Proceedings of the 2011 Winter Simulation Conference
S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, eds.

ARCHITECTURE FOR INTEGRATED MODELING, SIMULATION AND VISUALIZATION OF ENVIRONMENTAL SYSTEMS USING GIS AND CELL-DEVS

Mariano Zapatero
Rodrigo Castro

Departamento de Computación
FCEN - Universidad de Buenos Aires.
Ciudad Universitaria, Pabellón I.
(1428) Buenos Aires. ARGENTINA

Gabriel Wainer
Maysoun Houssein

Dept. of Systems and Computer Engineering
Carleton University. Centre of Visualization
and Simulation (V-Sim).
1125 Colonel By Dr. Ottawa, ON. CANADA

ABSTRACT

Online Geographic Information Systems (GIS) and their associated data visualization technologies are playing an increasingly important role in providing updated information for environmental models. The analysis of simulation results are often benefited from their georeferenced animated visualization. We present an architectural web-based integration of the DCD++ distributed modeling and simulation framework as the centerpiece of a GIS-based scientific workflow to study environmental phenomena. We demonstrate an end-to-end application of the proposed architecture by means of a wildfire spreading model, backed by online updates of different parameters affecting the environmental system under study. Google Earth and GRASS are the two GIS systems selected to highlight the flexibility of the integrated system.

1 INTRODUCTION

Many applications currently make use of modeling and simulation techniques for environmental applications. Many of them rely on geographic data such as terrain elevation, land use, climate, etc. There is a variety of specialized software for handling this type of data, known as Geographical Information Systems (GIS), which allow manipulating georeferenced information and performing different operations with maps. GIS are usually organized in multiple layers of data, and they are currently an indispensable tool for the scientific analysis of regional characteristics, in particular in the field of environmental applications. These systems can centralize all the environmental data available and make it accessible in several forms such as image maps, digital maps or raw data files.

Therefore, the simulations of processes that use environmental data as inputs can be very benefited if the data provided comes directly from an updated GIS, making the integration between GIS and the simulator a goal to be achieved. Although many GIS already contain embedded simulation capabilities, the reduced sets of available models are closed and the power of the simulator is constrained by the host computers (usually a PC) without capabilities for scaling up.

On the other hand, simulators that run separated from GIS tools provide precise results but they fall short in expressing them into the appropriate georeferenced context. This limitation can be overcome by porting raw simulation results into the corresponding georeferenced maps. In this way, many sources of information can be associated, even if they do not come from the simulation model, providing a much more powerful analysis experience.

The goal of this work is to develop an integrative architecture to orchestrate a general-purpose distributed simulator (DCD++, Madhoun 2008) with geographic information systems and data visualization technologies, to extract data simulate behavior and analyze results supporting the decision making for varied environmental scenarios. The simulation model is fed by a GIS application providing updated data, while the model design process enables integrating additional information layers in a straightforward way.

The task of analyzing simulation results is benefited from displaying them on an appropriate Geospatial Visualization System (GVS): the environmental phenomena being studied can be viewed in context with aggregated data about near relevant elements and hence potential environmental impacts can be evaluated more thoroughly. We chose Google Earth (Google Inc. 2011) as the GVS for evaluating simulation results. This choice was based on the popularity and ease of use of Google Earth, which can enhance the communication of results to non-technical users, providing instantaneous access to a database of information layers.

The concept behind the proposed architecture is to integrate: a) systems providing georeferenced annotated data with b) simulation models with behavior sensitive to that information (i.e. topology, vegetation, buildings, etc.), c) distributed simulation infrastructures and d) powerful visualization systems.

To demonstrate this concept, we introduce the integration of an open source GIS (GRASS - Geographic Resources Analysis Support System; GRASS GIS 2011), DCD++ as the simulation tool for environmental models based on the Cell-DEVS modeling paradigm (Wainer and Giambiasi 2002), and Google Earth as the GVS.

We introduce new interfaces for supporting this process and prove their usefulness by obtaining inputs from GRASS and simulating a wild forest fire, visualizing results in Google Earth. The new application provides a single and automated environment for the end-to-end process. The fire model uses environmental conditions to compute the ratio and the intensity of spread in any direction. The results obtained show how such an architecture can be used for environmental systems applications.

2 BACKGROUND

DEVS (Discrete-Event System Specification, Zeigler 2000) is a mathematical formalism for specifying discrete events systems using a modular description. A DEVS model is seen as composed of atomic submodels than can be hierarchically combined into coupled models. Cell-DEVS is a extension to DEVS that provides the capability to define cellular N-dimensional models as a composite of atomic cells, where each cell is defined as an atomic DEVS (see Figure 1), and defines a procedure to couple cells evolving the state of the influenced neighbors.

CD++ (Wainer 2002) is an open source environment capable of executing DEVS and Cell-DEVS models. It was originally designed as a standalone application and then evolved into DCD++, a parallel and distributed version that runs on networked clusters. DCD++ and its associated the architecture are shown in Figure 2. Because DEVS separates strictly models from simulators, and CD++ separates strictly simulation from the supporting hardware, all the basic concepts for simulating Cell-DEVS models are shared between CD++ and DCD++.

The CD++ simulator reads Cell-DEVS models from a *model file* (*.ma*) and optionally fills the cells at the initialization step with the content from an *initial values file* (the *.val* file). CD++ executes using message-driven simulation, and handling two classes of messages: control and content. Each of this has a timestamp associated. Control messages include: initialization (I), collect (@), internal (*) and done (D) which are used internally by the simulator. Content messages include: external -or input- (X) messages and output (Y) messages, which encode the data transmitted between atomic models. As in Cell-DEVS each individual cell is an atomic DEVS models, Y messages carry the information of each cell state-change for updating its neighbors, and they are the messages that allow reproducing exactly the results of a full simulation after it is run.

