SVG for Process Visualization

Keywords: monitoring, simulation, visualization, process modeling, GUI

Johan Koolwaaij, Ph.D.
Scientific Researcher
Telematica Instituut
Enschede

The Netherlands
Johan.Koolwaaij@telin.nl
http://www.telin.nl

Biography

Johan Koolwaaij (1973, M.Sc. 1996 Mathematics, Ph.D. 2000 Speech technology) is a member of scientific staff at the Telematica Instituut since June 2000. Previously, he obtained a Ph.D. in the field of automatic speaker verification at the University of Nijmegen. At Telematica Instituut he has research experience in several projects in the field of electronic business, XML (eXtensible Markup Language) standardization, ASP (Application Service Providing) , context-awareness and web services technologies in different domains including insurance, logistics, building, construction, telecommunications and tourism.

Peter Fennema, M.Sc.
Application Engineer
Telematica Instituut
Enschede

The Netherlands
Peter.Fennema@telin.nl
http://www.telin.nl

Biography

Peter Fennema (1961, M.Sc. 1987 Electrical Engineering) worked for several years in the field of computer aided quality and reliability engineering in various multinational companies. He obtained a bachelors degree in software engineering in 1998. Since 1998 he is employed at the Telematica Instituut. Main topics of interest are software engineering and software architecture.

Diederik van Leeuwen, M.Sc.

Application Engineer
Telematica Instituut
Enschede

The Netherlands
Diederik.vanLeeuwen@telin.nl
http://www.telin.nl

Biography

Diederik van Leeuwen (1969, M.Sc. 1996 Computer Science) is an Application Engineer at the Telematica Instituut in The Netherlands. In research projects Diederik realizes prototypes illustrating research results. Diederik has thorough knowledge of (software) architecture, database technology, XML technology, J2EE and .NET. Prior to joining Telematica Instituut in 2000, Diederik worked as a software developer at Telic Logistics International B.V.

Abstract

This paper reports on a persistent challenge for developers who want to visualize the process that is executed under the hood of an application. Typically, such a process is initiated by an end-user via the user interface. It is quite natural that the execution steps in the process are not visible from the perspective of the user interface, except when an error occurs. Hence, the end-user is often unaware of the complexity or the distributed character of the executed process. And we want to keep it that way, except in the cases where we use the application for demonstration, instruction or scientific purposes, with the focus on what exactly happens under the hood of the application when we perform a particular action. To visualize and animate the process in real-time, a monitoring framework is required that monitors the events in an application and translates these events into a graphical representation that can easily be understood and adapted by the demonstration or instruction group. In the latter phase, SVG can play an important role because of its extensive range of graphics concepts, its highly dynamic nature and because it is programmable through Java or JavaScript. We will present a generic infrastructure for monitoring events in applications that dynamically adapts an SVG representation of an underlying logical process model. In some cases, depending on the modeling language, the SVG front-end can even be generated automatically from the process model. This paper presents some of the most important features of a process monitoring and visualization framework with an SVG front-end.

Table of Contents

1. Process Monitoring Architecture

As applications grow more complex in the sense that they involve more distributed components and new technologies or paradigms, it becomes harder to explain the underlying process for demonstration or instruction purposes. A way to gain insight into the behavior of applications is to visualize the interactions among the different actors or components that are engaged in a process. Seeing the executed process helps to understand how the application works under the hood and what is new or innovative in a particular approach. For that purpose we have developed a monitoring infrastructure that enables applications and their components to control and animate a visual representation of a process model in a web browser.

There exist a number of approaches to process monitoring, including Polka, Modimos and Gridmapper. Polka is an animation toolkit that can be used to visualize programs from many different languages and on many different architectures **[1]** . Modimos is an architecture that allows for monitoring and visualization of a heterogeneous family of object-based environments **[4]** . Clinton Jeffery presents a fundamental model for execution monitoring and visualization **[2]** . And Gridmapper is a tool to visualize the behavior of large-scale distributed systems with a focus on visualizing the location and performance of the different nodes in grid applications **[3]** . All of these approaches are targeted to process visualization for application management, debugging, performance tuning or study of algorithms. And although elements from these approaches can be used, like event routing and general visualization principles, they do not serve the purpose of giving insight in the process that is initiated by specific user actions.

