An Agile Programming Model for Grid End Users

Zhiwei Xu, Chengchun Shu, Haiyan Yu, Haozhi Liu
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
{zxu, yuhaiyan}@ict.ac.cn {shuchengchun,liuhaozhi}@software.ict.ac.cn)

Abstract

Grid and service computing technologies have been explored by enterprises to promote integration, sharing, and collaboration. However, quick response to business environment changes is still a challenging issue. For end users, developing, customizing, and re-engineering applications remain a difficult and time-consuming task. Users still need to deal with excessive low-level details of platform-specific APIs. We present a high-level programming model together with a descriptive glueing language called GSML, to facilitate end-user programming. In this approach, applications could be visually composed from well-defined software components called "funnels" in an event-driven fashion. Application examples have shown that, by raising the level of abstraction as well as simplifying the programming model, GSML could empower end users to build grid applications on demand with improved productivity.

1. Introduction

Recent years have witnessed significant technology innovations on integrating distributed resources from multiple administrative domains into a virtualized and coherent computing environment, namely the grid. However, attempts to utilize grid technologies in daily enterprise operations are often frustrated by the difficulties of adapting underlying IT infrastructures as well as applications to business environment changes. For end users, developing, customizing and re-engineering applications are still a difficult task, even with state-of-the-art grid middleware and toolkits.

Enterprise information systems have to change with the trends of technology and market demands. These changes can be categorized into two levels: resource-level and application-level. The resource-level changes refer to the evolution of the underlying IT infrastructure, which consists of a variety of resources such as servers, desktops, networks, software and databases. Either regrouping of the whole organization, such as merging or cutting divisions, or small adjustment within some departments, such as upgrading database servers or changing the schema of a certain inventory table, result in resource-level changes that could demand the modification of business application codes. Application-level change is necessary when business models and business processes (e.g. workflows) have to be altered with respond to market conditions.

Both types of changes put an emergent demand for agile software development approaches that allow information systems to continue running smoothly, with little modification effort. Ideally, such modifications for meeting new business requirements could be accomplished by end users with minimal intervene of IT professionals.

Grid end users are those application users (e.g. managers, secretaries, salesmen, biologists, chemists) who use grid as part of their daily work but may not have professional background on computer programming. For them, building or modifying grid applications remain a difficult and time-consuming task. To build new applications, end users need to deal with excessive details of low-level APIs that are often platform-specific and have a high learning curve for them. For example, users must have extensive knowledge of XML, SOAP and Web Services when developing applications based on service-oriented grid middlewares. These technical details often divert user's attention from business needs of an application.

This paper presents an agile programming model together with a descriptive language called GSML, to facilitate end-user programming in an approach that applications could be visually composed from off-the-shelf software components called "*Funnels*". GSML (Grid Service Markup Language) is an XML-based markup language for users to specify how to access grid resources (including services) and how to glue heterogeneous modules together in an event-driven fashion. The core concepts of GSML, *funnel*, *event*, and *event-set*, are derived from a simplified model of

pi-calculus [9], which is suitable for describing the distributed and concurrent nature of grid applications. In addition, we have implemented a visual programming environment to support the authoring of GSML applications in an intuitive way. With GSML, users need not build a grid application from scratch, but can integrate many "ready for use" components as needed. These components have encapsulated low-level and labor-intensive coding work, and may therefore make the programming task easier even for non-programmers. Event-based connections among components allow for a more loosely coupled architecture than conventional RPC-style ones.

There are a variety of studies on end-user programming. Spreadsheet languages such as Forms/3 [2] and FAR [3] regard the spreadsheet paradigm as an easy, and computationally powerful devices to solve end-users' problems. KidSim [10] explores the Programming by Demonstration (PBD) method to empower children to program their simulations and games. Natural Programming [8] studies more natural approaches to programming based on empirical and psychological studies of programmers. The design and implementation of GSML Composer benefited from these studies and experiences.

Reference [6] categorizes grid programming models into the following classes: grid-enabled messages passing libraries, middleware, problem solving environments (PSE) and portals. Java CoG Kit [7] is one of the PSE tools, which aims to providing convenient access to grid system through commodity technologies such as the Java framework. Grid Portal Development Toolkit (GPDK) exemplifies portal technology which makes rapid portal development facilities possible by providing a suit of JavaBeans suitable for Java based grid computing environment. GSML enables end-user programming and agile programming through virtualization, visual-style programming, and a carefully designed small set of language concepts.

2. Design Considerations

To support business agility within the context of grid-enabled information systems, we identified a set of requirements for an "agile" programming model and design principles. We use the term agile to highlight the need to both quickly react to changes in resource level and quickly refactor programs in application level.