RESTful-CD++ (Al-Zoubi 2009) is a general middleware that exposes the simulator services as uniform resource identifiers (URIs) following the REST (Representational State Transfer) principles (Field-

ing 2000). It provides several services handled via HTTP methods such as authenticate, create and monitor simulations, send model and initial value files as inputs, and retrieve results from the simulator. RESTful-CD++ acts as a web wrapper to support DCD++, providing an easy way to run simulations in facilities accessible from Internet. It consists of two major components: a web service responsible for the functionality that wraps the functionality of the underlying simulator, and the DCD++ simulator itself, which is responsible for running the models and interacting with the web service component.

The integrator components are built on top of the RESTful web service, and are integrated with the web service components without introducing major changes into either the restful web service component or the underlying simulator software. This modular approach for developing the integration makes it easy to extend, simple to maintain, and flexible to mash up with other web-service based information systems.

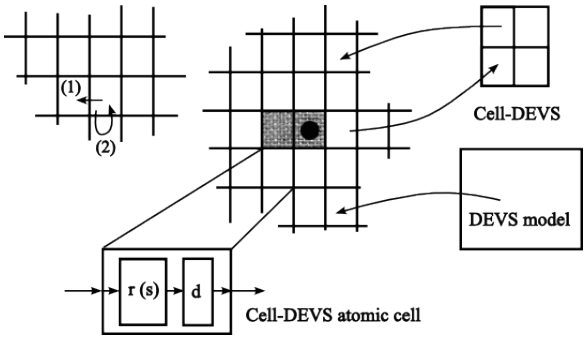

Figure 1: Informal definition of Cell-DEVS

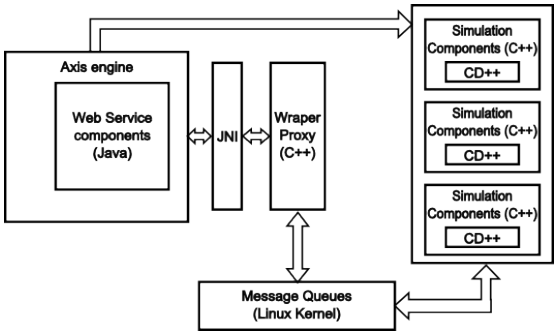

Figure 2: DCD++ Software Architecture

In Geographic Information Systems (GIS), spatial data is organized and presented graphically (maps, graphs, etc.) or as files with different formats. Some GIS allow obtaining information from outside the system using public APIs and, similarly, to incorporate information across different file formats. GRASS is a project of the Open Source Geospatial Foundation (OSGeo Foundation 2011) and is one of the most popular and widespread GIS. GeoTIFF (OSGeo Foundation GeoTIFF 2011) is an open standard developed by several organizations to establish a TIFF based interchange format for georeferenced raster images. GeoTIFF is also used to link raster images to known model spaces or map projections, and for describing those projections (OSGeo Foundation GeoTIFF Spec 2011).

Google Earth is a Geospatial Visualization System (GVS) that has evolved significantly in recent years reaching a massive use of large communities for both scientific and generic purposes. In addition to integrating data from different sources, Google Earth can add its own data (usually provided by its users' community) achieving significant coverage regarding geographical data, satellite images, photographs, statistics, ground infrastructure, etc. Google Earth uses KML (Keyhole Markup Language, OGC 2011) files, an open standard from the Open Geospatial Consortium. KML is an XML-based language focused on geographic visualization that includes annotation of maps and images. Its file format can be used to display geographic data in Earth browsers (such as Google Earth), using a tag-based structure with nested elements and attributes. The geographic visualization includes not only the presentation of graphical data, but also the control of the user's navigation (e.g., where to go, where to look).

Many GIS provide a range of operations to manipulate and process information that Google Earth does not provide. However, most of these operations -once performed within the GIS- produce data that can be exported into Google Earth. Considering its popularity, usability and variety of community-driven data available, Google Earth has been chosen as our GVS architectural component.

2.1 DEVS as the underlying framework for environmental systems modeling and simulation

Many previous efforts from the research community have presented tools and methodologies for modeling and simulation of environmental systems. The most widely adopted modeling paradigms are Cellular Automata (CA) and Agent-Based Modeling (ABM) [See Wainer (2006) and Castle and Crooks (2006) for a non-comprehensive review of tools for CA and ABM, respectively].

DEVS provides a combination of features that makes it a unique technology alternative for environmental modeling and simulation. The most salient features are paradigm heterogeneity, hierarchical structure and separation between models and simulators (Zeigler 2000).

Paradigm heterogeneity allows DEVS representing portions of large complex systems in a modular way adopting the most convenient paradigm for each subsystem. The Cellular Automata (CA) is one of such paradigms providing a convenient metaphor for representing spatially-distributed phenomena as a lattice of cells with dynamic behavior. Cell-DEVS is an extension of the classic CA paradigm implemented under the DEVS formalism, equipping CA with discrete-event capabilities and a more flexible handling of time (time is not constrained to fixed steps). Thanks to DEVS heterogeneity, additional subsystems modeled can be defined and interconnected with the CA portion of the system following (per instance) the Agent-Based Modeling (ABM) paradigm. Nevertheless, existing tools do not permit the sound and simultaneous mixing of both paradigms.

Furthermore, DEVS permits defining complex behavior as a form of emergent behavior produced by the interactions among simpler subsystems. Any subsystem belongs to a hierarchical structure, being at the same time a composition of finer grain subsystems and also a component of larger systems. DEVS also provides a self-contained and mathematically sound specification of models with strict separation from any simulation technology or computer language. This feature allows the model continuity of the studied environmental systems, fostering their reuse and incremental refinement by different research teams independently from the simulation technologies that might change across teams and time.

