Distributed Simulation and Web Map Mash-Up for Forest Fire Spread

Yosri Harzallah[1], Vincent Michel[2], Qi Liu[3], Gabriel Wainer[3]

University of New Brunswick, Fredericton, NB, Canada [1,2]
Ecole Nationale Supérieure des Mines, Saint-Étienne, France [1,2]
Carleton University, 1125 Colonel By Dr, Ottawa, ON, Canada [3]
yosri.harzallah@unb.ca [1], vincent@vincent-michel.fr [2], {liuqi, gwainer@sce.carleton.ca} [3]

Abstract

The emergence of recent XML-based technologies paved the way for new types of architectures and message exchanges on the Internet through Web services. Mash-Ups consist in combining existing systems capabilities in order to create new added value services. This paper focuses on a Mash-Up of different Web Services that can be used by emergency planning teams and firefighters to better understand the behavior of wildfires. This mash-up uses a distributed simulation system (DCD++) and it combines it with the Global Weather service, displaying the results on a Google Map. The weather data is combined in real-time with the theoretical fire spreading model to obtain precise results. An XML-based solution has been developed in order to serve performance, interoperability and usability around a two-tier architecture consisting of a Java Servlet and a Javascript client. This effort paves the road for even more advanced solutions integrating the simulation location environment with Geographic Information System (GIS) data, which would highly improve the results obtained by the emergency managers.

1. Introduction

The emergence of recent XML-based technologies paved the way for new types of architectures and message exchanges on the Internet. This eXtensible Markup Language has provided interoperability between partners and enabled companies to deploy a myriad of machine-consumable Web-based services, which can be later integrated in many different ways to produce multiple sets of services. A system reusing existing distributed services and combining them to provide added value through a web application is called a Mash-Up. For example, Web services for currency conversion and stock quotation can be associated to return financial information expressed in a target currency. We present the implementation of a Mash-Up applied to the field of wildfire spreading prediction (the reader can find a demo version available at [1]). This Mash-up combines three components: a simulator to predict the spread rate and direction of a wildfire, a worldwide weather service to adjust fire model parameters, and a Google Map to view the fire spread on-line.

2. Motivation

There is a growing need for emergency teams to have tools that can provide rapid and accurate information about the situation they are facing. In particular, in the last few years the increasing influence of global warming on the environment has produced wildfires with devastating consequences (including the recent 2007 fires in Greece, Utah, British Columbia and Southern California). In the case of wildfire management, fire managers and fire fighters must make important decisions quickly. However, wildfires are so unpredictable that reducing their impact is still a complex task. In order to assist the experts in the area, we need to provide on-line information about the fire evolution, including real-time updates with weather data and improved decision-making tools [2].

Fire evolution is a complex natural phenomenon that makes a mathematical analysis intractable. Instead, simulation provides a means to study fire spread by creating a software application that approximates the evolution of fire behavior. Simulation software has highly enhanced the understanding of the phenomenon, as facing simulation results to experimental data improves the model definition and helps understanding the fire behavior.

Fire models can be classified according to the methods used in their construction [3]. They can be *statistical* (i.e., without involving physical mechanisms) or *semi-empirical* (i.e., based on the principle of energy conservation). Many of the existing simulation models are based on Rothermel's stationary model, which integrates wind and slope parameters empirically [4]. Additional mechanisms of heat transfer and production can be incorporated to the models, in order to make them more robust.

The results obtained in the field of fire simulation are nowadays encouraging enough to contemplate real-world study cases. In real fire scenarios, visualization over a map interface and integration with live data can improve the ability of a fire control center to coordinate firefighters in the field. In order to do so, it is necessary to have a platform that can interface with a wide range of technically diverse data sources and present the computation results to decision makers in an intuitive and user-friendly manner. Nevertheless, a major drawback of most existing simulators is that they do not link the geospatial data with the simulated model in real time. As a consequence, the fire control center could not properly coordinate firemen teams on the field. The Mash-Up developed in this project aims at bridging this gap.

