18th World IMACS / MODSIM Congress, Cairns, Australia 13-17 July 2009
http://mssanz.org.au/modsim09

A Modular Spatial-Temporal Modeling Environment for GIS

Marchionni, B. [1], **Ames, D.** [1], **Dunsford, H.** [1]

[1] *Geospatial Software Lab, Department of Geosciences, Idaho State University, Idaho*
Email: marcbria@isu.edu

Development of an open source modeling environment for use with spatial-temporal data in a Geographic Information System (GIS) is presented. MapWindow GIS, a free and open source desktop GIS, has been used extensively in watershed modeling and is the underlying engine of the U.S. EPA BASINS system.

To date, legacy versions of MapWindow have lacked an integrated modeling environment suitable for linking together geospatial and temporal independent processes at a granular level. New developments in the open source MapWindow GIS 6 project have created the basic framework for an extensible modeling environment. This new modeling environment allows users to easily create models which can take advantage of spatial and temporal data objects and analytical tools.

The design approach involves the extensive use of interfaces which are essentially skeleton programming tools that detail how an object programmatically interacts with other objects, but not necessarily how it works internally. By using interfaces, the new MapWindow GIS modeler makes it relatively simple to take existing modeling processes, wrap them in an appropriate interface, and execute them as part of a more complex model. By using interfaces any one component of the modeler can be replaced by any other object that implements the same interface. The central underlying design consideration of the newest version of MapWindow GIS was to keep the entire project as modular as possible.

The developed modeler uses a simple interface to define how processes or tools should interact with other model components. The tool interface requires developers to specify a number of simple methods on their tool's object which are then called by the model when the tool is loaded. Developers need not spend time designing the user interface for their tool as it is automatically generated by the modeler when their tool is instantiated. Because all tools in the MapWindow modeler must implement the same interface, developers wishing to use a tool directly in their own application need not add the modeler if they do not so desire. By adding a reference directly to the tool they want to use they can gain access to all of the same methods that would be exposed to them if they worked with it through the modeler.

MapWindow GIS 6 and the modeler are entirely developed using the Microsoft .NET Framework which allows it to be run on a variety of operating systems including Windows, Linux or OS X (via the Mono compiler).

Keywords: *GIS, Interface, Model, GUI, MapWindow, Spatial, Temporal*

1. INTRODUCTION

The goal of the project described here was to develop a modeling environment for the next generation of the MapWindow project, MapWindow GIS 6. MapWindow GIS 4 is the current version of the project and is under continued development at Idaho State University in the Department of Geosciences. MapWindow GIS 5 was a short-lived prototype project that was never released publicly. Originally developed at Utah State University, and now maintained primarily at Idaho State University with an international development team, MapWindow GIS is a free and open source software project that is downloaded over 6000 times per month. It has an active community of users and developers on the MapWindow.org web site. The community collaborates on making updates and introducing new features. The existing project is divided into two components: the MapWindow GIS desktop application, and the ActiveX map control. These two components work together to form the entire project. This modularity allows the ActiveX control to be used in other stand-alone applications as well as within the main MapWindow GIS desktop application. (Ames et al., 2007)

The need to develop a modeling environment arose from other developments in the GIS community. A general need to simplify the task of using spatial and temporal processes had been brought forward by many MapWindow users and by several other communities who are using the MapWindow components in their own projects. Furthermore, the use of modeling in a GIS environment has been suggested by several other researchers including Xie and Brown in their 2007 paper noting, “simulation in spatial analysis and modeling has been one of the key approaches of many researchers of GeoComputation.” (Xie and Brown, 2007)

Requirements for the developed modeler are tied to other developments in the MapWindow 6 development effort and include:

- all user interfaces need to be as simple to use and as well documented as possible;
- users should need no programming experience to use the project;
- the project must have high portability, software should work on many different systems including MS Windows, Linux and Macintosh OS X;
- the code needs to be highly extensible and reusable;
- the code should be easy to maintain for new developers.

