get and Post actions. The Active Server keeps a list of all the Active Services and passes on Get and Post actions to the appropriate service.

10.4 Fire Simulation

The fire simulation is one of the components of the scenario. It is used by Controllers to predict how the fire will spread. The result of the fire simulation is a collection of timed annotations described in RDF that are stored in the KB and retrieved and viewed by Wildfmt. This section describes the fire spread model used for the fire simulation and how the RDF annotations are produced.

10.4.1 Fire Spread Model

Forests have different types of vegetation, such as trees, plants, grass, etc. which can catch fire and spread into surrounding areas. This vegetation – the fuel of the fire - catches fire and burns at different rates depending on flammability factors. Forests also feature rocks, roads and sometimes human-built fire-breaks; these have zero flammability factors and therefore prevent the fire from spreading further. A simple probabilistic-based fire-spread model known as Interacting Particle Systems (also called Probabilistic Cellular Automaton) [Siegrist, 2008] is used to simulate the forest fire. Interacting Particle Systems are spatial configurations of particles where the state of a

particular particle changes probabilistically subject to the state of its neighbours. The forest is modelled as a rectangular grid of cells. Each cell represents either a type of vegetation (fuel) or is part of a road or a fire-break (obstacle). Cells are affected by their neighbours from the north, south, east and west. The reason that only four neighbours are used is to keep the model simple. For a cell at point (i,j) the neighbours are : (i,j-1), (i,j+1), (i+1,j), (i-1,j) this is shown in Figure 10-2, left.

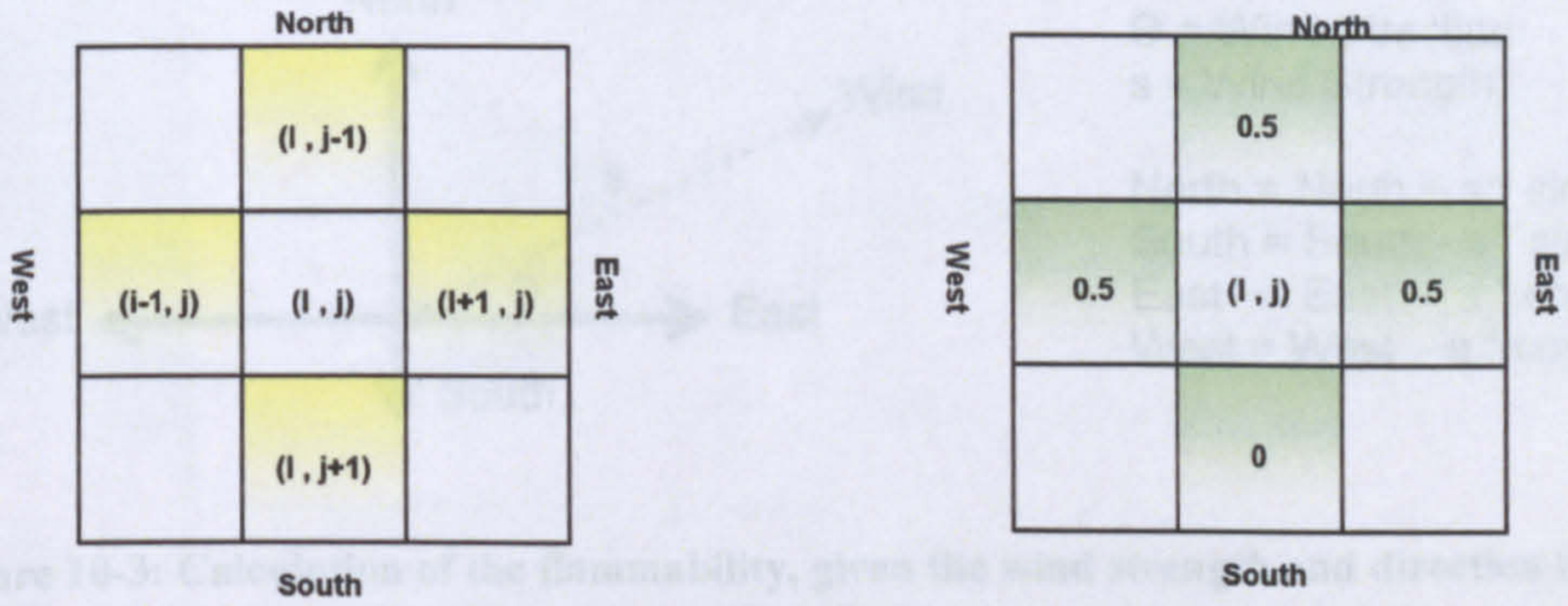


Figure 10-2: Neighbours (left), flammability grid (right)

Any cell in the grid of the forest model will have a fuel index which is used to determine the flammability of the cell (from 0: non-flammable, to 1: instantly combustible, see Figure 10-2, right). Furthermore, each cell has a state at any given time t. For example, for a cell that represents a tree (with 0.7 flammability), there will be the following states:

(1) Healthy at time t,

(2) Caught fire at t+1,

(3) Burning stage at t+2,

(4) Contagious stage (pass fire to neighbours) at t+3,

(5) Fire gets stronger at t+4,

(6) Burnt out at t+5.

For a cell that represents a road or a fire-break for instance there will be only one state, unaffected. The model is flexible and allows setting the number of states freely for each fuel type. The wind affects the fire spread model given its speed (strength) and direction. The wind strength ranges from 0 (no wind) to 1 (strongest wind) and the direction is from 0 to 360 degrees. The wind parameters (strength, direction) affect the flammability of the model cells according to the equations shown in Figure 10-3.

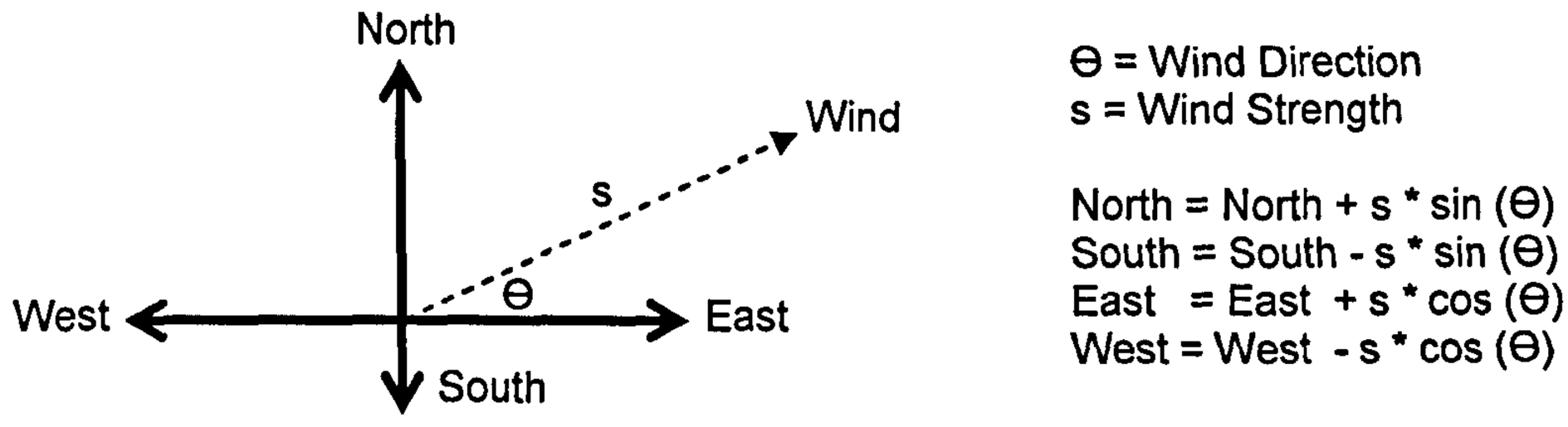


Figure 10-3: Calculation of the flammability, given the wind strength and direction for each direction

The directional probabilities may be used to model directional effects such as wind or terrain. The state of a cell in the grid changes from healthy to burnt-out in N timesteps. Associating a timestep with each fuel type makes the fire spread faster for fuels with small timestep values and slower for those with big timestep values. By giving the timestep a value as a period of time (1 minute, 10 minutes, etc.) it is then possible to know the time required for a particular fuel to be consumed totally. For example, if we take tree as a fuel type which needs 6 timesteps to be burnt-out and assume that the timestep has a value of 5 minutes, then the time required for the tree fuel to be consumed is 30 minutes.

10.4.2 Raster Image to Vector Image

From the description of the probabilistic forest fire model above, two requirements are needed to generate a visualization of the simulation as a raster image:

1. Two 2d arrays, one to store the distribution of the vegetation across the map

(Fuel Grid) and the second grid to keep track of the vegetation health state (State Grid).

2. A 2D buffer (can be also referred to as Raster Image) to store the output of the simulation as a raster image, see Figure 10-4 for an illustration.

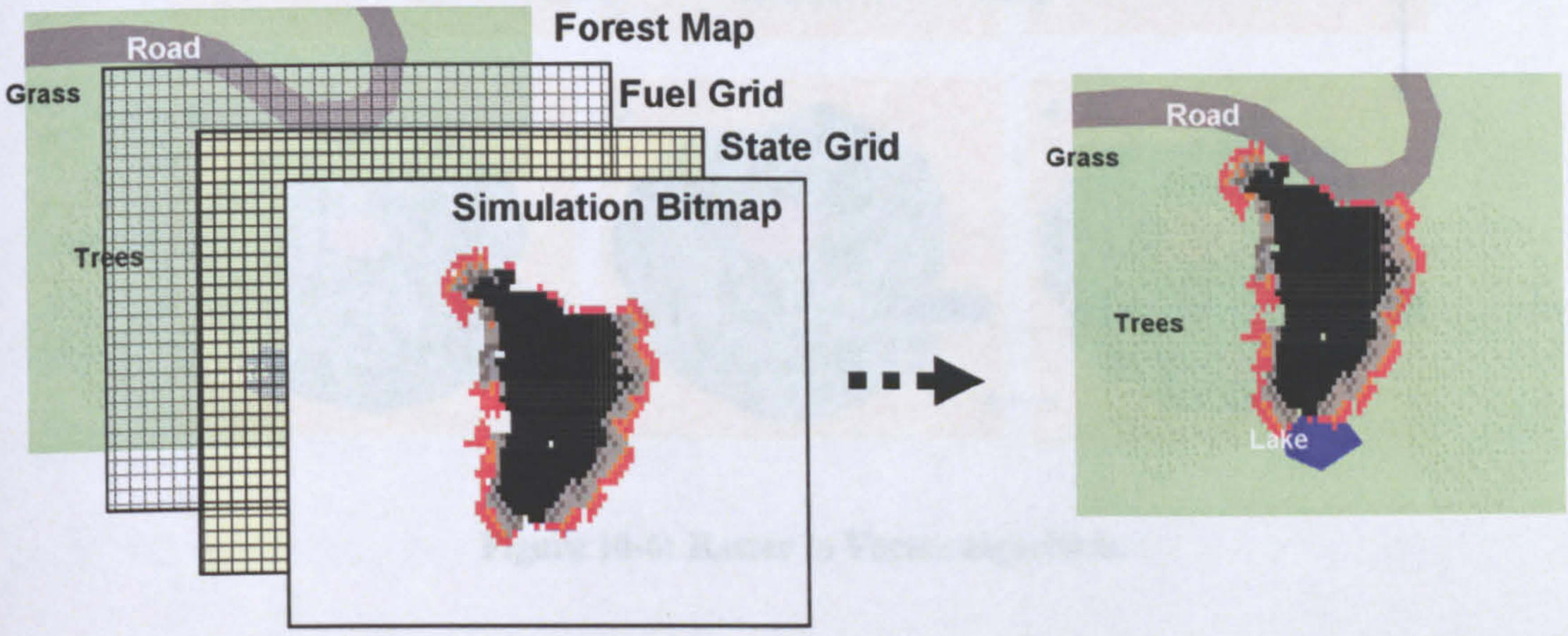
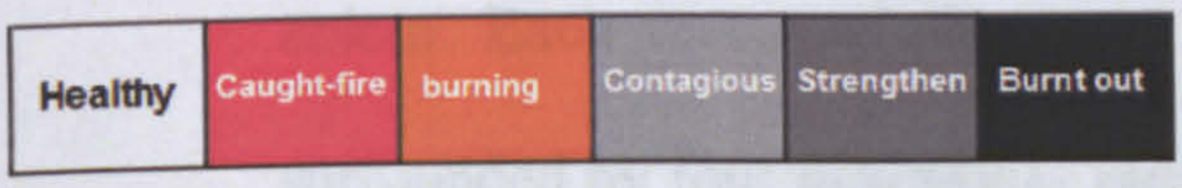


Figure 10-4: Forest Map, Fuel Grid, State Grid, Raster Image (left), Simulation overlaid the map (right)

Each cell in the Fuel Grid represents a fuel type (tree, rock, etc.) on the map as an integer number. Those numbers are associated with the flammability value of each fuel type (see Figure 10-4, right). For example, tree is represented in the grid as a number of value 4, and the flammability value of number 4 fuel is 0.6.

Colour Coding of Fire State Grid



Fuel Type	Value
Petrol	1
Grass	0.8
Plants	0.7
Trees	0.6
Obstacle	0
Water	0

Figure 10-5: Colour code for the Fire State Grid (left), Fuel Types table (right)

The state grid can be visualised by mapping different colours to state values. A raster image is created and filled with different colours that relate to the progress of the fire (see Figure 10-5). The images have the same size as the map and are divided into small rectangles. The amount of processing power required to run the simulation is

proportional to the number of grid cells.

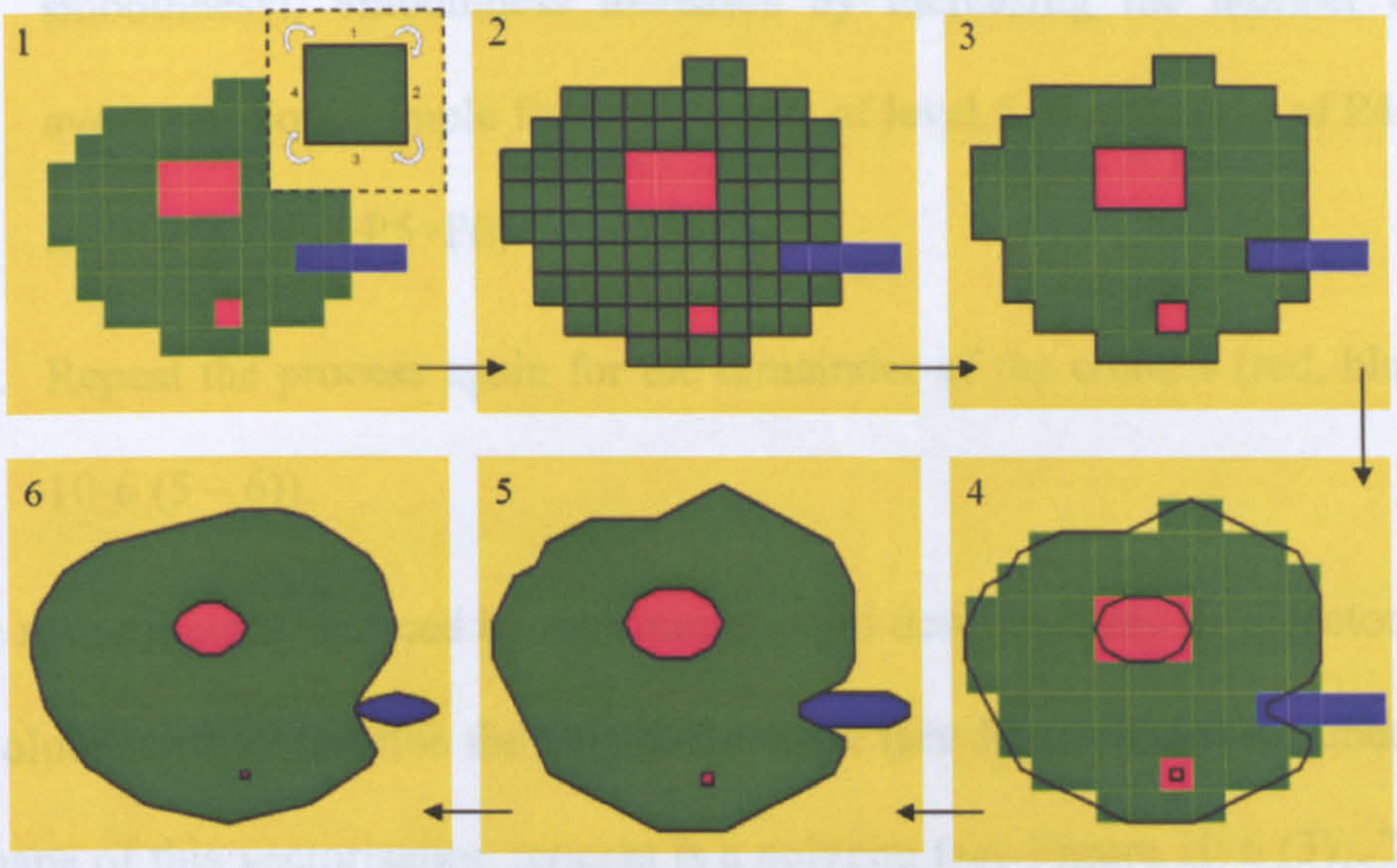


Figure 10-6: Raster to Vector algorithm.

A simple algorithm is used to convert raster images into vector images [cardhouse, 2006]. This algorithm converts each colour in the image into a unit square polygon of the same colour. For example, the images shown in Figure 10-6 (1) have three colours, green, red and blue. One needs to apply the vectorisation algorithm three times to convert this image to vectors. Here are the steps of the algorithm:

1. Create a representational square polygon (each side of the square is a vector of a single unit of length) for each pixel in the raster image of a certain colour. Each vector of the colour given (i.e. red, green and blue) will be surrounded by four directional vector lines each of 1 pixel length. The order of creating those lines is shown in Figure 10-6 (top, east, south and west) and the result of this step is shown in Figure 10-6 (2).

2. Merge all duplicate vectors that are located in the same location as the result of vectorising neighbouring pixels.

3. Join all vectors to one single polygon (Figure 10-6 (3)).

4. Apply a smoothing operation, averaging a number of neighbouring points

for each point in the polygon (Figure 10-6 (4) shows the result of level 1 smoothness). Smoothness increases by increasing the number of points averaged. For example for smoothness of level 5, the average of P4 (point 4) is: (P2+P3+P4+P5+P6)/5.

5. Repeat the process again for the remainder of the colours (red, blue, Figure 10-6 (5 – 6)).

From the raster image produced by the simulation as described above, a vector image of the red colour used to describe the caught-fire state (see Figure 10-4) is generated. The vector shape of this vectorisation process is a polygon (see Figure 10-6 (3)). The points constituting this polygon are used to construct a 'Path' element in SVG. The process is very simple; the points of the polygon are used to construct the path data and assign this to the 'd' attributes of the SVG Path element. For example if the polygon has the following points {(10,10), (40,10), (40,40),(10,40)} then the 'd' attribute of the 'Path' is = "M10,10 L40,10 L 40,40 L10,40 z". Below is the SVG code:

```
<path fill="none" stroke-color="black" stroke-width="2" d=" M10,10
L40,10 L 40,40 L10,40 z"/>
```

10.4.3 Steering the Simulation

There are 4 parameters that are used to configure the fire simulation (see Figure 10-7):

1. Simulation Time Step: used internally by the simulation to generate the fire predictions,
2. Real Time Steps: give the time for the real life scenario (in Figure 10-7, each 1 second (1000 millisecond) of the simulation corresponds to 5 minutes in scenario time),
3. Grid Cell Size: the size of the cells that are used internally by the simulator to calculate the size of the buffers used in the simulation, and finally

4. Path Smoothness Level: used with the Raster to Vector algorithm to convert the simulation data into vector graphics to be displayed in SVG. The higher the Smoothness Level value, the smoother the curves of the vector graphics generated.