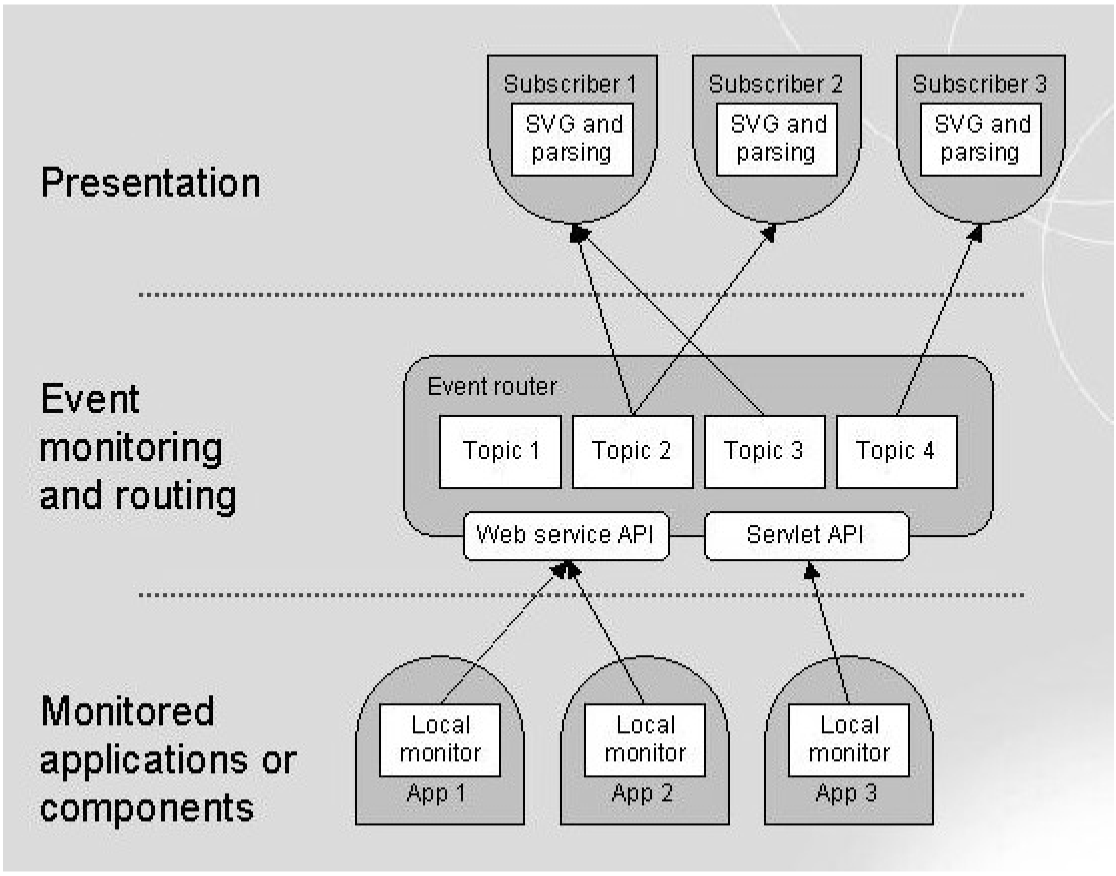

Figure 1: The monitoring infrastructure architecture

We have developed a monitoring infrastructure (MIS) that offers a number of features, including pushing of XML events (typically originating from an application that should be visualized) from a server to a client web browser over a persistent http connection, easy client side JavaScript XML parsing, mapping of XML events to graphical updates that control the SVG representation of the process, and also event queuing facilities to delay the event flow of high-speed applications, in order to adapt the visualization speed to human perception capabilities. To translate the incoming XML events into animation and manipulation of the SVG model, a dedicated XML based transformation language has been developed. The MIS architecture, depicted in **Figure 1** , consists of three layers. The bottom layer contains the monitored applications with their local monitors that send events to the second layer, the global event monitoring and routing layer. This layer collects all events, separates them into different topics and pushes the events to the presentation terminals subscribed to the corresponding topics, which form the third layer.

Monitored applications can publish events to topics on the event router via different types of interfaces, including a web service interface and a servlet interface. Using these interfaces the event router is accessible for heterogeneous applications, irrespective of language they are written in and the platform they use. Event publication requires code modification at cardinal points in the implementation, but by using the web service or servlet approach each event publication requires a minimal number of code lines. (The web service API is currently not part of the monitoring infrastructure as such, but an extension developed within the scope of the WASP project, see **Chapter 3** .) Subscribers can subscribe to topics, and receive events that are published towards those topics. This is similar to the behavior of a JMS (Java Messaging Service) server. The event router is based on pushlets **[10]** . We have also experimented with highly scalable but more expensive solutions as the KnowNow Event Router **[19]** , but for our purposes a Pushlet EventRouter appeared to be sufficient. Pushlets are a Java servlet-based mechanism where data is pushed directly from server-side Java objects to dynamic HTML within a client-browser without using Java applets or plug-ins. This allows for dynamic updating of a graphical process representation situated on the client side but initiated from the server (event router) side. The subdivision in topics makes it possible that more visualization terminals are connected while each of the terminals has its own view on the process events, or even receives a completely different event stream based on the same running process. Or the other way round, one visualization terminal can receive events from different processes or applications running in parallel.