Thus, taking into account the characteristics listed above, interfacing DEVS-based simulations with GIS by means of an extensible software architecture shall offer a flexible platform for studying environmental systems fostering interdisciplinary modeling and simulation.

3 INTERFACING CELL-DEVS SIMULATIONS AND GIS

In this section, we explain the proposed architecture, in which we import information from a GIS, use it to tune the parameters of a Cell-DEVS model, simulate it, and display georeferenced results. We briefly discuss the various systems involved and their relevant characteristics.

As we can see in Figure 3, the architecture includes a component *Integrator* with five modules: *Orchestrator* (making the process flow through its different stages), *UserInterface* and *SimulatorCommunication* (to interact with the users and with RESTful-CD++ respectively), *GIS2CD++* (to retrieve data from the GIS), and *CD++2GIS* (to visualize the simulation results).

The *Orchestrator* coordinates activities such as retrieving data from the GIS (either via an API or import/export files), communicating with the simulator (through RESTful-CD++), and transforming the outputs into KML files fed into Google Earth. Google Earth allows adding custom information to be displayed along with previously available data (from different sources, in particular from other GIS). This can be done through georeferenced graphic files, in particular GeoTIFF files (where image layers are overlapped) or KML files (specified as vectors) that define different spatial geometries and placemarks (possibly with extra data associated).

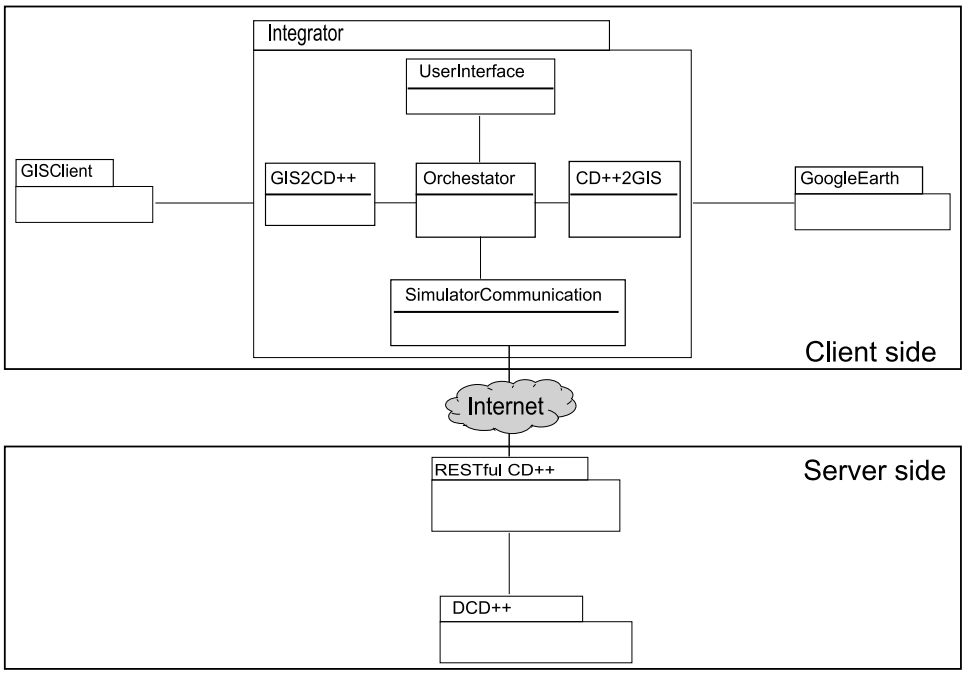

Figure 3: GIS-Simulation Architecture

This organization can connect to local or remote services and allows running simulations using local data (stored in the client computer, such as digital elevation models, georeferenced files, etc.) or any remote repository. This also gives the possibility to take advantage of more powerful infrastructure to run the simulation transparently. Google Earth allows analyzing possible future scenarios and their possible consequences thanks to the contextualization of the simulated data with information from annotated reality (urban areas, natural reserves, nuclear plants, etc.).

The *Integrator* first retrieves all the data required for the simulation from the GIS (e.g. the initial values for the land use). Then, *GIS2CD++* uses this information, adapting it for visualization purposes. *GIS2CD++* also receives the Cell-DEVS model to be simulated. With this information, the *Orchestrator* sends the model file and auxiliary data to the Server through RESTful-CD++, launches a remote simulation, and receives the results. *SimulatorCommunication* handles the sending of HTTP messages for different purposes: creating a remote workspace, receiving files from *Orchestrator* and hand them to the server, monitor the simulation evolution and getting results from the server (passing them to the Orchestrator).

Once the simulation finishes, the Orchestrator receives the results from RESTful-CD++, and it sends them to *CD++GIS* to transform them into a file that can be incorporated into a GIS (in our case, Google Earth using KML). In this way, we can integrates DCD++ with a GIS for obtain updated data and Google Earth to display environmental models simulations in a simple and efficient way, providing both the process to carry out simulations, the analysis of their results as well as allowing even more complex models to have available all the functionality of GIS. The Sequence Diagram in Figure 4 summarizes the process just discussed.

