

The Open Overlays Collaborative Workspace

Keywords: Collaborative working, RDF, SVG, Open Overlays

Professor Chris Cooper

Professor

[Oxford Brookes University](#)

Wheatley Campus

Oxford

UK

cscooper@brookes.ac.uk

Biography

Chris holds a degree in mathematics and physics from the University of Cambridge and a PhD from the University of London. For the past 25 years his main research interests have been in multiservice networks. From 1974 until 2001 he worked at the Rutherford Appleton Laboratory, England, before gaining a chair at Oxford Brookes University and becoming network strategist at UKERNA, the organization responsible for the UK academic network, JANET. Since 2004, Chris has been semi-retired and lives in the North West Highlands of Scotland, but continues to divide some of his time between teaching and research at Oxford Brookes and consulting for UKERNA on the continuing development of SuperJANET. He is currently a co-investigator on the UK EPSRC funded project Open Overlays, in which Oxford Brookes University and Lancaster University are collaborating on the development of dynamically reconfigurable middleware in support of service oriented distributed systems architecture applicable to next generation grid and web services.

Professor David A. Duce

Professor

[Oxford Brookes University](#)

Wheatley Campus

Oxford

UK

daduce@brookes.ac.uk

Biography

David Duce is Professor in Computing at Oxford Brookes University. He has been involved in the development of standards for computer graphics for 20 years, starting with the Graphical Kernel System (GKS). Together with Bob Hopgood and Vincent Quint, he submitted a proposal to W3C entitled Web Schematics, which launched the SVG activity. He has participated in the development of SVG 1.0 and represents Oxford Brookes University on the Advisory Committee of W3C. His research interests include web graphics and multiservice systems.

Mr Musbah Sagar

Research Assistant

[Oxford Brookes University](#)

Wheatley Campus

Oxford

UK

msagar@brookes.ac.uk*Biography*

Musbah Sagar is a Research Assistant at Oxford Brookes University. He has a BSc in Computer Engineering and an MSc in Web Technology from Oxford Brookes University, 2002. His work on his MSc dissertation, "An SVG Browser for XML Languages", was published in the Eurographics UK Chapter Conference, Birmingham, June 2003. He worked as a Web Programmer for Danfoss UK and then as a Research Assistant for Oxford Brookes University on a project called gViz; part of the e-science program. Currently he is working on an EPSRC funded project, Open Overlays, which involves Grid Computing and Web Technologies. He is particularly interested in Computer Graphics, Games Programming and Web Technologies.

Professor Gordon Blair

Professor

[Lancaster University](#)

Computing Department, InfoLab 21

Lancaster

UK

<mailto:gordon@comp.lancs.ac.uk>**Professor Geoff Coulson**

Professor

[Lancaster University](#)

Computing Department, InfoLab 21

Lancaster

UK

<mailto:geoff@comp.lancs.ac.uk>**Dr Paul Grace**

Research Assistant

[Lancaster University](#)

Computing Department, InfoLab 21

Lancaster

UK

<mailto:p.grace@comp.lancs.ac.uk>*Biography*

Paul is a research associate on the Open Overlays Project at Lancaster University developing next generation Grid middleware. He completed his PhD at Lancaster in

2004; this examined reflective middleware support for mobile applications.

Abstract

This paper describes a collaborative workspace tool being developed using SVG and RDF. The approach being taken is to regard 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. The approach and the current state of implementation work are described.

Table of Contents

1. Introduction

2. Application Scenario

3. The Collaborative Workspace Tool, svgCWE

3.1 Functionality

3.2 Architecture

3.3 RDF model

4. Implementation

5. Related Work

6. Conclusions and Future Work

Acknowledgements

Bibliography

1. Introduction

This paper describes a collaborative workspace tool being developed at Oxford Brookes University as part of the Open Overlays project [Grace et al, 2004]. The context of the project arises from the observation that next-generation Grid applications will operate within and across many heterogeneous network types; will employ a wide range of device types ranging from supercomputers to sensor motes; and will require many more interaction paradigms than just RPC and message-passing (e.g. publish-subscribe, multicast, tuple spaces etc.). The Open Overlays project is seeking to provide a Grid middleware infrastructure, Gridkit, that can span and integrate this growing diversity at both the infrastructure and the interaction paradigm levels.

Application scenarios are being used to motivate and evaluate the middleware development, in particular scenarios based on distributed and collaborative visualization. This choice is motivated by experience gained in earlier projects and the observation that team activity is important in many endeavours and gives rise to heterogeneous infrastructure platforms and their associated problems and issues. Earlier work in this field is described in [\[Brodlić et al \(2004a\)\]](#)[\[Brodlić et al \(2004b\)\]](#).

The first scenario being studied is based on wildfire management and one of the key components of this is graphical communication between groups of fire fighters and controllers who coordinate the work of the fire fighters. A key component of the scenario is a collaborative workspace tool, svgCWE, to support collaborative group working. This is the primary focus of this paper. The tool is constructed using SVG for presentation and RDF for describing the content of the workspace. This approach draws on ideas in W3C's Annotea project [\[Kahan et al \(2001\)\]](#), but develops them further by regarding all information in the workspace as an annotation of the workspace. This approach was taken in order to explore the flexibility and extensibility of RDF [\[RDF\]](#) for describing the information in the workspace and for querying this information to generate different views of the workspace for different actors. The Annotea approach was developed in an asynchronous context; everyone should be able to consult the annotations associated with a document and add their own. svgCWE is primarily a synchronous tool in that all members of the group should see content of the workspace as it is created and modified.