2. Dynamic visualization in SVG

Each visualization terminal runs a SVG representation and a JavaScript XML parser to translate the incoming events into graphical actions so that graphical objects become visible or invisible, start moving, get highlighted, et cetera. In this section, we discuss the creation of SVG representation from process models automatically, the mapping of XML events into graphical actions, the level of control of the end-user over the visualization, and some issues that deserve attention in follow-up work.

2.1 Process modeling

The SVG representation can be designed by a human for relatively simple processes. For more complex processes, however, it is more appropriate to generate the SVG automatically (e.g. using XSLT) from a formal (business) process model, expressed in XML based standards for process modeling. Examples of process modeling languages include ebXML BPSS **[16]** , BPEL4WS **[17]** , UML/XMI **[15]** or RSD studio **[13]** models. These standards all allow for a formal description of business processes, although the underlying meta model may vary quite a bit from one language to another, from web services based to transaction based modeling. Some of these standards already include a graphical representation. However, these representations are usually static or in a non-standard graphical format, and certainly not suited for dynamic, event-triggered visualization. Therefore, a SVG representation is translated from the graphical process model, or newly constructed from the logical process model. The SVG is extended with the proper hooks and APIs to serve as a process visualizer in the monitoring framework. **Figure 2** shows an example of a SVG image that is generated automatically from a RSD model using XSLT **[14]** . All process elements (actions, enabling relations, interfaces, et cetera) have a unique ID that will be used to highlight a specific element when the corresponding event is received.