3. Related Work

3.1. Fire model

The fire model of interest in this paper is described in [5]. It is an advanced fire model based on Rothermel's forest-fire propagation rules. The model has been created using the Cell-DEVS formalism [6]. Cell-DEVS is a variant of DEVS (Discrete-Event Systems specifications), a modeling and simulation formalism that provides a mathematical representation of discrete-event systems [7]. Numerous studies have showed that DEVS is suited for the simulation of fire spread (including [8, 9] and numerous others). Cell-DEVS is also widely used for this purpose thanks to its natural support for spatial relations and explicit timing constructions [2, 5, 10]. The formal specifications are focused on modeling systems organized as cell spaces (e.g., a burning area in a forest).

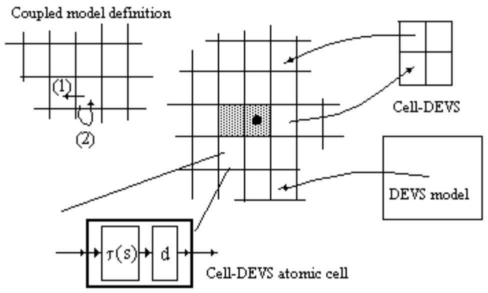


Figure 1. Informal definition of Cell-DEVS [6].

Cell-DEVS allows defining cellular models as a composite of atomic cells. Each cell is seen as having a set of N inputs to compute its future state (generally received from the neighboring cells). Upon reception of a new input, a local function is activated. The results are transmitted to the neighbors after a delay. Once the cell's behavior is defined, they can be put together to form a coupled model composed of an array of atomic cells, each of them connected to its neighborhood. As the cell space is finite, the borders should be provided with a different behavior than the rest of the space. Otherwise, the space can be defined as *wrapped*, meaning that cells in a border are connected with those in the opposite one.

The fire model uses environmental and vegetation conditions, and it computes the ratio of spread (i.e., the distance and direction the fire moves in a given timeframe) and the intensity of fire on each cell. Three parameter groups determine the fire spread ratio:
- vegetation type (caloric content, mineral content and density);
- fuel properties (the vegetation is classified according to its size);
- environmental parameters (wind speed, fuel humidity and field slope).

When Rothermel's rules are applied to a given fuel model and environmental parameters, it can determine the spread ratio in every direction. The spread ratio is then used to define the rules that dictate how fire spreads from one cell in the cellular space to its eight neighboring cells.

3.2. CD++ toolkit

The CD++ simulation toolkit [11] is a core component of our Mash-Up. CD++ is based on the DEVS and Cell-DEVS formalisms and thus allows for simulating the fire model introduced previously.

CD++ is available in different versions running on different platforms. There are standalone (single machine), parallel (over a cluster of machines) and distributed (over a network of computers) versions as described in [12]. The distributed version, called as DCD++, relies on the proper functioning of a Web service wrapper that interacts with the CD++ simulation engine and exposes its functionality to remote web service clients using SOAP messaging over the HTTP protocol, as shown in Figure 2.

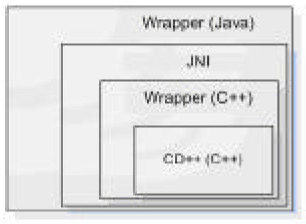


Figure 2. Components of the wrapper [12]

CD++ is written in C++ and the technology chosen to develop the Web service wrapper is Java. To interface these two components, the Java Native Interface (JNI) that allows Java programs to interact with C/C++ libraries was used. The wrapper consists of two separate layers. A layer to expose the CD++ simulator's functions and the other to expose the Java objects. This approach was followed because C++ is well-suited for efficient computations, while Java is widely supported by web service middleware for interoperability [12]. Figure 3 gives a closer look at the DCD++ software architecture.

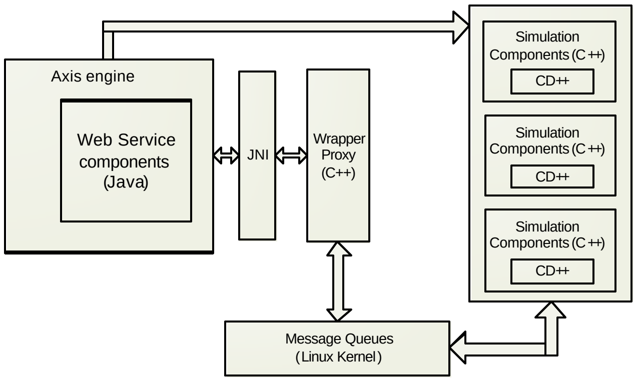

Axis engine
Web Service components (Java)
JNI
Wrapper Proxy (C++)
Simulation Components (C++)
CD++
Simulation Components (C++)
CD++
Simulation Components (C++)
CD++
Message Queues (Linux Kernel)

Figure 3. DCD++ software architecture [12]

Upon the authentication of a user, the simulation service creates a new session with a unique ID.