An important aspect of the Open Overlays middleware is support for configuration and run-time adaptation. svgCWE makes few assumptions about the structure of the RDF store and hence a variety of storage strategies based on approaches such as replication and distributed hash tables, can be utilised.

The remainder of the paper is structured as follows. [Chapter 2](#) describes the application scenario in more detail. The design of svgCWE is discussed in [Chapter 3](#) and the current implementation in [Chapter 4](#). [Chapter 5](#) discusses related work and [Chapter 6](#) concludes the paper and outlines some future work.

2. Application Scenario

The wildfire scenario was chosen because it exhibits a number of requirements that the Open Overlays project is addressing, in particular a wide range of interaction paradigms in addition to highly heterogeneous device and networking technologies.

The scenario is based on wildfire management in a remote region with poor accessibility. Fire fighters have very limited means at their disposal; aerial attack is not possible and so the main instruments for fire-fighting are hand beaters and fire breaks (often these are pre-cut, but become overgrown and require clearing). In similar real life situations, the fire fighter has little idea where the fire boundary is, there is no means for communication between different groups of fire fighters and there is no technological support.

For the Open Overlays scenario a number of advances are posited. Fire fighters carry PDA-like devices that enable communication with other fire fighters and with controllers whose role is to coordinate the work of the fire fighters. These devices support: screens on which information and commands are displayed; cameras to give the controllers a view of the fire; GPS to enable location tracking; and audio capabilities to enable group communication among the fire fighters and controllers. It is assumed that groups are dynamic, in other words individual fire fighters might move from group to group during an incident. In real life, wind velocity (speed and direction) is a major factor in determining the rate and direction in which a fire spreads. Wind velocity is rarely constant across the site of a fire and so availability of reliable current and forecast wind velocity at key locations around the site would be of major benefit to controllers. The scenario posits the availability of portable environmental sensors to provide controllers with information such as wind speed and direction. The sensors are placed manually by fire fighters and are networked wirelessly.

The scenario also assumes the availability of simulations to predict the evolution of the fire. There are a number of fire growth simulators available, for example Farsite [FARSITE]. Such fire simulators are resource-intensive applications. There are a number of different models of fire spread in use. Generally the terrain is modelled as a grid of cells and the model computes the spread of the fire from cell to cell, dependent on the type of vegetation, physical characteristics of the vegetation such as moisture content, and meteorological conditions such as wind speed and direction and rainfall. Fire simulation has been used as an exemplar in self-organising systems research (autonomic computing), for example [Hariri (2004)].

In the application scenario, we assume a fire spread simulator that is driven by input from the environmental sensors and forecast data that runs in a fixed-infrastructure Grid. Local controllers can call upon remotely-located experts to advise controllers on the evolution of the fire. Such experts would themselves be video-conferenced with each other and the controllers. Visualization plays an important role in such applications. Also visualization is often the primary way in which the progress of a simulation is monitored. Providing a Web Services interface between the simulation system and the visualization system offers a convenient way to monitor and steer a simulation, enabling a user to "log-in" through the visualization system to a back-end simulation, view its current state and perhaps make changes to the control parameters. This approach has been explored by Brodlie et al in the gViz library [Brodlie et al (2004a)].

As well as involving heterogeneous device and networking technologies, the scenario also requires a wide range of interaction paradigms, including reliable ad hoc multicast for command propagation, stream-based multicast for group audio communication, publish-subscribe for sensor data collection and SOAP-based messaging for communication with objects in the fixed Grid. These characteristics are ideal for exercising the middleware ideas being developed in the Open Overlays project.

Figure 1 below is a mock-up showing the kind of information that might be presented to a controller. The display shows positions of controllers, field, workers and sensors.

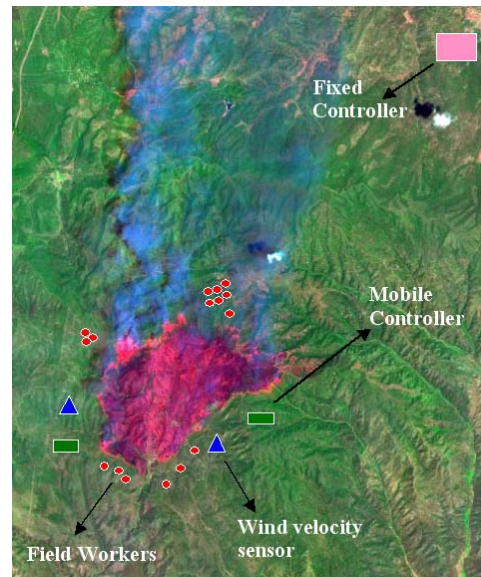


Figure 1: Mock-up of a controller's display

From the perspective of this paper, a key requirement of the scenario is the ability to share such visual presentations between the actors in the scenario: controllers and field workers.

3. The Collaborative Workspace Tool, svgCWE

3.1 Functionality

The collaborative workspace tool provides the basis for graphical communication between a group of users, for example a set of fieldworkers, or a controller and a set of fieldworkers. The aim is to be able to present map information and to overlay this with visualizations of sensor information, including positions of actors. Users should be able to sketch on the drawing surface, for example in order to give an indication of the local fire boundary, or to highlight particular features on the map.

Each user runs their own copy of svgCWE as explained in [Section 3.2](#). A snapshot of svgCWE in operation is shown below in [Figure 2](#).