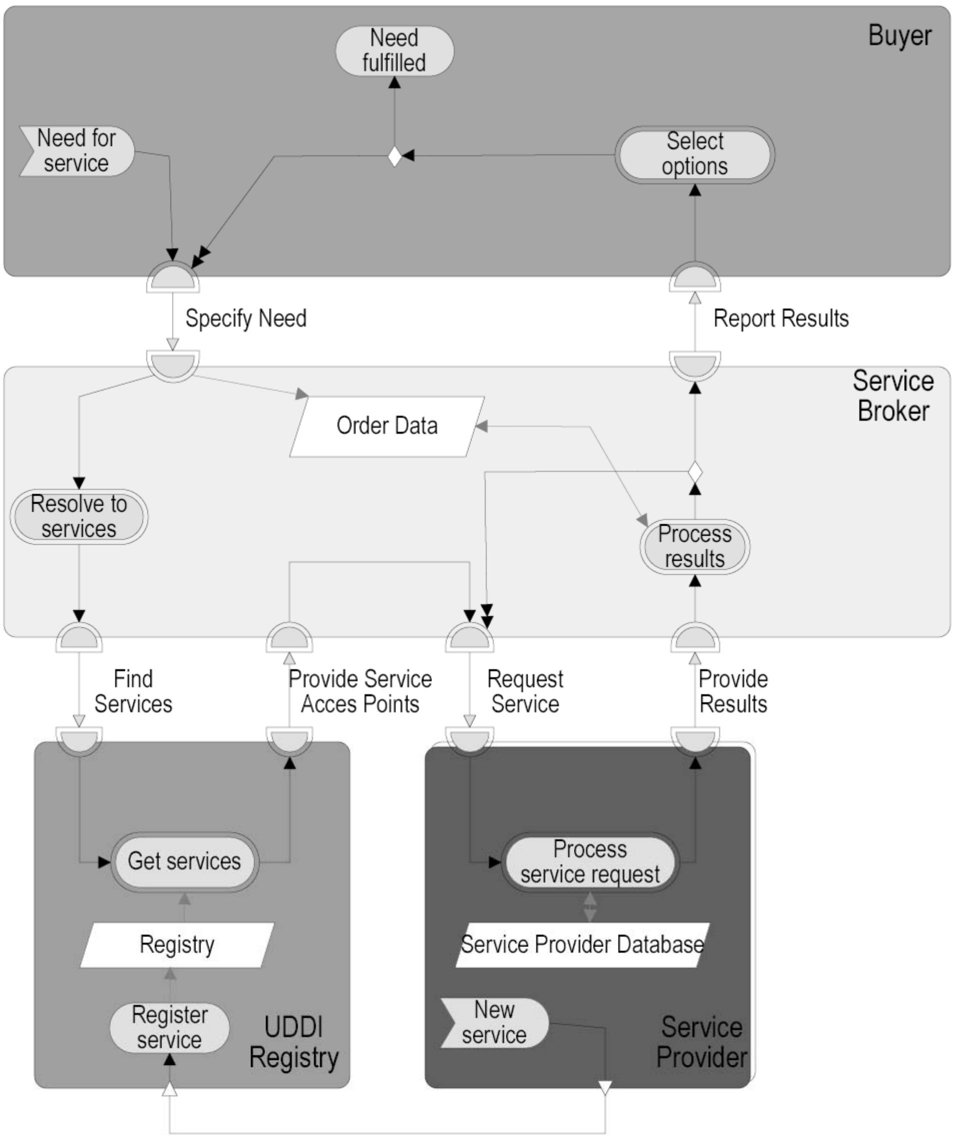

Figure 2: Process model in SVG

It is also possible to export SVG that enables multiple views on the same process. Sometimes the process view can be quite complex with a lot of action within and interaction between parties. See **Figure 3** for an example. In that case it might help to switch to the actor view in which only the high-level interactions between actors are shown. This can be done by clicking the actors button in **Figure 3** . In this example the process can also be simulated, not driven by events from the real time execution but by the process model itself. This can be useful for demonstration purposes when the application is not available or just as an animation to enhance the application's manual or website. The actor view is also suited to present a more fancy version of the process model. Instead of blocks communicating via arrows with other blocks, we can also handcraft a visually nice version of the actor model, with full graphic images of the involved parties and icons showing the different types of interactions. An example is shown in **Figure 8** . Such a handcrafted version requires more design effort, but can still use the same IDs for the actors and interactions and hence the same APIs to animate them. However, it requires SVG design tools that allow for a structured (or object oriented) SVG representation, see also **Section 2.5** .

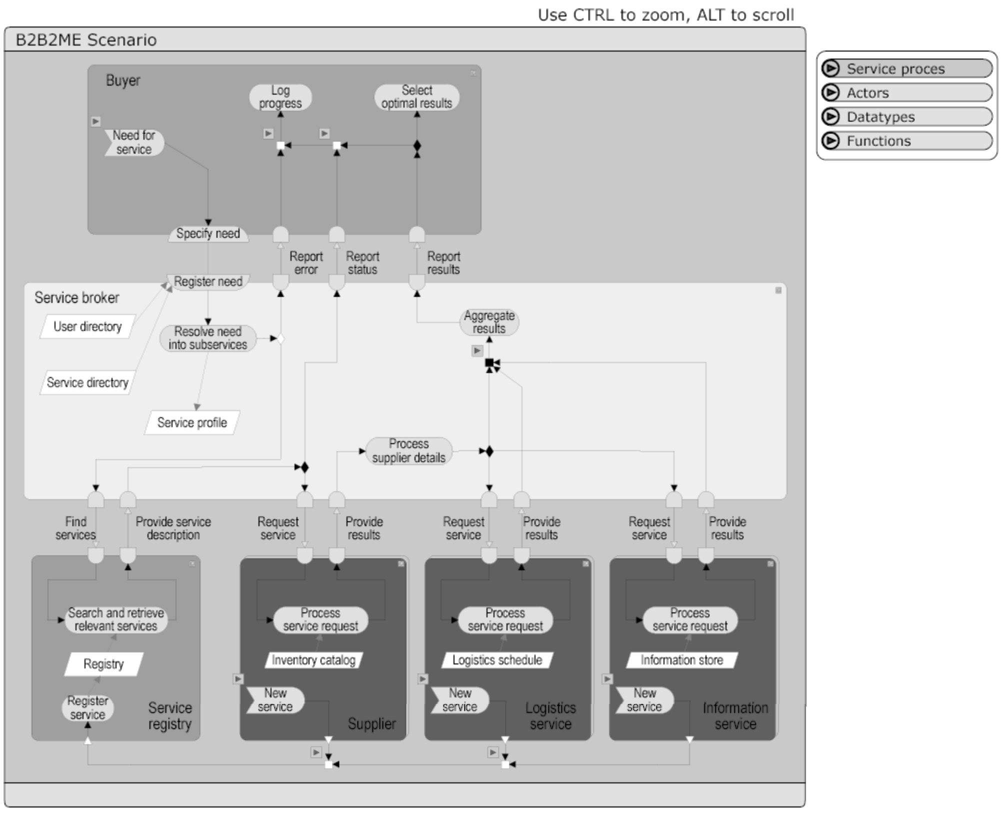

Figure 3: Model with process, actor and function views and process simulation. NB the enlarged version is available at b2b2me_scenario_large.svg