The modeling environment should also be designed such that it can be integrated into other applications with minimal dependency on external libraries, including the MapWindow.dll itself. Since the design of MapWindow GIS 6 was well underway at the time of the modeling environment’s conception, it was decided that components and data types from this new architecture would be used because of the advantages that it afforded.

2. MAPWINDOW GIS 6

MapWindow GIS 6 is the next generation of the MapWindow project. Early in the planning stages of MapWindow GIS 6 it became apparent that the technology behind the original MapWindow ActiveX map component would not be capable of meeting all of the new project’s requirements. Specifically since the original code was written as a Microsoft COM object it could never be cross platform compatible. It was primarily for this reason it was decided that a complete rewrite of the map component would be required.

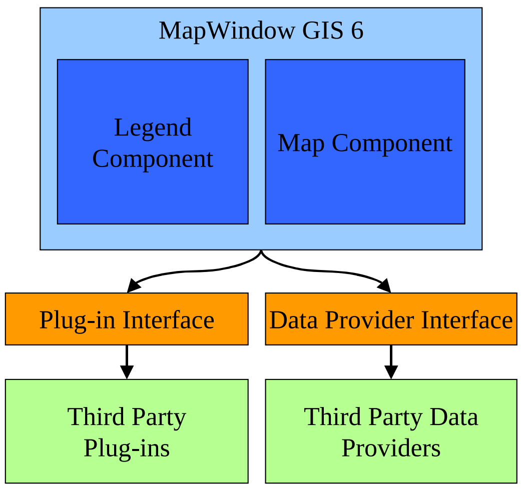


Figure 1. MapWindow GIS 6 Architecture

The design of the new architecture focused on an extremely modular system using class interfaces. Figure 1, highlights the interface architecture of MapWindow GIS 6. This design allows for any single component to be replaced by another that uses the same interface. This design stemmed from the successful plug-ins methodology from the original MapWindow GIS 4 that allowed third party developers to extend the functionality of the application by writing their own class which implements the plug-in class interface. (Ames et al., 2007)

3. THE MAPWINDOW MODELER PROJECT

3.1. Project Requirements

The MapWindow Modeler environment has been developed specifically to meet the requirements of several uses cases identified by the United States Environmental Protection Agency (EPA) Data for Environmental Modeling (D4EM) project. Some of the key requirements and constraints of the system are defined as follow:

- the tool should be written in Microsoft .NET so that it can be ported to Windows Mobile and Mono for Linux;
- the available tools and available data types should be extensible;
- the tool should integrate both spatial and temporal components;
- the tool should be easy to use for end users;
- the tool should be compatible with existing versions of MapWindow GIS;
- the tool should be robust and easy for new developers to add to and enhance.

Several existing open source projects were identified to see if they could meet the requirements for a modeling tool for MapWindow. Sextante (Olaya and Gimenez, 2007) meets all of the requirements, except that it is written in the Java language and hence would not meet the requirement of being written in Microsoft .NET. No other open source modeling products that were available were written in Microsoft .NET and could handle both spatial and temporal data interaction. The OpenMI system (Gregersen et al., 2007) was examined but it, too, failed to meet all of the requirements of the system as its scope went well beyond a simple graphical tool for link spatial and temporal processes.

3.2. Use Cases

There are three primary use cases for the modeling environment. The first covers the modelers' use while integrated into MapWindow GIS 6. In this mode a standard extension to the MapWindow GIS 6 desktop application will include the modeling environment. These two environments are tightly linked to allow data from the MapWindow GIS 6 map component to be added seamlessly from the modeler and conversely allow data from the map to be used in models. The second use case involves integration with the legacy code of MapWindow GIS 4. This is similar to integration with version 6 of MapWindow, but only a specific subset of the data will be made available to the MapWindow GIS 4 application because of format compatibility issues. The final use case covers using the modeler as a stand-alone component for use in third party applications. Since all possible uses of the modeler in other applications cannot be considered, the modeler must be as versatile and customizable as possible.