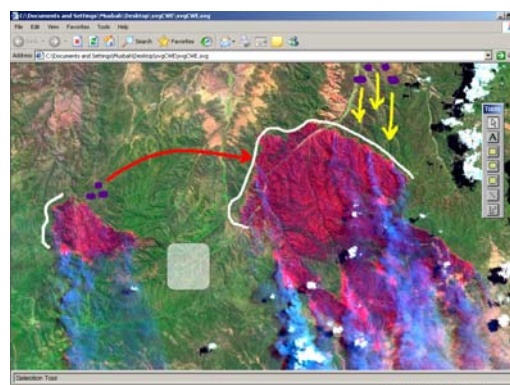


Figure 2: svgCWE snapshot

Figure 2 shows a background map displayed in the workspace with sketch annotations added by different users.

The initial design for svgCWE assumes that a single workspace is associated with a single group of users. The composition of the group may change over time; users may join and leave a group. Within a workspace, information is structured as a set of contexts. A context consists of background information provided by applications (for example a map of the hazard region showing vegetation type, road network, population centres; or the output from a simulator) and sketch annotation created by the users.

The functionality of svgCWE is split into two parts, import tools and drawing tools. The import tools introduce the background information and the drawing tools the sketch annotation. Initially the import tools just deal with static background information (images and SVG documents) rather than output dynamically updated by, say, a Web Service.

The tool palette, seen on the right hand side of **Figure 2**, provides a collection of drawing tools text, simple geometric shapes and freehand curves. Shapes can be selected, moved, resized, deleted, etc. through mouse and keyboard interaction. Attributes can be controlled through an attribute palette as shown in **Figure 3**. The Up and Down buttons change the order in which elements are displayed. The button "Get RDF" is used for debugging purposes.

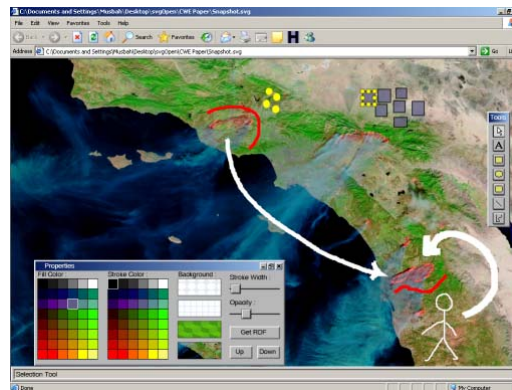


Figure 3: Attribute palette

svgCWE also provides replay functionality so that users can view the evolution of the current state of the workspace. One application of this in the wildfire scenario is for post incident analysis.

3.2 Architecture

Before we describe the architecture of svgCWE we describe the overall architectural framework within which it is being constructed. This is shown in **Figure 4**.

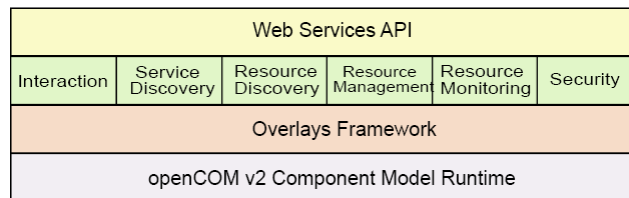


Figure 4: The Gridkit architecture

The Gridkit architecture is built from a component model called OpenCOM v2, which uses a minimal runtime environment supporting loading and bindings of lightweight software components at runtime. OpenCOM v2 is sufficiently lightweight that it can be supported even by very primitive devices. The layers above OpenCOM v2 are constructed using OpenCOM v2. The overlays framework supports the deployment of overlay networks. Without going into details, an overlay network is a virtual communications structure that is "laid over" one or more underlying physical networks (for example the Internet and/or a wireless ad hoc networking environment). The benefits of this approach are that it can mask the heterogeneity of the underlying networked infrastructure by separating engineering implementation from high-level functionality, it can provide specific network services (such as multicast) in network environments where they do not have native support, and it is inherently configurable and can be adapted at runtime.

The vertical frameworks above the overlays framework provide functionality in orthogonal areas. These frameworks are included as necessary in order to provide the services required in specific situations. The Web services API layer above the vertical frameworks provides a familiar programming interface for Grid applications.

The interaction framework provides an extensible framework for so-called plug-in interaction paradigms (PIPs). Again without going into details, this approach allows different interaction types to re-use overlay configurations already available, for example a topic-based publish-subscribe PIP and a reliable multicast PIP might both share a multicast tree overlay. Another part of the project is looking at the resource discovery, management and monitoring frameworks, which feed into dynamic control of overlay configuration.

In order to support the collaborative workspace tool we have designed a group abstraction on top of the interaction types framework. The group abstraction provides a group management interface with operations to create, delete, and monitor groups. A group interface provides operations to join and leave a group and to send data to the members of a group. The latter operation uses an interaction types interface.

An overview of the architecture of svgCWE is shown in [Figure 5](#). svgCWE is based on HotDraw [[HotDraw](#)][[Johnson \(1992\)](#)], a framework for structured drawing editors originally developed in SmallTalk, but later ported to Java [[Java HotDraw](#)]. A subset of the code has been ported to JavaScript to provide the foundation for svgCWE. The HotDraw code will run in either a Web browser with SVG support or in a Java component using the Batik SVG engine. Instances of svgCWE communicate with each other and an RDF repository using a group abstraction interface over Gridkit.

The RDF repository records events in the collaborative workspace for a number of functions including:

1. a source from which members of a group can obtain the current state of the group's workspace, e.g. when changing context, or when joining a group;
2. a source from which events in the workspace can be replayed at a later time. One use of this in the application scenario is for post incident analysis.;
3. a source from which group members can selectively display items in the workspace, e.g. items created by a particular member of the group.