2.2 Mapping of XML events to graphical updates

The XML events, originating from the application to be visualized, have to be translated to graphical updates in the SVG representation. The MIS uses JavaScript in a web browser to handle the incoming events. The JavaScript code is imported into the SVG document. See the SVG snippet below.

```
<svg xmlns:xlink="http://www.w3.org/1999/xlink" onload="selectDefaultView()"
        width="500" height="450" viewBox="0 0 500 450">
        <script xlink:href="../js/xml.js" type="text/javascript"/>
        <script xlink:href="../js/views.js" type="text/javascript"/>
        <script xlink:href="../js/eventsToActions.js" type="text/javascript"/
        <svg id="street" width="500" height="450" viewBox="0 0 500 450">
                <svg id="street_root" width="500" height="500" fill="none">
                        …... ...
                </svg>
        </svg>
</svg>
```

Figure 4: The SVG representation of the process model

To explain the most important libraries: xml.js is a JavaScript library for XML parsing and eventsToActions.js contains the JavaScript code that executes the translation of incoming XML events and manipulation of SVG objects. This file was generated at design time, based on a dedicated transformation language. The transformation language is XML based, and is best explained by an example:

```
<eventsToActions>
        <case>
                <event type="START_CONTENT_UPLOAD_DEAMON">
                </event>
                <action method="setAttribute">
                        <actionTarget fileId="street" objectId="rectContentUp
                        <actionData value="fill"/>
                        <actionData value="limegreen"/>
                </action>
        </case>
        <case>
                <event type="STOP_CONTENT_UPLOAD_DEAMON">
                </event>
                <action method="setAttribute">
                        <actionTarget fileId="street" objectId="rectContentUp
                        <actionData value="fill"/>
                        <actionData value="orange"/>
                </action>
        </case>
        …... ...
        <case>
                <event type="SET_MESSAGE">
                        <eventData name="message"/>
                </event>
                <action method="getFirstChild().setData">
                        <actionTarget fileId="street" objectId="txtMessage"/>
                        <eventDataUse name="message"/>
                </action>
        </case>

</eventsToActions>
```

Figure 5: Translating events to actions

The mapping file shown above is specified at design time by the developer. After specifying the mapping the developer runs a tool (MIS Monitor) that translates the XML mapping file to a

JavaScript file using XSLT. A snippet of the resulting code for this example is shown below. To understand the relation between the XML specification and the generated JavaScript code, which is quite straightforward, please note the event types to start and stop the content upload daemon and to set a message.

```
function handleEvent(event) {
        eventDocument = new XMLDoc(event);
        eventRoot = eventDocument.docNode;
        eventType = eventRoot.getAttribute("type");
        eventSources = eventRoot.getElements("eventSource");
        eventDatas = eventRoot.getElements("eventData");

        if      (       eventType=="selectView"
                &&      eventDatas.length>0
                &&      eventDatas[0].getAttribute("name")=="view"
                ) {
                selectView(eventDatas[0].getAttribute("value"));
        }

        if      (       false
                ||      (       true
                        &&      eventType=="START_CONTENT_UPLOAD_DEAMON"
                        )
                ) {
                        object = svgDocument.getElementById('street_rectConte
                        object.setAttribute('fill','limegreen');
        }

        if      (       false
                ||      (       true
                        &&      eventType=="STOP_CONTENT_UPLOAD_DEAMON"
                        )
                ) {
                        object = svgDocument.getElementById('street_rectConte
                        object.setAttribute('fill','orange');
        }

        …... ...

        if      (       false
                ||      (       true
                        &&      eventType=="SET_MESSAGE"
                        &&      hasElement(eventDatas,"message")
                        )
                ) {
                        object = svgDocument.getElementById('street_txtMessag
                        object.getFirstChild().setData(getElement(eventDatas,
        }
}
```

Figure 6: Generated JavaScript for the SVG monitor

This is a relatively simple example with an elementary process model underneath. For more complex process models an automatic transformation can be used to generate a template for eventsToActions.xml. This template can then be completed by the developer who knows how events translate into actions on the different elements of the process model.

2.3 Event queuing and delay

An important issue in the visualization of applications for demonstration purposes is the adaptation of the event flow of high-speed, optimized applications to human perception capabilities. There are basically 2 ways to resolve this issue.

The first solution is to pause the application for a certain period of time after sending out an event. This is often undesired in real-life applications (since in most cases our goal is to send out events while minimizing the impact on the application's behavior), but in the case of an application that is developed for demonstration purposes only it is a legitimate and simple approach. The MIS contains Java classes that implement this behavior (publish and pause).