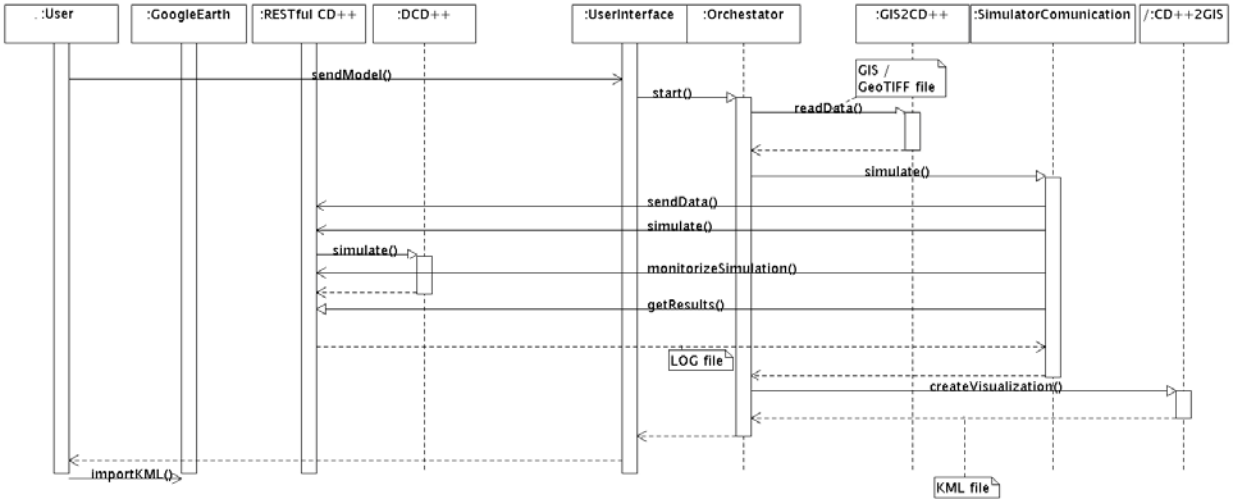

Figure 4: Sequence Diagram for the End-to-End Process

3.1 Integrator

The *Integrator* is the main component of the application's high-level architecture, as it integrates the modules providing different features such as communication with other applications, interaction with users, orchestration of sequential activities, logging and error handling for all stages. The Integrator is composed by the following modules:

1. *GIS2CD++*, which is responsible for reading and preprocessing the input data from the GIS
2. *Orchestrator*, which is responsible for the sequencing of steps to carry forward the process
3. *SimulatorCommunication*, which handles all interaction with the simulator
4. *CD++2GIS,* which is responsible for processing the simulation results obtained from the simulator and generating their display information
5. *UserInterface* concentrates all the interaction with the user

3.2 GIS2CD++

This module is responsible for loading data into the application, and transforming it into files understandable by the simulator. To do this, we defined modules for dealing with different standard input file formats.

The ideal solution for GIS data access is to use its native API to access all its exposed information and functionality. Since each GIS has its own peculiarities in both information organization and structure (and supporting file formats), we decided to implement the logic in a generic class that does not assume anything about the format of the information, and propose an interface to use specialized methods that rely on specific GIS. Thus, for each particular GIS a new class must be provided that implements the abstract methods processing the information according to the format used by that system.

As not all GIS have a public API, we need an alternate method for retrieving information. We used a common method to share information, using georeferenced as well as a data input procedure to the Integrator. Though this option is more limited than using APIs, it offers a generic approach that requires no previous knowledge about the GIS tool. We chose GeoTIFF as the standard file format, as it is supported by most GIS, including GRASS (Neteler2010), for both input and output purposes.

Besides the georeferenced contextual data, the simulation model itself has to be provided. The idea is to minimize the efforts required to make a model sensitive to the georeferenced data. To do this, further processing is necessary to create an enriched version of the model, with parameters sensitive to the properties of the given geographical zone. This processing includes transforming the maps obtained to the

format used by the simulator as initial data values (*.val* file), the final Cell-DEVS model (*.ma* file), and an information file (*.info* file) where global geographical references are kept.

Once these files are created, the Orchestrator sends them to the simulator and the simulation process is performed. The following describes the implementation of this module, whose class diagram is shown in Figure 5.

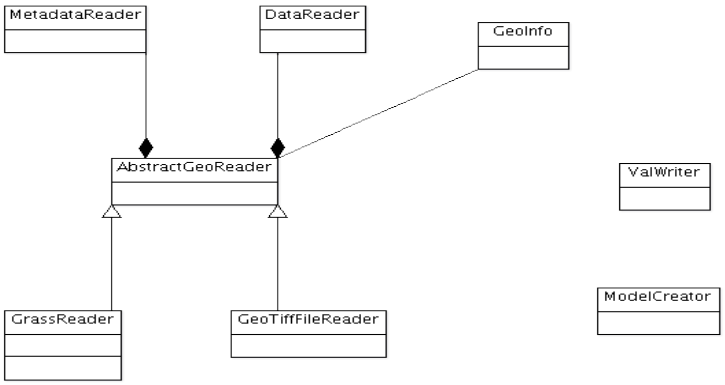

Figure 5: *GIS2CD++* class diagram

AbstractGeoReader implements the general logic of the data input through two subclasses: *MetadataReader*, whose responsibility is to obtain geographical references and *DataReader* to get data on each of the cells. These classes do not make any assumptions on the input data format, but provide an interface through certain abstract methods that must be implemented by subclasses.

This structure can be used to add new file formats and classes to connect directly to other GIS, a geographic database provider or any other source of information easily. This is because, although each source of information keeps data in various formats, the main logic of data processing and model files creation for the simulator does not change. For instance, to incorporate a new GIS, a new class should be added providing the correct geographical references and values for each location (i.e., a class that extends *AbstractGeoReader* and implements the abstract methods).

The *GeoTiffReader* class extends the *AstractGeoReader* class, implementing these abstract methods to obtain information from georeferenced files in GeoTIFF format. *GrassReader* class does the same, retrieving the data through the GRASS API.