To exercise the Open Overlays framework, the RDF repository can be either a centralised repository, replicated, or distributed; the latter making use of a distributed hash table overlay (DHT). RDFPeers [Cai and Frank (2004)] is an example of a distributed RDF repository that is based on an extension of the DHT approach.

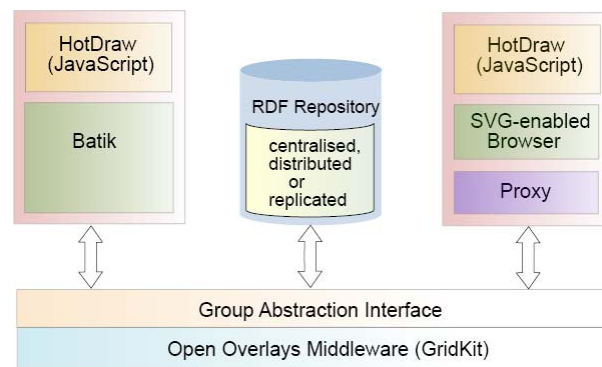


Figure 5: svgCWE architecture overview

The more detailed internal architecture of svgCWE is described in [Chapter 4](#).

The RDF structures used are discussed in the next section.

3.3 RDF model

W3C's Annotea system [Kahan et al (2001)] introduced the use of RDF to model annotations of documents as a class of metadata. Annotations were viewed as statements made by an author about a document and could be stored externally to the document in one or more annotation servers. Users could query the annotation server to retrieve, modify or delete existing annotations and add a new annotation.

The simplest form of RDF is the triple model. A triple asserts that a resource (the subject) has a given property (the predicate) with a given value (the object). The resource can be any Web resource identified by a URI and the value may be a literal string (which may include XML markup) or may be the URI of another Web resource. The property names are also Web resources.

The approach we have followed in the design of svgCWE considers the workspace of a

collaborative group to be a Web resource. We think of each member of the group having a view of this workspace. Each member of the group can add display information in their own view which will be transmitted to the views of the other members of the group. Background information added to the workspace is also transmitted to the views of all the members of the group.

RDF is used to describe the content of the workspace and the history of the workspace. This is quite an abstract way of thinking about a collaborative workspace. We considered a more concrete representation in which the document displayed in each instance of svgCWE is considered as a resource, and RDF is used to describe each node in that document; but this was rejected for reasons of simplicity and flexibility, especially in replay. The approach we have taken forces a more holistic view of the workspace and to an extent discourages the existence of private information within the shared workspace, though it would be straightforward to extend the approach to allow this by introducing metadata to describe ownership and access privileges.

The RDF structure we are using at the time of writing this paper is shown below in [Figure 6](#). The graph visualization was created using W3C's IsaViz visual authoring tool for RDF.



Figure 6: Basic RDF class structure

The figure shows the RDF class structure. The workspace (class Workspace) is the key resource. Assertions are then made about the contexts (class Context) associated with a workspace, and the contents of each context. Graphical fragments (class GraphicalFragment) form the content of a context, and these in turn are described by snapshots of their history (class HistoryNode). Assertions are also made about the group (class Group) of people (class Person) associated with a workspace.

[Figure 7](#) shows the RDF Schema definitions of the properties that are used. The properties are of type `rdfs:Property` and restrictions are placed on the domains and ranges of each; `rdfs:domain` and `rdfs:range` specify the class of the resources that may appear as subjects and objects respectively of the property to which they refer. The properties and classes are explained further in the discussion of the example workspace description shown in [Figure 8](#).

RDF Collection, in which the sequence is ordered in the unique order in which actions are serialized. In a sense the timestamp approach moves the problem elsewhere, i.e. establishing and maintaining global time and deciding how to treat two snapshots timestamped with the same global time, but it offers some flexibility for updating distributed repositories.

We use assertions from the Dublin Core vocabulary to describe when a change takes place. Dublin Core provides a date property that characterises a point in the lifecycle of its subject. svgCWE requires a more specific meaning of date than this, hence the `svgcwe:timestamp` property is a subclass of `dc:date` (this is not shown in [Figure 7](#)). The format shown for timestamp values is the XSchema basic type `dateTime`.

The geometrical content of snapshots of graphical fragments is described using the RDF geometry vocabulary, `RDFGeom` [[Goad \(2004\)](#)]. This defines a set of RDF classes and properties for geometry and is based in part on SVG. Styling properties (`fillColor` is shown for illustration here and in [Figure 7](#)) are described using SVG property names in the `svgcwe` vocabulary. An RDF/XML representation for one of the history nodes in [Figure 8](#) is shown below.

```
<svgcwe:HistoryNode>
  <svgcwe:svgFragment rdf:parseType="Literal" xmlns:svgcwe="...">
    ... </svgcwe:svgFragment>
  <svgcwe:fillColor>turquoise</svgcwe:fillColor>
  <geom2d:x>25</geom2d:x>
  <geom2d:width>300</geom2d:width>
  <svgcwe:timestamp>2005-01-28T20:00:00Z</svgcwe:timestamp>
  <svgcwe:history-of rdf:nodeID="A0"/>
  <dc:creator rdf:resource="http://svgcwe/person/1"/>
  <geom2d:y>50</geom2d:y>
  <geom2d:height>400</geom2d:height>
</svgcwe:HistoryNode>
```

The fact that the `HistoryNode` in this example represents a box is an assertion about the object of the "history-of" property that these assertions are associated with. In [Figure 8](#) this is the blank node with type `geom2d:Box`. There is an implicit constraint that the type of a graphical fragment cannot be changed, e.g. a box cannot change into a circle.