The second solution is to queue the high-speed events before sending them out to the web browser. The MIS contains Java classes implementing queuing and dequeuing. The dequeuing speed can be controlled externally. The Java class library for delaying events by means of queuing can be used in any implementation specific configuration.

2.4 SVG GUI controls

To obtain the best demonstration results it is necessary that the end-user (or the demonstrating person) can adapt the visualization in the sense that he is able to select different views on the process, start a simulation without executing a real process or adapt the speed of the demonstration. This could be done by offering HTML controls that interact with the SVG representation, but sometimes it is more natural to have the controls embedded in the SVG representation. In **Figure 3** there is a control to select the desired view on the model that also could have been implemented in HTML, but the small controls to start the simulation of (parts of) the scenario are more natural to appear in the graphical representation itself. Also from a developer's perspective, it could be desirable to have all functionality within one SVG file and not having to maintain scripts in both the SVG and the HTML, or a collection of HTML files when working with frames. In **Figure 8** all controls are embedded in the SVG code. It is an extension of the SVG widgets that are offered by Kevin Lindsey **[11]** . It has controls to select the view, to choose for simulation instead of monitoring and one to adapt the visualization speed. This last control is a slider, implemented in SVG. The slider is not among the standard HTML controls, but is available in Xforms **[18]** , a specification for next generation web forms that clearly distinguishes between what a form does, and how it looks like. However, Xforms is still a standard on its own, and not (yet) supported by the standard internet browsers. SVG is there, ready to use, and it enables developers to invent new controls such as sliders in all sorts of forms, dimmer switches or graph navigation.

2.5 Presentation issues

SVG has been found to be a powerful and flexible technology to display dynamic process models controlled by real-time events from running applications, that deliver insight in the processes executed under the hood of these applications, especially in a distributed web environment. However, a small number of open issues still remain, including the generation of a surveyable SVG representation of rather complex processes, the visualization of parallel events arriving in the same message queue, and for the developers, intelligent development environments that support easy SVG scripting with debugging capability. In general, SVG is very well suited for visualization purposes. Version 1.0 still has a number of shortcomings, like absence of automatic text wrapping and standard user controls, but these issues seem to be solved in version 1.2 [12] . To facilitate the mapping of events to graphical updates in a handcrafted SVG representation it is also important that the SVG representation is well structured, in the sense that graphical objects are modeled as SVG objects with well defined IDs, and not as -for example- an unstructured collection of SVG paths. This requires SVG design tools that support roundtrip editing, that is allow graphical editing, DOM tree editing and text editing simultaneously. Such tools are beginning to appear, e.g. Jasc Webdraw [20] and XStudio [21] , and they are useful but have not yet reached a satisfactory maturity level.

3. Application areas

The MIS architecture has been tested and extended in several demonstrations of (distributed) applications, including business web services, electronic marketplaces, and mobile service delivery. This section discusses a few of our demonstrators that have deployed the MIS successfully.

To start with mobile service delivery, the WASP project [8] proposes a web services based application environment to facilitate and speed-up the development and deployment of context-aware applications for mobile users. Context-aware applications adapt to the user, his/her preferences and the (changing) environment. WASP has built a demonstrator that supports a tourist to plan a town walk along locations that are of interest to him, see **Figure 7** . And during the walk he is supported with navigation information, information about nearby locations and services that might be relevant for him. From the perspective of the user interface it is quite difficult to tell which steps are executed when a tourist requests his personalized town walk. The MIS is used to visualize all the -otherwise invisible- steps such as authorization, profile management, payment, context retrieval, point of interest search, dynamic service integration, or navigation support.