- A separate workspace containing the source files is created on the server for the new session.
- The wrapper creates two message queues in the Linux kernel to implement a bidirectional communication channel between the web service component and the corresponding simulation component.
- The user sets the parameters of the simulation by uploading configuration files using a predefined format.
- The user starts the simulation.
- When the simulation is over, the user can get the simulation results from the server along with a session log file that records detailed client operations.

The user interacts with the simulator through SOAP messages as defined in a WSDL file that specifies the interface of the simulation service. The following Figure 4 is an excerpt of this WSDL file where three SOAP messages are defined by specifying the name and type of the parameters and return values. As we can see, DCD++ makes use of SOAP attachments through the DataHandler type of the JavaBean Activation Framework to exchange data in complex type formats (in this case, .cpp, .h, .log, .ma, etc.).

```
<wsdl:definitions
targetNamespace="http://www.sce.carleton.ca/ars/CDpp">
            <wsdl:message name="retrieveLogFileRequest">
                  <wsdl:part name="in0" type="xsd:int"/>
            </wsdl:message>
            <wsdl:message
                    name="getCurrentSimulationTimeResponse">
                   <wsdl:part
name="getCurrentSimulationTimeReturn"
                        type="soapenc:string"/></wsdl:message>
            <wsdl:message name="setGridConfigFileRequest">
                   <wsdl:part name="in0" type="xsd:int"/>
                   <wsdl:part name="in1" type="soapenc:string"/>
                   <wsdl:part name="in2"
                         type="apachesoap:DataHandler"/>
            </wsdl:message>
…
```

Figure 4. Excerpt of the WSDL file.

The simulation results are stored in a log file that records the detailed message exchanges between model components. These messages can be used to visualize the simulation progress, as shown in Figure 5. In this case, after 02:23:946, cell (17,12) is activated (* message). As a result of the spreading rules, the cell informs its current spreading rate (message *Y* with value *3.3991*), and it passivates (*D*one message, informing that the next activation time is at *infinity*). The spreading rate is informed to the neighbors (i.e., cells *(16,11)*, *(16,12)*, etc.) using an *X* message carrying the spread rate value.

```
...
* / 02:23:946 / top to forestfire
* / 02:23:946 / forestfire to forestfire(17,12)
Y / 02:23:946 / forestfire(17,12) / out / 3.3991 to forestfire
D / 02:23:946 / forestfire(17,12) / infinity to forestfire
X / 02:23:946 / forestfire / neighborchange / 3.3991 to
forestfire(16,11)
X / 02:23:946 / forestfire / neighborchange / 3.3991 to
forestfire(16,12)
...
```

Figure 5. Excerpt of the LOG file

Wrapping CD++ into a Web service allows for the execution of complex models on heterogeneous machines across multiple domains. Furthermore, it enables users to use a browser as a high-end client to invoke the simulation and visualize the results.

3.3. Global Weather Web service

The British company WebserviceX.net [13] is one of the publicly available Web services that offer access to a worldwide live weather information. This is achieved by exposing two methods called as

`GetCitiesByCountry` and `GetWeather` respectively. Both support GET, POST and SOAP requests.

The first method is used to retrieve a list of cities in XML format when given a country name as input. Figure 6 shows the XML response of a `GetCitiesByCountry` query for Luxembourg. The element `NewDataSet` lists the only city available in Luxembourg as a child element.

```
<?xml version="1.0" encoding="utf-8"?>
  <NewDataSet>
    <Table>
      <Country>Luxembourg</Country>
        <City>Luxembourg / Luxembourg</City>
    </Table>
  </NewDataSet>
```