Assertions about the geometrical properties of the graphical fragment such as the x and y coordinates and width and height of the box, and appearance properties such as the fill colour of the box, effectively "lift" attribute values from the SVG level to the RDF level. The reason for doing this is to enable us to reason about the fragment within RDF. An example is given later in this section. An XML SVG fragment that describes the graphical fragment is also made available as the value of an assertion about the fragment; this avoids having to reconstruct the fragment from the RDF property values and also means that the properties of a graphical fragment that are exposed as RDF properties can be restricted.

The RDF model is limited in the kinds of queries that can be applied because the geometric properties of a primitive depend upon its type, for example, paths and boxes have different geometric properties. This can make it difficult to retrieve all geometric information in one query. The SPARQL query language [[Prud'hommeaux and Seaborne \(2004\)](#)], under development within W3C, provides additional functionality (optional matching) that appears to offer a solution.

The upper part of [Figure 8](#) shows assertions made about the members of the workspace. Members are regarded as resources. For illustration these are identified by URIs such as <http://svgcwe/person/1>. Assertions are then made about the name of the person, their role in the scenario, etc. The name property is taken from the friend of a friend (FOAF) vocabulary [\[FOAF\]](#). FOAF also provides vocabulary for describing groups. This has not been used as yet within svgCWE, but is an area for further study. At the time of writing this was also an area under development in FOAF itself. There are also assertions linking each history node object and the person who created it.

This RDF model is not finalized. It is being revised in the light of experience and further exploration of functionality for the svgCWE workspace.

We have performed some initial experiments to explore the flexibility of the model, using the query language RDQL (as implemented in the Jena toolkit [\[Jena\]](#)). RDQL queries are expressed as graph patterns. The result of a query is the set of tuples that match the pattern. For example, a query to find the creation date, type, and geometric attributes of all the history nodes created by Fred Smith which make assertions about the workspace identified by the URI <http://svgcwe/ws/1> could be written as follows.

```
SELECT ?date, ?t, ?x, ?y, ?w, ?h
WHERE (?c svgcwe:context-of <http://svgcwe/ws/1>)
      (?geom svgcwe:content-of ?c)
      (?geom rdf:type ?t)
      (?g svgcwe:history-of ?geom)
      (?g geom2d:x ?x)
      (?g geom2d:y ?y)
      (?g geom2d:width ?w)
      (?g geom2d:height ?h)
      (?g dc:creator ?person)
      (?g svgcwe:timestamp ?date)
      (?person foaf:name "Fred Smith")
USING svgcwe FOR <http://svgcwe/vocabulary#>,
      geom2d FOR <http://fabl.net/vocabularies/geometry2d/1.1/>
      rdf FOR <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
      dc FOR <http://purl.org/dc/elements/1.1/>
      foaf FOR <http://xmlns.com/foaf/0.1/>
```

The SELECT statement identifies the variables to return. The WHERE clause describes the graph patterns to match. Variable names are prefixed by "?". Where the same variable name is used in different patterns, the value of the variable must be the same in each triple pattern for a successful match. The result of this query applied to the graph in [Figure 6](#) is the single tuple:

Tuple

date	t	x	y	w	h
2005-01-28T20:00:00Z	http://fabl.net/vocabularies/geometry2d/1.1/Box	25	50	300	400

Table 1

To illustrate the extensibility of the approach, and an additional type of query, we could easily

add assertions about the geographical location of each member of the group, then we could search for history nodes created by members located within a particular region. For example, the query:

```
SELECT ?xloc ?yloc ?date ?t, ?x, ?y, ?w, ?h
WHERE (?c svgcwe:context-of <http://svgcwe/ws/1>)
      (?geom svgcwe:content-of ?c)
      (?geom rdf:type ?t)
      (?g svgcwe:history-of ?geom)
      (?g geom2d:x ?x)
      (?g geom2d:y ?y)
      (?g geom2d:width ?w)
      (?g geom2d:height ?h)
      (?g svgcwe:timestamp ?date)
      (?g dc:creator ?p)
      (?p svgcwe:location-x ?xloc)
      (?p svgcwe:location-y ?yloc)
AND
      ?xloc >= 200 &&
      ?xloc <= 300 &&
      ?yloc >= 200 &&
      ?yloc <= 400
USING svgcwe FOR <http://svgcwe/vocabulary#>,
      geom2d FOR <http://fabl.net/vocabularies/geometry2d/1.1/>
      rdf FOR <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
      dc FOR <http://purl.org/dc/elements/1.1/>
      foaf FOR <http://xmlns.com/foaf/0.1/>
```

would search for history nodes created by anyone located in the region 200 to 300 in x and 200 to 400 in y. The "AND" clause in the query filters tuples based on the values of the location-x and location-y properties. Of course if members are allowed to change location, a more complex RDF representation than this simple extension is required, but the example suffices to indicate a degree of flexibility and extensibility.