GeoTiffReader implements these operations through the GDAL (GDAL 2011) library for raster geospatial data formats, including techniques for efficient handling of large files.

GeoInfo is a convenience class for storing geographic contextual data such as coordinates and the resolution of the area on which to work.

ValWriter class is responsible for generating the cell's initial values file (.val) from data obtained by the *DataReader* class. This process is absolutely decoupled from the input file format.

Finally, *ModelCreator* class is responsible for creating the model file containing the structural and behavioral information to run the simulation, along with references to other files generated.

Figure 6 describes the interaction between the various classes.

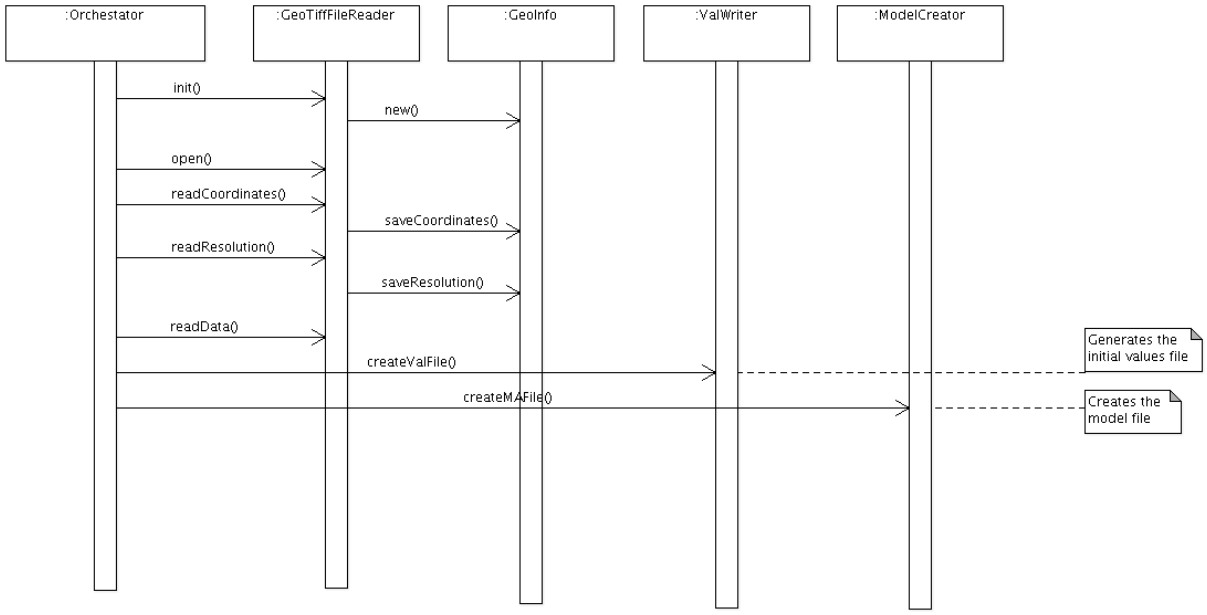

Figure 6: Interaction inside *GIS2CD++*

3.3 Orchestrator

The *Orchestrator* class is responsible for sequencing the stages for preprocessing information, the generation of files for the simulator, the communication with the simulator and the transformation of results into a displayable format. At each stage, it performs a series of checks to validate the proper execution of the previous stage. For instance, before sending the files to the simulator, *Orchestrator* verifies all the files created. The detection of errors triggers the stopping of the execution and/or their report for debugging purposes. As all the other classes implemented, it keeps a sequential record of the operations performed.

In addition to establishing and releasing the communication transparently, the Orchestrator provides the simulator with required files, launches and monitors the simulation, and receives the simulated results. All the communication with the simulator is done in this module, encapsulating this functionality. This is an enabler for creating mashable web-based applications where results coming from heterogeneous sources are mashed-up producing a unified result. The strict isolation of the simulation mechanisms from the results manipulation procedures allows modifying both layers at will without breaking the integration.

3.4 Communication with RESTful-CD++

The communication with the simulator infrastructure is done through RESTful-CD++, which accepts HTTP messages using REST principles, processes them, and forwards action commands to the simulator to satisfy the services requested by the messages. This way, the simulation facility is made available online.

The first step requires registration of the client side, thus the *Orchestrator* provides a username and a password. Upon completion of the login step, a session and a workspace are created into the server that will uniquely identify a set of simulations. Next, the Orchestrator sends the model file and the cell initial values file to the server side so the simulation can be executed.

While the simulator is running the *Orchestrator* monitors the progress of that activity, and at the end of the simulation, retrieves the log file generated. That log file will be used as an input for the *CD++2GIS* module.

3.5 CD++2GIS

All the information produced by the simulator must be interpreted and transformed to be displayed back into a GIS and/or GVS environment. Here, we discuss the architecture for this output stage, and then a particular implementation based on Google Earth using KML files.

For visualization purposes, the relevant information that is generated by changes in the cell values are represented by output messages in the simulation. Therefore, CD++2GIS first parses the data generated by the simulator preserving output messages that represent state changes. This responsibility falls on the *LogParser* class, which stores the changes into the supporting structure class *LogInfo,* facilitating further processing. The next step is to translate *LogInfo* into the desired output visualization format. This is a responsibility of *AbstactGeoWriter* class, an abstract class providing an interface to translation methods (analogous to the *AbstactGeoReader* class discussed in Section 3.3 for GIS2CD++). Our case study focuses on the generation of KML files by means of the *KMLCreator* class (see Figure 7).