Figure 6. Result of a `GetCitiesByCountry` query

As shown in Figure 7, the second method returns an XML string, representing the weather information for a previously retrieved city name: last update time, wind speed and direction, temperature, humidity and other parameters.

```
<?xml version="1.0" encoding="utf-16"?>
  <CurrentWeather>
    <Location>Luxembourg / Luxembourg, Luxembourg
        (ELLX) 49-37N 006-13E 379M</Location>
      <Time>Mar 26, 2008 - 07:20 PM EST / 2008.03.27 0020
   UTC
      </Time>
      <Wind> Variable at 1 MPH (1 KT):0</Wind>
      <Visibility> 4 mile(s):0</Visibility>
      <Temperature> 37 F (3 C)</Temperature>
      <DewPoint> 35 F (2 C)</DewPoint>
      <RelativeHumidity>93%</RelativeHumidity>
      <Pressure> 29.47 in. Hg (0998 hPa)</Pressure>
      <Status>Success</Status>
  </CurrentWeather>
```

Figure 7. Result of a `GetWeather` query

3.4. Google Map API

Google Map is a powerful service that can be embedded in Web applications. Based on the Google AJAX API Loader, Google Map v2.0 provides a clear and simple Javascript API to interface the main Web service map [14]. Capabilities of personalizing the map behavior and adding customized content have been exploited in this project to display fire spread evolution dynamically.

4. The Service Oriented Architecture for Simulating and Visualizing Forest Fire

4.1. Architecture

To realize this Mash-Up, several requirements have to be met:

- For the simulator:
  - The format of the file returned by the simulator has to be converted to be correctly displayed on the Google Map. Due to Javascript's poor performance and so as not to overload the client, a server has to be involved to realize this task.
  - The programming language used to implement the application on the server must have libraries supporting SOAP calls and especially DataHandlers in the methods parameters. So Javascript is excluded.
- For the Google Map API:
  - The API is written in Javascript. So to display dynamical information on the map, this requires querying the server through AJAX.

Considering the second requirement, it has been decided to use Java for its portability and a Servlet as a HTTP server for its straightforward implementation. On the one hand, Java is used to query Web services with AXIS library, and on the other hand, it receives AJAX queries, processes it and returns the response. Moreover, designing the Mash-Up by distributing the main tasks between several Servlets reduces the development costs. Adding a new functionality simply means creating a new Java Servlet.

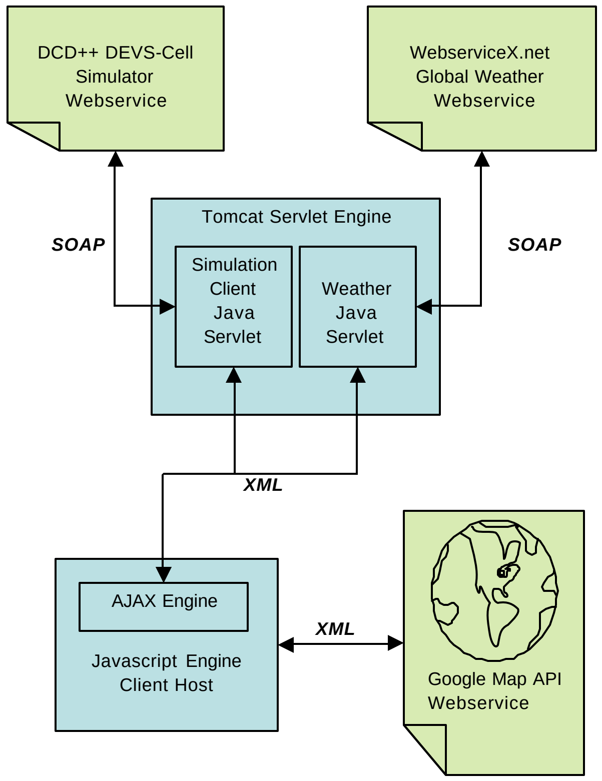

DCD++ DEVS-Cell Simulator Webservice
WebserviceX.net Global Weather Webservice
SOAP
SOAP
Tomcat Servlet Engine
Simulation Client Java Servlet
Weather Java Servlet
XML
AJAX Engine
Javascript Engine Client Host
XML
Google Map API Webservice