3.3. Software Design Technique

Since many different users and developers will be working with the system, the initial design specifications needed to be well defined at the outset. Once this initial design was completed a small group of developers created a set of simple tools to test the general design. It was at this stage that critical modifications were made to the design to address specific problems that developers and users were facing.

This approach of rapid prototyping allows for feedback from testers and other developers while ensuring a quick time to deployment. Initial development efforts took only six months. Once this critical initial development stage was completed the second phase continued until such time as all parties involved are satisfied with the resulting architecture. Once completed most major design considerations were done and the over all architecture finalized. While changes can still be made at this point they must take into account the existence of other dependant components that need to be integrated and cannot have their functionality impaired. For example, if a new version of an interface is created once this second stage of development is completed it must ensure that any components using the already existing interface must continue to function seamlessly.

This second stage of development is potentially the most important as it is feedback from developers creating tools that will ensure the ease of use of the interface for new developers wishing to create tools or data types for the system.

4. SOFTWARE DESIGN

4.1. Modeler Design

The MapWindow modeler is composed of two inter-related parts: the ToolManager and the Modeler. The ToolManager lists all of the available tools to the user while also providing access to tools in the Modeler. The Modeler displays, loads, saves, and executes models in a graphical environment.

The Modeler and ToolManager itself are actually .NET form component. Like other programming objects in the .NET environment it has a graphical representation that allows programmers to drag and drop it onto a form without writing any code. This greatly reduces the time needed for programmers to develop an application that uses the modeler. Figure 2 shows an instance of the ToolManager on the left displaying the tools it has found, and the Modeler on the right displaying a simple model containing one tool.

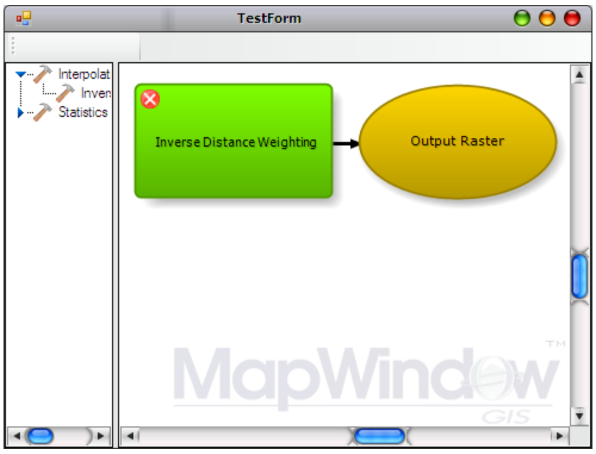


Figure 2. The ToolManager and Modeler running in Mono on Mac OSX

4.2. Building for Extensibility

Since the use cases for the modeler cover many different applications it was imperative that the modeler be designed such that it can be extended in several different ways, such as:

- tool definitions;
- parameter definitions;
- user interface representation of parameter definitions.

To allow each of these areas to be expanded upon, several programming concepts needed to be employed. These concepts are widely used through the architecture of MapWindow GIS 6 so programmers familiar with this environment can more easily add functionality to the modeler.