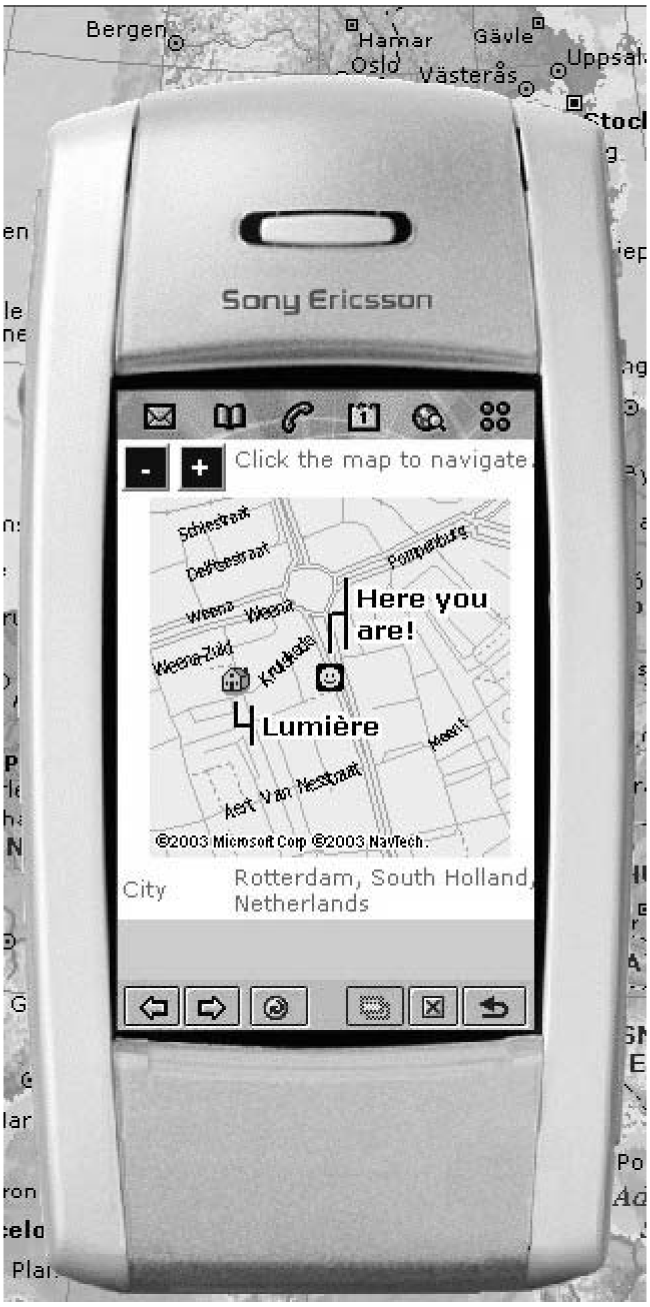
Sony Ericsson
Click the map to navigate.
Here you are!
Lumière
City
Rotterdam, South Holland, Netherlands
©2003 Microsoft Corp ©2003 NavTech.

Figure 7: GPRS device with a context-aware application to support tourists

Within the GigaTS project [5] we have developed a demonstrator of the new possibilities of web service integration. A service broker offers value added services on top of the diverse and dynamic service offerings of third parties, like product suppliers, information services, transports services, et cetera. When the end-user requests information about a specific product, the broker searches real-time for suppliers that offer that particular product, and supplies the end-user with the product offerings plus all kinds of extra services, like business information, ordering services, or transport possibilities. This is also a typical example of a complex search operations and service composition based on new technologies like SOAP, WSDL and UDDI. The MIS has been used to explain more of the operation of these different technologies, both on a high level for large group demonstrations and on the detailed level for personal demonstrations with more in-depth discussion.

The TIPSCI project [9] aims to prototype an integrated content delivery solution that will allow easier, safer and more secure exchange of consumer digital content in the home - such as music, video and games. Issues to be solved include security of content, digital rights management, quality of service provision, and speed and manageability of home communications networks. These are enabling technologies that are usually below ground level. The MIS architecture is used to visualize what these enabling technologies contribute to secure exchange of digital content in the home, making it clear how content is transferred between different nodes in a household and how the different technologies act and interact. The example visualization (**Figure 8**) offers the possibility for different views depending on the interest of the public and also to simulate instead of monitor the executed process, which might be useful for stand-alone demonstrations.