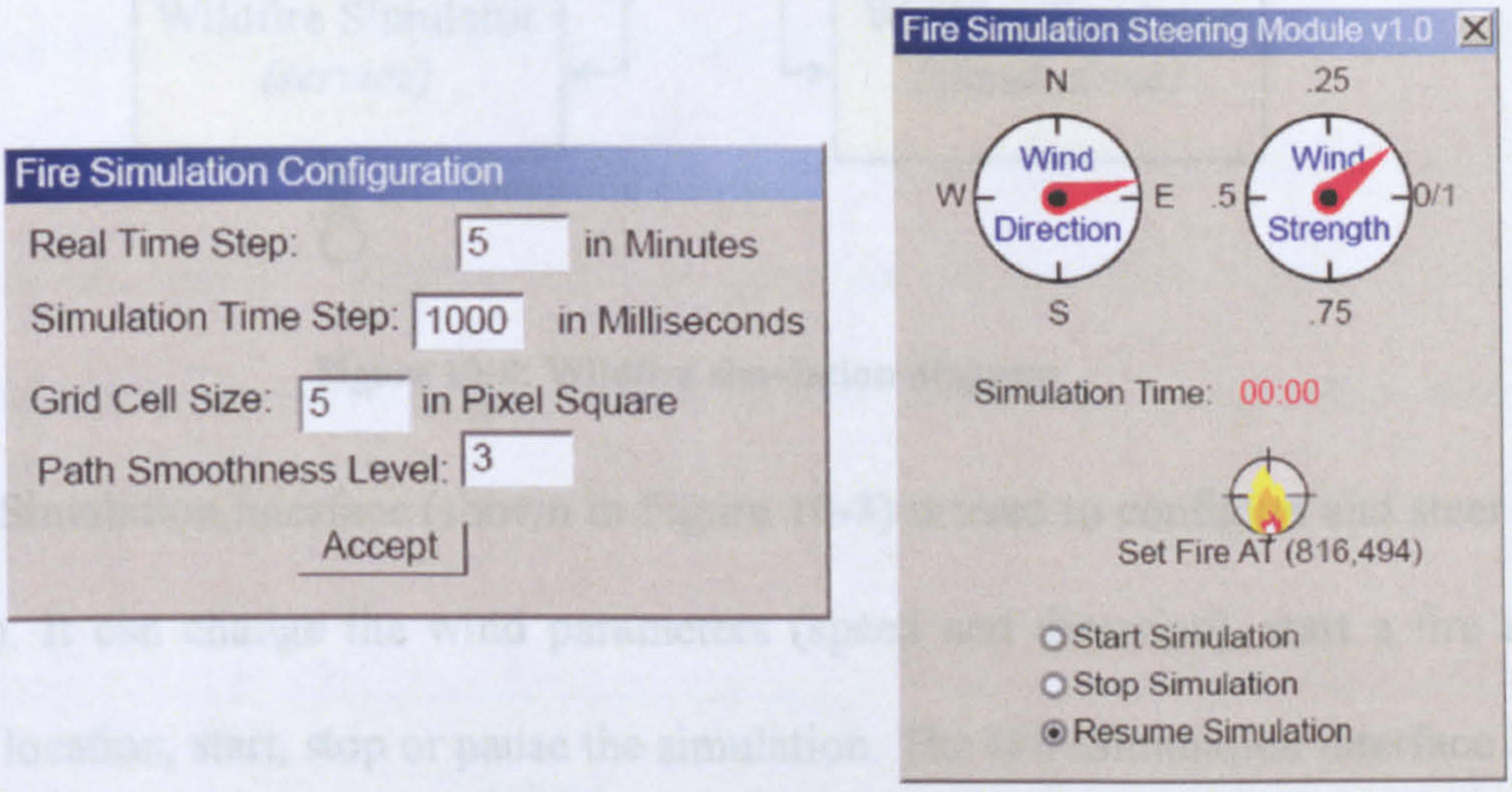


Figure 10-7: Simulation Configuration (left), Steering the Fire Simulation (right)

The Simulation Steering Module (Figure 10-7, right) controls the wind strenth and direction, starts, stops and resumes the simulation. The user can drag the target icon (with flames in the middle) onto the location where the fire is required to start. The use of a slider widget instead of the dial widget to change the wind strength (as shown in Figure above) would be more appropriate.

10.4.4 Implementation

Section 10.2 introduces the terms: fire simulator and fire emulator. The fire simulator is used to predict the spread of fire and the fire emulator is used to replace (or emulate) the real fire. Because the underlying fire model is the same for both processes it was decided to implement an interface to the fire model that the simulator and the emulator can both plug-into. This allows for easy steering of the simulation model. See Figure 10-8.

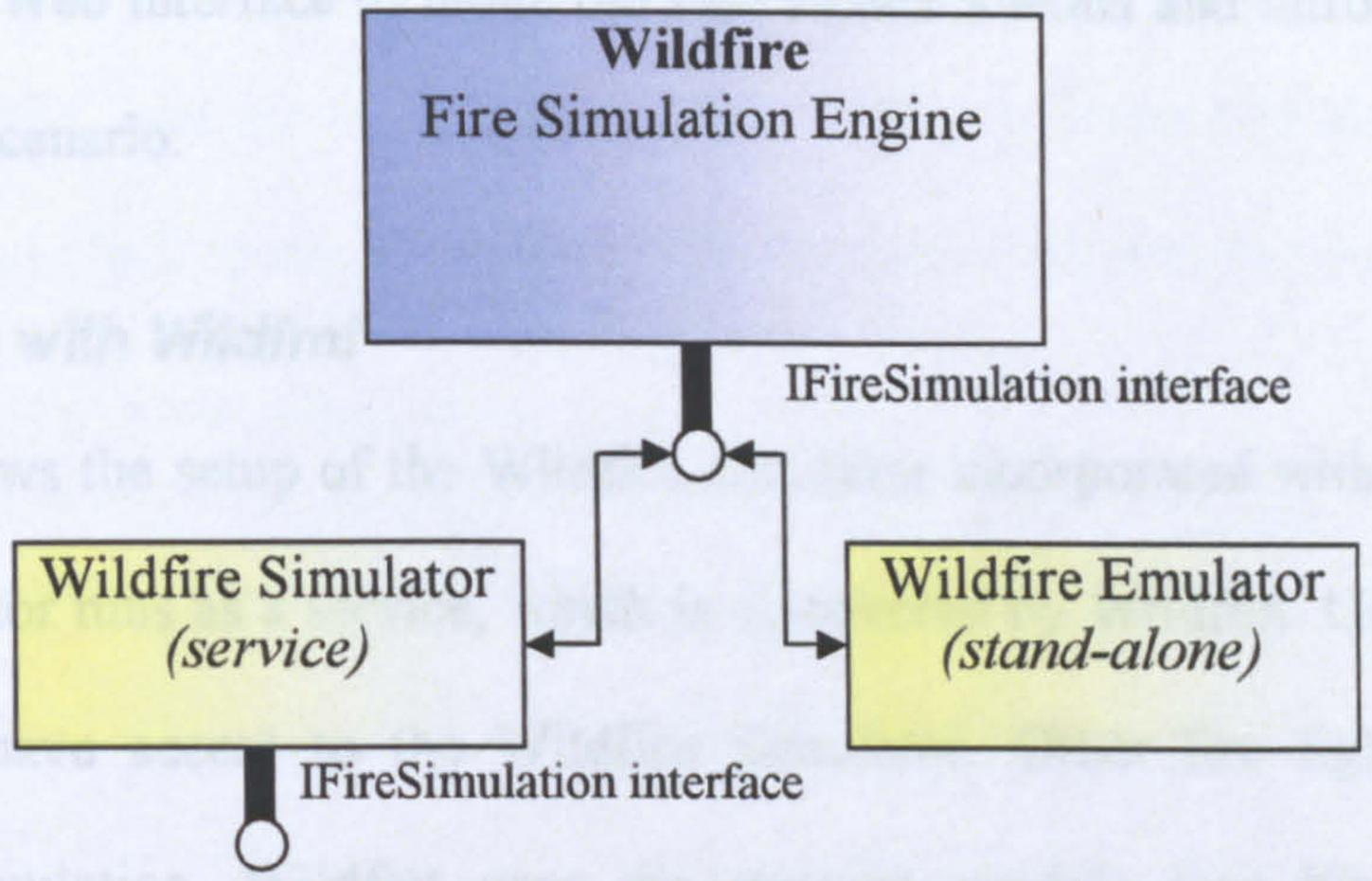


Figure 10-8: Wildfire simulation diagram

The IFireSimulation interface (shown in Figure 10-8) is used to configure and steer the simulation. It can change the wind parameters (speed and direction), start a fire at a particular location, start, stop or pause the simulation. The IFireSimulation interface is:

```
public interface IFireSimulation {
public void   start(); // Start or resume the simulation
public void   stop();  // Stop and reset the simulation
public void   pause(); // Pause the simulation, 'start' to resume
public void   changeWindDirection(double d);
public void   changeWindStrength(double s);
public void   setFireAt(int x,int y);
public void   configureSimulation(int realTimeStep,int
simulationTimeStep,int cellSize,int pathSmoothness,String fMap);
public String simulationData(String data);
// Change the flammability of the fuel map to 0 in order to fight the
fire.
public void fightFire(int value,Polygon pol); }
```

The Fire Simulation Engine was implemented in Java and it allows controlling the spread of fire by introducing fire-breaks. Fire-breaks stop the spread of fire as they have zero flammability. Fire-breaks are regions, described by vector polygons, with specific flammability values (zero or more). This gives a more accurate simulation reflecting what is happening on the ground with the real-life scenario (fire fighters fighting the fire with hand-beaters, fire-breaks, etc.). This is used with the stage-servers, where the fire

fighter uses the Web interface to move the Fire fighter's icons and introduces the fire-breaks into the scenario.

10.4.5 Binding with Wildfmt

Figure 10-9 shows the setup of the Wildfire simulator incorporated with Wildfmt. The Wildfire simulator runs as a service, which is discovered by Wildfmt. Only Controllers (admin users) have access to the Wildfire Simulator. Other fire fighters can only observe the simulation. Wildfmt uses the steering module (see Figure 10-17) to communicate with the Wildfire simulator and send the initial configuration settings (timestep, cell size, etc.) and later steer the simulation once it has started.

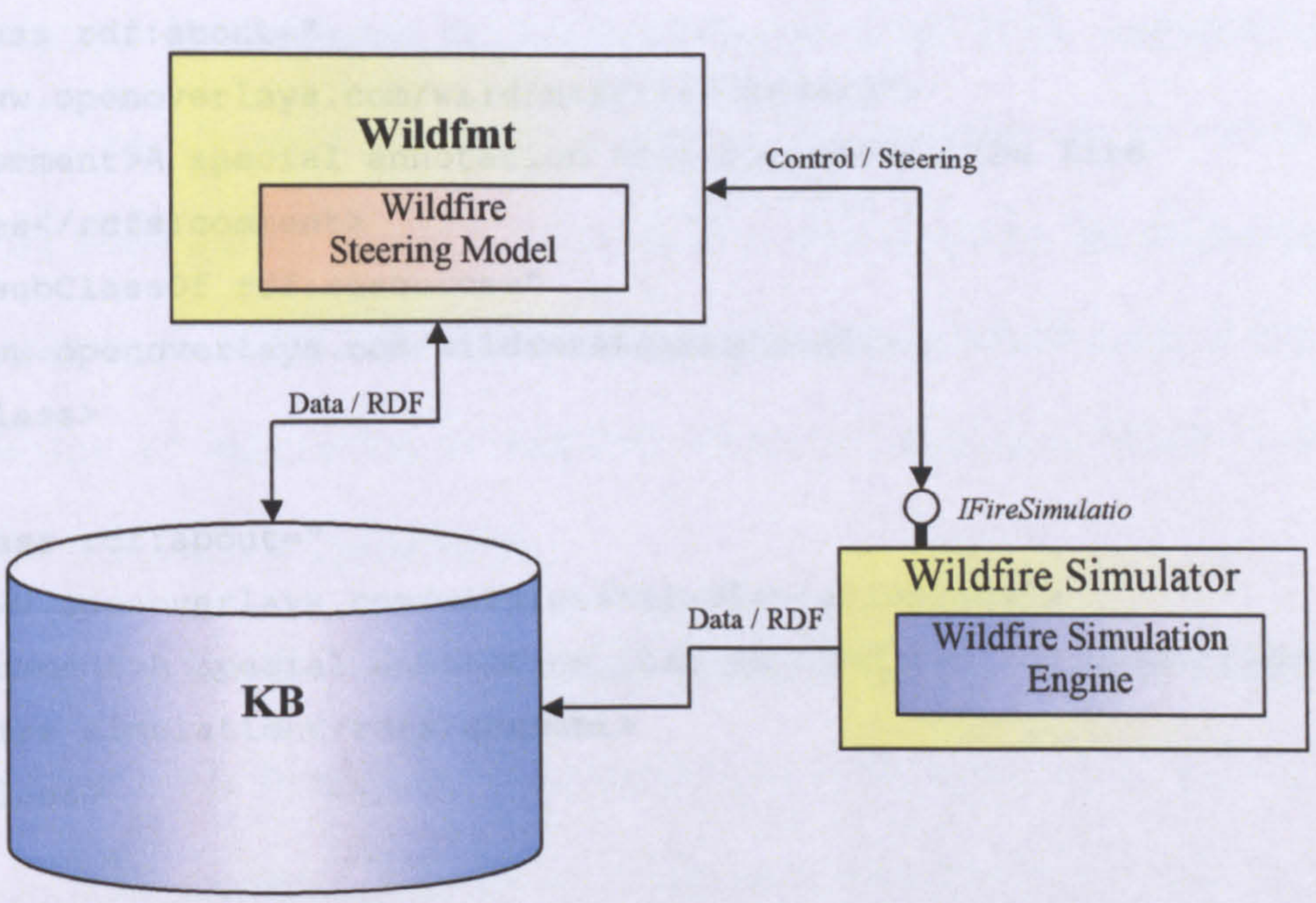


Figure 10-9: The binding between the KB, Wildfmt and the fire simulator.

The result of the simulation is produced as RDF annotations to the Wildfmtworkspace, and stored directly into the KB. By making the Fire Simulation available through the KB, Controllers and Fire fighters can register a request into the KB to express an interest to retrieve results from the wildfire simulator as they become available.

10.4.6 Fire Simulation RDFS

Below is the RDFS used for the fire simulation. There are some repeated data models from Wildfmt and CWE to show the connections between the two data models.

```
<rdfs:Class rdf:ID="Workspace">
 <rdfs:comment>Each collaborative group that uses CWE has a single
Workspace resource.</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="
http://www.openoverlays.com/wildfmt#Annotation">
 <rdfs:comment>The superclass of all application
annotations</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="
http://www.openoverlays.com/wildfmt#FireBoundary">
 <rdfs:comment>A special annotation that represents the fire
boundaries</rdfs:comment>
  <rdfs:subClassOf rdf:resource="
http://www.openoverlays.com/wildfmt#Annotation"/>
</rdfs:Class>

<rdfs:Class rdf:about="
http://www.openoverlays.com/wildfmt#FireSimulationData">
 <rdfs:comment>A special annotation that is used for the data produced
by the fire simulation</rdfs:comment>
</rdfs:Class>

<rdf:Property rdf:about="
http://www.openoverlays.com/wildfmt#simulation-data">
 <rdfs:domain rdf:resource="
http://www.openoverlays.com/wildfmt#FireSimulationData"/>
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="
http://www.openoverlays.com/wildfmt#simulation-time">
 <rdfs:domain rdf:resource="
http://www.openoverlays.com/wildfmt#FireSimulationData"/>
```

```
 <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Literal"/>
</rdf:Property>
```

The fire boundary type (FireBoundary) is the annotation type used to represent the fire simulation prediction on the workspace of Wildfmt. The FireSimulationData holds the points of the vector polygon resulting from the fire simulation and raster to vector image conversion (see Section 10.4.2) as a literal. The simulation-data and simulation-time properties link the fire simulation boundary to the data and time which are both saved as text.

10.5 Scenario Execution

At the start of the demonstration, the KB is configured to run in the replicated mode (Section 6.4). The KB could also be set to work in the centralised or the distributed mode. Which mode the KB should be working in is determined by the circumstances and the requirements in the real world. For example, the replicated mode is suitable when reliability of information is required while the distributed mode is more appropriate if scalability is a prerequisite.

The five actors in the Application Scenario (four fire fighters and one controller), have to run Wildfmt and the emulation (USCM) tools side by side on their Dell laptops. The emulation tool allows actors to change location, watch the fire emulation as it progresses and fight the fire by creating fire-breaks. The controller takes control over Loge to control the fire emulation. The emulation tools (Loge and the USCM tool) can be accessed through the Webpage designed specifically for this demonstration. The Web site is served by the Active Server (described above) via the URL: http://localhost:8080/index.htm , see Figure 10-10.

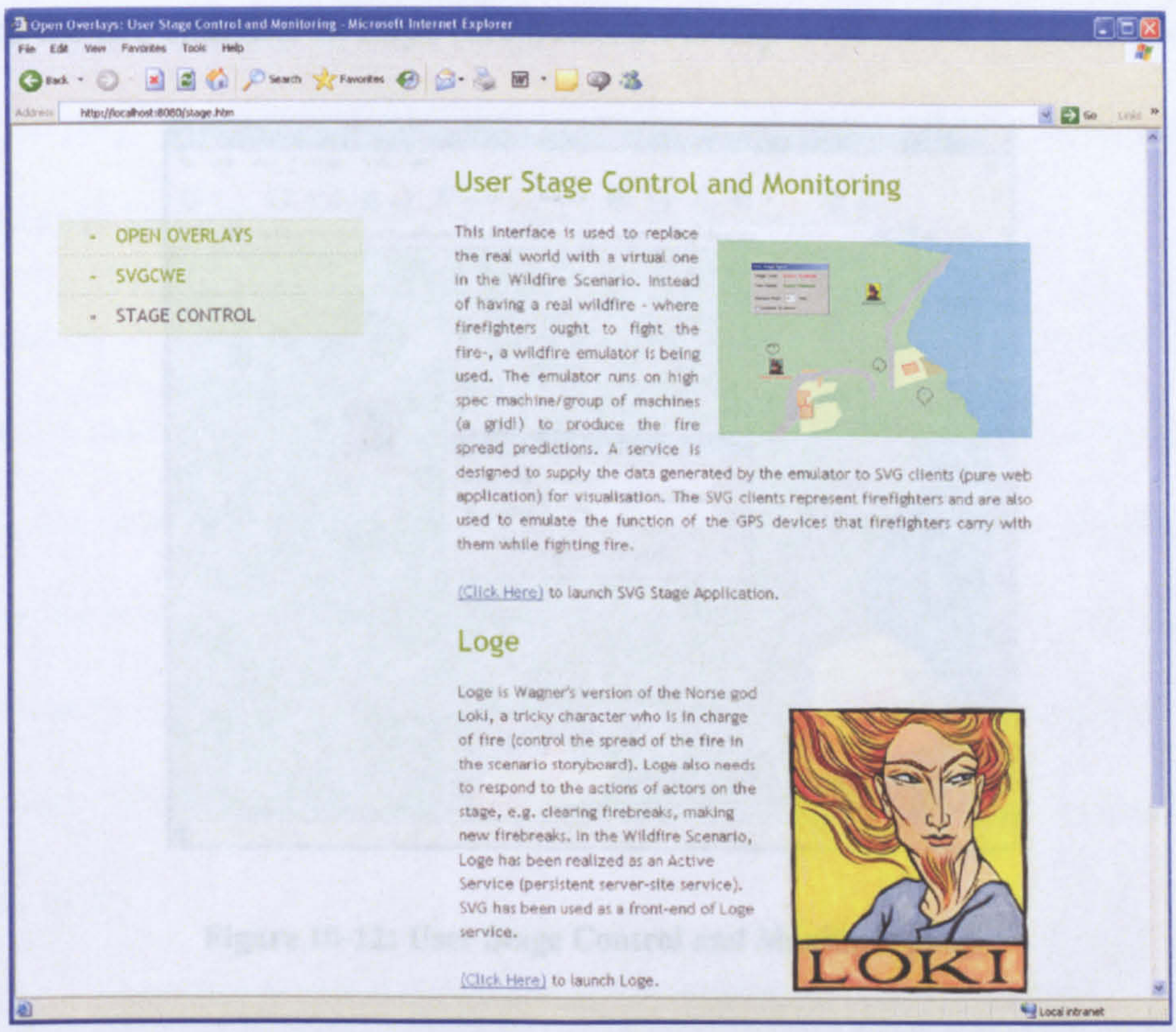


Figure 10-10: The Webpage, showing the links to launch USCM tool and Loge.

