

CAFISE: An Approach to Enabling Adaptive Configuration of Service Grid Applications

HAN YanBo (韩燕波), ZHAO ZhuoFeng (赵卓峰), LI Gang (李刚), XING DongShan (邢东山), LV QingZhong (吕庆中), WANG JianWu (王建武), XIONG JinHua (熊锦华) and LIU Hao (刘浩)

Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100080, P.R. China

E-mail: yhan@ict.ac.cn

Received December 26, 2002; revised May 28, 2003.

Abstract Aiming at building up more powerful, open-standard-based and generic infrastructures for application integration, service grids address the challenges in large-scale coordinated sharing and on-demand composition of network-based application services. The related endeavors have opened up new ways of application development, deployment and integration. In connection with the new level of scale, openness and dynamism brought forward by service grids, adaptive service configuration is of essential importance to applications. This paper proposes an approach called CAFISE, which tries to better facilitate on-demand configuration and dynamic reconfiguration of service grid applications. In CAFISE, a business design and its supporting software system are considered in a coherent way, and a convergent relation, which helps to map business-level configurations to software-level configurations, is highlighted. The paper is particularly devoted to presenting and discussing the principles, reference model, modeling language and supporting application framework of CAFISE. Since practical usefulness is highly valued in the development of CAFISE, the application of the approach to a real-world scenario is also presented in the paper.

Keywords service grid, on-demand service configuration, dynamic service reconfiguration, convergent modeling, adaptive application framework

1 Introduction

It has been widely realized that traditional Internet-based applications have not taken full advantage of the Internet infrastructure. With technological and social advances, some new developments can be observed. Large collections of computational and information resources are made available as services and can flexibly be utilized over the Internet. On the other hand, our modern society tends to depend on more agile, diverse and integrated value chains that can be formed by assembling these dispersed services. The striving goal of engineering Internet-based applications has thus shifted to locating, incorporating, adapting and integrating distributed and heterogeneous resources to satisfy diverged and personalized requirements of organizations and individual users.

Aiming at realizing large-scale, coordinated resource sharing in a wide-area network, grid computing technologies^[1,2] are among the most prominent advances toward next-generation Internet-based applications. Grid computing originated

from high-performance scientific computing field and is evolving into enabling technologies with a wider spectrum from computational grids to higher-level platform technologies, such as service grids^[3,4]. Here, service grids refer to the open computing infrastructures that enable easy plug-in and deployment of individual and dispersed services, interoperation of heterogeneous services, dynamic formation of virtual and collaborative service assemblies, as well as on-demand composition of services. We then call the applications based upon such infrastructures as service grid applications.

Service grids help to open up new ways of application development and deployment. In connection with the new level of scale, openness and dynamism brought forward by service grids, system flexibility and adaptability become more critical. We know that changes of computer applications are often unavoidable and tend to become more frequent as the scale of application systems increases. However, an old but not yet well-resolved problem is that the side effects of changes in business-level requirements or business policies are un-proportionally

This work is supported by the National Natural Science Foundation of China under Grant No.60173018, the Young Scientist Fund of ICT under Grant No.20026180-22, and the Key Scientific and Technological Program for the Tenth Five-Year Plan of China under Grant No.2001BA904B07.

amplified in the underlying software systems. That is, a minor change in a business may cause a great deal of change efforts in its software system. In a grid environment, the border and the constituent parts of an application system are no longer so fixed as in traditional software systems due to the above-mentioned openness and dynamism. To sum up, on-demand access to and dynamic integration of distributed services have become a key issue in constructing next-generation Internet-based applications. Adaptable service orchestration, versatile business policy specification, and flexible configuration management are hereon some of the key problems to be dealt with in a grid-based environment.

As a matter of fact, solutions for making computer applications to adapt reactively and sometimes proactively to changes of the businesses they support have been sought for years. Many useful technologies, including object-oriented technologies, workflow technologies and some others, have been invented. However, the flexibility attained with these modern technologies is often mitigated greatly after the implementation or compilation stage. Concerning on-demand configuration of Internet-based information systems, the problems we are facing are even more complicated, and effective and systematic approaches are still far away from mature. This paper re-addresses the old flexibility problem in the above-mentioned new setting and presents an approach called CAFISE (Convergent Approach for Information System Evolution). It is organized as follows. Firstly, a brief overview of the CAFISE approach is given in Section 2. Section 3 discusses the CAFISE model, the CAFISE language and some modeling principles. Section 4 describes an application framework, which helps to construct and reconfigure a service grid application on the basis of the CAFISE model. Section 5 examines a case study and briefly describes how to construct and reconfigure an application with the CAFISE approach in practice. Finally, some concluding remarks are given in Section 6.