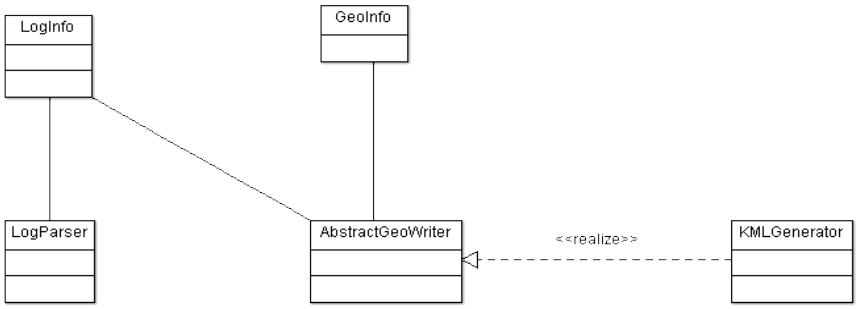

Figure 7: *CD++2GIS* class diagram.

KMLCreator takes this structure (*LogInfo* class) and the *.info* file with the georeferences, processes them and generates a KML file with georeferenced and timed representation of each simulated cell state change. The process consists in translating each output message into KML tags. Once the KML file is created, it can be imported into Google Earth allowing the visualization of the simulated results evolving on a layer impressed over standard layers (e.g. satellite views, street maps, etc.). Google Earth provides mechanisms to make layers evolve forward and backward in time, which is useful to analyze the progress of a simulation interactively (e.g., a cell temperature in a fire spreading model) pausing it at any point.

The classes that implement the translation process of log files into the KML format are described in Figure 8. The *Mapping* class is responsible for identifying all the points in a simulated area and for identifying which ones are POI (Points Of Interest); e.g. the starting coordinate is a POI, and each time a new cell experiments a state change it becomes a new POI (a placemark is also defined for that place). This class is responsible for identifying the borders of the area under study. The *Mapping* class then calls the *KmlTagCreate* class, which creates one tag for each point on the map along with its description, and if the point is a POI, the class sets a placemark for it.

The KML Converter features several checking points, ensuring the quality of the files generated, logging errors if any (to improve debugging tasks).

Finally, *KmlCreator* writes the physical KML file and is also responsible for completing the KML file correctly (even when intermediate errors are trapped). This procedure generates geographical areas that emulate those simulated in cellular spaces of a Cell-DEVS model.

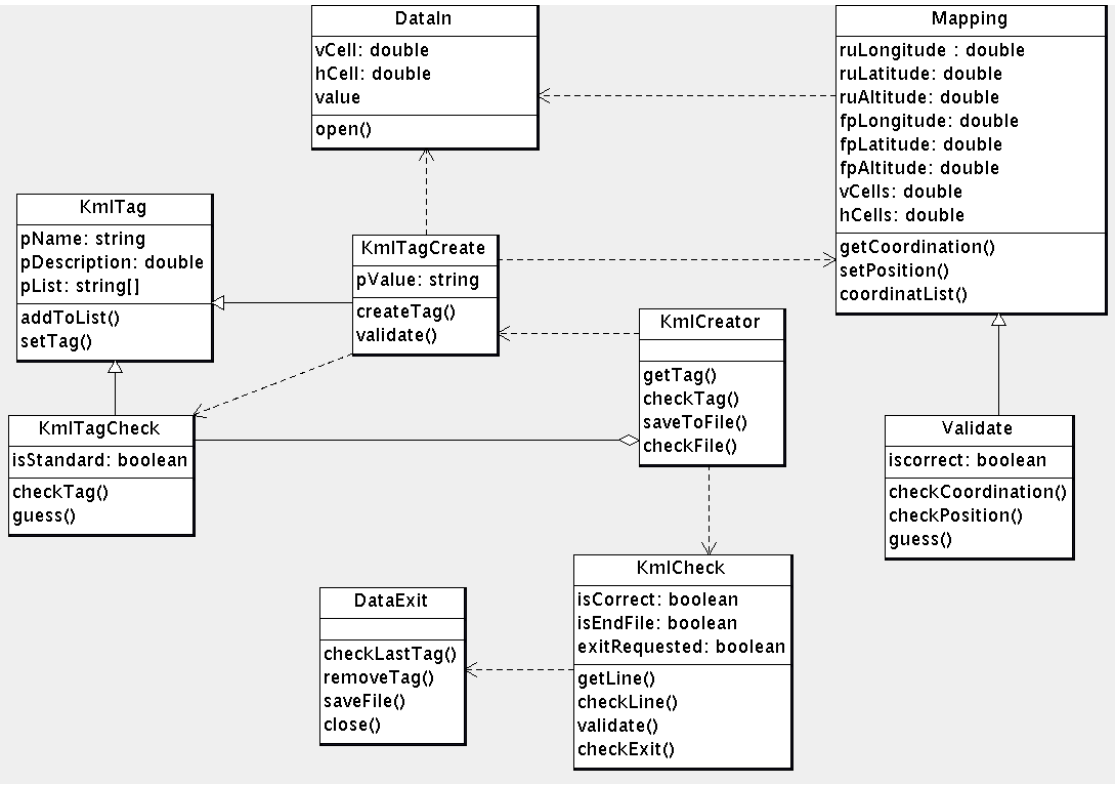

Figure 8: KML Generator class diagram

4 CASE STUDY: FIRE MODEL SIMULATION

For our testing environment, we use the RESTful CD++ to run a model developed to study the spreading of fires in forests (Wainer 2006, Ameghino 2001). The model is designed using Cell-DEVS, which provides the expressive power for imitating phases of the actual behavior of forest parcels in response to external events like heat or the effect of wind on the area, among other external parameters.