The previous example illustrated a query at what we might term the application level, in that case to select nodes based on application level location information. Another query at this level would be to display all uncleared firebreaks. Typically a firebreak will have a particular kind of graphical representation, so the query could be translated to a graphical query if there is some distinguishing characteristic of the graphical representation of firebreaks that is not shared by representations of any other kind of application entity. Suppose that firebreaks and only firebreaks have colour lime. The following RDQL will extract all graphical fragments that are coloured lime.

```
SELECT ?xloc ?yloc ?date ?t, ?x, ?y, ?w, ?h
WHERE (?c svgcwe:context-of <http://svgcwe/ws/1>)
      (?geom svgcwe:content-of ?c)
      (?geom rdf:type ?t)
      (?g svgcwe:history-of ?geom)
      (?g geom2d:x ?x)
      (?g geom2d:y ?y)
      (?g geom2d:width ?w)
      (?g geom2d:height ?h)
      (?g svgcwe:timestamp ?date)
      (?g svgcwe:fillColor "lime")
USING svgcwe FOR <http://svgcwe/vocabulary#>,
      geom2d FOR <http://fabl.net/vocabularies/geometry2d/1.1/>
      rdf FOR <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
      dc FOR <http://purl.org/dc/elements/1.1/>
```

```
foaf FOR <http://xmlns.com/foaf/0.1/>
```

This is not however a very robust or extensible approach and relies on knowledge that is not represented in RDF, in this case the assertion that only firebreaks have colour lime. A more interesting approach is to model the scenario at the application level by one set of assertions, at the graphical presentation level by another set of assertions, with a third set of assertions connecting the two levels. RDFGeom [Goad (2004)] shows one example of this kind. A simple example from the wildfire scenario which represents a named firebreak and a link to the corresponding geometry is shown below.

```
<rdf:Description rdf:about="http://svgcwe/scen1/break1">
  <cwewf:name>fire break 1</cwewf:name>
  <cwewf:representation rdf:nodeID="A0"/>
</rdf:Description>
```

This asserts that the name of the firebreak is "fire break 1" and the graphical representation is the node with identifier A0, in this case the box node shown in [Figure 8](#). The query:

```
SELECT ?fb ?date ?t, ?x, ?y, ?w, ?h
WHERE (?c svgcwe:context-of <http://svgcwe/ws/1>)
      (?geom svgcwe:content-of ?c)
      (?fb cwewf:name "fire break 1")
      (?fb cwewf:representation ?geom)
      (?geom rdf:type ?t)
      (?g svgcwe:history-of ?geom)
      (?g geom2d:x ?x)
      (?g geom2d:y ?y)
      (?g geom2d:width ?w)
      (?g geom2d:height ?h)
      (?g svgcwe:timestamp ?date)
USING svgcwe FOR <http://svgcwe/vocabulary#>,
      geom2d FOR <http://fabl.net/vocabularies/geometry2d/1.1/>
      rdf FOR <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
      dc FOR <http://purl.org/dc/elements/1.1/>
      foaf FOR <http://xmlns.com/foaf/0.1/>
      cwewf FOR <http://svgcwe/wildfire/vocabulary#>
```

will retrieve the history of the graphical representation of this firebreak. This is a very simple example but it illustrates the general idea.

4. Implementation

The internal architecture of svgCWE is illustrated in [Figure 9](#).

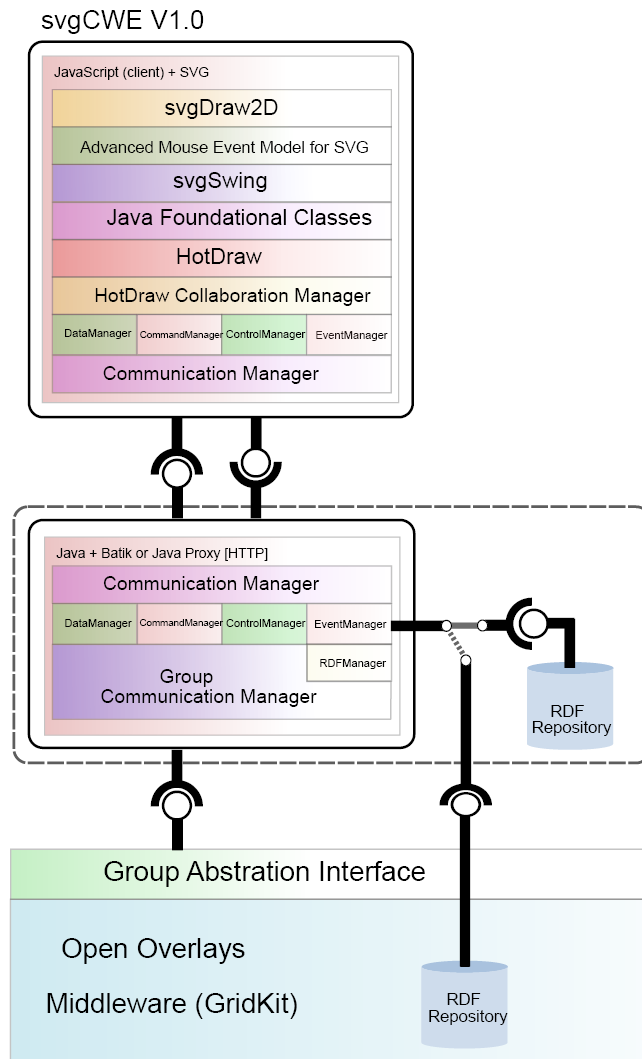


Figure 9: svgCWE internal architecture

The implementation is split into an svgCWE client component which is written in JavaScript and may run either in a SVG-enabled web browser or in Batik in a Java environment and a communication component. The client is built using HotDraw which we have ported to JavaScript. To facilitate this an extensive set of JavaScript libraries has been created, which implement appropriate parts of the Java Foundation Classes and JavaSwing classes to provide GUI support. These libraries have evolved over a period of time and have been used in other projects also. The input model was completely redesigned when implementing svgCWE and is described in a separate paper [Sagar et al 2005].

Communication between Java and JavaScript is achieved through a communication manager class which has been implemented in both Java and JavaScript; both implement a Transport interface which supports two methods, send and receive. The Java side is within the communication component. When the svgCWE is running in a web browser, this communication is mediated through a Web server proxy.