2 Rationales of the CAFISE Approach

An information system or an application system in general is developed to support a certain business. In the traditional way of information system development, business models are derived based on business analysis results first. It is software architects who then transform business models into software architecture models, either explicitly or implicitly. Based on the resulting software architec-

ture models, individual applications are developed by software engineers. So far, so good. However, there exists a conceptual gap between the business domain and the software domain as shown in Fig.1. As mentioned earlier, changes in businesses take place frequently. In order to reflect and keep up with business changes, their underlying software systems are subject to frequent changes. As a common practice, when confronted with changes in requirements and technology updates, application developers try to modify the existing software systems and make patches to them. Over time and as the number of patches and modifications reaches a certain limitation, complexity sharply increases, the original structure is destroyed and becomes inconsistent, and at last the software system reaches its life-end at a young age. One of the main reasons is that since software architecture models are based on previous business models, mismatches may come forth when business models are changed while software architecture models remain the same. With the presence of such mismatches, it is difficult and risky to modify an existing software system. The domain gap has been a main impediment for software systems to keep pace with the changing businesses in time. In a service-grid-enabled environment, this problem is even more crucial due to the openness and dynamism. In addition to the problems of change and adaptation, service-grid-based applications also need to handle the requirements of just-in-time application construction on the basis of user's spontaneous requirements.

With the paradigm shift towards grid-based computing, an application can be seen as a spontaneous configuration of its constituent computational and information resources. By explicitly separating configuration schemas from applications, we obtain a separate layer of abstraction for the sake of flexibility. Different configurations correspond to different applications. A lot of research efforts have been made to study various strategies of application configuration to facilitate adaptive changes to applications^[5,6]. Oreizy employed software architecture models to specify application configurations, and used an architecture description language (ADL) to directly describe the configuration models^[7,8]. As such, application configuration can be specified in terms of an architecture model. Software architecture is used to support application reconfiguration at runtime. Shrivastava also proposed to use software architecture models to support dynamic application reconfiguration^[9]. In [10], Warren presented a specialized model for

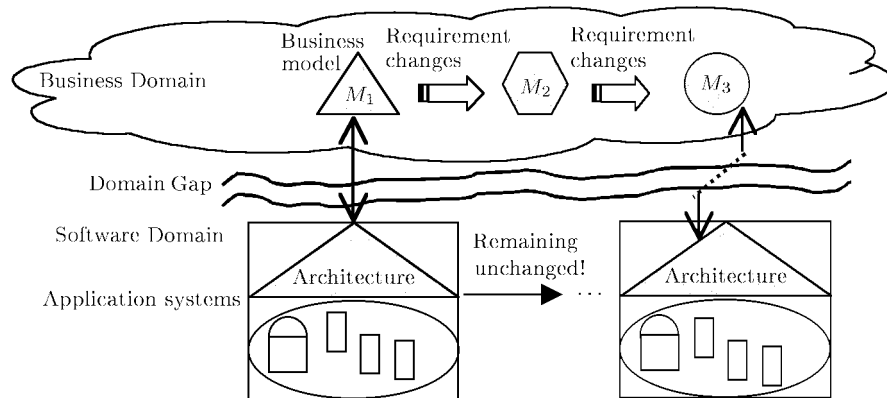


Fig.1. Model mismatches in the traditional way of information system development.

application configuration, paying particular attention to maintaining application's consistency in performing dynamic reconfiguration. Both structural constraints and behavioral information are captured in the configuration model. Although the above-mentioned efforts can help to ease application (re)configuration, they only focus on the concepts and mechanisms of the software domain. In fact, the majority of change requirements come from the business domain. Effective approaches to keeping software systems updated in accordance with new business requirements are not yet mature. On-demand application configuration and dynamic reconfiguration that are required in a service-grid-enabled environment are still difficult to achieve.