Figure 8. An overview of the Mash-Up architecture

4.2. Common Servlet development framework

Communication between Servlets and clients are made through the POST method. Like any Web service, the parameter “method” specifies Java application’s behavior. If the method targets a call to a distant Web service, the next POST parameters will have the same name as the Web service WSDL expects.

```
<response type=”result”>
…
</response>

<response type=”error”>
   <faultString>This is a description of the error</faultString>
   <faultTrace></faultTrace>
</response>

<response type=”notAuthenticated” />
```

Figure 9. The three types of XML messages exchanged between the client and the Servlets

A common set of XML messages have been defined to provide a coherent Servlet communication interface with the client. Figure 9 shows an example of the three message types differentiated by the `type` attribute in the root element: the value “result” will be used to denote a successful query. A type “error” implies a Servlet processing error, where the child element `faultString` gives the reason for the error. A type value “notAuthenticated” indicates that the user is not connected. The attribute `type` in the root element allows for efficient selection of appropriate actions at the early stage of the Javascript callback process.

Java Exceptions are handled by a common class called as `ClientException`. Should any error arise during data processing, this exception is thrown and handled at the top level method of the Servlet to return an `XMLResponse` error message.

4.3. Simulation Client Servlet

The Simulation Servlet aims at providing Web service query capabilities to the Javascript client side, which does not support SOAP messages. As a result, DCD++ WSDL has been exploited to generate the Java stubs used to communicate with the simulator. The interface of the simulation service includes the following methods [12].

- `authenticate`: authenticates the user and initializes a new session for each successful login.
- `setMAFile`: uploads a model definition file to the simulation service.
- `setSupportFile`: uploads other supporting files (e.g., the initial cell values of a Cell-DEVS model) required by the simulation engine.
- `setGridConfigFile`: uploads an XML file containing the configuration of the distributed grid of computers used during the simulation.
- `startSimulationService`: sends a start simulation signal to the simulator.
- `retrieveLogFile`: retrieves the generated simulation log file.
- `getStatus`: returns the current simulation status.
- `deleteSession`: deletes the temporary files created on the server.
- `retrieveSessionLogFile`: retrieves operation information or error messages logged during the current session.
- `setExecutionTime`: sets the end time of the simulation.
- `killSimulation`: forces a simulation to stop prematurely.
- `getCurrentSimulationTime`: returns the current simulation time.
- `logOff`: logs off the user and erases the session.

In addition, several methods have been added to match the client side requirements. The methods `setMAFileMIME`, `setSupportFileMIME` and `setGridConfigFileMIME` were added to upload the corresponding files from the client to the simulation server through the Servlet engine server. We also modified `setMAFile`, `setSupportFile` and `setGridConfigFile` to take a string instead of a file as input and to upload a file on the server.

Another method added is `retrieveCustomXMLDrw`. This method does not match any of the Simulator Web service interface. It is derived from the fact that the format of the simulation results is inadequate to be processed on the client side. A CD++ utility tool bridges this gap by generating a plain text matrix representation of the cell space and a graphical 2D representation. However, even for a middle-sized fire model the file size can easily reach more than 2MB, which is bulky to be transmitted over the network. `retrieveCustomXMLDrw` brings a solution to this problem. It retrieves the simulation log file and processes it with a customized parser. Instead of sending plain text data, an XML document is generated and sent. For each simulation time step, only those values that have been changed are specified in the XML document. Consequently, the 2MB text file becomes a 49KB XML document. More than just an optimization in the file size, this also allows for communication using standard XML messages.

```
<matrices number='387' x='30' y='30'>
   <mat step='1' time='00:00:00:000'>
      <point x='19' y='10' value='1.0' />
   </mat>
   <mat step='2' time='00:01:11:973'>
      <point x='18' y='11' value='2.200000047683716' />
   </mat>
   <mat step='3' time='00:02:23:946'>
      <point x='17' y='12' value='3.4000000953674316' />
   </mat>
   <mat step='4' time='00:02:59:049'>
      <point x='18' y='10' value='4.0' />
      <point x='19' y='11' value='4.0' />
   </mat>
….
```

Figure 10. Log file transformed into an XML document.

On the client side this XML file is processed and shown on the map. The attributes of the root element give the number of steps in the simulation and the size of the matrix representing the burning area. Each `mat` tag represents the system status at a specified time. The `point` child elements contain the information on modified cells. Each `point` is mapped to a `GPolygon` object defined in the Google Map API and is shown on the map. A reference to each polygon is kept so that the user can run the simulation backward or forward. The data structure holding the `GPolygon` objects is a multidimensional array indexed with the step number and the order of the `point`s in the step.

