GE Energy

Smallworld Core Spatial Technology™ 4

Moving from monolithic applications to component–based development

GER-4234 (10/04)

Moving from monolithic applications to component–based development

Abstract

Early adopters of Geospatial Information Systems (GIS) technology faced the daunting and expensive process of digitizing information held typically as paper-based records. As this data capture phase approached completion, businesses began to demand a return on the investment they made in their data and their GIS. At first, small, simple applications appeared that began to exploit the value of this data by automating selective engineering tasks. These applications were often developed in isolation with little or no mechanisms for supporting integration and with user interfaces that were often confusing and inconsistent. This fundamental architectural weakness was exposed as GIS started to move out of the engineering department and adopt a more important role within the enterprise as a whole.

The Smallworld architecture developed by GE Energy acknowledges this new found importance by providing a powerful, extensible component-based framework that meets the demands of today's user-focused, enterprise-based GIS applications.

Simple beginnings yielded mixed blessings

The first GIS applications were simple affairs. Each application typically mapped on to a single task and was operated by a few engineers.

A typical application might answer a question such as "If an outage occurs at this point on the network, which of my customers will be affected?" These kinds of applications were typically developed in isolation with little thought given to other existing applications (generally because it was thought at that time that there was no need). Over time, more and more applications were developed as business raced to unlock the value held in their data. Some of these new applications provided new pieces of functionality, others overlapped more or less with existing applications and duplicated code. As time went it on it became obvious to application architects that it would be very advantageous if these applications could talk to each other and share existing functionality and data. Most GIS vendors approached this problem by developing piecemeal point-to-point connections between applications. If application A needed to talk to application B, then a point-to-point connection AÆB was developed to help out. If A now needed to talk to C, then a new point-to-point connection appeared AÆC. This kind of architecture happily managed the few point-to-point connections that were originally needed, but the increase in complexity of GIS applications and the business need to integrate with each other quickly resulted in a complex web of interconnections that became increasingly expensive to extend and maintain.

As if the integration question was not problematic enough, the whole issue was compounded by an architecture that squandered the opportunity to reuse code by producing made-to-order applications that were heavily tailored to one particular customer and difficult to customize or configure.

Application user interfaces that were simple and effective got corrupted by ad hoc additions that resulted in a confusing, overly complex experience for many users.

Moving forward with components, frameworks, databases and small focused applications

The Smallworld architecture has matured over time keeping pace with demands of the more powerful applications that require good integration and fast deployment. Recognizing at a very early stage that businesses were demanding more from their GIS, a bold initiative was started to move its core foundation forward to an even more flexible and extensible

application architecture capable of meeting and exceeding the most demanding requirements of the modern enterprise.

With the Smallworld architecture this transi-tion has reached a significant milestone with the introduction of a suite of cutting edge

technologies designed to simplify integration, encourage code reuse, reduce development times and ease deployment. These key technologies are:

- Component-based applications.
- Frameworks and databases.
- Configuration and customization using open standards.
- Small focused applications.

Building large with smaller blocks

The overriding principal of a component-based application model is to move away from applications implemented as a single, inflexible piece of code to more manageable, more flexible applications constructed from many smaller, reusable components. Each component is structured to represent a self-contained piece of functionality that has been designed to be generic enough to be reused by more than one application. For example, the outage application cited earlier on in this paper might be composed of a network trace component (to find affected customers), a map component (to see the extent of the outage), a report component (to produce a mail shot informing customers of repair work) and a highly focused user interface to allow the application to be easily driven by the end user.

With the Smallworld architecture, each component is implemented as an object class that inherits its basic support infrastructure from an existing core class provided by the Smallworld core API. Application developers can then expose the required functionality through the component using a well defined API. It is, therefore, important that careful thought is given to the definition of a component's API to make it as generic and reusable as possible. This inevitably means more design effort initially, but it is an investment that pays a recurring dividend over time as development costs are dramatically reduced through code reuse, streamlined integration, reduced maintenance and increased quality.

A super superstructure

Components by themselves are isolated pieces of functionality. The structure that brings these components together to form a useful application within the Smallworld architecture is called a framework. A framework is a powerful mechanism in its own right, but also manages many of the day-to-day housekeeping chores associated with creating, configuring, managing and discarding the set of components that it oversees.