Taylor brought forward the concept of convergent engineering in 1995 for bridging the gap between the business domain and the software domain^[11]. Convergent engineering promotes an integrated approach to designing more flexible software that can better cope with business changes. It is considered as a good foundation for just-in-time application construction and on-demand adjustment on user's behalf. The essential idea is to treat a business design and its supporting software as a whole. In [12], Hubert reified the concept of convergent engineering and built a convergent software architecture directly reflecting some business-level concepts in an object-oriented manner. However, constructing business objects that can reflect both business perspective and software perspective coherently is still a complicated undertaking and remains the job of software professionals. But it is business experts who need a feasible way to express their new demand. An object-oriented convergent design is often difficult for them to master. Moreover, dynamic application reconfiguration is not thoroughly explored in previous researches.

From a user-centric perspective, the CAFISE approach is intended to help a business user to build his or her own personalized and situation-specific application by configuring available services in terms of personal preference and business-related process logic. The focus is on facilitating business-user-oriented and just-in-time configuration, and dynamic reconfiguration of service grid applications. In order to enable business people to perform on-demand configuration of service grid applications, business designs and application implementations are considered in a convergent way. CAFISE allows a business expert to transparently examine all scattered resources that are accessible and available to him or her and to configure them in an easy and straightforward manner. The advantage lies in that business people, to some extent, no longer have to rely on software professionals whenever they have change net requirements for changes.

Fig.2 illustrates the central idea of CAFISE. We coined the term *power user* to highlight the user-end programming. A power user is a business user who is capable of configuring his business applications with proper tool support but knows little about software programming. CAFISE aims at supporting power users in (re)configuring a service-grid-enabled application on their demand. We call the process, with which a power user configures his or her application, as user-end programming. Instead of directly introducing object-oriented concepts and technologies into the business domain as in [12], we propose to build a convergent model — the CAFISE model — to cover business issues and software issues simultaneously. It has three related parts: a business-level model, a software-level model and a convergent relation as glue. Business-level models are designed to reflect business requirements with a minimum set of user-end pro-

gramming concepts and mechanisms, which are easy for a power user to understand and master. A power user can (re)configure service grid applications by building or editing business-level models. Software-level models are abstractions of service composition. Based on network-based autonomous services, service composition is seen as a new paradigm to construct an application by aggregating services and defining their cooperation pattern^[13–15]. The resulting application is either a new customized application or another service with a larger granularity. CAFISE narrows the gap between the two levels with a convergent relation. The convergent relation helps to translate simple and intuitive business-level elements into more concrete and executable elements at the software level.

CAFISE promotes a new thinking in application assembly and adjustment. In CAFISE, what a user perceives as applications is in fact manifes-

tations of the CAFISE framework acting on the CAFISE model. Instead of seeking help from software professionals for building applications, power users can define application specifications by themselves. After a power user constructs an intuitive application configuration from the viewpoint of his business, the CAFISE framework helps to manifest, map and interpret a corresponding software-level model. The convergent relation plays a key role in transforming the business-level model into the software-level model. To present and construct the CAFISE model, a modeling language called the CAFISE language is proposed. Besides the model, the language and the CAFISE framework as shown in Fig.2, a virtual service organization (also called service community)^[16], which helps to discover, organize and manage services in a service grid, serves as a basis of the CAFISE approach.

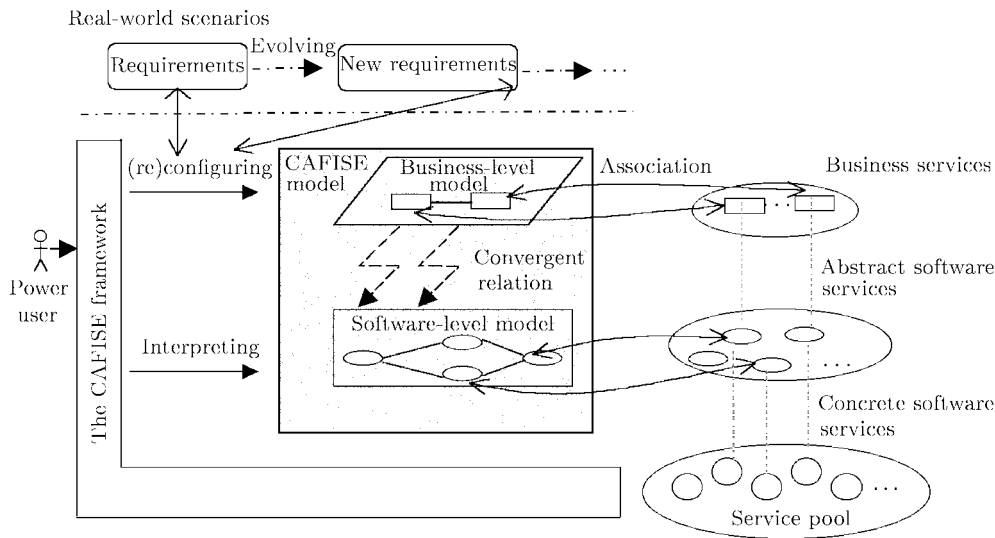


Fig.2. User-centric construction and adjustment of service grid applications with CAFISE.