2.1 Resource Virtualization

Virtualization is a well-accepted design principle both in traditional operating systems and programming languages. In the GSML programming model, an end user only sees *virtual resources* that are location independent. A virtual resource (e.g., with a name "PriceQuery") is mapped to a physical resource (e.g., the price query service on machine *A* with endpoint address of http://host-*A*:8080/pq) at runtime, the process of which is called *resource binding*. Resource bindings are automatically enforced with the support of runtime system software.

With virtualization, application codes do not have to be hard-wired to physical resource addresses and interfaces. In addition, GSML applications referring to virtual resources could benefit from systematic load balancing (by choosing alternative physical services with lower load), fault tolerance (by switching to a new physical service in response to service failure), locality of service access (by locating a nearby physical service), etc. All such details are transparent to the application logic.

2.2 Loosely-coupled Components and Event-based Architecture

The volatile behaviors of grid resources, such as joining or leaving the grid at any time, imply that the components of grid applications need also to be loosely coupled. Most of existing component architectures, like CORBA, DCOM, RMI and Web Services, are based on a point-to-point communication model, which is characterized by a tight conceptual coupling between the component that requests a service (the client) and the component that satisfies such a request (the server). Many situations require the availability of a more decoupled model [4]. For example, in a dashboard application where live information of stock quotes from world-wide markets are aggregated and analyzed, the communication among the components may involve more than two parties, and may be driven by the contents of the information rather than by the identity of information producers and consumers.

In GSML, we propose to utilize an event based model for gluing components, to support loosely coupled, scalable interactions among distributed parts of an application. A language construct called *funnel* is introduced for the purpose of specifying business components, and for disseminating, merging, splitting or filtering events.

2.3 High-Level and Visual Programming

End users require a high level programming model, where by grid applications are constructed by reusing modular business components, instead of developing

from the scratch with general-purpose programming languages like C and Java. A key issue is to define a suitable abstraction, which is called funnel in GSML.

The success of spreadsheets demonstrates visual style of programming could be easier to understand and generate for humans, especially for non-programmers [1]. Adding visual features into a programming model provides intuitive graphical primitives and also enforces programming discipline, which brings about application robustness.

With the GSML model, we designed a visual programming environment that allows users to create applications by dragging components (funnels), dropping them into a workspace, and connecting them together to build an event flow graph.

3. The GSML Programming Model

3.1 Constructs

A GSML application is an end user defined grid application, which is specified as an XML document and run in a GSML browser.

```
<gsml xmlns="http://gsml.ict.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://gsml.ict.org GSML.xsd">
  <head>
    <title>dashborad-1</title>
  </head>
  <body>
    <row>
      <cell>
        <funnel id="pWeather" type="WSInvoker">
          <eventset>
            <event source="pStock" type="WSResult"/>
            <target destination="pHTML" type="script">
              <para>
                <name>code</name>
                                <value><![CDATA[stocks.innerHTML=
                  "${pStock.WSResult.getStocksReturn}"]]></value>
              </para>
            </target>
          </eventset>
          .......
        </funnel>
        <funnel id="pHTML" type="HTMLFunnel"></funnel>
        <funnel id="pTimer" type="TimerFunnel"></funnel>
        <funnel id="pStock" type="WSInvoker"> </funnel>
      </cell>
    </row>
  </body>
</gsml>
```

Fig.1. A GSML application example

The core construct of GSML is *funnel*. A funnel is an independent function module, which can be either *atomic* or *composite*. Several funnels can interact with one another by generating, filtering and consuming *events*. This is how the control and data flows for an integrated GSML application are specified. An atomic funnel is a basic module predefined by the system (the GSML runtime system or a grid environment). A composite funnel is a graph of event-sets and funnels.

An *event-set* defines the dynamic interaction of funnels in a grid application. It is made up of a set of input events and output events. The interaction occurs when all its input events occur. Then all its output events will be tested according to the corresponding Boolean expressions associated with the events' parameters. If an expression results in True, an event then will be sent to the destination.

An input event has three attributes: the identifier, the type and the source of the event. Every input event is generated by some source funnel, which also specifies the event type.

An output event describes what, where and when the event will be transmitted. It has three attributes: a event description, the destination identifier, and a Boolean expression to determine whether or not the transmission occurs. An event description is a set of parameters (name-value pairs). The parameters serve as customized arguments of the event.

