BEAP: An End-User Agile Programming Paradigm for Business Applications

Cheng-Chun Shu¹,² (舒承樞), Hai-Yan Yu¹ (余海燕), and Hao-Zhi Liu¹ (刘昊植)
¹Research Center for Grid and Service Computing, Institute of Computing Technology, Chinese Academy of Sciences
Beijing 100080, P. R. China
²Graduate University of the Chinese Academy of Sciences, Beijing 100039, P. R. China
E-mail: shuchengchun@software.ict.ac.cn; yuhaiyan@ict.ac.cn; liuhaozhi@software.ict.ac.cn
Revised May 15, 2006.

Abstract 
Business applications are subject to changes with technology trends or market demands. However, quick response to these changes is still a challenging issue. Most of the existing architectures (e.g., CORBA, Web Services) still expose the developers to excessive low-level details and force a tight coupling between program modules. For end users, developing, customizing, and reengineering applications remain difficult and time-consuming tasks. A high-level programming model is presented, together with a descriptive programming paradigm called BEAP, 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, BEAP could empower end users to build business applications on demand with improved productivity.

Keywords 
visual programming, event-based, virtualization, component, agile, pi-calculus

1 Introduction

Business applications obtain a number of advantages by integrating resources from multiple administrative domains into a virtualized coherent computing environment as the grid does: higher reliability, larger scalability, better performance and improved efficiency. However, the building of the large-scale business applications is often impeded by the excessive low-level technical details of the programming tools. Attempts to utilize the distributed computing technologies in daily 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 reengineering applications are still difficult and time-consuming tasks, even with state-of-art business middleware and toolkits.

Business applications are subject to changes when the technology trend or market demands are altered. 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 trimming 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 in respond to market condition changes.

Both types of changes put an emergent demand for agile software development approaches that allow the applications to continue running smoothly, with few modification efforts. Ideally, such modifications for meeting new business requirements could be accomplished by end-users with minimal intervention of IT professionals.

Business end-users are those application users (e.g., managers, secretaries, salesmen, biologists, chemists) who use business applications as part of their daily work but may not have professional background on computer programming. For them, building or modifying the applications remains 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 middleware. These technical details often divert user’s attention from business needs. Furthermore, most middleware systems and toolkits (e.g., CORBA, MPI, and Java RMI) for business applications force tight, coupled ties between software components and are unable to survive the dynamic interactions among components and the frequent components departures or failures. These technologies fail to perform well in distributed environment where the communication is unreliable, the latency and bandwidth are heterogeneous. An agile programming environment for business applications must support the loosely coupled connections among components and volatile prop-
erties of resources and interactions.

This paper presents an end-user agile programming paradigm for business applications called BEAP. To support loosely coupled connections between components, event-based model is adopted not only for building the underlying communication infrastructure, but also for defining the core constructs to specify the interactions among components. The core constructs of BEAP, funnel, event, event-set and session, are derived from a simplified model of pi-calculus [1], which is suitable for describing the distributed and concurrent nature of components in business applications. We have implemented a visual programming environment to support the authoring of business applications in an intuitive way. Programmers need not build business applications from scratch, but integrate many ready-to-use components predefined in BEAP. These components, which function as high-level resource abstraction, have encapsulated low-level and labor-intensive coding work, and may therefore make the programming task accessible even for non-programmers. Examples show that BEAP enables end-user agile programming through virtualization, visual-style programming, and a small carefully designed set of language concepts.

2 Requirements and Design Considerations

This section outlines the requirements to support an agile programming for business applications. The aim is to identify a set of design principles for the paradigm. By agile, we highlight the need to both quickly react to changes in resource level and quickly refactor programs in application level.

2.1 Requirements

Scalability. The paradigm needs to deal with a large number of resources in the applications. The challenge to accommodate a large number of resources lies at not only the complexity of heterogeneous and distributed resources, but also the complexity of growing number of connections among resources.

Extensibility. Business applications tend to evolve because of both application-level and resource-level changes. The paradigm should be extensible to support new resources and new forms of interactions, which demands a pluggable architecture and a high-level resource abstraction.

End-user friendliness. The dilemma of traditional paradigms for business applications is that technicians know little about the business logic while end-users cannot deal with the technical details of implementing the logic. One solution is to lower the technical barriers and empower end-users to customize their own applications.

2.2 Design Principles

2.2.1 Resource Virtualization

Virtualization is a well-accepted design principle in both operating systems and traditional programming languages. In BEAP, 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, BEAP 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.

2.2.2 Loosely-Coupled Component and Event-Based Architecture

The volatile behaviors of business resources, such as joining or leaving the environment at any time, imply that the components of business 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 [2]. For example, in a dashboard application where live information of stock quotes from worldwide markets is 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 BEAP, we propose to utilize an event-based model for glueing 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 and filtering events.

2.2.3 High-Level and Visual Programming

End-users require a high-level programming model, by which BEAP 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 BEAP.

The success of spreadsheets demonstrates visual style of programming could be easier to understand and generate for humans, especially for non-programmers [3]. Unlike traditional textual programming techniques,