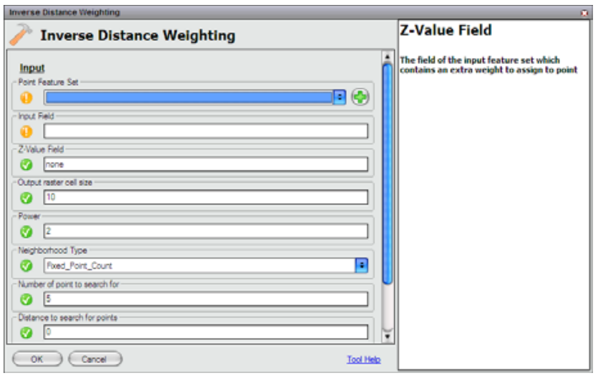
To accomplish this, a class interface was defined for tool definitions and parameter definitions called ITool and IParameter respectively. Using interfaces, blank class templates which programmers can populate with functions (Liquori and Spiwack, 2008), allows developers to rapidly develop software which implements the needed operations of software they are interacting with. (Greenberg, 2007) “A well-recognized method for reducing program complexity involves structuring the model as a set of distinct modules with well-defined interfaces.” (Maxwell, 1999) Since tools and parameter types can be generated in a variety of different ways, the Modeler never loads ITools or IParameters from disk directly; rather, it relies on a ToolManager to handle loading, and instantiating tools, and parameter types as needed. The ToolManager loads tools by scanning specified folders for assemblies that implement the IToolProvider interface. Once a class that implements this interface is found it is instantiated and queried for a list of the tools it is capable of providing. This allows tool providers to create tools from a wide variety of sources. A default tool provider is included in the ToolManager. This tool provider scans specified folders for assemblies which implement the ITool interface directly. Loading of parameter definitions and their user interface is also done the same way with the ToolManager looking for assemblies that implement the IParameterProvider interface, and a default provider which scans for IParameter implementing assemblies.

4.3. The ITool and IToolProvider Interfaces

The goal of the ITool interface is to remove the burden of creating a user interface and maintaining tool interoperability from the tool developer. Developers designing tools need only implement the ITool interface when designing their tool and the ToolManager generates the graphical user interface automatically for them when the tool is instantiated. Figure 3 illustrates the form that is automatically generated when the Inverse Distance Weighting tool is created. Note the help text on the right is automatically displayed when the user highlights a particular input parameter. Status lights on the left side of the parameter field display the parameters' validity. The ITool interface contains all the information necessary for running and displaying a tool.

The IToolProvider interface allows tools to be generated in a wide variety of ways. While the default ToolProvider searchers folders for assemblies that contain ITools, there are many other ways that tools could be generated. For example a ToolProvider could connect to a Web Processing Service, get a list of available tools, and then generate a corresponding set of ITools which would then be in charge of instantiating for the ToolManager.

Tools can also be generated by the Modeler. This is done by saving the model which includes several tools to a XML file which can then be opened by the ToolManager as a stand alone Tool, capable of being executed independently of the Modeler. As long as each of the Tools used to create the model are available to the ToolManager at execution time, the new Tool can be run.


Inverse Distance Weighting

Input

Point Feature Set:

Input Field:

Z-Value Field: none

Output raster cell size: 10

Power: 2

Neighborhood Type: Fixed_Point_Count

Number of point to search for: 5

Distance to search for points: 0

Z-Value Field

The field of the input feature set which contains an extra weight to assign to point

Figure 3. The Inverse Distance Weighting tool dialog running in Mono on Mac OSX

4.4. The IParameter and IParameterProvider Interfaces

Parameters are the input and output of a tool and need to be defined so that they can have an appropriate visual representation. For example a numerical parameter should allow for a minimum and maximum value to be specified tolimit the users' input to a certain range. It should also be capable of specifying a default value and be represented on the tool dialog by a text box that will only accept numerical values. This can be accomplished by creating a parameter object that specifies these constraints and contains a control object that represents how the parameter should be represented in the tool dialog. Figure 4 displays the IParameter base interface, which all parameters must implement. Figure 5 displays the List Parameter component which implements the IParameter interface. IParameters are responsible for generating two graphical components, one for input and one for output parameter configurations. This ensures that parameters must act differently before inputs and outputs can be handled.