Different parameters affecting the fire spreading come from the weather state. The following ones are taken into consideration to calculate the ratio of spreading for the fire:

1. particles properties (amount of heat, minerals and density)
2. type of fuel (includes the size of the vegetation)
3. natural environment (wind speed, land use, slope, humidity)

In this model, no fire suppression or containment will be considered. To feed real data into the model, we chose a GRASS GIS with the sample dataset "North Carolina, USA" (a dataset that comes readily available with the standard version of the tool). We then select a section of the *landuse96_28m* map because it has two water surfaces surrounded by flammable land. This is useful to illustrate the differences in fire propagation according to the type of surface, evidencing the sensitiveness of the model to the land use information provided by GRASS. The expected simulation result is the propagation of fire through the land and not over water. Figure 9 shows the entire initial map and the subarea selected for simulation.

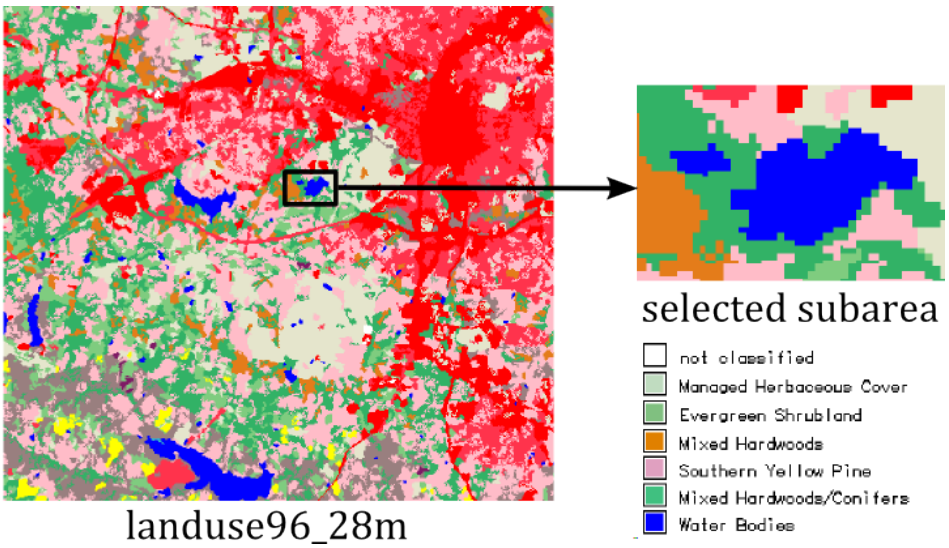

Figure 9: Land-use map (left) and selected subarea for fire spread simulation (right)

The simulation is executed over a DCD++ environment and the result of processing the resulting log file generating a KML output file as shown in Figure 10.

```
[...]
<Placemark>
  <styleUrl>#fuego</styleUrl>
  <description>TIME=00:01:11:973 CELL=37,24 VALUE=11.19956</description>
  <TimeSpan>
    <begin>2011-06-13T00:00:00.000</begin>
    <end>2011-06-13T00:01:11.973</end>
  </TimeSpan>
  <altitudeMode>relativeToGround</altitudeMode>
  <Point>
    <coordinates>-78.686443747576374,35.763245589852001,0</coordinates>
  </Point>
</Placemark>
<Placemark>
  <styleUrl>#fuego</styleUrl>
  <description>TIME=00:02:23:946 CELL=36,25 VALUE=12.39912</description>
  <TimeSpan>
    <begin>2011-06-13T00:01:11.973</begin>
    <end>2011-06-13T00:02:23.946</end>
  </TimeSpan>
  <altitudeMode>relativeToGround</altitudeMode>
  <Point>
    <coordinates>-78.686339196403981,35.763350141024394,0</coordinates>
  </Point>
</Placemark>
[...]
```

Figure 10: portion of generated KML file.

This KML file is then loaded into Google Earth obtaining the following results shown in Figure 11. The simulation was run for analyzing the spreading of fire during 01:00:00. It can be easily seen that fire propagates around the lakes as expected, effectively taking into account the land use information obtained automatically through GRASS. It can also be seen and that given the wind direction adopted, the large

urbanized areas at the bottom-right are not compromised, but the ones at the middle-top start are affected by fire. 01:10:00.