It is convenient to structure messages into different classes for command, event, data and control messages. The command class is an application-level concept and is specific to the wildfire scenario. Commands are issued by controllers to (groups of) fire-fighters. Messages in

this category require particular delivery guarantees and hence particular properties of the overlay network by which they are distributed. Control messages are internal to svgCWE. Data messages distribute non-RDF data to group members. Event messages distribute geometrical properties from the workspace to the RDFManager. User actions in the workspace, such as sketching using a drawing tool, are represented by events.

The interface to the Gridkit layer is through Java classes. An instance of svgCWE communicate with other instances by passing messages and RDF between GroupCommunicationManager classes.

Figure 9 uses notation from the OpenCOM component model shown in **Figure 10**.

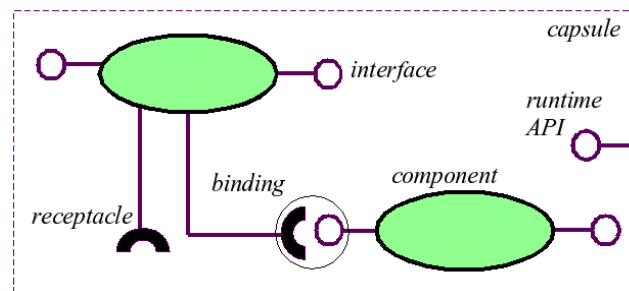


Figure 10: The OpenCOM component model

Components are language-independent encapsulated units of functionality and deployment that interact with other components exclusively through "interfaces" and "receptacles". **Figure 9** just uses the notions of receptacle, interface and binding. Interfaces are expressed in terms of sets of operation signatures and associated datatypes; Components can support multiple interfaces: this is useful in embodying separations of concern (e.g. between base functionality and component management). Receptacles are "required" interfaces that are used to make explicit the dependencies of a component on other components. Finally, bindings are associations between a single interface and a single receptacle. The points to note in **Figure 9** are the bindings between the svgCWE client and communication component, and between the communications component and RDF repository and group abstraction interface over Gridkit. The group abstraction receptacle and RDF repository receptacle ensure that the svgCWE components are agnostic to the kind of overlay used to realize group communication and the kind of RDF repository (e.g. centralized, replicated, or distributed).

The steps below describe how a new piece of geometry (an oval) is added to the workspace in the current implementation. The steps are described in terms of the svgCWE internal architecture shown in **Figure 9**.

1. The user selects the Creation Tool and creates an Oval figure in the svgCWE workspace. (The Creation Tool is on the floating toolbar shown in **Figure 2**.)
2. This generates a create event containing the properties of the new geometrical object, which is passed to the Java environment.
3. If the user is using a web browser, then the create event is sent to a web server proxy using the HTTP protocol. The server is written in Java and should join the group that the

user is a member of.

4. Processing now takes place within the Java environment. The event is routed to RDFManager which generates an RDF representation of the new geometrical object created in JavaScript.
5. The RDFManager sends the RDF representation to the GroupCommunicationManager which in turn distributes it to all participants in the group.
6. A copy of the RDF is then stored in the RDF Repository.
7. When new RDF is received by the GroupCommunicationManagers of each of the other users in the group, the process is reversed culminating in the creation of new geometry in the SVG display environments of each user.
8. In the current implementation, each user keeps a copy of the new RDF received in their local RDF Repository. The architecture implemented so far is thus a replicated repository. **Figure 9** allows more general kinds of RDF repositories, including centralized, replicated and distributed, through the Gridkit layer.

The current state of the tool allows users to replay the content of their local RDF Repository. The user can clear the workspace of its content and request the RDFManager to perform a replay action. The replay request is sent to the RDFManager through the ControlManager. The ControlManager sends a message to the RDFManager to stream the content of the local RDF Repository to the workspace of svgCWE through EventManager. The content is then ordered by timestamp to determine the presentation order.

The current implementation uses the RDF API and repository in the Jena Semantic Web Framework toolkit [**Jena**]. At the time of writing, the implementation is not using RDQL, but we are experimenting with RDQL stand-alone prior to integration into the implementation to support replay and selective display (see **Section 3.3**). The main HotDraw functionality required for sketch annotation has been implemented and additional support for multiple contexts and multiple workspace is being added. Early implementation work used the JGroups reliable multicast communication toolkit [**JGroups**]; we are now porting svgCWE to the Gridkit environment, facilitated by the use of a common group management layer.

5. Related Work

The relationship to the Annotea work has been described above.

Qiu, Carpenter and Fox [**Qiu et al (2003)**] describe a shared SVG browser. They describe the decomposition of the Batik browser based on the Model-View-Controller (MVC) paradigm. The view corresponds to the user interface and the model to a Web service interface. The model and view are linked by a publish/subscribe messaging system called NaradaBrokering. The preparation and interpretation of messages together with the messaging system correspond to the controller component. Collaboration is supported by replicating Web services and delivering events generated on a master view client to all instances of the model, which then service the associated view components. They also describe an alternative approach which is to use NaradaBrokering to multicast the messages from a single model instance to all collaborating view components. The svgCWE approach differs from this work in two respects: firstly the

Open Overlays middleware infrastructure is very different to the NaradaBrokering infrastructure; secondly Qiu et al were not exploring the use of RDF in their work. Fox's group also advocate the use of SVG for interchange between 2D presentation tools [[Lee et al \(2002\)](#)], an approach we also advocate in the Open Overlays scenario.

6. Conclusions and Future Work

We perceive a number of benefits in this approach for this scenario.