Figure 10-10 shows the Webpage running in IE Explorer 6. The controller launches Loge to control the fire emulation. Loge allows the controller to watch the location of fire fighters and the fire-breaks they create.

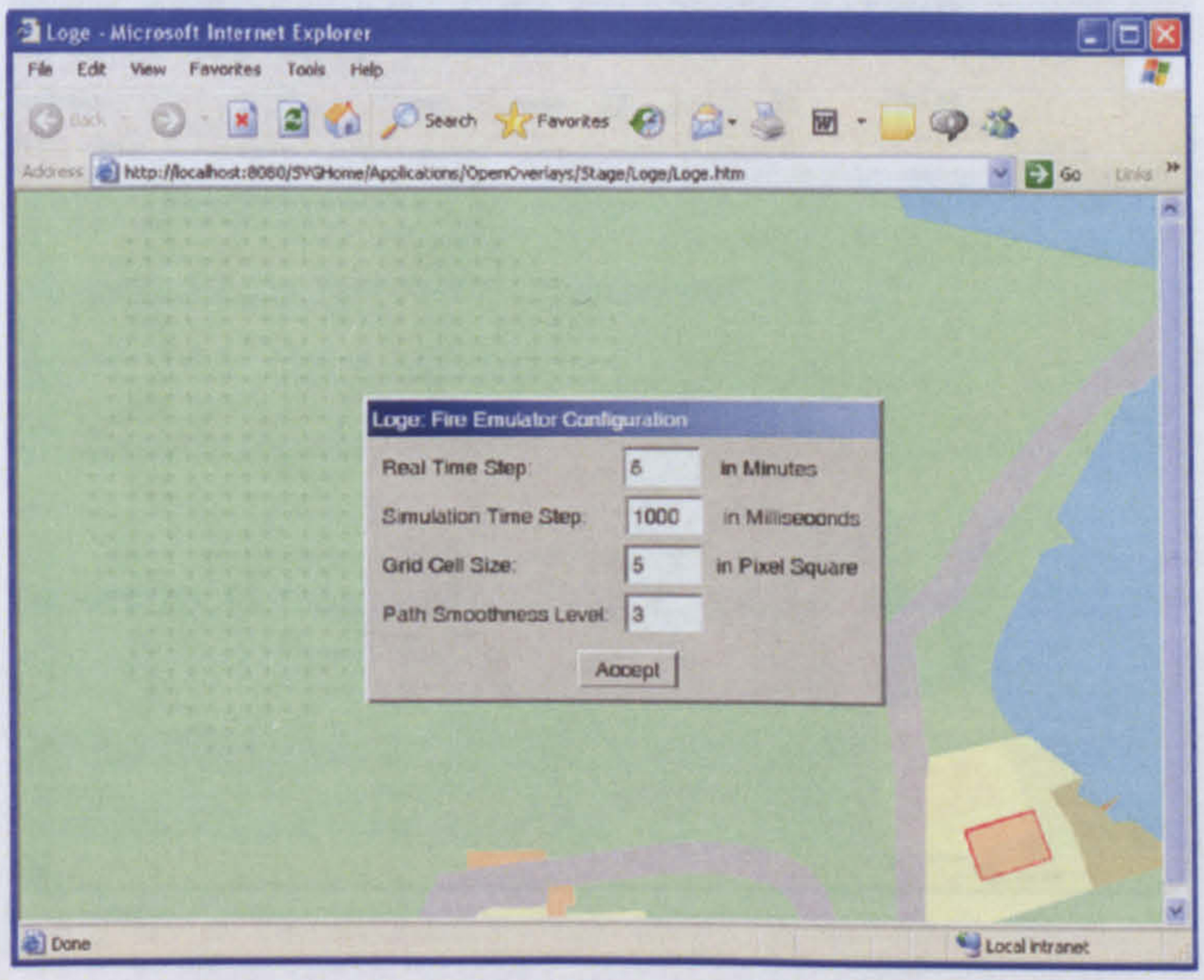


Figure 10-11: Loge fire emulator configuration dialoge box.

Figure 10-11 shows the Loge application. There are 4 parameters that are used to

configure the fire emulator in Loge (see Section 10.4.3).

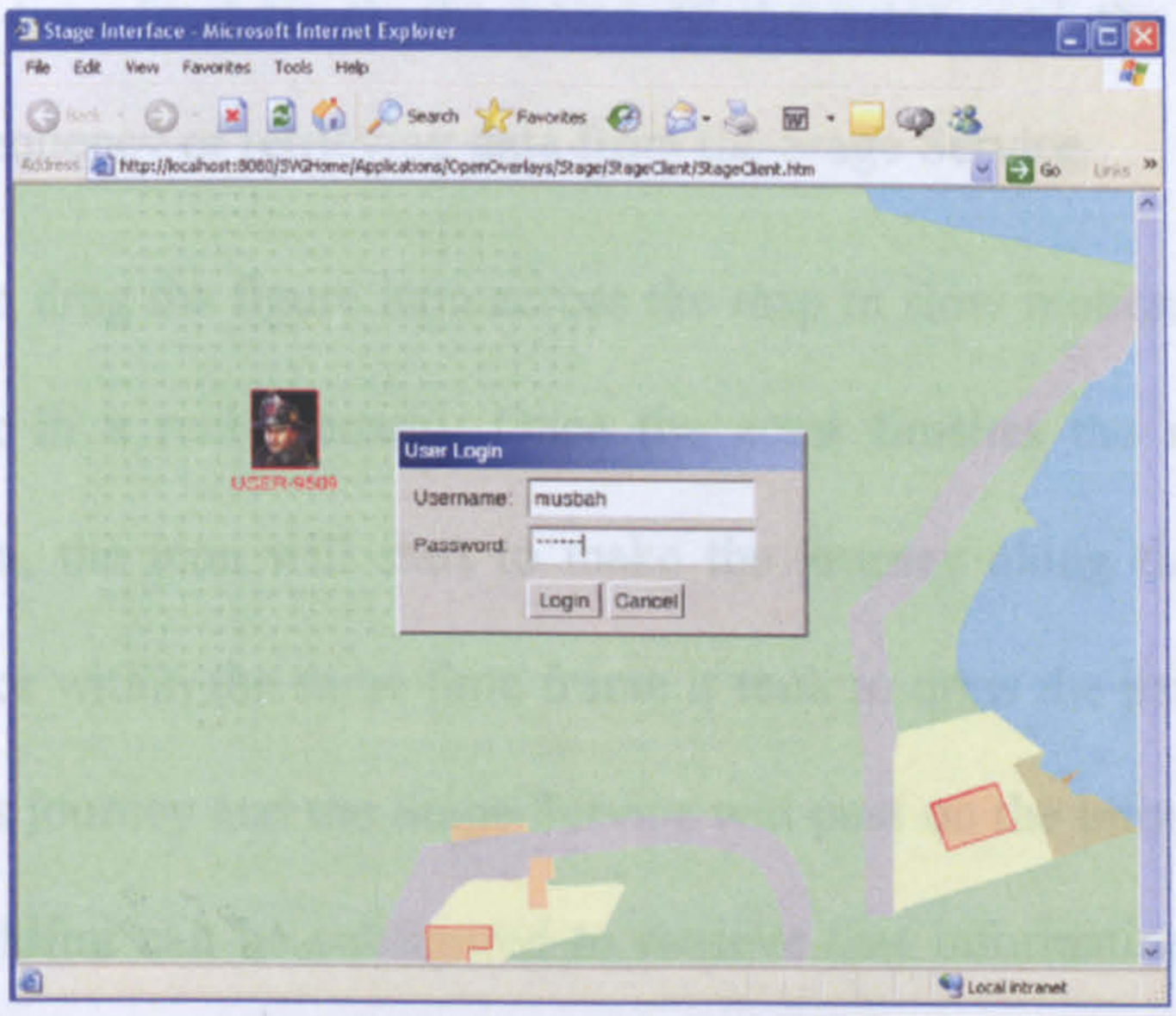


Figure 10-12: User Stage Control and Monitoring tool

Figure 10-12 shows the login page of the emulation tool, USCM. The actors will log into the emulation by entering their username and password (retrieved from the KB similarly to logging into Wildfmt). Once logged in, they will be presented with the site map and icons that represent them and which reflect their GPS locations on the map.

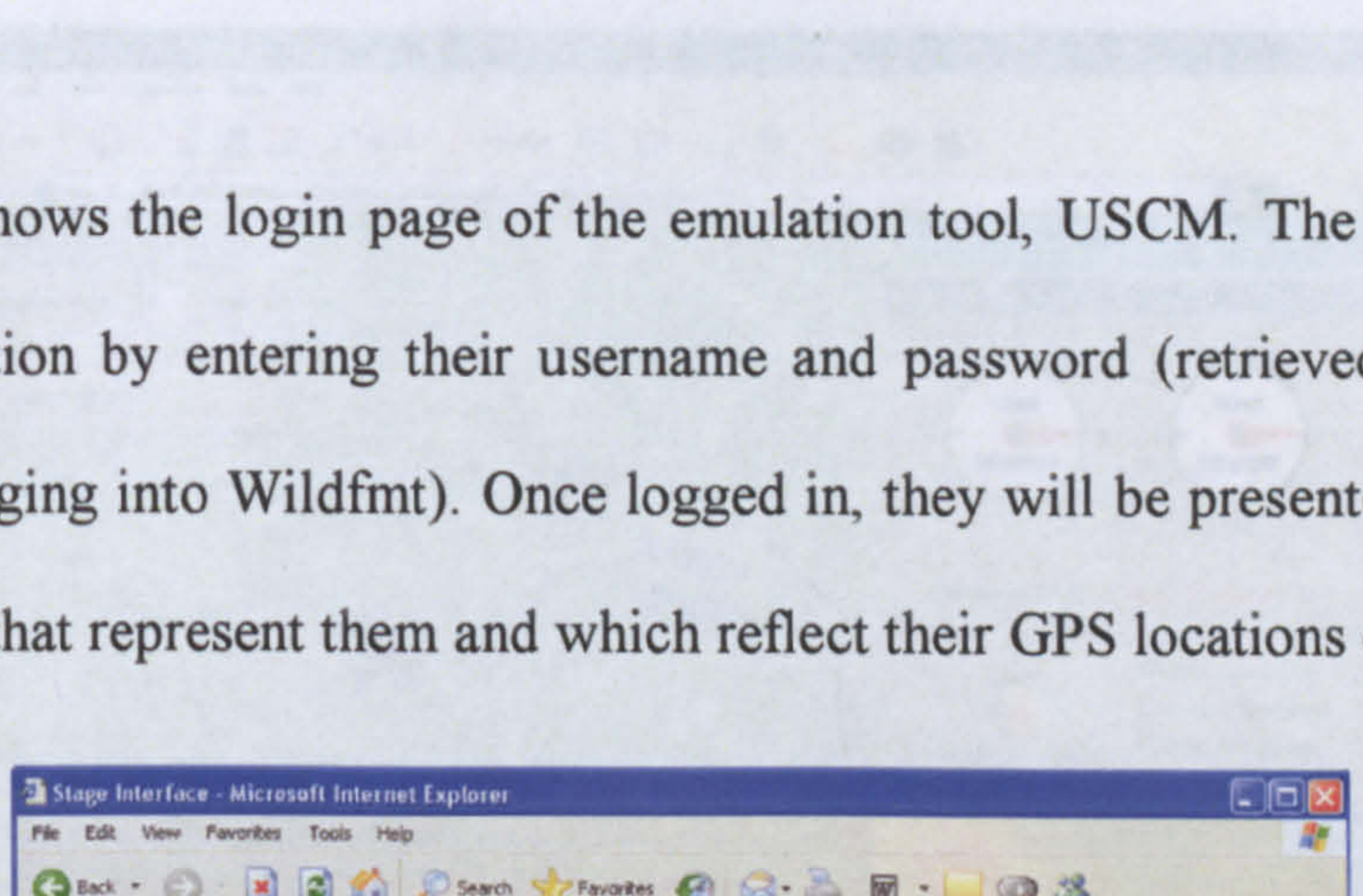


Figure 10-13: Two actors are logged into the User Stage Control and Monitoring Tool.

Figure 10-13 shows two actors successfully logged into USCM tool, the other 3 join

successfully right afterwards. The Stage control window at the top left of the screen shows the stage time (real time), the name of the actor, and the refresh rate that determines the frequency of retrieving data from the Stage Service.

Actors can drag the figure icon across the map in slow motion to show the path they are to walk in a real scenario. Once the actor finishes the motion action by releasing the icon, the icon will start to make the journey along the imaginary path drawn by the actor within the same time frame it took to draw the path. All actors will be able to see this journey and the Stage Service will pass on the information to the KB as GPS data. Wildfmt can be configured to retrieve that information by registering a SPARQL query.

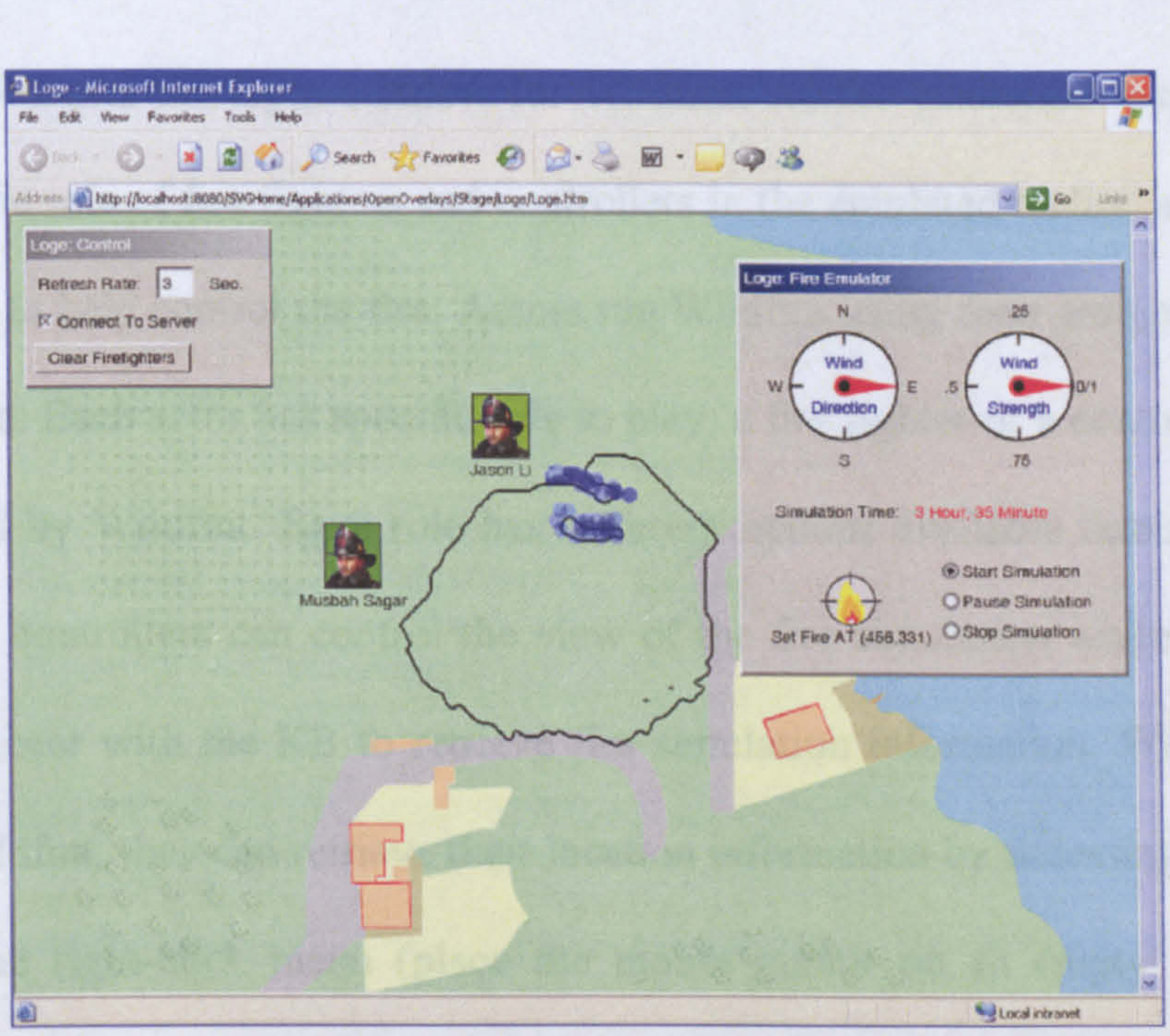


Figure 10-14: Loge tool in action.

Figure 10-14 shows the view of the controller running Loge. The window on the right side is the simulation steering module. Loge can view the fire fighters and the location of the fire (black border in the middle of the map). It can also see the fire-breaks created by fire fighters (blue obstacles). The control window on the left side of the screen determines how often the application can retrieve data from the Stage Service.