To finish, the system keeps track of the user's authentication using Servlet sessions. The Servlet looks up the session object associated with every request. This object is created upon authentication and stores the session ID until the user logs off. This restricts the use of the simulator to authorized users.

4.4. Global Weather Servlet

The Global Weather Web service is used to provide the user with details about weather conditions, especially the wind speed and direction, as these parameters are needed to generate correct simulation results.

The main purpose of the Global Weather Servlet is to interface the Web service with the client and return an HTML response to reduce the processing time required by Javascript. The POST `method` parameter can be set to either `GetCitiesByCountry` or `GetWeather`. Once the call of the Web service is finished, an XSLT processor is launched with a style sheet according to the method and the final presentation requirements.

In the case of the first method, each `City` element is transformed with XSLT to be inserted into a `<select>` HTML tag. So `option` elements are created. City names are also sorted for better usability. Before being embedded into a `response` element to create the response, tag delimiters of the XSLT transformation are replaced by HTML special characters `&lt;` and `&gt;`. This technique is useful for the Javascript client. It aims at parsing the document quickly with DOM without having to browse the entire tree and reprocessing it.

```
…
<xsl:template match="/NewDataSet">
   <xsl:for-each select="Table">
      <xsl:sort select="City" />
      <xsl:apply-templates mode="city"/>
   </xsl:for-each>
</xsl:template>

<xsl:template match="City" mode="city">
   <option>
      <xsl:attribute name="value">
         <xsl:value-of select="text()" />
      </xsl:attribute>
      <xsl:value-of select="text()" />
   </option>
</xsl:template>
<xsl:template match="*" mode="city" />
```

Figure 11. Part of the city presentation XSL style sheet

For the second method, another style sheet is used to extract the content of each information element included in the Weather Web service response (Figure 7), and it inserts a simple line-break `<br/>` between each of them. In case of a city where no weather information is available, the content of the response will be "No Data Result". The result is finally returned to the client. The content of the `Location` element is kept in another sub-element of the response to be able to extract it later in Javascript and to query the Google Map Web service to move to the target location on the map interface.

```
<?xml version="1.0" encoding="utf-8"?>
   <response type="result">
      <location> Luxembourg / Luxembourg, Luxembourg
</location>
      <text>
         Time: Mar 27, 2008 - 12:20 PM EST / 2008.03.27 1720
         UTC&lt ;br /&gt ;
         Wind: from the W (280 degrees) at 9 MPH (8 KT):0&lt ;br
/&gt ;
         Visibility: greater than 7 mile(s):0&lt ;br /&gt ;
         SkyConditions: mostly cloudy&lt ;br /&gt ;
         Temperature: 37 F (3 C) &lt ;br /&gt ;
         DewPoint: 35 F (2 C) &lt ;br /&gt ;
         RelativeHumidity: 93%&lt ;br /&gt ;
         Pressure: 29.53 in. Hg (1000 hPa) &lt ;br /&gt ;
      </text>
   </response>
```

Figure 12. Example of the result returned by the Global Weather Servlet

XSL technology has been chosen to process XML since it is an XML standard and can be authored efficiently.

4.5. Client Side

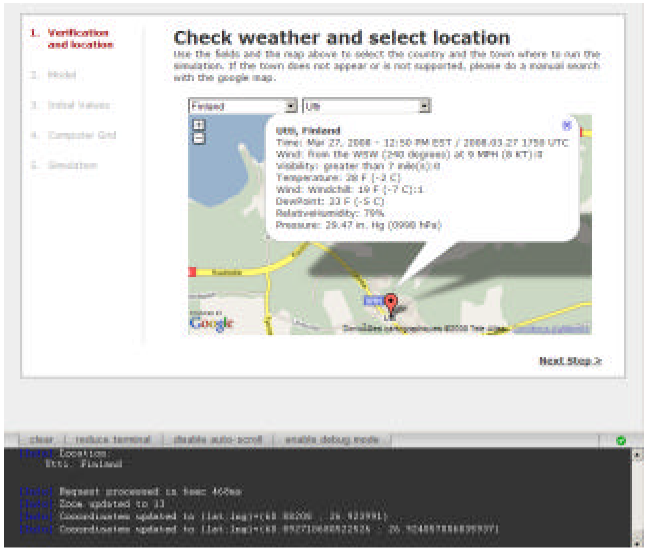
The client side must complete several objectives, including:

- Interfacing with the Google Map Web service
- Providing a simple GUI for the user
- Querying the two Servlets with AJAX

The client has been implemented around four key standards: XHTML for content organization, CSS for presentation, Javascript for data processing, and XML for information streams in order to guarantee a complete interoperability with future work or technologies involved. Frames have been used to present a console displaying information and error messages coming from Javascript functions. Frames are organized as follow:

- Main Frame: main frame container
  - o Main Content: displays and reloads user pages
  - o Console Frame: console frame container
    - Console Menu: contains console controls and status.
    - Console Content: displays information and error messages.

Figure 13 shows the frame organization, including the Console Frame (in the bottom panel) and the Main Content Frame (in the center panel) for user information and controls.



Check weather and select location

Use the fields and the map above to select the country and the town where to run the simulation. If the town does not appear or is not supported, please do a manual search with the google map.

Utti, Finland
Time: Mar 27, 2008 - 12:50 PM EST / 2008.03.27 1750 UTC
Wind: from the WSW (240 degrees) at 9 MPH (8 KT):0
Visibility: greater than 7 mile(s):0
Temperature: 28 F (-2 C)
Wind: Windchill: 19 F (-7 C):1
DewPoint: 23 F (-5 C)
RelativeHumidity: 79%
Pressure: 29.47 in. Hg (0998 hPa)

```
[Info] Location:
   Utti: Finland

[Info] Request processed in time: 468ms
[Info] Zoom updated to 13
[Info] Coordinates updated to (lat,lng)=(60.88206 , 26.923991)
[Info] Coordinates updated to (lat,lng)=(60.89271068852252 , 26.92485788689937)
```

Figure 13. Screenshot of the client side interface

More than just a means to log information, the console implements a system of variable storage. The Javascript object Console exhibits three methods to achieve this goal: `create("myVar")`, `set("myVar", "value")` and `get("myVar")`. Since the console frame is not reloaded, the variables keep their values and can be reused in the user frame. The button "`enable debug mode`" allows the user to display Java exception traces if there is one.

When the user connects to the system for the first time, a user name and a password are required to create a session on the Simulator Web service. If the authentication is successful, a session is created on the Servlet to keep the trace of the user. The following steps are necessary to initiate the simulation.

1. The location of the wildfire is specified via two HTML select fields. Queries are sent to the Global Weather Servlet to retrieve the list of available cities and the current weather condition. Weather information is printed directly on the Google Map. The map center and zooming scale are updated automatically. The user can now update the fire model with the weather information (wind direction and speed, humidity, air temperature, etc.). A specific location can also be selected manually on the map but weather information may not be available for that location.
2. The fire model can be uploaded to the server in two possible ways: using a file in the local file system, or copying the model directly in the text field. A confirmation message is displayed in the console.
3. Similarly, the user uploads a grid configuration file, which defines the mapping of the fire model on the remote hosts involved in the distributed simulation.
4. The simulation can be started upon the successful completion of the previous steps. By clicking on the start button, the simulation is launched through the Simulation Servlet and the user is redirected to the map interface. When the simulation is finished and the information retrieved successfully, the map controls are enabled for the user.

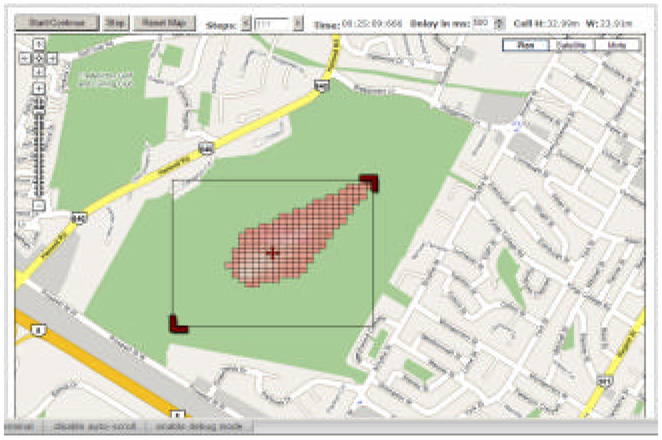


Figure 14. Screenshot of the simulation map interface

Note that the user can go back to a previous step at any time to modify the data and/or to upload a different file.