Figure 4. The IParameter base component as it appears in the Microsoft Visual Studio designer. The base component is never seen in the modeler

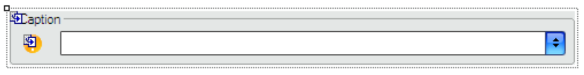
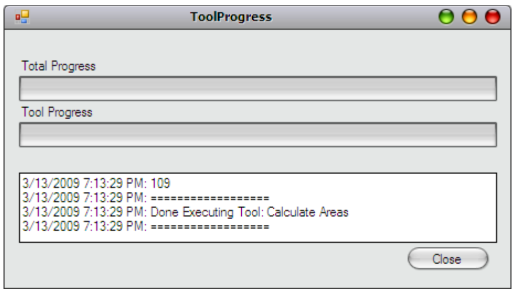


Figure 5. The List Parameter component as it appears in the Microsoft Visual Studio designer

The IParameterProvider interface defines how ParameterProviders define parameters for tools. Tools that use parameter types defined by a specific ParameterProvider must reference that provider and design time. When loading, the ToolManager first generates a list of all Parameter types that have been defined, querying each of the ToolProviders that have been detected, and requesting a list of parameters that the provider supports. Once all of the known parameter types have been loaded, the ToolManager then loads the ToolProviders. This ensures that all of the parameter types are known before the tools are instantiated.

4.5. Tool and Model Execution

Once a tool is called to be executed, either on its own or integrated into a model, a background thread is started to carry out the tool's execution. A separated thread is used to ensure that the tool progress dialog remains responsive to user activity. Messages from the background thread are relayed to the foreground progress dialog thread to allow progress indicators to be updated by the tool. Figure 6 displays the progress indicator form running a tool. In the event of user cancellation, tools are responsible for cleanly exiting.



```
3/13/2009 7:13:29 PM: 109
3/13/2009 7:13:29 PM: ===================
3/13/2009 7:13:29 PM: Done Executing Tool: Calculate Areas
3/13/2009 7:13:29 PM: ===================
```

Figure 6. Progress indicator dialog

4.6. Modeler Architecture Overview

The Modeler and ToolManager are intentionaly modular, and any single component can be replaced with another, which satisfies the interface requirements of that component. Not all components are necessary for the entire environment to work. If, for example, the Modeler was not included in a project because it was not needed, the ToolManager could be included by itself as a visual component or as a instantiated object invisible to the end user. This high level of interchangability ensures that the components of the modeling environment can be used to meet the widest ranges of developer needs. Figure 7 displays the overall architechture of the entire MapWindow Modeling project.