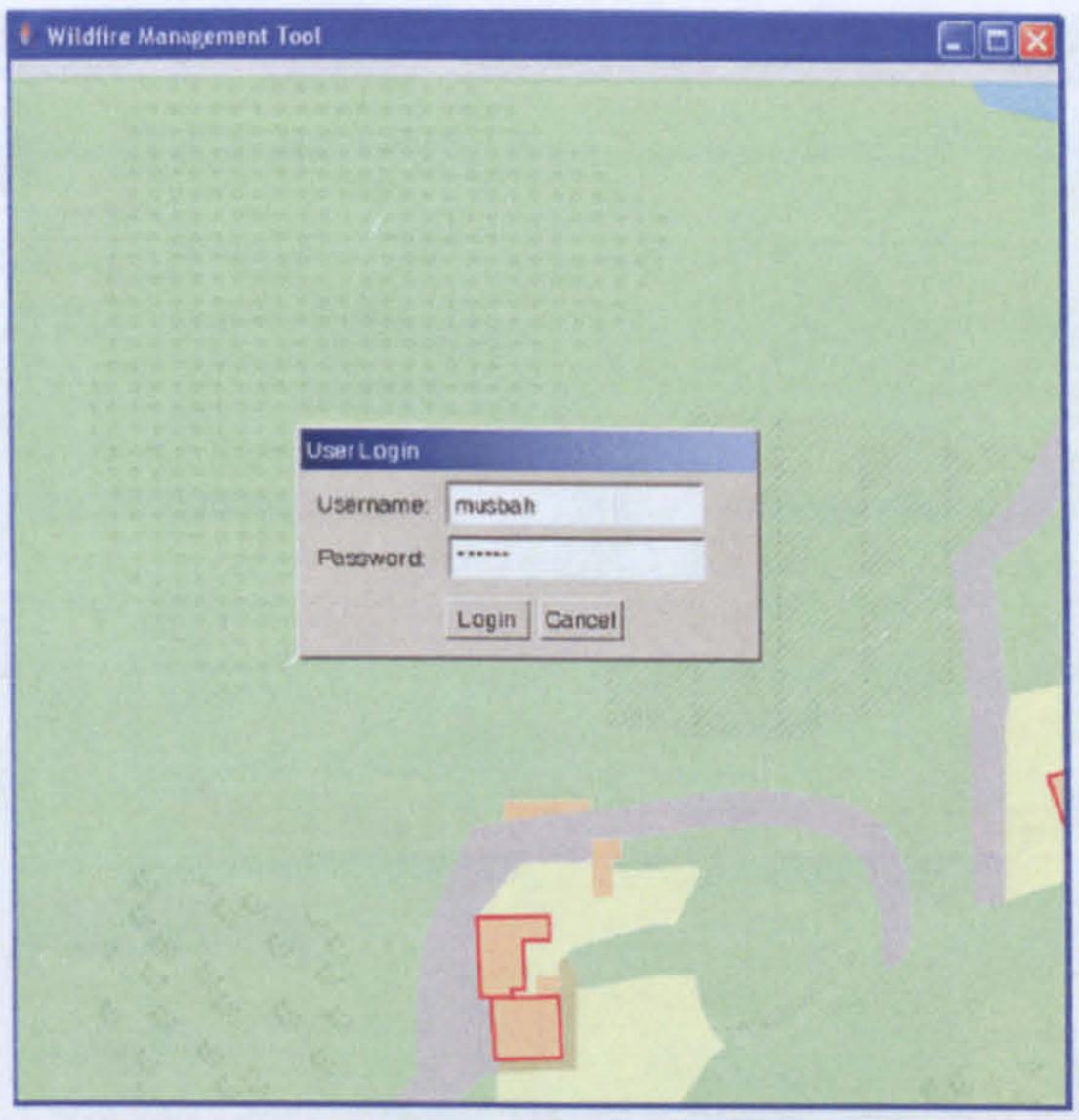


Figure 10-15: Wildfmt login window.

Figure 10-15 shows the login window for Wildfmt. As the emulation takes place and actors play the role of fire fighters and controllers in the emulated reality, actors start to use Wildfmt to help control the fire. Actors run Wildfmt using their assigned username and password. Each actor has specific role to play, a fire fighter or a controller and this is recognised by Wildfmt. Each role has different options available through Wildfmt. For instance controllers can control the view of the fire simulation while fire fighters can only register with the KB to retrieve fire simulation information. When the actor logs into Wildfmt, they can retrieve their location information by accessing the Settings option on the right-click menu (place the mouse cursor on an empty space in the workspace area and press the mouse right button).

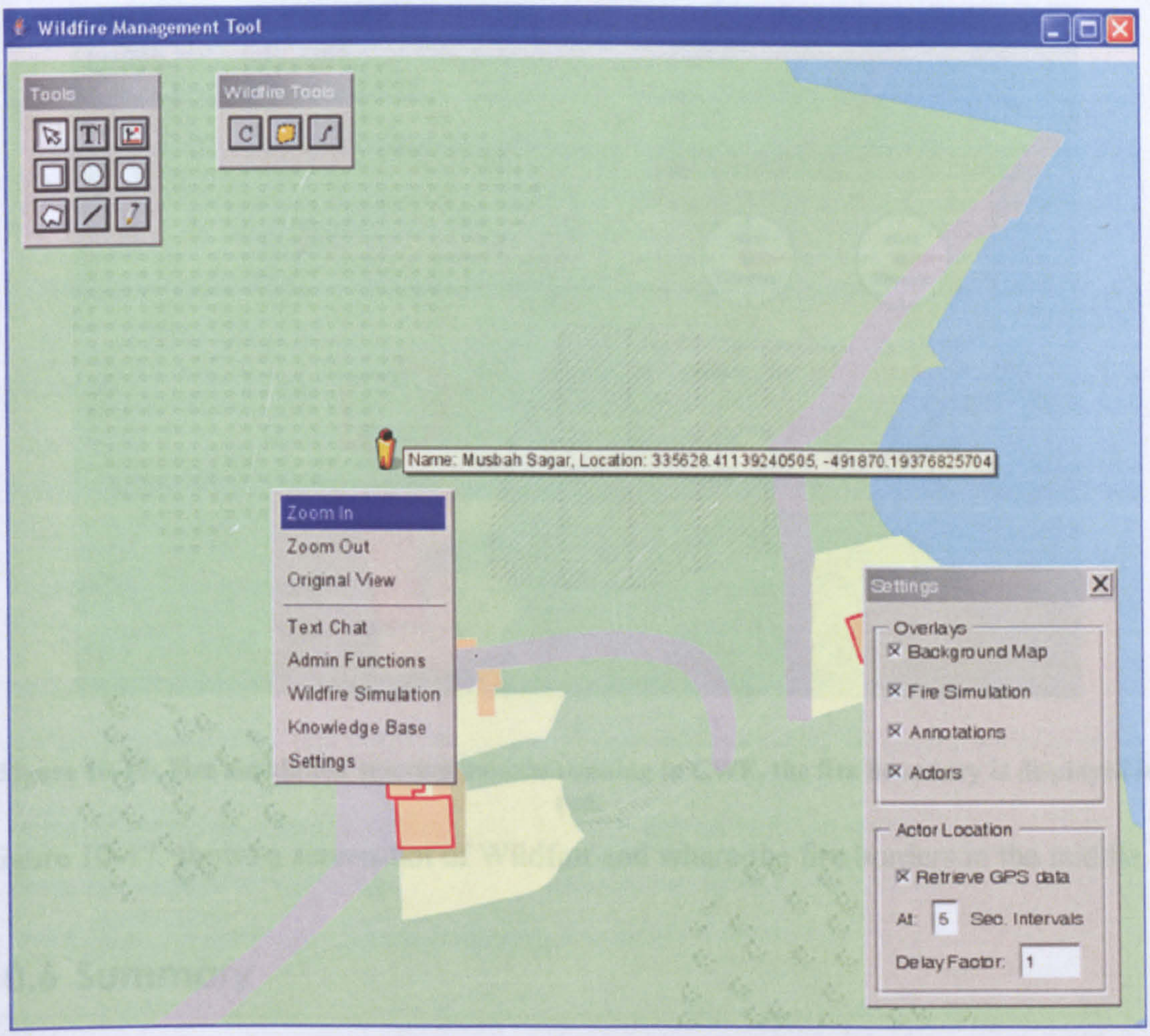


Figure 10-16: Widfmt Setting Window .

The Settings Window (see Figure 10-16) allows the actor to show or hide the background layers, for instance, the background map, fire simulation, annotations or actor locations. Fire simulation annotations and the location information are retrieved from the KB.

Figure 10-17: Fire simulation steering module running in CWE, the fire boundary is displayed in red.

Figure 10-17, shows a screenshot of Wildfmt and where the fire borders in the middle.

10.6 Summary

This chapter has described the successful attempt to deploy and use the KB and Wildfmt. The KB was only tested in replicated mode and performance measurements were not taken due to the complications and the many different aspects of our model. However, the five actors who participated in the demonstration successfully carried out an emulation of the detailed Application Scenario (see Section 10.1), using the emulation tools and Wildfmt. Also, the KB successfully stored Wildfmt annotations, as well as the locations of the fire fighters and the fire simulations predictions. This demonstrates that Web technologies can address the challenges expressed in the Application Scenario to build ACTs. The following chapter summarises the work presented in this thesis and sets directions for future research work.

11

Conclusion and Future Work

This chapter presents the main findings of this research, the research contributions, a summary of our approach and the open research issues and future work.

11.1 Main Findings of the Research:

The research work presented in this thesis investigated the design and development of collaborative applications. In this research a novel engineering method has been proposed to address the data aspect of building ACTs. The novel approach was built via the development of a four-layer model — a new generic architectural framework, a novel methodology and a set of engineered technologies to build ACTs. These are based on cutting-edge Web technologies and a Grid middleware infrastructure following the Open Overlays concept. Substantial code development and testing have been carried out to achieve the findings of this research.

The research has demonstrated that building collaborative applications is indeed a complex process and involves several complicated research issues. The research first analysed current collaborative applications to identify problems that were addressed by this research. The following restrictions were drawn from the review of the current approaches for building collaborative applications:

1. Target single platforms,
2. Designed to suit one type of devices,
3. Have a rigid choices of architecture (centralised or distributed),
4. Cannot be modified or changed to adapt to changes in the surrounding environment or any new requirements.

These observations and limitations were investigated and have been addressed in this research by proposing a new type of collaborative applications that are adaptive to HHEs. These collaborative applications exhibit many adaptive characteristics including:

1. Working across a wide variety of network infrastructures,
2. Operating on a spectrum of device types,
3. Acquiring data and information from heterogeneous sources without prior knowledge of their data schemes and structure and
4. Running in changing environments and having to adapt to varying requirements (i.e. reliability, scalability, etc.) that necessitate different styles of software architecture.

Based on the above analysis and observations this research has developed a new Web-based engineering method built through the development of a four-layer model. The four-layer layer model introduced in Chapter 3 is the answer this research provides. The model satisfies the research objectives described in Section 1.4. Furthermore, the benefit of this model is that it separates out a range of concerns so that they can be addressed individually. The four layers provided by this model each is concerning different aspects of building ACTs. The bottom layer is the Middleware layer which provides support for working on a range of network infrastructures and builds collaborative applications with flexible architecture that can change as required. The

second layer is the Group Communication layer. This layer hides the complexity of the Middleware layer and enables applications to collaborate by providing means for communication. On top of the Group Communication layer is the Knowledge Representation layer. This layer addresses two major concerns in building ACTs: the data modelling aspect and the data storage facilities (points 2 and 3 in the Aim and Objectives Section 1.4). It provides an approach to express and structure data and knowledge for collaborative applications to allow for data from a range of sources to be mixed freely and queried by the application without previous knowledge of its model and structure. This layer also addresses the storage facilities (the KB) that are required by ACTs for storing knowledge. The KB does not commit itself to a particular architecture (distributed or centralised) but rather it can be configured to operate in centralised, distributed or replicated structures. The top layer is the Presentation and Interaction layer which addresses point 1 in the Aim and Objectives Section 1.4. Different devices ranging from desktop computers, laptops, through to mobile devices have different specifications such as screen sizes and resolutions. This layer provides a method to develop user interfaces which can fit different types of devices. The Middleware and Group Communication layers were vital to the work achieved on the top two layers: the Knowledge Representation layer and the Presentation and Interaction layer. The Killer App (Chapter 10) addresses point 4 in the Aim and Objectives (see Section 1.4).

11.2 Contributions

This research has addressed a number of critical issues. The contributions of this work are summarised as follows.

1. The Platform-independent Data Model (RDFPIDM): a new method for using the RDF Data model as a platform and programming language independent

data framework, that features implicit support for collaboration and is suitable for different programming languages. It offers a new and open approach to the way computer applications are designed and constructed. It provides advantages over typical Type Systems such as more expressive powers to convey complex concepts and relations in the design and development stages, data and information querying.

2. The Knowledge Base (KB): characterises a reconfigurable knowledge storage system to store and query RDF data. The KB can operate in three modes: i) centralised, ii) distributed and iii) replicated. Furthermore, it can switch from one mode to the other without affecting the data consistency. Collaborative applications can use the KB service for collaboration and to store/retrieve their RDF data in a transparent manner regardless of its mode of operation.

3. The Oea framework: used to build adaptive and scalable user interfaces using SVG technology. This framework incorporates sub-contributions such as:

    i. A new model to write JavaScript code with an enhanced Object Oriented style (called ClassBJS, see Section 7.5), a Class-based Object Oriented model for a JavaScript environment that bear a strong resemblance to the OO model of languages such as C++ and Java. This approach enables programmers to port applications from other OOP languages such as Java and C++ to JavaScript/SVG. A large-scale Java application (JHotDraw) has been ported using this approach (see Chapter 8).

    ii. A novel method that provides a universal support for SVG format

to generate graphics and visual presentations instead of the approach that has been around for decades which uses custom-made, platform-dependent (and platform-independent) graphics packages (effectively, replacing package support with format support). This involves an extensive toolkit (called svgDraw2D, see Section 7.4.1) for manipulating SVG by providing a higher level of APIs and classes which helps to decouple the manipulation of DOM and SVG interfaces from writing graphical applications. The use of SVG makes it possible to accommodate the display requirements of virtually all types of devices that support SVG with all screen sizes and resolutions.

iii. A user-friendly new mouse event model (domMouse) that simplifies handling mouse events in an SVG environment and which solves the notorious out-of-synchrony problem largely present in current SVG applications (see Section 7.6).

iv. A Graphical User Interface (GUI, see Section 7.3.2) toolkit built to write graphical applications for SVG, featuring a wide set of widgets that are reliable, flexible, extensible and easy to use (i.e. EditBoxes, Windows, Buttons, etc.). This toolkit has been built on top of the svgDraw2D toolkit and domMouse. Developers can employ those widgets to create graphical applications that support user interactions through graphical interfaces in all SVG-enabled devices.

11.3 Our Approach in Five Points

This section summarises our approach in five points. To build ACTs one needs to:

1. Adopt our four-layer model.

2. Use RDF and RDFS for the design, data modelling and data representation.

3. Use our approach to build flexible data repositories (the KB) which can adapt architecturally (centralised, distributed and replicated) to changing environments and requirements.

4. Use the Oea framework to develop adaptive and scalable user interfaces.

5. Use the generic Group Abstraction Interface as a collaboration enabler.

11.4 Open Research Issues and Future Work

There are many areas where further research can be beneficial or which have not been covered fully and require future engineering work.

1. Investigating the use of ontology languages such as OWL as an alternative to RDFS for data modelling when developing ACTs in order to target specific communities to maximize knowledge reuse in the development process. OWL would provide more power in terms of expressing relationships between classes. Logic engines can also be used to analyse the knowledge and inference relationships between various data. The way this can be done is by replacing RDFS with OWL in the design stage. All classes of the collaborative application and their relationships would be written in OWL. The Jena engine supports OWL and can be used for the implementation and SPARQL can still be used to query the data.

2. More work on the KB is required. The current KB can work in three different modes: centralised, distributed or replicated. However, in order to switch between these modes, the collaborative application is forced into the quiescent state manually in order for the reconfiguration to take place. Further work is

required so that it would be possible to perform the transitions between the different modes of operation automatically. The trigger for the change could be driven by changes in the surrounding environment. This will include working on enhanced and reliable algorithms that would keep the integrity of the data stored in the KB in the transitions between modes. Also, the current KB lacks two important aspects of any storage facility: (1) Handling failure (2) Acknowledgment for completions of transactions (i.e. insert, delete, update etc). This should also be considered for future work [Porter, Taiani and Coulson, 2006].

3. Automatically binding of RDF classes and types: currently, the RDF classes and data developed in the design stage of the collaborative applications are manually mapped into the programming language specific types and classes (Java, JavaScript, etc.). There are recent methods that allow RDF classes and types to be bound to programming languages automatically. Further investigation to the possible advantages of binding the RDF data models with the underlying programming language, such as Java [Cowan, 2008] is required.

4. Additional investigation is required in order to enable the Oea framework to operate on small devices such as PDAs and mobile phones. Issues of user interface design and interactions for small devices are separate but vital and they have to be considered. There is a huge market for mobile phones applications at present. Much work has been invested in the Oea framework and a small amount of work is now required to make it compatible with small devices. Potentially, applications written with the improved Oea framework can adapt to work on mobile devices and larger computers without any changes. This would make the work of this research valuable to millions of users around the world.

5. This research did not address the security concerns that accompany the

development of collaborative applications. Further work on this aspect of building collaborative applications can be hugely beneficial. Improvements to our model would include providing secure group communication and providing choices for data encryption to secure the RDF data stored in the KB.

6. This research has demonstrated a new approach to port Java applications to SVG and JavaScript following the Oea framework. The process involved manually converting Java classes and code into JavaScript based on the ClassBJS model and the support of other Oea framework packages. Although JavaScript and Java languages have similar syntax, however, the process of porting Java code into JavaScript is lengthy and laborious. One improvement to this not ideal circumstance is to develop a program that would do this job automatically. This will save time and effort and can benefit this research greatly in bringing more Java applications to work in SVG using the model developed in this research (Oea framework).

7. Recently, Scalable Vector Graphics (SVG) Tiny 1.2 Specification has become W3C Recommendation in 22 December 2008. More investigation is required to adapt the Oea framework to work on SVG Tiny 1.2 (see Section 7.1).

8. A release of the Oea framework code to the general public would be a good step forward. This will help to develop the framework further and engage the Web community into testing it.

Glossary of Terms

Terms marked with '' are especially defined and used in this thesis only.*

802.1	A family of standards which was developed by IEEE to cover wireless networking.
ABR	Available Bit Rate.
ACT*	Adaptive Collaborative applicaTion is a computer collaborative application designed to overcome the challenges posed by Highly Heterogeneous Environments.
ad hoc	A Latin phrase used to describe a solution or a design for a specific problem or task and which cannot be used to serve other purposes.
AI	Artificial intelligence.
Ajar*	A JavaScript package for RDF that supports interactions with external knowledge storage facilities (i.e. KB, see Chapter 6) using RDF and SPARQL as a query language.
Ajax	Asynchronous JavaScript and XML.
API	Application Programming Interface.
ASP	Active Server Pages, is Microsoft's first server-side script engine for dynamically-generated web pages.
ATM	Asynchronous Transfer Mode.
AWT	Abstract Window Toolkit is a Java GUI toolkit.
Bluetooth	A wireless protocol for short distances communication and data transmission.
Chord	A protocol which proposes a solution to the problem of efficient data location on P2P networks.
Chord DHT	An implementation of DHT on Chord P2P network.
ClassBJS*	Class-based Object Oriented JavaScript.
CORBA	Common Object Requesting Broker Architecture is a standard defined by the Object Management Group to enable software components written in multiple computer languages and running on multiple computers to work together.
CoRDF*	Collaborative RDF, an engineering approach addressing the data and knowledge heterogeneity aspect of building ACTs.
CPU	Central Processing Unit.
CSCW	Computer Supported Cooperative Work is the area of computer science that specialises in designating and developing collaborative applications to support cooperative group work.
CSS	W3C standard for Cascading Style Sheet.

CWE*	Collaborative Workspace Environment is a graphical tool that supports effective communication using graphical annotations on a shared work surface. CWE serves as an exemplar for ACT.
DHT	Distributed Hash Table.
DOM	Document Object Model - W3C standard for a platform- and language-neutral interface that will allow applications to dynamically access and update the content, structure and style of XML documents.
domMouse*	An advanced mouse event model for DOM that resolves the out-of-sync problem (see Section 7.6), and which supports an elegant set of mouse events that makes the mouse event handling process straightforward.
drawops	Drawing operation is each action to draw, delete or move a drawing in WB.
EPSRC	Engineering and Physical Sciences Research Council.
e-Science	The UK e-Science Programme which started in 2001.
FOAF	The Friend of a Friend, an ontology to describe persons, their relationships and activities with others.
GAI*	Group Abstraction Interface.
GI *	Group Interface is used to allow users to retrieve references to group members and broadcast messages to individual members or to the whole group.
GMI *	Group Management Interface is used to manage groups (creation, deletion, etc.).
GPS	Global Positioning System is a global navigation satellite system that enables GPS receivers to determine their current location, time, and velocity.
Grid	A paradigm that is used to harness scattered resources such as, supercomputers, storage resources, data sources, sensors and privileged devices.
Gridkit	A middleware infrastructure, based on the Open Overlays concept.
Groupware	Software that helps groups of people work together.
GUI	Graphical User Interface.
H.320	Video conferencing protocols suite.
HHE*	Highly Heterogeneous Environment exhibits many dimensions of heterogeneity such as network heterogeneity, device type heterogeneity, data heterogeneity and architectural heterogeneity.
HTML	Hyper Text Markup Language.