Figure 1 presents a simple example to illustrate the syntax of a GSML document. It periodically retrieves and updates weather and stock information. The application is built with five ready-to-use funnels: one TimerFunnel, one System, one HTMLFunnels and two WSFunnels. The dynamic interactions between these funnels are specified in the four event-sets: one for calling the corresponding web services based on the timer event information, and two for scripting the web pages to update the dashboard with the newest information. How this application is built in GSML composer and how it runs in GSML Browser are presented in section 4.

3.2 Layered model

Figure 2 gives the three-layer GSML model. GSML Resource Layer is a resource space layer, in which each resource is abstracted into a ready-to-use component. The component can be virtualized desktop applications (e.g. Excel, Matlab, Web Browser), a Web service, or grid services. To hide low-level details of resources, the GSML Resource Layer provides resources abstractions with a uniform interface: the components work as funnels which interact by generating, filtering and consuming events.

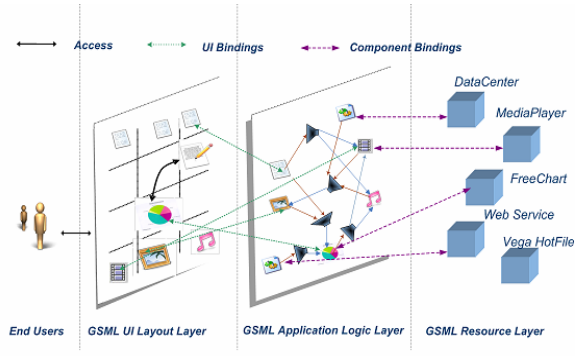

Fig.2. Architectural model of GSML

GSML Application Logic Layer is the gluing layer where loosely coupled, dynamic interactions between distributed components (funnels) are customized by end users. This layer defines what components are used by the application by mapping funnels to resources. This layer is also responsible to customize how the interactions among the components are performed, via funnels and event-sets.

GSML UI Layout Layer is to define the presentation layout of components and their display. Currently, a two dimensional table is used in the form of cells in rows with adjustable width and height. Each cell is assigned to zero or more funnels.

4. Implementation

4.1 GSML Composer

We implement GSML Composer to provide a visual programming environment for the development of GSML applications, using the Eclipse Graphical Editing Framework (GEF) [5]. The composer represents the funnels and event-sets as visual nodes, and events as connections. The composer lists all the ready-to-use components and associates all the possible events with the components' connections. A GSML application is constructed by dragging and dropping the visual nodes of components into the workspace, connecting them with events. End users can interactively manipulate and modify the visual objects, as GSML Composer provides the immediate visual feedback at liveness level 3 [1]. Figure 3 shows a snapshot of GSML Composer during authoring the information dashboard example.

When the building or modification is done, GSML applications can be saved in composition files, exported as GSML documents, or directly loaded into the GSML Browser to run.

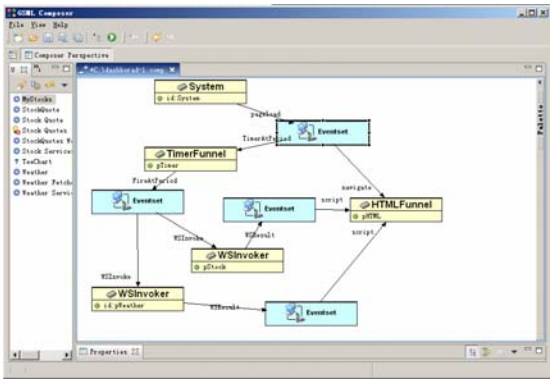

Fig. 3 A Snapshot of the GSML Composer

4.2 GSML Browser

The GSML Browser provides a runtime system for GSML applications. It (1) initializes and manages the funnels required by the application, and (2) handles event-based interactions among the funnels.

After the GSML document is loaded and parsed by the GSML Browser, a Funnel Manager is started. It loads each funnel and starts it in a separate thread context. An Event Manager manages the events from funnels, check if an interaction is ready, and deliver output events to destination funnels.

GSML Browser builds TCP connections as communication links between the funnel and the Event Manager. The Event Manager starts with a public port for connection with the funnels in GSML applications, and each funnel will locate the Event Manager and establish a connection when it starts. Through the connection, the funnels transmit the events to the Manager which in turn forwards the output events to the destination funnels. The connections are loosely coupled and dynamic because they are specified by the users of components, and dynamically created during execution, which are subject to rapid adaptations to application changes. Figure 4 shows a snapshot of the information dashboard application running in the GSML Browser.