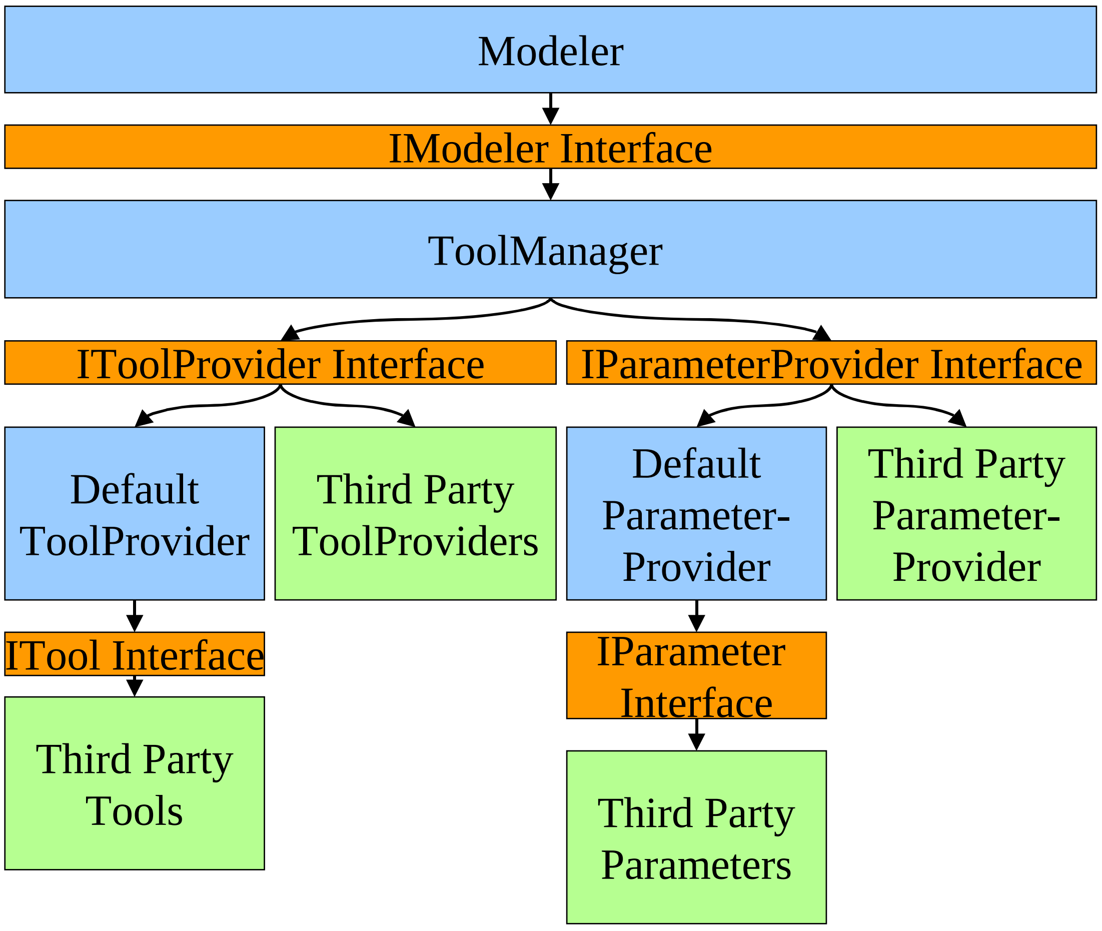


Figure 7. MapWindow Modeler Architecture

5. DISCUSSION AND CONCLUSIONS

The MapWindow GIS Modeler is a versatile modeling environment, which can handle many different data types. Due to its modular and extensible architecture it can use tools of many different designs. The design flexibility not only allows tools to function in a wide variety of different ways, but it allows tools and their associated parameters to be generated from any number of sources. Its ease of use for end users and developers, as well as its integration with MapWindow GIS 6 and MapWindow GIS 4, ensures that the widest range of users will have access to the program.

By building on the successful design of previous generations of MapWindow GIS, the MapWindow Modeler benefits from all of the development expertise, keeping the designs that were the most effective while eliminating some of the more constrictive problems. It is one more tool available to both developers looking to create new modeling tools and end users wishing to create models with such tools.

REFERENCES

Ames, D., Michaelis, C. and Dunsford, T., 2007. Introducing the MapWindow GIS Project. The Journal of the Open Source Geospatial Foundation, 2.

Greenberg, S., 2007. Toolkits and interface creativity. Multimedia Tools and Applications, 32(2): 139-159.

Gregersen, J.B., Gijsbers, P.J.A. and Westen, S.J.P., 2007. OpenMI: Open modelling interface. Journal of Hydroinformatics, 9(3): 175-191.

Liquori, L. and Spiwack, A., 2008. Extending FeatherTrait Java with Interfaces. Theoretical Computer Science, 398(1-3): 243-260.

Maxwell, T., 1999. A paris-model approach to modular simulation. Environmental Modelling & Software, 14(6): 511-517.

Olaya, V. and Gimenez, J.C., 2007. SEXTANTE: a gvSIG-based platform for geographical analysis, Free and Open Source Software for Geospatial, Victoria, Canada.

Xie, Y. and Brown, D.G., 2007. Simulation in spatial analysis and modeling. Computers, Environment and Urban Systems, 31(3): 229-231.