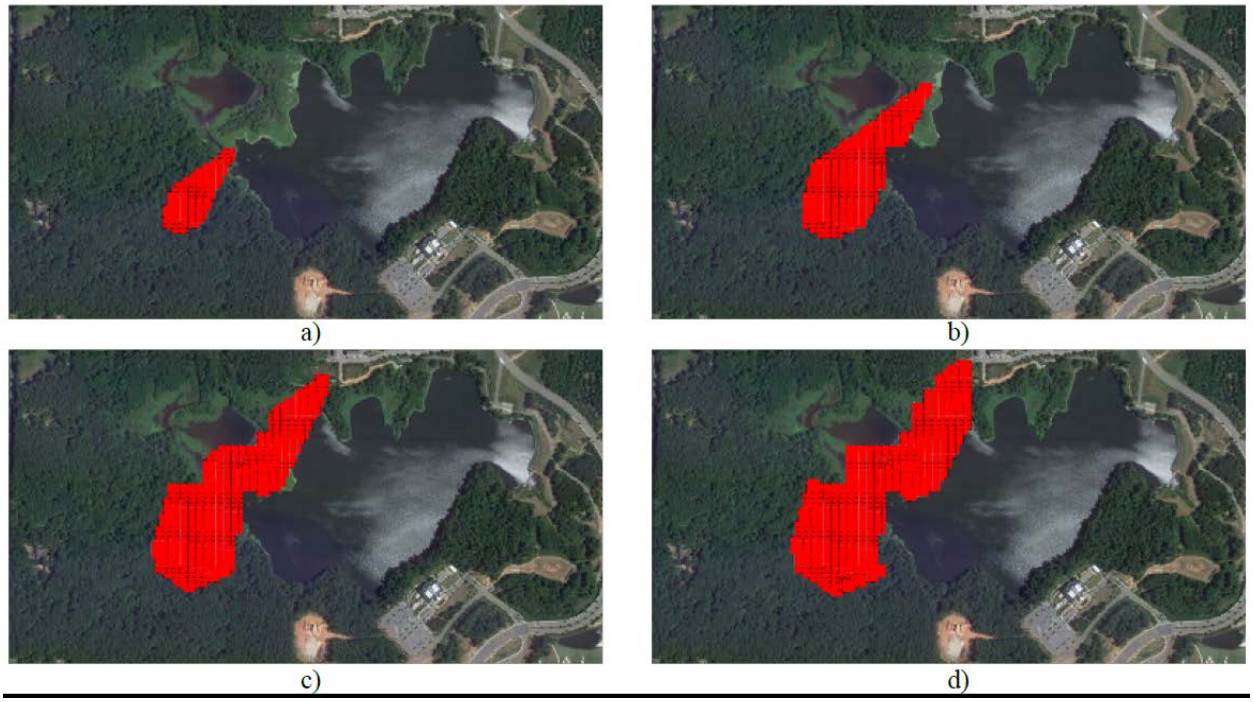

Figure 11: Fire model simulation results shown in Google Earth.
Snapshots taken at times a) 00:20:23 b) 00:34:47 c) 00:54:32 and d) 01:00:00

5 CONCLUSIONS

The proposed architecture efficiently helped integrating previously existent tools for modeling, simulation and visualization of environmental systems. With minimal modifications, a fire spread model was made sensitive to parametrized land use information, and was used in an automated process retrieving real-world land use information from an open source GIS tool (GRASS), simulating the model on a selected area, and displaying results in a publicly available Geographical Visualization System (Google Earth).
We can then take advantage of Google Earth's multi layered feature to overlap simulation results over large amounts of visually rich geographical information supplied by satellites and the user's community.
The new automated process greatly reduces efforts for repetitive and error-prone tasks such as file handling, transformation and communication. The DCD++ distributed and web-enabled framework for modeling and simulation was effectively used as the centerpiece of the GIS-based scientific workflow helping studying and environmental phenomenon. GRASS and Google Earth proved being excellent choices offering flexible input/output interfaces. Yet, the architecture obtained is flexible enough to incorporate new tools for both geographical data retrieval and animated georeferenced simulation visualization. The clear decoupling of the geographical data handling modules from the web-enabled distributed simulation facility makes this architecture highly scalable, and suitable for mashing up with other sources of information to produce more complex and visually rich results.

REFERENCES

Al-Zoubi, K., Wainer G. 2009. "Using REST Web-Services Architecture for Distributed Simulation". In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed*

Simulation (PADS '09). IEEE Computer Society, Washington, DC, USA, 114-121, Lake Placid, NY.

Ameghino, J., Troccoli, A. and Wainer, G. 2001. "Models of Complex Physical Systems Using Cell-DEVS". In *Proceedings of the 34th IEEE/SCS Annual Simulation Symposium* (ANSS'01), Seattle, WA: 266-273.

Castle, C.J.E. and Crooks, A.T. 2006. "Principles and Concepts of Agent-Based Modelling for Developing Geospatial Simulations" UCL Working Paper Series. Paper 110. Centre for Advanced Spatial Analysis. University College London.

Fielding, R. T. 2000. "Architectural Styles and the Design of Net-work-based Software Architectures". Ph.D. thesis, University of California, Irvine. Available via http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm Accessed June 2011.

GDAL. 2011. "Geospatial Data Abstraction Library." Accessed June 2011. http://www.gdal.org/

Google, Inc. 2011. "Google Earth." Accessed June 2011. http://earth.google.com

GRASS GIS. 2011. "Geographic Resources Analysis Support System." Accessed June 2011. http://grass.fbk.eu/.

Madhoun, R., K. Al-Zoubi and G. Wainer. 2008. "Distributed simulation of DEVS and Cell-DEVS models in CD++ using Web-Services". *Simulation Modeling, Practice and Theory*, 16 (9), 1266–1292.

Neteler, M. and H. Mitasova. 2010. *Open Source GIS: A GRASS GIS Approach*, 3rd. Edition. Springer Publishing Company, Inc.

OGC. 2011. "KML." Accessed June 2011. http://www.opengeospatial.org/standards/kml

OSGeo Foundation GeoTIFF. 2011. "GeoTiff.". Accessed June 2011. http://geotiff.osgeo.org.

OSGeo Foundation GeoTIFF Spec.Spec. 2011. "GeoTIFF Format Specification" Accessed June 1. http://www.remotesensing.org/geotiff/spec/geotiffhome.html.

OSGeo Foundation. 2011. "The Open Source Geospatial Foundation." Accessed June 2011. http://www.osgeo.org/

Wainer, G. 2002. "CD++: A Toolkit to Develop DEVS Models". *Software: Practice and Experience*, 32 (13), 1261-1306.

Wainer, G. 2006. "Applying Cell-DEVS methodology for modeling the environment." *Simulation*, 82 (10): 635-660.

Wainer, G. 2009. *Discrete-Event Modeling and Simulation: a Practitioner's approach*. CRC Press. Taylor and Francis.

Wainer, G. and N. Giambiasi. 2002. "N-dimensional Cell-DEVS Models". *Discrete Event Dynamic Systems*, 12 (2): 135-157.

Zeigler, B.P., H. Praehofer and T. Kim. 2000. *Theory of modeling and simulation*, 2nd Edition, Academic Press.