IDL	Interface Description Language.
IEEE	Institute of Electrical and Electronics Engineers.
ILS	Internet Location Server.
IP	Internet Protocol.
IRDFStore*	A Java interface used to allow data to be inserted, updated, removed and queried.
IRDFStoreClient*	A Java interface used to allow receiving callback replays from a remote RDF Service.
JGroups	A Java package for reliable group (or multicast) communication.
JMS	Java Message Service API, a Java Message Oriented Middleware API for sending messages between clients.
JVM	Java Virtual Machine.
KB*	Knowledge Base is a reconfigurable knowledge storage system to store and query RDF data.
KBR	Key Based Routing.
LAN	Local Area Network.
LBL	Lawrence Berkeley Laboratory.
MBone	Multicast Backbone.
MCU	Multipoint Control Unit, a central server used with H.320 protocol to support group communication in NetMeeting.
Middleware	A software layer that addresses network and operating system heterogeneity.
MOM	Message Oriented Middleware.
MPI	Message Passing Interface.
MVDHT*	Multi-value Distributed Hash Table.
N3	Notation 3 language.
NetMWB*	Microsoft NetMeeting Whiteboard.
Oea	The ancient name of Tripoli (Libya), a city founded in the 7th century BC by the Phoenicians.
Oea framework *	A framework to develop Web-based user interfaces for ACTs using SVG.
OMG	Object Management Group is a consortium for setting standards for distributed object-oriented systems and model-based standards.
OOP	Object Oriented Programming.
Open Overlays	A new approach that addresses the issue of network and platform heterogeneity in the Grid middleware.
OpenCOM	A platform and programming language independent component model that can be used to construct component-based systems which can be reconfigured at runtime.

OpenGL	Open Graphics Library is cross-platform and cross-language standard specification for writing 2D and 3D graphics applications.
OWL	Web Ontology Language is a W3C standard used for authoring ontologies.
P2P	Peer-to-Peer.
PDA	Personal Digital Assistant is a handheld computer.
PHP	A scripting language originally designed for producing dynamic web pages.
RAM	Random Access Memory.
RDF	Resource Description Framework.
RDFPeers	A Scalable Distributed RDF Repository based on A Structured P2P Network.
RDFPIDM*	The RDF Platform-Independent Data Model proposes to use RDF and RDFS technologies as the data model for the design and the implementation of ACTs.
RDFS	Resource Description Framework Schema.
RemoteRDFStore*	A Java class implementation used to interact with a remote RDF Service.
Scribe	A large-scale, decentralized application-level multicast infrastructure; built on top of Pastry.
Skype	Internet messaging software that allows users to make telephone calls based on P2P architecture.
SOAP	W3C standards for exchanging structured information in a distributed environment using XML technologies.
SPARQL	W3C standard for SPARQL Query Language for RDF.
SQL	Structured Query Language.
SVG	Scalable Vector Graphics, a W3C vector graphics standard
svgDraw2D *	A lightweight 2D graphics package to generate graphics and drawings.
svgSwing*	A GUI framework that provides the Oea framework applications with a reliable, stable and interactive user interface widgets (i.e. textbox, button, window, etc.).
T.120	The data protocol responsible for multimedia conferencing and is used by NetMWB.
TBCP	Tree Building Control Protocol.
TCP	Transmission Control Protocol.
UDP	User Data Protocol Transmission Control Protocol.
VRML	Virtual Reality Modelling Language.
W3C	The World Wide Web Consortium.

WAN	Wide Area Network.
WB	LBL Whiteboard.
Whiteboard	A computer application that mimics the capabilities of a physical whiteboard.
Wildfmt *	Wildfire Management Tool is an application built on the principle ideas of CWE.
Wireless ad hoc network	Specialized wireless networks where nodes forward data to nearby neighbouring nodes based on connectivity.
XHTML	Extensible Hypertext Markup Language, is a markup language that conforms to XML syntax and has the same expression as HTML.
XMPP	Extensible Messaging and Presence Protocol.

References

[Aberer and Hauswirth, 2002]. *Karl Aberer and Manfred Hauswirth* (2002) ,"An Overview on Peer-to-Peer Information Systems" , Workshop on Distributed Data and Structures, Switzerland, volume 14, pp.171-188, Available at http://lsirpeople.epfl.ch/hauswirth/papers/WDAS2002.pdf

[ACCDC]. "Atlantic Canada Conservation Data Centre" , Available at http://www.accdc.com/contact/index.html [22 September 2008]

[Alexander, Jshikawa et al., 1977]. *Christopher Alexander, Sara Jshikawa, Murrar Silverstein, Max Jacobson, Ingrid Fiksdahl-King and Shlomo Angle* (1977) ,"A Pattern Language" , Oxford University Press, USA

[Antoniou and van Harmelen, 2004]. *Grigoris Antoniou and Frank Van Harmelen* (2004) ,"The Semantic Web Primer" , MIT Press, ISBN 0262012103

[ASV3]. "Adobe SVG Viewer 3.0" , Available at http://www.adobe.com/svg/viewer/install/mainframed.html [22 September 2008]

[Ban]. *Bela Ban* ,"JGroups, a toolkit for reliable multicast communication" , Available at http://www.jgroups.org/ [22 September 2008]

[Bannon and Schmidt, 1989]. *Liam Bannon and Kjeld Schmidt* (1989) ,"CSCW: Four Characters in Search of a Context" , First European Conference on CSCW, Gatwick, UK

[Bannon, Ehn et al., 1988]. *Liam Bannon, Pelle Ehn, Irene Greif, Robert Howard and Mark Stefik* (1988) ,"CSCW: What does it mean?" , ACM conference on CSCW, Portland, Oregon, United States

[Batik]. "Apache Batik SVG Toolkit" , Available at http://xmlgraphics.apache.org/batik/ [22 September 2008]

[Bell, 2005] *Douglas Bell* (2005) ," Software Engineering for Students " , Addison-Wesley, ISBN 0321261275

[Berners-Lee, 1994]. *Tim Berners-Lee* (1994) ,"The World Wide Web Consortium (W3C)" , Available at http://www.w3.org/ [22 September 2008]

[Berners-Lee, Fielding and Masinter, 2005]. *Tim Berners-Lee, Roy Fielding and Larry Masinter* (2005) ,"Uniform Resource Identifier (URI): Generic Syntax" , Available at http://www.ietf.org/rfc/rfc3986.txt [22 September 2008]

[Berners-Lee, Hendler and Lassila, 2001]. *Tim Berners-Lee, James Hendler and Ora Lassila* (2001) ," The Semantic Web" , SCIENTIFIC AMERICAN -AMERICAN EDITION-, volume 284(5), pp. 28--37

[Bitflash]. "Bitflash" , Available at http://www.bitflash.com [18 October 2008]

[Blair, Coulson et al.]. *Gordon Blair, Geoff Coulson, Laurent Mathy, Paul Grace, Wai-Kit Yeung, Barry Porter, Wei Cai, Chris Cooper, David Duce, Musbah Sagar et al.* ,"Open Overlays Project" , Available at http://www.comp.lancs.ac.uk/computing/research/mpg/projects/openoverlays/index.htm [22 September 2008]

[Brant, 2006]. *John Brant* (2006) ,"HotDraw Applications" , Available at http://st-www.cs.uiuc.edu/users/brant/HotDraw/HotDraw-applications.html#HotPaint [22 September 2008]

[Brickley and Epinions]. *Dan Brickley and R.V. Guha Epinions* ,"Resource Description Framework Schema (RDFS) Specification 1.0, W3C Candidate Recommendation" , Available at http://www.w3.org/TR/2000/CR-rdf-schema-20000327/ [22 September 2008]

[Cai and Frank, 2004]. *Min Cai and Martin Frank* (2004) ,"RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network" , 13th international conference on World Wide Web, New York, NY, USA, pp.650 - 657 , ISBN 1-58113-844-X

[cardhouse, 2006]. *Cardhouse* (2006) , "Raster To Vector Algorithm " , Available at http://cardhouse.com/computer/vector.htm [2 December 2008]

[Castro, Druschel et al., 2002]. *Miguel Castro, Peter Druschel, Anne-marie Kermarrec and Antony Rowstron* (2002) ,"SCRIBE: A large-scale and decentralized application-level multicast infrastructure" , IEEE Journal on Selected Areas in Communications (JSAC) , volume 20, issue 8, pp.1489-1499

[C-sharp]. " C# Language Specification" , Available at http://en.csharp-online.net/CSharp_Language_Specification [20 November 2008]

[Coulson, Blair et al., 2004]. *Geoff Coulson, Gordon Blair, Paul Grace, Ackbar Joolia, Kevin Lee and Jo Ueyama* (2004) ,"OpenCOM v2: A Component Model for Building Systems Software" , IASTED Software Engineering and Applications, Cambridge, MA, ESA

[Cowan, 2008]. *Taylor Cowan* (2008) ,"Jenabean: Easily bind JavaBeans to RDF" , Available at http://www.ibm.com/developerworks/library/j-jenabean.html [22 September 2008]

[Crockford]. *Douglas Crockford* ,"Classical Inheritance in JavaScript" , Available at http://www.crockford.com/javascript/inheritance.html [4 January 2008]

[Crockford]. *Douglas Crockford* ,"Private Members in JavaScript" , Available at http://javascript.crockford.com/private.html [4 January 2008]

[Date and Darwen, 1997]. *C. J. Date and Hugh Darwen* (1997) ,"A Guide to SQL Standard, 4th Edition" , Addison-Wesley, ISBN 0-201-96426-0

[De-Meuter, D'hondt et al., 2003]. *Wolfgang De-Meuter, Theo D'hondt, Jessie Dedecker, Manfred Broy and Alexandre V. Zamulin* (2003) ,"Intersecting Classes and

Prototypes" , Fifth International Conference on Perspectives of System Informatics, in Andrei Ershov , Siberia, Russia

[Druschel and Rowstron, 2001]. *Peter Druschel and Antony Rowstron* (2001) ,"Past: Persistent and anonymous storage in a peer-to-peer networking environment" , The 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII), pp.65--70

[E.Krasner and Pope, 1988]. *Glenn E.Krasner and Stephen T. Pope* (1988) ,"A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80" , Object Oriented Programming, volume 1(3), pp.26--49

[Ecma262]. *Ecma262* ,"Ecma-262" , Available at http://www.ecma-international.org/publications/standards/Ecma-262.htm [22 September 2008]

[ECMAScript4]. *ECMAScript4* ,"ECMAScript Edition 4" , Available at http://www.ecmascript.org/ [22 September 2008]

[Edwards]. *Dean Edwards* ,"A Base Class for JavaScript Inheritance" , Available at http://dean.edwards.name/weblog/2006/03/base/ [4 January 2008]

[Elliott, Eckstein et al., 2002]. *James Elliott, Robert Eckstein, Marc Loy, David Wood and Brian Cole* (2002) ,"Java Swing", O'Reilly Media, Inc, ISBN 0-596-00408-7

[Elonen, 2001]. *Jarno Elonen* (2001) ,"NanoHTTPD" , Available at http://elonen.iki.fi/code/nanohttpd/

[Emia]. *Systems Emia* ,"The Renesis SVG 1.2 Player" , Available at http://www.gosvg.net [22 September 2008]

[Emmerich and Gruhn, 2004]. *Wolfgang Emmerich and Volker Gruhn* (2004) ,"Engineering Distributed Objects" , Wiley, ISBN 0471986577

[Eriksson, 1994]. *Hans Eriksson* (1994) ,"Mbone: The Multicast Backbone" , Communications of the ACM, volume 37, pp.54-60

[Ferraiolo, 2008]. *Jon Ferraiolo* (2008) ," How Ajax Changes the Game for SVG " , 6th International Conference on Scalable Vector Graphics, SVG Open 2008, Available at http://www.svgopen.org/2008/papers/63-How_Ajax_Changes_the_Game_for_SVG/ [13 January 2009]

[Ferraiolo, Duce et al., 2005]. *Jon Ferraiolo, David Duce, Bob Hopgood and John Bowler* (2005) ,"Scalable Vector Graphics (SVG) Full 1.2 Specification, W3C Working " , Available at http://www.w3.org/TR/SVG12/ [22 September 2008]

[Ferraiolo, Duce et al.]. *Jon Ferraiolo, David Duce, Bob Hopgood and John Bowler* ,"Scalable Vector Graphics (SVG) 1.0 Specification, W3C Recommendation" , Available at http://www.w3.org/TR/2001/REC-SVG-20010904/ [22 September 2008]

[Fettes and Mansfield, 2004]. *Alastair Fettes and Philip Mansfield* (2004) ,"SVG-Based User Interface Framework" , SVG Open, Available at http://www.svgopen.org/2004/papers/SPARK/

[Flanagan, 2001]. *David Flanagan* (2001) ,"JavaScript: The Definitive Guide" , O'Reilly Media, Inc, ISBN 0596000480

[Foster, 2006]. *Ian Foster* (2006) ,"Globus Toolkit Version 4: Software for Service-Oriented Systems" , IFIP International Conference on Network and Parallel Computing, pp.pp 2-13, Available at http://www.globus.org/alliance/publications/papers/IFIP-2006.pdf

[Frankel and Pepper, 2000]. *Justin Frankel and Pepper* (2000) ,"Gnutella" , Available at http://en.wikipedia.org/wiki/Gnutella [22 September 2008]

[Gabriel, E. et al., 2004]. *Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Sahay, Prabhanjan Kambadur, , et al.* (2004) "Open {MPI}: Goals, Concept, and Design of a Next Generation {MPI} Implementation" , 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, pp.97--104

[Gamma and Eggenschwiler]. *Erich Gamma and Thomas Eggenschwiler* ,"Java Graphical Editing Framework (JHotDraw)" , Available at http://www.jhotdraw.org/ [22 September 2008]

[Gamma, Helm et al., 1994]. *Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides* (1994) ,"Design Patterns: Elements of Reusable Object-Oriented Software" , Addison-Wesley, ISBN 0201633612

[Garrett, 2005]. *Jesse James Garrett* (2005) ," Ajax: A New Approach to Web Applications" , Available at http://www.adaptivepath.com/ideas/essays/archives/000385.php [20 November 2008]

[Gertzen]. *Joshua Gertzen* ,"Object-Oriented Super Class Method Calling with JavaScript" , Available at http://truecode.blogspot.com/2006/08/object-oriented-super-class-meth [4 January 2008]

[Glover, Miller et al., 2005]. *Derekl Glover, Davidl Miller, Dougl Averis and Victorial Door* (2005) ,"The interactive whiteboard: a literature survey " , Technology, Pedagogy and Education, volume 14, pp.155-170(16)

[Grace, Coulson et al., 2004]. *Paul Grace, Geoff Coulson, Gordon Blair, Laurent Mathy, Wai Kit Yeung, Wei Cai, David Duce and Chris Cooper* (2004) ,"GRIDKIT: Pluggable Overlay Networks for Grid Computing" , International Symposium on Distributed Objects and Applications (DOA), Larnaca, Cyprus, pp.1463--1481

[Grace, Coulson et al., 2005]. *Paul Grace, Coulson Geoff, Gordon Blair, Barry Porter, Wei Cai, David Duce, Chris Copper, Muhammad Younas, Musbah Sagar and Wei Li* (2005) ,"Open Overlay Support for the Divergent Grid" , UK E-Science All Hands Meeting , Available at http://csdl.computer.org/comp/proceedings/icdcsw/2003/1921/00/19210382abs.htm

[Grace, Hughes et al., 2008]. *Paul Grace, Danny Hughes, Barry Porter, Gordon Blair, Geoff Coulson and Francois Taiani* (2008) ,"Experiences with Open Overlays: A Middleware Approach to Network Heterogeneity" , ACM International EuroSys Conference '08, Glasgow, Scotland

[Greif, 1988]. *Irene Greif* (1988) ,"Computer Supported Cooperative Work: A Book of Readings" , Morgan Kaufmann Publishers, ISBN 0934613575

[Grudin, 1994]. *Jonathan Grudin* (1994) ,"Computer-supported cooperative work: history and focus" , Computer, volume 27, pp.19-26

[H.320]. "H.320" , Available at http://www.itu.int/rec/T-REC-H.320-200403-I/en [22 September 2008]

[Ingalls, Kaehler et al., 1997]. *Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace and Alan Kay* (1997) ,"Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself " , OOPSLA '97, ACM SIGPLAN Notices, pp.318--326, Available at http://users.ipa.net/~dwighth/squeak/oopsla_squeak.html

[Carroll and Reynolds, 2004]. *Jeremy Carroll and Dave Reynolds* (2004) ,"Jena: Implementing the semantic web recommendations" , pp.74--83

[Jacobson and McCanne, 1994]. *Van Jacobson and Steven McCanne* (1994),"LBL Whiteboard, Lawrence Berkeley Laboratory" , Available at http://ee.lbl.gov/wb/ [22 September 2008]

[Jannotti, Gifford et al., 2000]. *John Jannotti, David K. Gifford, Kirk L. Johnson, M. rans Kaashoek and James Jr. O'Toole* (2000) ,"Overcast: Reliable Multicasting with an Overlay Network." , 4th conference on Symposium on Operating System Design & Implementation, volume 4, pp.197-212

[JME]. "Java Platform, Micro Edition" , Available at http://java.sun.com/javame/index.jsp [30 April 2009]

[Johansen, 1988]. *Robert Johansen* (1988) ,"Groupware: computer support for business teams" , The Free 1, a division of Macmillan, Inc (New York)., pp.192

[Johnson, 1992]. *Ralph Johnson* (1992) ,"Documenting Framworks using Patterns" , Object Oriented Programming Systems Languages and Applications, Vancouver, British Columbia, Canada , pp.63 - 76

[Kaiser, 2001]. *Wolfram Kaiser* (2001) ,"Become a programming Picasso with JHotDraw" , Available at http://www.javaworld.com/javaworld/jw-02-2001/jw-0216-jhotdraw.html [19 September 2008]

[Kesselman and Foster, 1998]. *Kesselman, Carl and Foster, Ian* (1998) ,"The Grid: Blueprint for a New Computing Infrastructure" , Morgan Kaufmann Publishers, ISBN 1558604758

[Kim, 1979]. *Won Kim* (1979) ,"Relational Database Systems." , ACM Computing Surveys, volume 11, pp.187-211, Available at db/journals/csur/Kim79.html

[Kim, 1993]. *Won Kim* (1993) ,"Object-Oriented Database Systems: Promises, Reality, and Future" , 19th International Conference on Very Large Data Bases , San Francisco, CA, USA, pp.676 - 687

[King, 1980]. *W. Frank King III* (1980) ,"Relational Database Systems: Where We Stand Today." , ZFZP Congress on Information Processing, pp.369-381

[Knublauch, Oberle et al., 2006]. *Holger Knublauch, Daniel Oberle, Phil Tetlow and Evan Wallace* (2006) ,"A Semantic Web Primer for Object-Oriented Software Developers" , Available at http://www.w3.org/TR/sw-oosd-primer/ [22 September 2008]

[Le-Hors, Wood et al., 2004]. *Arnaud Le-Hors, Lauren Wood, Gavin Nicol, Inso EPS, Jonathan Robie, Philippe Le-Hégaret, Mike Champion and Steve Byrne* ,"Document Object Model (DOM) Level 3 Core Specification, Version 1.0 W3C Recommendation " , Available at http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/ [22 September 2008]