1. SVG provides a convenient presentation environment for 2D graphics that is device-independent and can be built into stand-alone or Web based clients.
2. RDF should provide a framework for managing the display of diverse kinds of information in a collaborative setting, where there will be relationships between the roles that actors take and the kinds of information that they require.
3. RDF also provides a framework for expressing relationships between graphical and other kinds of information.
4. Use of RDF also opens up the possibility for richer ways of managing graphical presentations in the future, for example through ontology-based semantic markup.

Implementation of the first version of svgCWE has been completed. The essential HotDraw functionality required for svgCWE is functioning; each member of the workspace has their own local RDF repository. Future plans include experimenting with different kinds of RDF repositories, including DHT-based repositories.

Experience so far suggests that the RDF-based approach is very flexible, and extensible. Descriptions of the contents of a workspace and the members of the workspace can be integrated within a single framework, and hence content for display can be selected based on properties of the content and properties of the members. The examples given show the start we have made in the use of application level semantic markup, but so far we have been exploring ideas and more attention now needs to be paid to the precise vocabularies to be used in the application scenario and their definitions (for example using Semantic Web technology such as OWL) and also to existing vocabularies on which the work should build.

Acknowledgements

We are grateful to Andrea Berardi (Open University) and Jay Mistry (Royal Holloway University of London) for discussions about the application scenario, and Brian Matthews (CLRC Rutherford Appleton Laboratory) for discussions about RDF. Funding support from the EPSRC Fundamental Computer Science for e-Science Programme is gratefully acknowledged.

Bibliography

[Brodli^e et al (2004a)]

K. Brodli^e, D. Duce, J. Gallop, M. Sagar, J. Walton, J. Wood (2004), *Visualization in Grid Computing Environments*, IEEE Visualization 2004, pp. 155-162. ISBN 0-7803-8788-0

[Brodli^e et al (2004b)]

K.W. Brodli^e, D.A. Duce, J.R. Gallop, J.P.R.B. Walton, and J.D. Wood (2004), *Distributed and Collaborative Visualization*, Computer Graphics Forum, 23(2), pp. 223--251

[Cai and Frank (2004)]

M. Cai, M. Frank (2004), *RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network*, WWW2004, ACM Press, pp. 650-657, ISBN 1-58113-844-X

[FARSITE]

FARSITE, Fire Area Simulator, <http://fire.org/> and <http://farsite.org/>

[FOAF]

FOAF Vocabulary Specification, <http://xmlns.com/foaf/0.1/>

[Goad (2004)]

C. Goad (2004), *RDFGeom (RDF Geometry Vocabulary)* , <http://fabl.net/lib/geometry/1.1>

[Grace et al, 2004]

P. Grace, G. Coulson, G. Blair, L. Mathy, W.K. Yeung, W. Cai, D. Duce, and C. Cooper (2004), *GRIDKIT: Pluggable Overlay Networks for Grid Computing*, in Proceedings Distributed Objects and Applications (DOA'04), Lecture Notes in Computer Science 3291, Springer-Verlag. ISBN: 3-540-23662-7. Available at <http://www.springerlink.com/index/6QBX3EE75JQXU0MG>

[Hariri (2004)]

S. Hariri (2004), *Autonomic Runtime System: Design and Evaluation for SAMR Applications*, Unconventional Programming Paradigms, 15 - 17 September 2004, Mont Saint-Michel, France. Slides at <http://upp.lami.univ-evry.fr/Documents/Harriri.ppt>

[HotDraw]

HotDraw home page, <http://st-www.cs.uiuc.edu/users/brant/HotDraw/HotDraw.html>

[Java HotDraw]

HotDraw, <http://c2.com/cgi/wiki?HotDraw> (svgCWE is based on Java HotDraw version 5.1)

[Jena]

Jena Semantic Web Framework <http://jena.sourceforge.net/>

[JGroups]

JGroups - A toolkit for reliable multicast communication , <http://www.jgroups.org/javagroupsnew/docs/index.html>

[Johnson (1992)]

R.E. Johnson (1992), *Documenting Frameworks using Patterns*, OOPSLA '92, pp. 63-76, ISBN:0-201-53372-3; also in ACM SIGPLAN Notices, 27(10), pp. 63-76

[Kahan et al (2001)]

J. Kahan, M-R Koivunen, E. Prud'Hommeaux and R.R. Swick (2001), *Annotea: An Open RDF Infrastructure for Shared Web Annotations*, WWW 10, ACM Press, 2001, pp. 623-632, ISBN 1-58113-348-0

[Lee et al (2002)]

S. Lee, G. Fox, S. Ko, M. Wang, X. Qiu (2002), *Ubiquitous Access for Collaborative Information System Using SVG*, SVG Open 2002, http://www.svgopen.org/2002/abstractlist_byauthor.html

[Prud'hommeaux and Seaborne (2004)]

E. Prud'hommeaux and A. Seaborne (2004), *SPARQL Query Language for RDF*, W3C. Available at <http://www.w3.org/TR/2004/WD-rdf-sparql-query/>

[Qiu et al (2003)]

Xiaohong Qiu, Bryan Carpenter and Geoffrey C. Fox (2003), *Collaborative SVG as A Web Service*, SVG Open 2003, <http://www.svgopen.org/2003/proceedings.html>

[RDF]

Resource Description Framework, <http://www.w3.org/RDF/>

[Sagar et al 2005]

M. Sagar, C. Cooper., D. Duce (2005), *Advanced Mouse Event Model for SVG*, 4th Annual Conference on Scalable Vector Graphics, Enschede, the Netherlands, August 2005. <http://www.svgopen.org/2005/proceedings.do#paper72>

XHTML rendition made possible by [SchemaSoft's Document Interpreter™](#) technology.