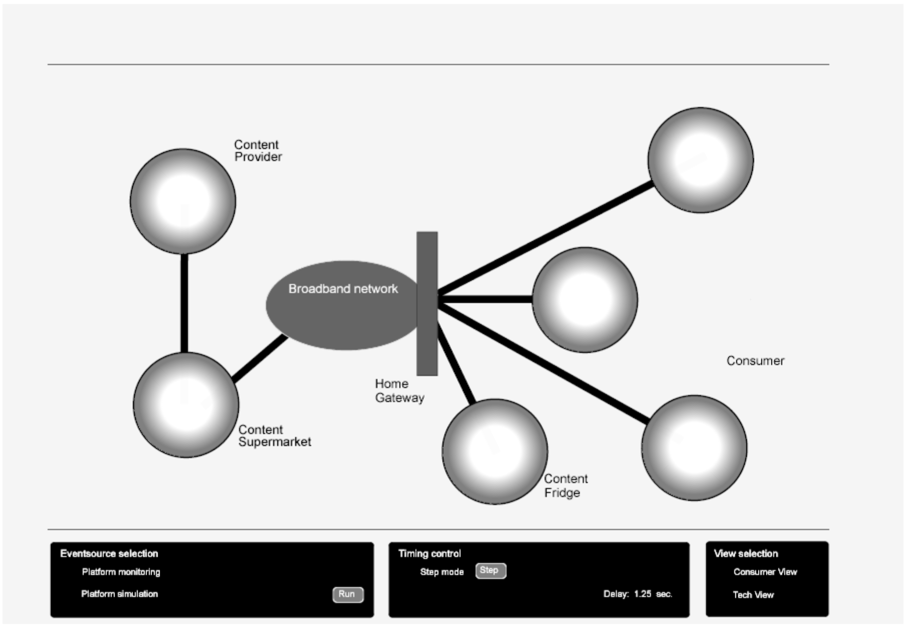

Figure 8: Visualization of monitoring or simulation with SVG controls (controls are not active). NB the enlarged version is available at tipsci.svg

4. Conclusion

We have presented the MIS architecture that offers real-time process monitoring and visualization in distributed applications for demonstration or instruction purposes. It uses intelligent event routing to transfer events from the execution platform to the visualization terminal. It allows for multiple views on the same process that can be adapted according to the need of the demonstration or instruction group. The visualization terminal shows a SVG front-end that can be generated from a process model and animated driven by the events that are fired by the executed process. SVG is very well suited because of the possibility of automatic creation, its richness of graphic concepts, its highly dynamic nature and because it is programmable through JavaScript. A number of issues have been identified that need further investigation in the next versions of the MIS, which is still under development. And some of the issues are 'automatically' solved with the introduction of SVG 1.2. More information on the MIS package can be obtained from [22] .

Acknowledgements

The authors are grateful to the projects GigaTS [5] , WASP [8] , Tipsci [9] , GigaMobile [6] and Uluru [7] for serving as a fruitful testing environments for the monitoring infrastructure toolkit .

Bibliography

[1]
Visualization of Parallel and Distributed Programs, http://www.cc.gatech.edu/gvu/softviz/parviz/parviz.html

[2]
Monitoring and Visualizing Program Execution: an Exploratory Approach, Clinton L. Jeffery, http://www.cs.arizona.edu/icon/ftp/doc/pv.pdf, Ph.D Dissertation, 1993

[3]
GridMapper: A Tool for Visualizing the Behavior of Large-Scale Distributed Systems, William Allcock et al., http://www.computer.org/proceedings/hpdc/1686/16860179abs.htm, IEEE HDPC'02, p. 179, 2002

[4]
Monitoring and Visualizing Heterogeneous Distributed Object Applications, Jakub Szymaszek, http://www.ics.agh.edu.pl/~jasz/papers/ersads95.ps.gz, .ERSADS'95, L'Alpe d'Huez, France, April 1995

[5]
Giga Transaction Services, http://gigats.telin.nl

[6]
Giga Mobile Services, http://gigamobile.telin.nl

[7]
Platform for mobile audiovisual services, http://uluru.telin.nl

[8]

Web Architectures for Services Platforms, http://wasp.freeband.nl

[9]

Telematica Instituut, IBM, Philips Secure Content Initiative, http://www.ibm.com/news/nl/10092002_nl_nl_philips.html

[10]

Pushlets: Send events from servlets to DHTML client browsers, http://www.javaworld.com/javaworld/jw-03-2000/jw-03-pushlet.html

[11]

Set of experimental SVG GUI widgets, Kevin Lindsey, http://www.kevlindev.com/gui/index.htm

[12]

Scalable Vector Graphics 1.2, http://www.w3.org/TR/SVG12

[13]

RSD studio 3.0, https://doc.telin.nl/dscgi/ds.py/Get/File-22415

[14]

An extensible import and export facility for RSD Studio, https://doc.telin.nl/dscgi/ds.py/Get/File-24671

[15]

XML Metadata Interchange 1.2, http://www.omg.org/technology/documents/formal/xmi.htm

[16]

ebXML Business Process Specification Schema , http://www.ebxml.org/specs/index.htm

[17]

Business Process Execution Language for Web Services , http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[18]

XForms: The Next Generation of Web Forms, http://www.w3.org/MarkUp/Forms/

[19]

KnowNow Event Router, http://developer.knownow.com/devguide/knrouter/EventRouterPG.html

[20]

Jasc WebDraw, http://www.jasc.com/products/webdraw

[21]

XStudio, http://www.evolgrafix.com/

[22]

Monitoring Infrastructure, http://portal.demo.telin.nl/mis