[Lindsey, 2000]. *Kevin Lindsey* (2000) ,"KevLinDev GUI" , Available at http://www.kevlindev.com/gui/index.htm [22 September 2008]

[Lindsey]. *Kevin Lindsey* ,"JavaScript Inheritance" , Available at http://www.kevlindev.com/tutorials/javascript/inheritance/ [4 January 2008]

[Maloney, 1995]. *John Maloney* (1995) ,"Morphic: The Self User Interface Framework" , Self 4.0 Release Documentation, Available at ftp://ftp.squeak.org/docs/Self-4.0-UI-Framework.pdf

[Manola and Miller]. *Frank Manola and Eric Miller* ,"RDF Primer, W3C Recommendation" , Available at http://www.w3.org/TR/rdf-primer/ [22 September 2008]

[Mathy, Canonico and Hutchison, 2001]. *Laurent Mathy, Roberto Canonico and David Hutchison* (2001) ,"An Overlay Tree Building Control Protocol" , Third International Workshop on Networked Group Communication, pp.76--87

[McBride, 2002]. *Brian McBride* (2002) ,"Jena: A Semantic Web Toolkit." , IEEE Internet Computing, volume 6, pp.55-59, Available at http://www.computer.org/internet/ic2002/w6055abs.htm

[Michael, Blair et al., 2001]. *Clarke Michael, Gordon Blair, Geoff Coulson and Nikos Parlavantzas* (2001) ,"An Efficient Component Model for the Construction of Adaptive Middleware" , IFIP Middleware, Heidelberg, Germany

[Mistry and Berardi, 2005]. *Jayalaxshmi Mistry and Andrea Berardi* (2005) ,"Assessing Fire Potential in a Brazilian Savanna Nature Reserve" , Biotropica, volume 37, pp.439-451, ISBN 0006-3606

[MSPaint]. "Microsoft Paint" , Available at http://www.lkwdpl.org/classes/MSPaint/paint.html [22 September 2008]

[OperaMobile]. "Opera Mobile" , Available at http://www.opera.com/products/mobile/ [19 November 2008]

[Orfali and Harkey, 1997]. *Robert Orfali and Dan Harkey* (1997) ,"Client Server Programming With Java and Corba" , Wiley, ISBN 0471163511

[OWL]. "OWL Web Ontology Language" , Available at http://www.w3.org/2004/OWL/ [22 September 2008]

[Porter, Taiani and Coulson, 2006]. *Barry Porter, Francois Taiani and Geoff Coulson* (2006) ," Generalised Repair for Overlay Networks" , Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems, pp. 132 – 142, IEEE Computer Society Washington, DC, USA, ISBN 0-7695-2677-2

[Prud'hommeaux and Seaborne, 2006]. *Eric Prud'hommeaux and Andy Seaborne* (2006) ,"SPARQL Query Language for RDF, W3C Working Draft " , Available at http://www.w3.org/TR/rdf-sparql-query/ [22 September 2008]

[Prud'hommeaux and Seaborne]. *Eric Prud'hommeaux and Andy Seaborne* ,"SPARQL Query Language for RDF" , Available at http://www.w3.org/TR/rdf-sparql-query/ [22 September 2008]

[Ratnasamy, Francis et al., 2001]. *Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp and Scott Shenker* (2001) ,"A scalable content-addressable network" , Applications, technologies, architectures, and protocols for computer communications, San Diego, California, United States, pp.161-172, Available at http://www.acm.org/sigs/sigcomm/sigcomm2001/p13-ratnasamy.pdf

[Sagar, Duce and Cooper, 2005]. *Musbah Sagar, David Duce and Chris Cooper* (2005) ,"Advanced Mouse Event Model for SVG" , 4th Annual Conference on Scalable Vector Graphics, SVG Open 2005, Enschede, the Netherlands, August 2005.

[Sagar, Duce et al., 2008]. *Musbah Sagar, David Duce, Mohammed Younas and* (2008) ,"The Oea Framework for Class-Based Object Oriented Style JavaScript for Web Programming" , Computer Standards & Interfaces (2008), doi:10.1016/j.csi.2008.03.014

[schemagen]. "Jena schemagen" , Available at http://jena.sourceforge.net/how-to/schemagen.html [10 November September 2008]

[Schich and Cyganiak, 2008]. *Maximilian Schich and Richard Cyganiak* ,"Sparql Update Language" , Available at http://esw.w3.org/topic/SparqlUpdateLanguage [22 September 2008]

[Siegrist, 2008]. *Kyle Siegrist,"* The Fire Process" , Available at http://www.math.uah.edu/stat/particles/Fire.xhtml [22 September 2008]

[Skype]. "Skype" , Available at http://www.skype.com/ [2 December 2008]

[Snirm and Otto, 1998]. *Marc Snirm and Steve Otto* (1998) ,"MPI-The Complete Reference: The MPI Core" , MIT Press, ISBN 0262692155

[Stoica, Morris et al., 2001]. *Ion Stoica, Robert Morris, David Karger, M.Frans Kaashoek and Hari Balakrishnan* (2001) ,"Chord: A Scalable Peertopeer Lookup Service for Internet Applications" , SIGCOMM, pp.149--160, Available at http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

[Summers, 1998]. *Robert Summers* (1998) ,"Official Microsoft NetMeeting Book" , Microsoft Press, ISBN 1-57231-816-3

[Szyperski, 1997]. *Clemens Szyperski* (1998) ," Component Software: Beyond Object-Oriented Programming" , Addison-Wesley, ISBN 0201178885

[Taivalsaari, Mikkonen et al., 2008]. *Antero Taivalsaari, Tommi Mikkonen, Dan Ingalls and Krzysztof Palacz* (2008) ,"Web Browser as an Application Platform:The Lively Kernel Experience" , Sun Microsystems Laboratories Technical Report, Available at http://research.sun.com/techrep/2008/smli_tr-2008-175.pdf

[Talk]. "Google Talk" , Available at http://www.google.com/talk/ [22 September 2008]

[Tomek, 1999]. *Ivan Tomek* (1999) ,"Visualworks Smalltalk: An Introduction" , Addison-Wesley, ISBN 0201895455

[van Harmelen and McGuinness, 2004]. *Frank Van-Harmelen and Deborah L. McGuinness* (2004) ,"Web Ontology Language, W3C Recommendation" , Available at http://www.w3.org/TR/owl-features/ [22 September 2008]

[VML]. "Vector Markup Language (VML)" , Available at http://www.w3.org/TR/1998/NOTE-VML-19980513 [30 April 2009]

[Wilson, 1991]. *Paul Wilson* (1991) ,"Computer Supported Cooperative Work: An Introduction" , Springer, ISBN 0792314468

[XMLsdtRDFOWL]. "XML Schema Datatypes in RDF and OWL" , Available at http://www.w3.org/TR/swbp-xsch-datatypes [10 November 2008]

[XMPP]. "Extensible Messaging and Presence Protocol (XMPP)" , Available at http://www.xmpp.org/ [22 September 2008]

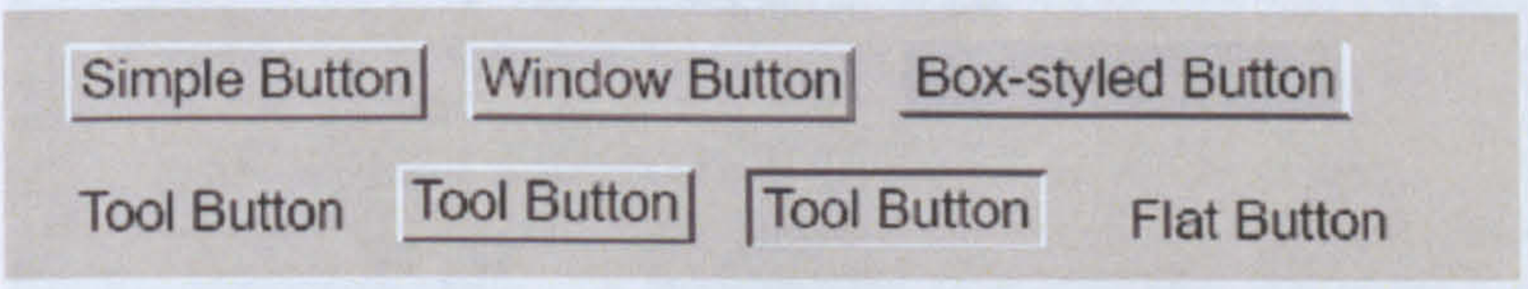
[XQuery]. " XQuery 1.0: An XML Query Language" , Available at http://www.w3.org/TR/xquery/ [23 April 2009]

[Zukowski, 1997]. *John Zukowski* (1997) ,"Java AWT Reference" , Available at http://www.oreilly.com/catalog/javawt/book/index.html [22 September 2008]