Frameworks are themselves derived from components and this powerful concept allows the manufacture of composite components comprised of an assembly of smaller components. This is a useful capability especially in the case of very complex systems.

Frameworks also provide the essential infrastructure to allow components to communicate, broadcast events and share data with each other using an underlying architecture based on an elegant databus technology.

One important idea to emphasize at this point is that components rarely interact directly with each other (thus avoiding the morass associated with point-to-point connections). Instead, this inter-component interaction is an important responsibility of the encompassing framework and is implemented by allowing events and data broadcast by one component to be channeled through a common databus to another interested component for processing.

This works in the following way:

- Components first register with the databus the types of data that they will produce and the types of data that they will consume (are interested in).
- Components that want to share interesting data

with other components simply push it on to the databus along with some metadata that classifies it (for example, as the current map selection).

- Components that have expressed an interest in this data will then be notified when it becomes available allowing the component to access the data and perform some processing on it (for example, updating a user interface to show details of the new selection).

Consumer components are not restricted to having to wait for a producer component to make data available. A consumer component can also pull data from the databus when needed (not strictly asynchronous). In this circumstance the consumer component explicitly requests the databus to make data of a particular type available (again, for example, the current selection). The databus will then oblige by forwarding the request to the appropriate producer component who, in turn, will return the data to satisfy the original request.

This loosely coupled approach is an important architectural advantage as it simultaneously helps to ensure the independence of each component and discourages direct dependencies between components that all too often lead to migration, maintenance and integration problems.

Using open standards to configure inner workings

Many conventional GIS architectures that evolved from supporting simple applications continue to promote modifications to source code or editing proprietary system files as an effective way to configure and customize parts of an application. However, the realities of the enterprise environment makes this an unappealing proposition for most IT departments irrespective of the programming language or how widespread the use of the files in question. This is especially true for those departments with limited resources and finite skill sets. The Smallworld architecture was designed with these practical limitations in mind and implements configuration and customization using the latest XML open standard. Each component is configured using a snippet of XML that is then added to a master XML file that contains the configuration information for all the components required by an application. This approach is attractive to IT departments since XML is a widely understood standard and there are a wide variety of inexpensive tools for easily editing its content.

Configuration files can be kept centrally to support groups of similar users or stored locally to provide support to an advanced user.

Focusing on the user

Large monolithic applications are not only unwieldy to develop, integrate and deploy but also tend to cultivate ungainly, confusing and unproductive user interfaces.

These monster applications often have as their roots humble origins. Small applications designed to help with a particular task might have originally had productive user interfaces, but the limitations of the underlying architecture together with time and budgetary constraints meant that the path of least resistance often led to applications being augmented with extra functionality that neither fitted well with the application's original purpose nor its intended new role. Over time this degenerative process continued, resulting in a plethora of mega-applications with muddled, bloated user interfaces that end users found confusing, inconsistent and unproductive.

The Smallworld component based architecture allows the application engineer to once again put the user first by providing small, highly task-focused applications that improve the end user's experience and productivity. Application architects can now quickly swap in and out components, change their configuration and so on to meet the changing needs of the user simply by modifying configuration files.

Components can have modular user interfaces that can be hosted in containers such as panels, tabs or dialogs and can be tailored to specific situations. This

mechanism allows components to be easily moved around the user interface to optimize workflow.

Components can expose functionality that can be linked easily to common user interface elements such as pull down and popup menus, toolbars and accelerator keys again simply by editing configuration files.

A component's properties can be set by the application developer by modifying snippets of XML and, where appropriate, may also be exposed to the end user through a common dialog provided by Smallworld core API (example properties might include display units, current co-ordinate system and so on). Modified user preferences are restored when the user starts a new session.

Small focused applications require less documentation, less training, lead to fewer end user errors and bind more tightly to the existing business process.

Conclusion

Where GIS architectures have not kept pace is with the needs of the modern enterprise. They have continued with large, monolithic applications integrated using a multitude of point-to-point connections and presented with confusing and muddled user interfaces, which seriously diminish the business benefits of using GIS.

The Smallworld architecture delivers a powerful, highly flexible component-based architecture that eases integration and supports the advanced business processes executed by today's enterprises: providing easy to use applications focused on the user.