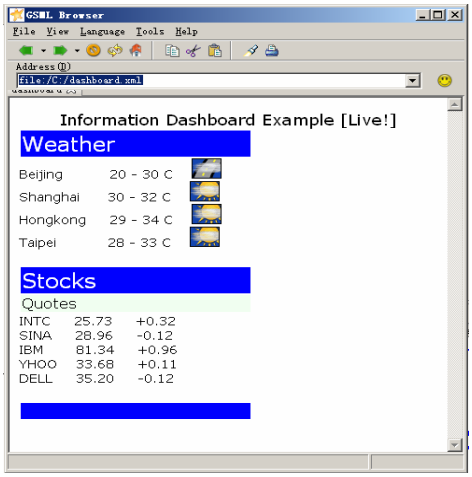

Fig.4. A Snapshot of the GSML Browser

5 Conclusion

This paper presents a high-level visual programming model that allows end users to integrate business components. The objective is to support business agility in a grid environment.

Three technical requirements are identified to accommodate business agility needs, namely, resource virtualization, loosely coupled component architecture, and high-level visual programming. A markup language called GSML, together with its runtime system (the GSML Browser) and developer tool (the GSML Composer), is designed and implemented. The core language construct is funnel, while a key mechanism is event-based interaction among components (funnels).

End User Friendliness. As is shown in the Information Dashboard example, the GSML Composer can lower the technology barrier for end users. First, it lists all the ready-to-use components and their interfaces in visual way, with business-level terminology. Second, it gives each element of a GSML page a visual representation, thereby an end user can manipulate the data model intuitively. Third, the instanced information of the funnels (e.g. the WSFunnel) can be automatically filled through UDDI Web Services facility of GSML Composer. The simplified requirement can be afforded by grid end users who even have little knowledge of low-level details of grid architecture, protocols and programming. In addition, GSML Composer improves the correctness of grid applications by exposing end users to the visual representations for GSML elements instead of the error-prone, laborious XML syntax.

Supporting Agile Programming. The GSML Composer enables rapid development of grid applications. For application changes at the presentation level, we only need minor changes, such as adjusting the table layout visually or load new web page templates created by web page authoring tools. For the application logic changes, we can add new funnels and new connections, or/and adjust the connection and parameters of the interaction flows. For resources changes in the system, no application modification is need in many cases, due to resources virtualization.

Expressiveness. The expressiveness of GSML can be evaluated in two levels. The first level is to evaluate what GSML can express by composing a given set of funnels. To aid user friendliness, GSML provides very intuitive primitives, and try to provide too much expressiveness. Another level is to evaluate what applications the GSML can build given that all the required funnels are available. The expressiveness at the level is determined not only by the loosely-coupled, dynamic interactions, but also by the funnels which can be developed with traditional programming languages (Java, JSP, etc.) and existing programming models (RMI, Web Service, etc).

To sum up, with funnels and event-sets, GSML can be used to build many grid applications by an efficient partition of labor between end users and technical staff. Technology savvy people are responsible for developing components (expressed in GSML as funnels) that can be shared by many grid applications. End users can use the GSML composer to quickly develop and maintain personalized grid applications.

References

[1] M. Burnett, "Visual Programming", *Encyclopedia of Electrical and Electronics Engineering*, John Wiley & Sons, 1999.

[2] M. Burnett, and H. Gottfried, "Graphical Definitions: Expanding Spreadsheet Languages through Direct Manipulation and Gestures", *ACM Transactions on Computer-Human Interface 5(1)*, March 1998.

[3] M. Burnett, S. K. Chekka, and R. K. Pandey, "FAR: An End-User Language to Support Cottage E-Services.", *HCC 2001*, 2001, pp. 195-202.

[4] G. Cugola, E. Nitto, A. Fuggetta, "The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS", *IEEE Transactions on Software Engnineering,vol. 27*, 2001, pp. 827-850

[5] Graphical Editing Framework. http://www.eclipse.org/gef/, 2005.

[6] D. Laflorenza, "Grid Programming: Some Indications Where We are Headed", *Parallel Computing, Vol. 28*, 2002, pp. 1733-1752.

[7] G.V. Laszewski, I. Foster, J. Gawor, and P. Lane, "A Java Commodity Grid Kit," *Concurrency and Computation: Practice and Experience, vol. 13*, 2001, pp. 643–662.

[8] B. Myers, "Towards More Natural Functional Programming Languages", *The ACM International Conference on Functional Programming (ICFP 2002)*, Pittsburgh, PA, October, 2002,

[9] D. Sangiori, and D. Walker, "The Pi-calculus: A Theory of Mobile Processes", Cambridge University Press, 2001.

[10] D. Smith, A. Cypher, and J. Spohrer, "Kidsim: Programming Agents without a Programming languages", *Communications of the ACM 37(7)*, July 1994, pp. 54-67.