Appendix I: svgSwing Picture Gallery

1. Label

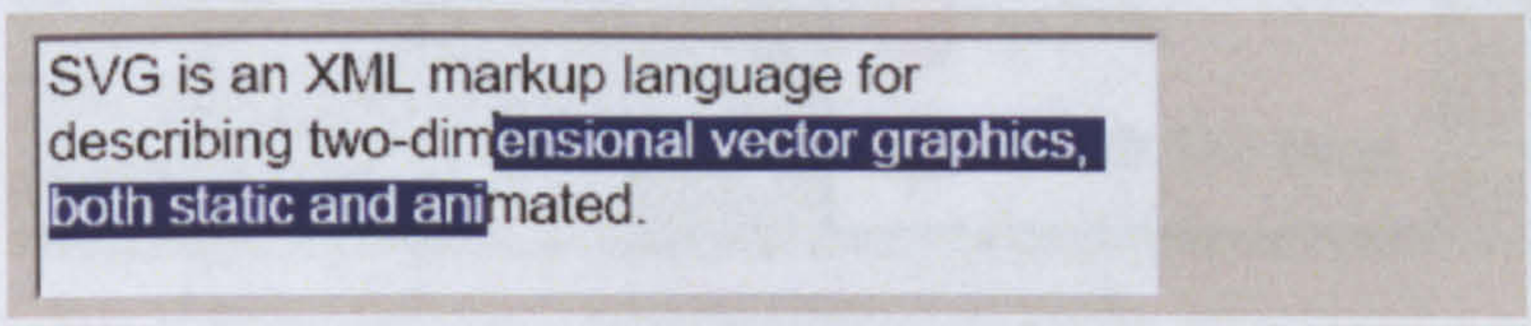
2. Button



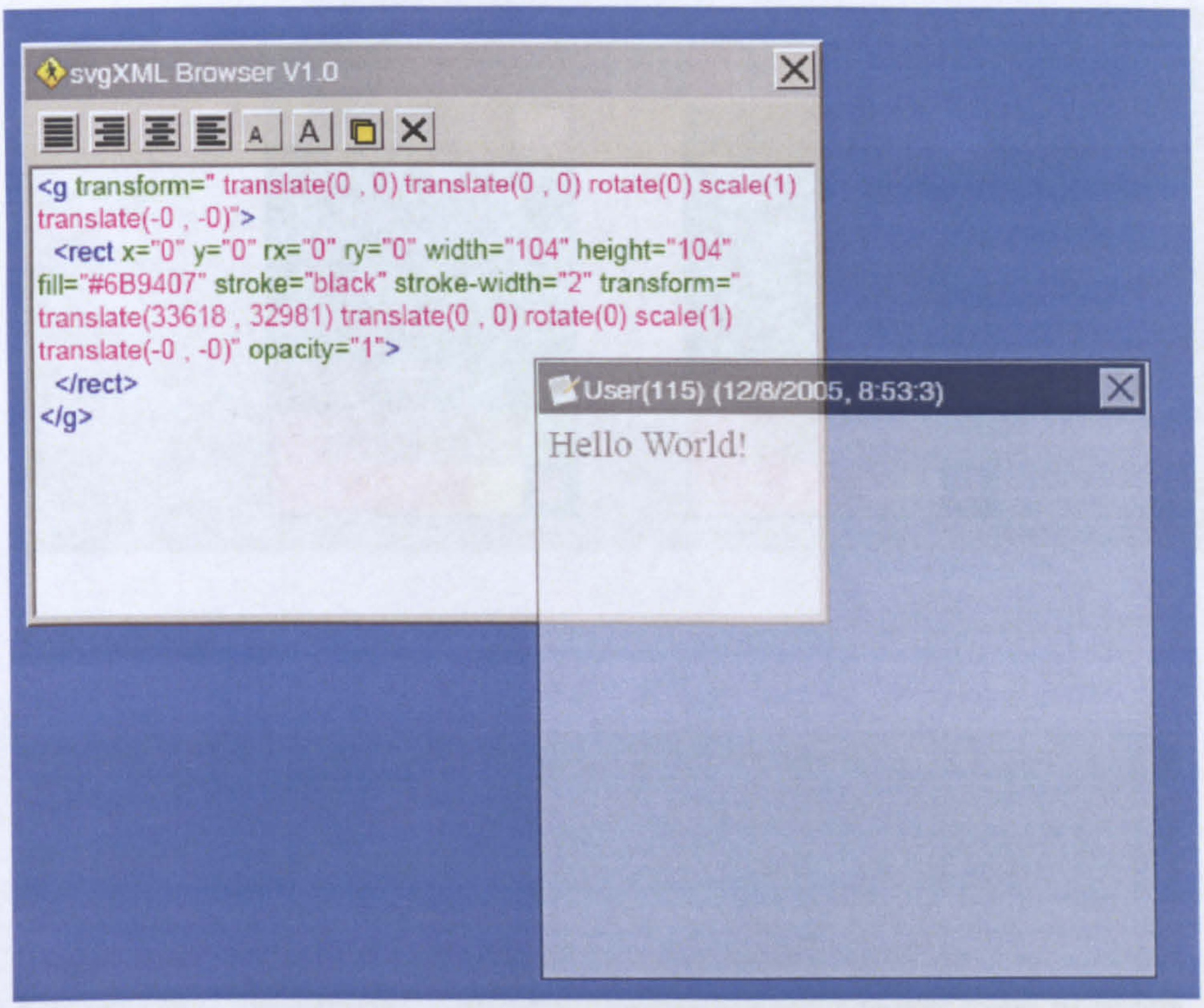
3. CheckBox

4. RadioButton

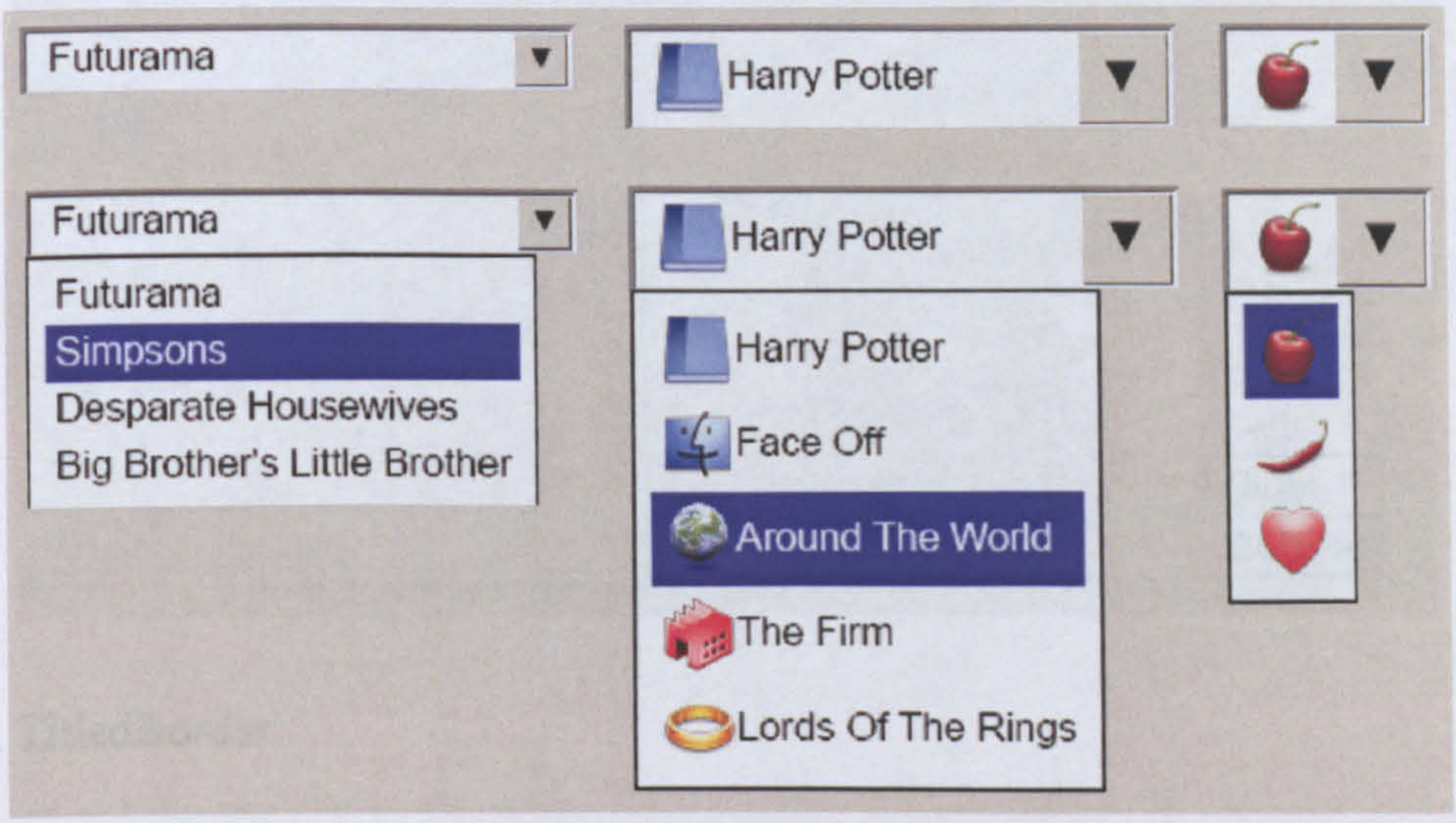
5. TextBox



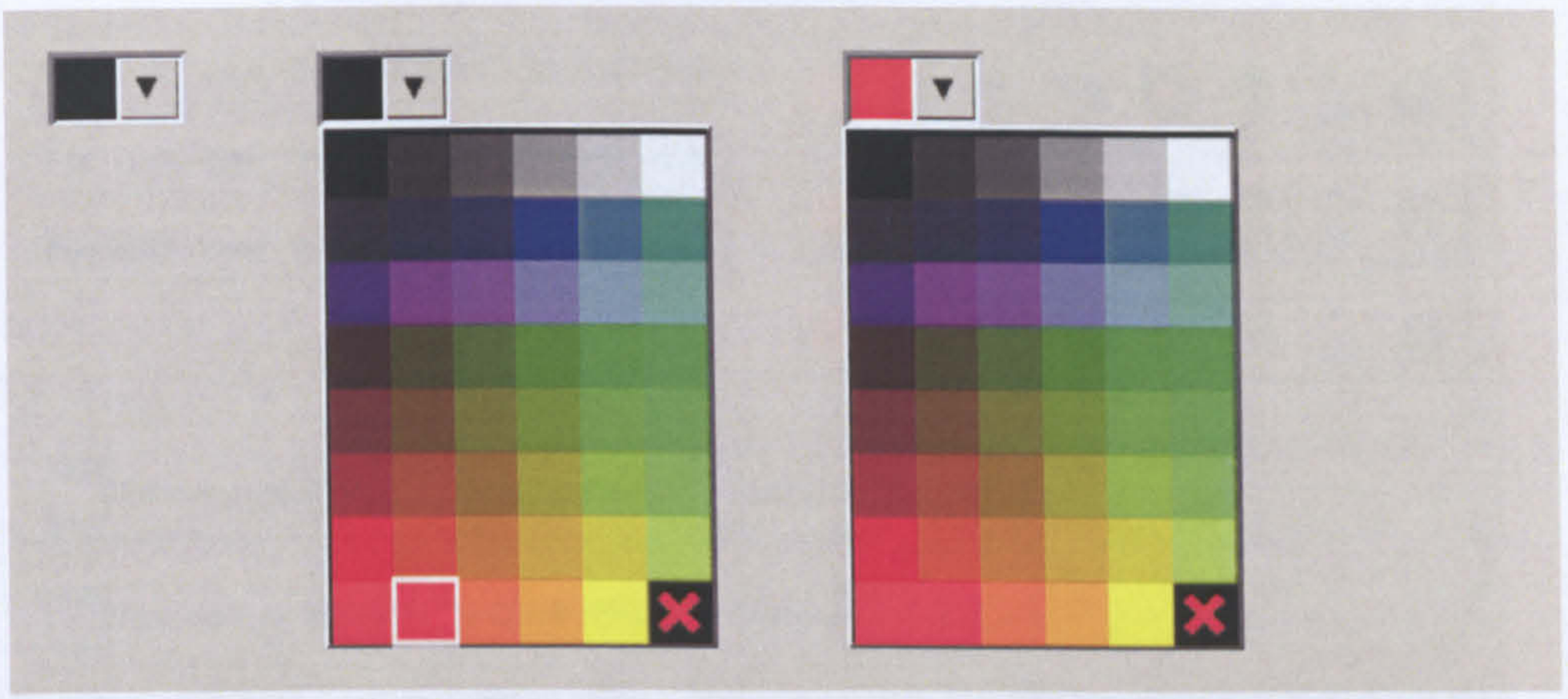
6. Windows



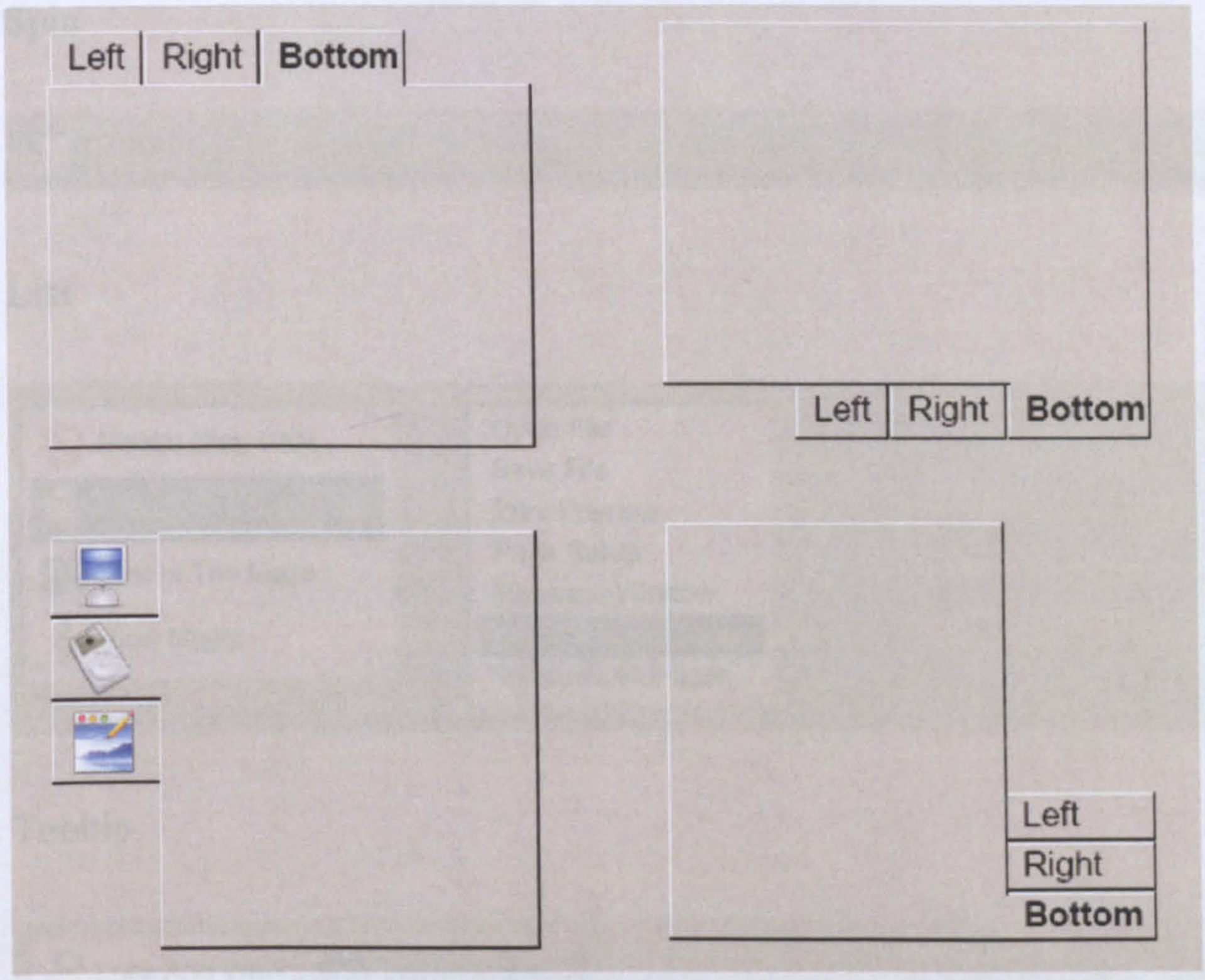
7. ComboBox



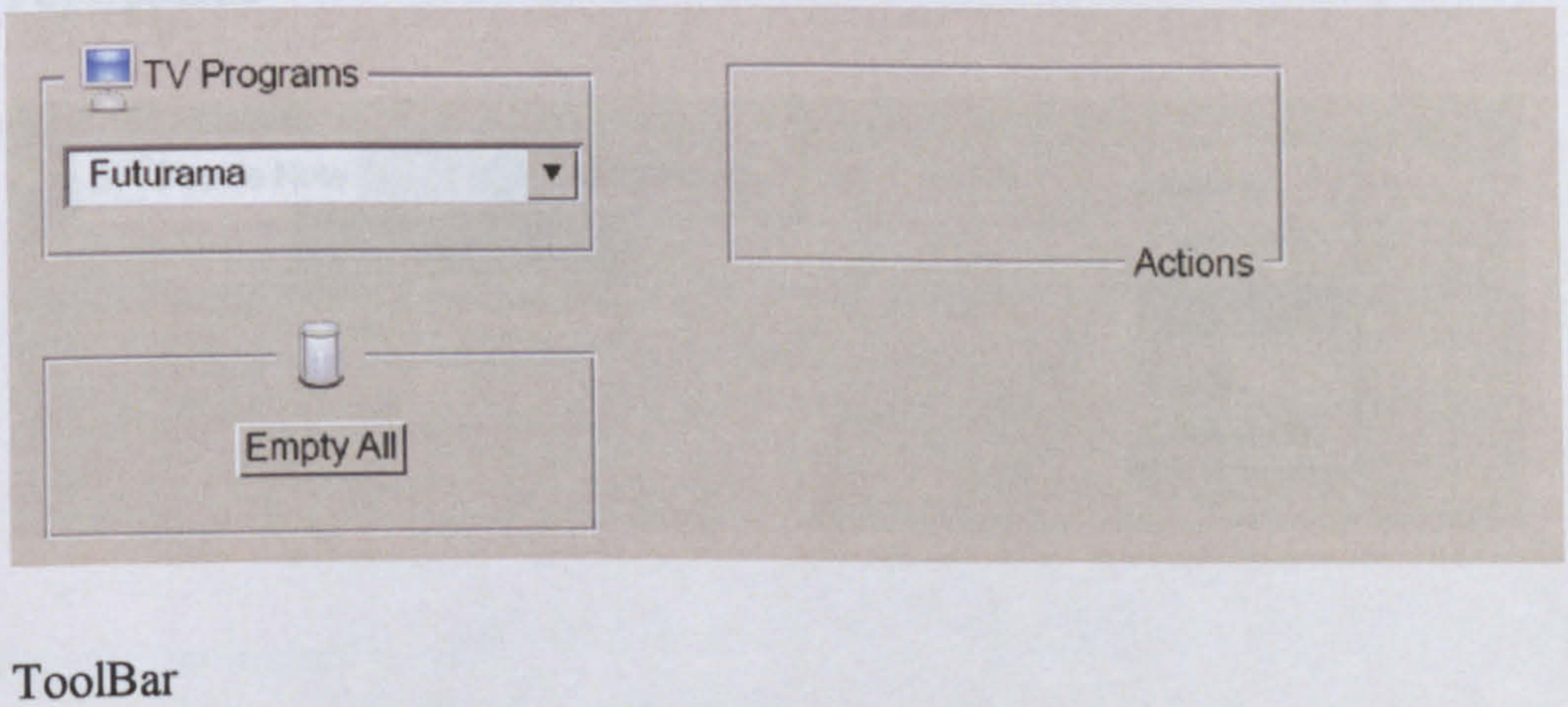
8. ColorComboBox



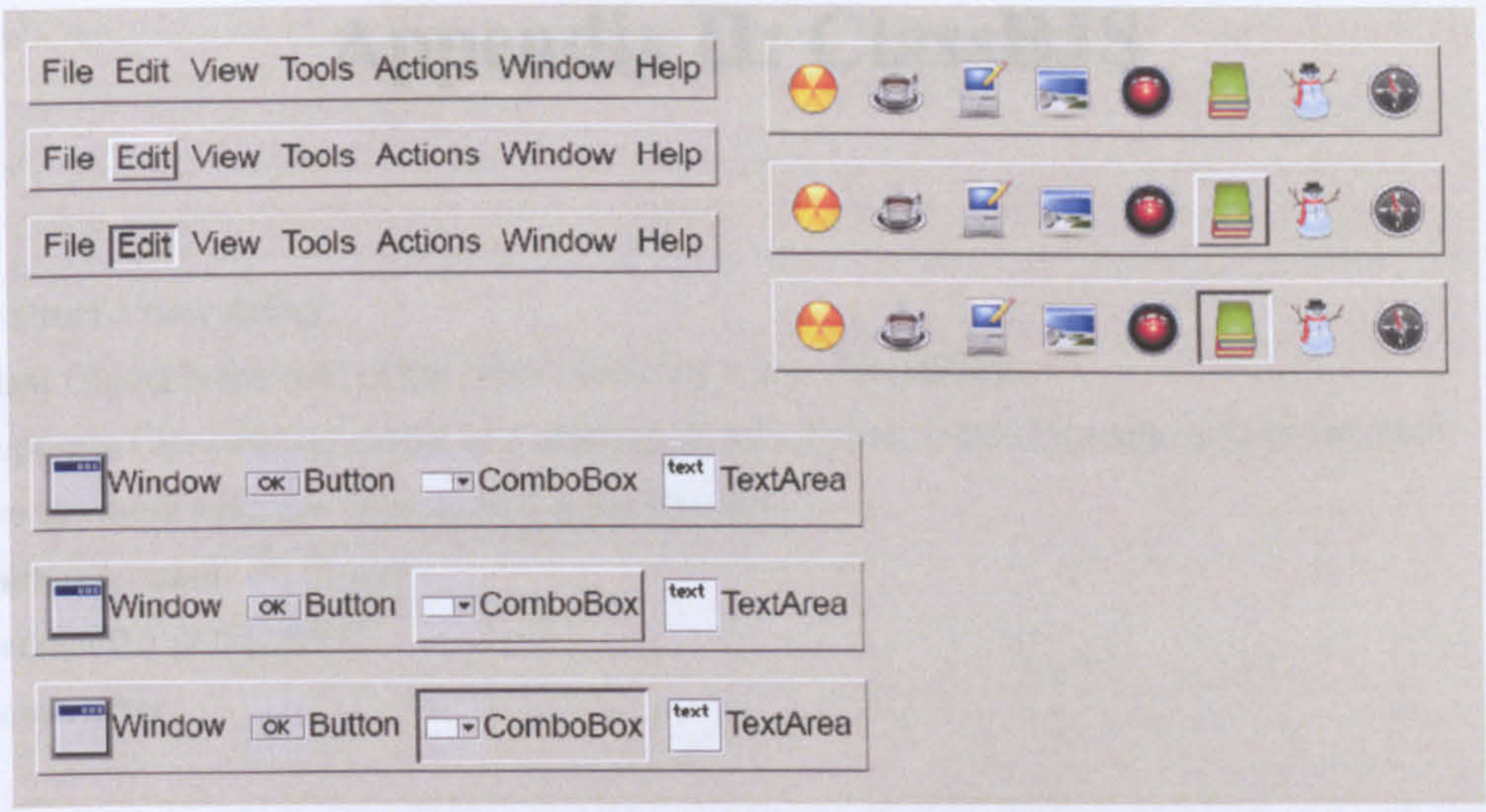
9. TabbedPane



10. TitledBorder

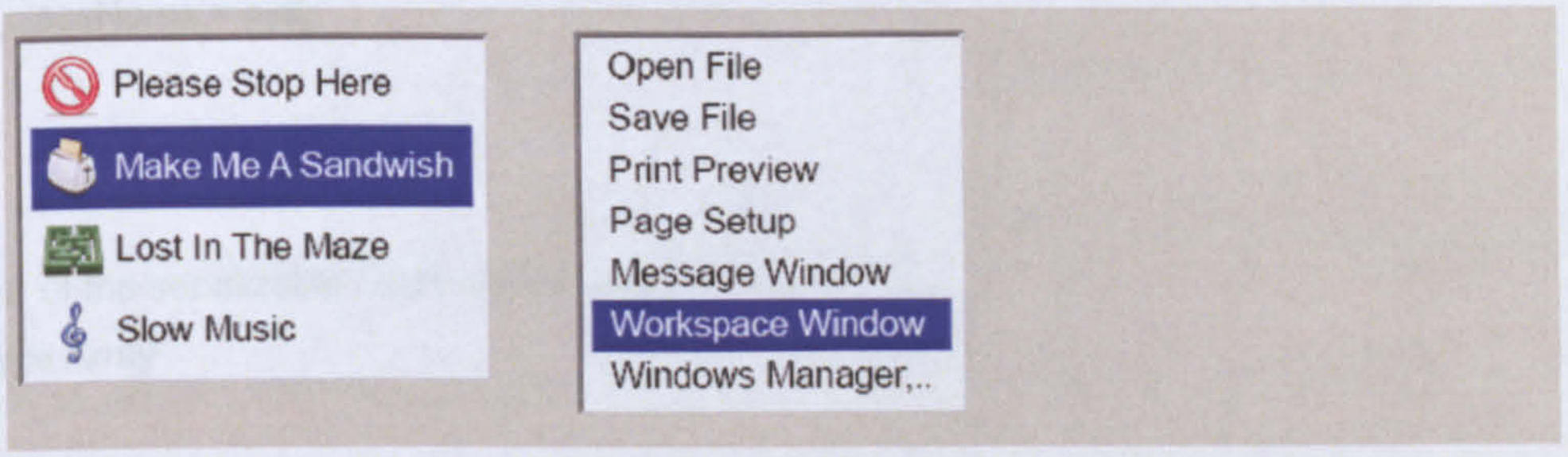


11. ToolBar

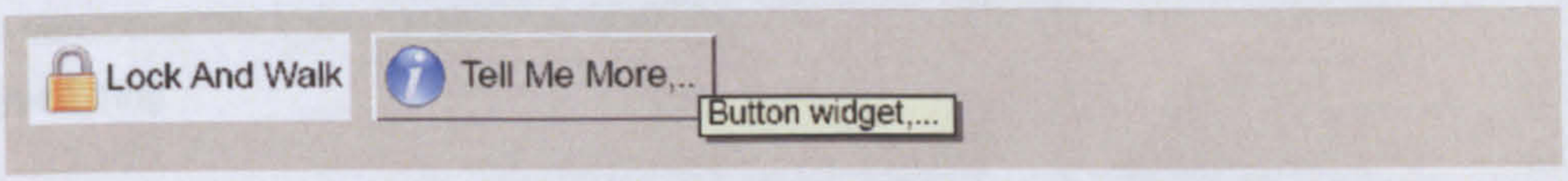


12. Spin

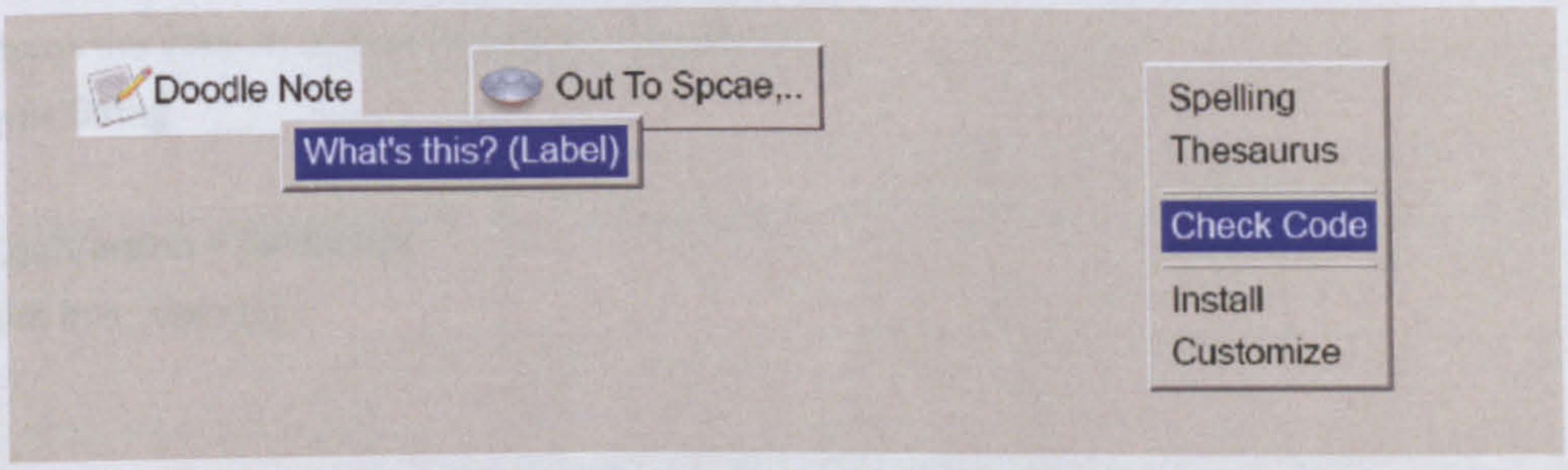
13. List



14. Tooltip



15. PopUpMenu



Appendix II: ClassBJS

```
1  /**
2  * Construct a new object.
3  * @class Object is the root of the class hierarchy in the Oea library.
4  * It supports Class-based model and enables its subclasses to be cloneable and serializable.
5  * have to inherit from the Object class (lang.Object);
6  * @author Musbah Sh. Sagar
7  * @version 0.1 [22/3/2006]
8  * @constructor
9  */
10
11 function Object(){
12
13 /**
14 * The class name of this object
15 * @type Class
16 */
17
18 this._className = null;
19
20
21 /**
22 * A list of the serializable fields of this object
23 * @type Array
24 */
25 this._serializable    = null;
26
27 /**
28 * The version of this class. Used for serialisation
29 * @type String
30 */
31 this._version    = "1.0";
32
33 /**
34 * Returns the version of this class (for serialisation)
35 * @type String
36 */
37  this.getVersion = function(){
38   return this._version;
39  }
40
41 /**
42 * Returns the version of this class
43 * @param {String} modify the version of this object
```

```
44 * @type void
45 */
46  this.setVersion = function(v){
47   if(v != undefined)
48       this._version   = v;
49  }
50
51 /**
52 * Get an instance of the Class class of this object
53 * @type Class
54 */
55  this.getClass = function(){
56   return new Class(this);
57 }
58
59 /**
60 * Obtain the class name of this object
61 * @type String
62 */
63  this.getClassName = function(){
64   return this._className;
65  }
66
67 /**
68 * Mark fields as serializable (DO NOT call this method from the class constructor; call it from the object
constructor 'this.constructor')
69 * @param {Object, Object, etc} field1, field2 etc
70 * @type void
71 */
72  this.markSerializable = function(){
73
74        if(this._serializable == null)
75          this._serializable = new Array();
76
77    for(var i=0;i<arguments.length;i++)
78          this._serializable.push(arguments[i]);
79  }
80
81 /**
82 * Saves the current state of this object to a string in XML format
83 * @type String
84 */
85  this.serialize = function(){
86          var list = this._serializable;
87      var   xml     =   "<Object   class='"+   this._class   +   "'   package='"+this._package.getName()+   "'
version='"+this._version+"'> \n";
```

```
88          xml += " <Fields> \n";
89        for(var i in list)
90          xml += "  <Field name='"+list[i]+"'>"+this[list[i]]+"</Field>\n";
91          xml += " </Fields> \n";
92          xml += "</Object>";
93        return xml;
94  }
95  /**
96  * Restores an equivalent object from a serialized string
97  * @param {String} xml is the serialized string of an instance of this class
98  * @type void
99  */
100 this.deserialize = function(xml){
101  var doc = parseXML(xml);
102
103  if( doc == undefined || doc == null) return;
104       var obj = doc.getElementsByTagName("Object").item(0);
105
106        if (obj != null){
107         var _class   = obj.getAttribute("class");
108         var _package = obj.getAttribute("package");
109         var _version = obj.getAttribute("version");
110         // stop if the class or the package or the version is different. Throw an Exception!
111          if( _class != this._class || _package != this.getPackageName() || _version != this._version) {
112     alert("Error: deserialisation process failed, serialized object does not match current object (Class
"+this._class+")");
113               return;
114               }
115        // Get filed elements
116    var fields  = obj.getElementsByTagName("Field");
117        for(var i=0;i<fields.length;i++){
118         var field = fields.item(i);
119               if(field != undefined && field != null && field.firstChild != null)
120                this[field.getAttribute("name")] = field.firstChild.nodeValue;
121               }
122        }
123 }
124
125 /**
126 * Creates a copy (clone) of this object
127 * @type Object
128 */
129 this.clone = function(){
130        // Create a new instance of the type/class of this object
131   var clone = new this._package[this._class]();
132        // Copy attributes to the clone object
```

```
133         for(var i in this)
134          if(!(this[i] instanceof Function))
135           clone[i]=this[i];
136         return clone;
137  }
138
139 /**
140 * Get an instance of the Class class of this object
141 * @type Class
142 */
143  this.getClass = function(){
144   return new Class(this);
145  }
146
147 /**
148 * Get an instance of the Class class of this object
149 * @type Class
150 */
151  this.initClass = function(obj){
152   return  Object.initClass(obj);
153  }
154
155
156 /**
157 * Convert this object to a string
158 * @type String
159 */
160  this.toString = function(){
161   return "";
162  }
163
164 }/* End of Class 'Object' */
165
166 /**
167 * Static member of Object
168 * Adds facilities to Javascript classes to enhance its OOP interaction model to Class-based.
169 * @param {Function} obj
170 * @param {String} objType
171 */
172
173
174  Object.initClass =  function(obj,objType){
175   // Get the class name
176   var callerClass = "";
177         var className  = "";
178   if(Object.initClass.caller != undefined){
```

```
179         callerClass = Object.initClass.caller;
180          if(callerClass == undefined){
181           alert("can not read the caller name (Object.js, line 181)");
182           return null;
183          }
184   var callerBody = callerClass.toString();
185   //  extract the full name of the caller (the qualified class/function name, i.e. Object etc):
186   // 1- Get the text between 'function' and '('
187   var st = callerBody.indexOf("function")+8;
188   var en = callerBody.indexOf("(");
189   // 2- Remove any whitespaces
190   var className  = callerBody.substring(st,en).replace(/^\s+/g, '').replace(/\s+$/g, '');
191
192        } else {
193         if( objType != undefined ){
194          className = objType;
195         }
196                 else   {
197           alert("JavaScript interpreter does not support 'caller' method (Object.js, line 197)");
198                 return
199                 }
200        }
201
202
203
204   obj._className = className;
205   obj.getClassName = function(){return this._className}
206   return window[className].prototype;
207 }
```

Appendix III: SVG Document for Oea Applications

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"
[
<!ATTLIST svg xmlns:a3 CDATA #IMPLIED a3:scriptImplementation CDATA #IMPLIED>
<!ATTLIST script a3:scriptImplementation CDATA #IMPLIED>
]>
<?AdobeSVGViewer save="snapshot"?>
<svg version="1.2" pointer-events="all" onload="main();" width="100%" height="100%"
    xml:space="preserve"  zoomAndPan="disable" viewBox ="0 0 640 480"
    xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:math="http://http://www.w3.org/Math"
    xmlns:xhtml="http://http://www.w3.org/XHTML"
    xmlns:a3="http://ns.adobe.com/AdobeSVGViewerExtensions/3.0/">

<!-- ************************************************************************ -->
<script type="text/ecmascript"  xlink:href="../../../Lib/Initialise.js"                    />
<!-- ************************************************************************ -->
<script type="text/ecmascript"  xlink:href="../../../Lib/sys/InitialiseSvgDraw2d.js"       />
<script type="text/ecmascript"  xlink:href="../../../Lib/sys/InitialiseSvgSwing.js"        />
<!-- ************************************************************************ -->
<script type="text/ecmascript"  xlink:href="../../../Lib/fClasses/FClasses/Graphical/Point.js"   />
<script type="text/ecmascript"  xlink:href="../../../Lib/fClasses/FClasses/Graphical/Rect.js"    />
<!-- ************************************************************************ -->
<script type="text/ecmascript"  xlink:href="../../../Lib/fClasses/Color/Palette.js"        />
<!-- ************************************************************************ -->
<script type="text/ecmascript"  xlink:href="../../../Lib/fClasses/FClasses/SVG/SvgNode.js"     />
<script type="text/ecmascript"  xlink:href="../../../Lib/fClasses/FClasses/SVG/SvgUtilities.js" />
<script type="text/ecmascript"  xlink:href="../../../Lib/fClasses/FClasses/SVG/SVGDefs.js"     />
<!-- ************************************************************************ -->
<script type="text/ecmascript"  xlink:href="../../../Lib/fClasses/Graphical/RectNode.js"       />
<!-- ************************************************************************ -->
<script type="text/ecmascript"  xlink:href="../../../Lib/fClasses/FClasses/Node/Node.js"       />
<!-- ************************************************************************ -->
<script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Layer/Layer.js"             />
<!-- ************************************************************************ -->
<script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Page/Page.js"               />
<!-- ************************************************************************ -->
```

```
40 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Desktop/Desktop.js"                  />
41 <!-- ************************************************************************** -->
42 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Font/Font.js"                        />
43 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Font/FontMetrics.js"                 />
44 <!-- ************************************************************************** -->
45 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Graphics/Graphics.js"                 />
46 <!-- ************************************************************************** -->
47 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/Shape.js"                      />
48 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/Line.js"                       />
49 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/Oval.js"                       />
50 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/Circle.js"                     />
51 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/Polygon.js"                    />
52 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/Path.js"                       />
53 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/WinBorder.js"                  />
54 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/StepBorder.js"                 />
55 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/BoxBorder.js"                  />
56 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/RRectangle.js"                 />
57 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/Rectangle.js"                  />
58 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/Text.js"                       />
59 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/TextView.js"                   />
60 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Shapes/Image.js"                      />
61 <!-- ************************************************************************** -->
62 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/Cursor/Cursor.js"                     />
63 <!-- ************************************************************************** -->
64 <script type="text/ecmascript"  xlink:href="../../../Lib/svgDraw2d/ToolTip/ToolTip.js"                   />
65 <!-- ***************************[ JAVA ]*************************************** -->
66 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/Util/Vector.js"                            />
67 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/Util/Hashtable.js"                         />
68 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/Util/EventListener.js"                     />
69 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/Util/EventObject.js"                       />
70 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/Util/Enumeration.js"                       />
71 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/Util/Enumerator.js"                        />
72 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/Util/ReverseEnumerator.js"                 />
73 <!-- ***********************[ AWT Framework ]*********************************** -->
74 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Geom/Point2D.js"                       />
75 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Geom/Point.js"                         />
76 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Geom/Dimension2D.js"                   />