Before showing the simulation results on the map interface, the user can adjust the simulation area, as seen in Figure 14. The scaled values of the width and height of a cell are updated and displayed along with the simulation time and steps when the animation is running. The log file and session log file generated by the Simulation Service can also be downloaded for further investigation. The animation rate can be adjusted with the delay cursor in the top panel.

5. Conclusion

We have introduced a Mash-Up combining three Web services to improve the ease of use and effectiveness of wildfire simulation. The Web Services were designed to transform information and to bring an advanced visualization support to a Cell-DEVS based fire model. Around a two-tier architecture consisting of a Java Servlet and a Javascript client, different XML-based technologies such as XHTML, SOAP, XPATH and XSLT have been used to enhance performance, interoperability and usability.

Based on a predefined model and reducing the number of input parameters can make the simulator easier to use and make it more likely to be used in a real-world application for emergency purposes.

The Mash-Up developed in this project can be enhanced by adding new sources of information on the simulation parameters in addition to the weather conditions, so that the user can define a more accurate model. Since the slope field and the vegetation type are also important parameters of the simulation model [11], we can envision the integration with a GIS Web service to incorporate the topography of the field and mashing it to our system.

Another functionality that warrants further research is to allow the model to be dynamically modified according to the information reported by the weather service on the wind speed and direction. This work is currently in progress and the functionality should be available soon.

References

[1] Fire Simulation Mash-up (Demo Version). [Online] available at http://isel.cs.unb.ca/~michelv/. 2008.

[2] A. Muzy, G. Wainer, E. Innocenti, A. Aiello, J-F. Santucci, "Dynamic and discrete quantization for simulation time improvement: fire spreading application using the CD++ tool". in *Proceedings of 2002 Winter Simulation Conference*, San Diego, U.S.A. 2002.

[3] A.M. Grishin, "Mathematical modeling of forest fires and new methods of fighting them". Publishing house of the Tomsk state university. Albini Ed. pp. 81-91. 1997

[4] R. Rothermel, "A mathematical model for predicting fire spread in wild-land fuels". Research Paper INT-115. Ogden, UT: U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station. 1972.

[5] G. Wainer, "Applying cell-DEVS methodology for modeling the environment", in *SIMULATION*, October 2006, Vol. 82, No. 10, pp. 635-660.

[6] G. Wainer, N. Giambiasi, "N-dimensional Cell-DEVS models". Discrete Event Dynamic Systems. Springer Netherlands. ISSN 0924-6703. Vol. 12. No. 2. 2002.

[7] B.P. Zeigler, H. Praehofer, T.G. Kim, *Theory of modeling and simulation:Integrating discrete event and continuous complex dynamic systems*, Second Edition, Academic Press, 2000.

[8] M. Vasconcelos, J. Pereira, B. Zeigler, "Simulation of fire growth using discrete event hierarchical modular models" in *Advances in Remote Sensing*, EARSeL, 1995, Vol. 4, No. 3, pp. 54-62.

[9] L. Ntaimo, B. Khargharia, B. Zeigler, M. Vasconcelos, "Forest fire spread and suppression in DEVS" in *Simulation, Transactions of the SCS*, 2004, Vol. 80, No. 10, pp. 479-500.

[10] J. Ameghino, A. Troccoli, G. Wainer "Models of complex physical systems using Cell-DEVS" in *Proceedings of Annual Simulation Symposium*, 2001, Seattle, WA, U.S.A..

[11] G. Wainer, "CD++: a toolkit to define discrete-event models" in *Software, Practice and Experience*, Wiley, November 2002, Vol. 32, No. 3, pp. 1261-130.

[12] R. Madhoun, B. Feng, G. Wainer, "Web-Service-Based Distributed CD++" in *Artificial Intelligence, Simulation and Planning*, AIS 2007, Buenos Aires, Argentina.

[13] WebserviceX.net [Online] available at http://www.webservicex.net

[14] Google Map API, [Online] available at http://code.google.com/apis/maps