77 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Geom/Dimension.js"                     />
78 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Geom/Rectangle2D.js"                   />
79 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Geom/gRectangle.js"                    />
80 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Geom/gPolygon.js"                      />
81 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Insets.js"                             />
82 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Color.js"                              />
83 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Event/MouseListener.js"                />
84 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Event/MouseMotionListener.js"          />
85 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Event/MouseEvent.js"                   />
```

```
86 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Event/KeyEvent.js"                />
87 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Event/ActionEvent.js"             />
88 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Event/KeyListener.js"             />
89 <script type="text/ecmascript"  xlink:href="../../../Lib/Java/AWT/Event/ActionListener.js"          />
90 <!-- **********************[ svgSwing Look And Feel ]*************************** -->
91 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/LookFeel/ButtonSkin.js"          />
92<script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/LookFeel/SimpleButtonSkin.js"     />
93 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/LookFeel/WinButtonSkin.js"       />
94 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/LookFeel/BoxButtonSkin.js"       />
95 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/LookFeel/ToolButtonSkin.js"      />
96 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/LookFeel/FlatButtonSkin.js"      />
97 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/LookFeel/WindowSkin.js"          />
98 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/LookFeel/DefaultWindowSkin.js" />
99 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/LookFeel/SimpleWindowSkin.js"  />
100 <!-- **********************[ svgSwing Framework ]****************************** -->
101 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/ListenerManager.js"             />
102 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/EventManager.js"                />
103 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/FlowLayout.js"                  />
104 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/BoxLayout.js"                   />
105 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/Component.js"                   />
106 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/Canvas.js"                      />
107 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/Container.js"                   />
108 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/Panel.js"                       />
109 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/Icon.js"                        />
110 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/Label.js"                       />
111 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/Button.js"                      />
112 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/CheckBox.js"                    />
113 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/RadioButton.js"                 />
114 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/ButtonGroup.js"                 />
115 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/Toolbar.js"                     />
116 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/TabbedPane.js"                  />
117 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/Pane.js"                        />
118 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/TitledBorder.js"                />
119 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/Separator.js"                   />
120 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/List.js"                        />
121 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/PopUpMenu.js"                   />
122 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/TextBox.js"                     />
123 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/ComboBox.js"                    />
124 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/Spin.js"                        />
125 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/ColorComboBox.js"               />
126 <script type="text/ecmascript"  xlink:href="../../../Lib/svgSwing/Window.js"                      />
127 <!-- ******************[ Tools ]******************************************** -->
128 <script type="text/ecmascript"  xlink:href="../../../Lib/Tools/DebugWindow.js"                    />
129 <script type="text/ecmascript"  xlink:href="../../../Lib/Tools/svgXMLBrowser.js"                  />
130 <script type="text/ecmascript"  xlink:href="../../../Lib/Tools/Launcher.js"                       />
131 <!-- ********************[ GEF ]*********************************************** -->
```

```
132 <!--                    [ Framework ]                                    -->
133 <script type="text/ecmascript"  xlink:href="../InitialiseHotDraw.js"      />
134 <script type="text/ecmascript"  xlink:href="../Framework/Figure.js"       />
135 <script type="text/ecmascript"  xlink:href="../Framework/Drawing.js"      />
136 <script type="text/ecmascript"  xlink:href="../Framework/FigureEnumeration.js"  />
137 <script type="text/ecmascript"  xlink:href="../Framework/FigureChangeListener.js"  />
138 <script type="text/ecmascript"  xlink:href="../Framework/DrawingChangeListener.js"  />
139 <script type="text/ecmascript"  xlink:href="../Framework/FigureChangeEvent.js"  />
140 <script type="text/ecmascript"  xlink:href="../Framework/DrawingChangeEvent.js"  />
141 <script type="text/ecmascript"  xlink:href="../Framework/Tool.js"         />
142 <script type="text/ecmascript"  xlink:href="../Framework/DrawingView.js"  />
143 <script type="text/ecmascript"  xlink:href="../Framework/DrawingEditor.js"  />
144 <script type="text/ecmascript"  xlink:href="../Framework/Handle.js"       />
145 <script type="text/ecmascript"  xlink:href="../Framework/Locator.js"      />
146 <!--                       [ Util ]                                      -->
147 <script type="text/ecmascript"  xlink:href="../Util/ReverseVectorEnumerator.js"  />
148 <script type="text/ecmascript"  xlink:href="../Util/PaletteButton.js"     />
149 <script type="text/ecmascript"  xlink:href="../Util/Command.js"           />
150 <script type="text/ecmascript"  xlink:href="../Util/FloatingTextField.js"  />
151 <script type="text/ecmascript"  xlink:href="../Util/Geom.js"              />
152 <!--                       [ Standard ]                                  -->
153 <script type="text/ecmascript"  xlink:href="../Standard/FigureTransferCommand.js"  />
154 <script type="text/ecmascript"  xlink:href="../Standard/DeleteCommand.js"  />
155 <script type="text/ecmascript"  xlink:href="../Standard/DuplicateCommand.js"  />
156 <script type="text/ecmascript"  xlink:href="../Figures/GroupCommand.js"   />
157 <script type="text/ecmascript"  xlink:href="../Figures/UngroupCommand.js"  />
158 <script type="text/ecmascript"  xlink:href="../Standard/FigureEnumerator.js"  />
159 <script type="text/ecmascript"  xlink:href="../Standard/ReverseFigureEnumerator.js"  />
160 <script type="text/ecmascript"  xlink:href="../Standard/FigureChangeEventMulticaster.js"  />
161 <script type="text/ecmascript"  xlink:href="../Standard/AbstractFigure.js"  />
162 <script type="text/ecmascript"  xlink:href="../Standard/CompositeFigure.js"  />
163 <script type="text/ecmascript"  xlink:href="../Standard/StandardDrawing.js"  />
164 <script type="text/ecmascript"  xlink:href="../Standard/AbstractTool.js"  />
165 <script type="text/ecmascript"  xlink:href="../Standard/SelectionTool.js"  />
166 <script type="text/ecmascript"  xlink:href="../Standard/StandardDrawingView.js"  />
167 <script type="text/ecmascript"  xlink:href="../Standard/ToolButton.js"    />
168 <script type="text/ecmascript"  xlink:href="../Standard/PaletteIcon.js"   />
169 <script type="text/ecmascript"  xlink:href="../Standard/SelectAreaTracker.js"  />
170 <script type="text/ecmascript"  xlink:href="../Standard/CreationTool.js"  />
171 <script type="text/ecmascript"  xlink:href="../Standard/ScribbleTool.js"  />
172 <script type="text/ecmascript"  xlink:href="../Standard/DragTracker.js"   />
173 <script type="text/ecmascript"  xlink:href="../Standard/AbstractHandle.js"  />
174 <script type="text/ecmascript"  xlink:href="../Standard/LocatorHandle.js"  />
175 <script type="text/ecmascript"  xlink:href="../Standard/RadiusHandle.js"  />
176 <script type="text/ecmascript"  xlink:href="../Standard/HandleTracker.js"  />
177 <script type="text/ecmascript"  xlink:href="../Standard/AbstractLocator.js"  />
```

```
178 <script type="text/ecmascript"  xlink:href="../Standard/RelativeLocator.js"                 />
179 <script type="text/ecmascript"  xlink:href="../Standard/BoxHandleKit.js"                    />
180 <script type="text/ecmascript"  xlink:href="../Standard/TextHolder.js"                      />
181 <script type="text/ecmascript"  xlink:href="../Standard/NullHandle.js"                       />
182 <script type="text/ecmascript"  xlink:href="../Standard/ActionTool.js"                       />
183 <!--                              [ Figures ]                                                -->
184 <script type="text/ecmascript"  xlink:href="../Figures/FigureAttributes.js"                  />
185 <script type="text/ecmascript"  xlink:href="../Figures/AttributeFigure.js"                   />
186 <script type="text/ecmascript"  xlink:href="../Standard/DecoratorFigure.js"                  />
187 <script type="text/ecmascript"  xlink:href="../Figures/RectangleFigure.js"                   />
188 <script type="text/ecmascript"  xlink:href="../Figures/PolyLineFigure.js"                    />
189 <script type="text/ecmascript"  xlink:href="../Figures/EllipseFigure.js"                     />
190 <script type="text/ecmascript"  xlink:href="../Figures/ImageFigure.js"                       />
191 <script type="text/ecmascript"  xlink:href="../Figures/RoundRectangleFigure.js"              />
192 <script type="text/ecmascript"  xlink:href="../Figures/TextFigure.js"                        />
193 <script type="text/ecmascript"  xlink:href="../Figures/TextTool.js"                          />
194 <script type="text/ecmascript"  xlink:href="../Figures/FontSizeHandle.js"                    />
195 <script type="text/ecmascript"  xlink:href="../Figures/PolyLineHandle.js"                    />
196 <script type="text/ecmascript"  xlink:href="../Figures/PolyLineLocator.js"                   />
197 <script type="text/ecmascript"  xlink:href="../Figures/LineFigure.js"                        />
198 <script type="text/ecmascript"  xlink:href="../Figures/BorderDecorator.js"                   />
199 <script type="text/ecmascript"  xlink:href="../Figures/BorderTool.js"                        />
200 <script type="text/ecmascript"  xlink:href="../Figures/GroupFigure.js"                       />
201 <script type="text/ecmascript"  xlink:href="../Figures/GroupHandle.js"                       />
202 <script type="text/ecmascript"  xlink:href="../Contrib/PolygonFigure.js"                     />
203 <script type="text/ecmascript"  xlink:href="../Contrib/PolygonHandle.js"                     />
204 <script type="text/ecmascript"  xlink:href="../Contrib/PolygonTool.js"                       />
205 <script type="text/ecmascript"  xlink:href="../Contrib/PolygonScaleHandle.js"                />
206 <script type="text/ecmascript"  xlink:href="../Contrib/FloatingTextArea.js"                  />
207 <script type="text/ecmascript"  xlink:href="../Contrib/TextAreaTool.js"                      />
208 <script type="text/ecmascript"  xlink:href="../Contrib/OpacityHandle.js"                     />
209 <script type="text/ecmascript"  xlink:href="../Contrib/TextAlignHandle.js"                   />
210 <script type="text/ecmascript"  xlink:href="../Contrib/TextAreaDecorator.js"                 />
211 <!--                              [ Application ]                                            -->
212 <script type="text/ecmascript"  xlink:href="../Application/DrawApplication.js"              />
213 <!--                             [ Demo Program ]                                            -->
214 <script type="text/ecmascript"  xlink:href="DemoDrawingView.js"                             />
215 <script type="text/ecmascript"  xlink:href="DemoApplication.js"                             />
216 <script type="text/ecmascript"  xlink:href="DemoAttributes.js"                               />
217 <script type="text/ecmascript"  xlink:href="HotDraw.js"                                      />
218
219 </svg>
```

PAGES NOT SCANNED AT THE REQUEST OF THE UNIVERSITY

SEE ORIGINAL COPY OF THE THESIS FOR THIS MATERIAL