3 CAFISE Model

3.1 Model Definitions

As shown with the dark rectangle in Fig.2, the CAFISE model has three constituent components: a business-level model, a software-level model and a convergent relation. It is formally defined as a three-tuple: $M = (B, A, \tau)$ with $B \cap A = \emptyset$ and $B \neq \emptyset$, where the meanings are as follows.

1) $B = (B_A, B_S, B_E, B_R, f_B)$ represents a business-level model, satisfying $B_A \neq \emptyset$, $B_A \cap B_S = \emptyset$, $B_A \cap B_E = \emptyset$, $B_S \cap B_E = \emptyset$.

- B_A is a finite and nonempty set of business

activities. A business activity defines what is to be done at a business step.

- B_S is a finite set of business services. A business service provides a set of functionalities that can be used to accomplish a business activity. It can be atomic or composite. While atomic business services serve as the most elementary service units, a composite business service aggregates multiple business services that can be either atomic or composite. In CAFISE, an application can also be viewed as a specific form of composite business services. To users, both atomic and composite business services take on a uniform appearance.

- B_E is a finite set of business information enti-

ties. A business information entity corresponds to an information resource consumed or produced by a business activity.

- $B_R = \{B_R^1, B_R^2, B_R^3\}$ is concerned about three types of business relations, respectively describing relationships between business activities, between business activities and business services, and between business activities and business information entities. The three types of business relations are explained in some more detail below:

$B_R^1 \subseteq B_A \times B_A$ defines relationships between business activities.

$B_R^2 \subseteq B_A \times B_S$ associates business activities with business services. A business activity is fulfilled by a business service, either atomic or composite.

- $B_R^3 \subseteq B_A \times B_E$ defines relationships between business activities and business information entities. A business activity may have relationship with one or more business information entities it consumes and/or produces.

$f_B: 2^{B_A} \times 2^{B_A} \rightarrow \{\text{conditional sequence, conditional split, conditional join}\}$ is a function determining routing strategies between business activities, and results in three types of control nodes connecting business activities. *conditional sequence* is a directed link from a source business activity to a destination business activity. The business activities are performed one by one under certain conditions. Note that a source activity and a destination activity can be the same, resulting in a conditional repetition. *conditional split* means that some of the business activities behind this point can be performed concurrently under certain conditions. *conditional join* means that only some of business activities before this point need to be finished under certain conditions.

2) $A = (A_S, A_C, A_M, A_R, f_A^1, f_A^2)$ represents a software-level model, satisfying $A_S \cap A_C = \emptyset$, $A_S \cap A_M = \emptyset$, $A_C \cap A_M = \emptyset$.

- A_S is a finite and nonempty set of abstract software services. An abstract software service is composed of functional and non-functional specifications of a software service. Functional specifications describe functional constraints and input/output messages, and non-functional specifications include non-functional properties and constraints, such as service provider's information, service categories, QoS information, etc.

- A_C is a finite set of concrete software services. A concrete software service corresponds to an implementation of an abstract software service belonging to A_S . It can be a stand-alone service,

such as an Internet-accessible Java application, or a composite service that consists of a set of other services.

- A_M is a finite set of messages. A message is composed of message type, message dispatcher, message receiver and message body that contains parameters and protocol-related data. Protocol-related data mark the protocols that are used to exchange messages.

- $A_R = \{A_R^1, A_R^2, A_R^3\}$ represents three types of relationships:

$A_R^1 \subseteq A_S \times A_S$ defines control relationships between abstract software services, which reflect their execution order.

$A_R^2 \subseteq A_S \times A_C$ associates abstract software services with concrete software services.

An abstract software service can be associated with one or more concrete software services. Note that they are not necessarily bound with each other fixedly. At runtime, it is possible to do dynamic binding based on functional and non-functional specifications of an abstract software service.

$A_R^3 \subseteq A_S \times A_M$ represents message relationships that specify mappings between input and output messages of abstract software services and also describes the message to be exchanged between abstract software services.

$f_A^1: 2^{A_S} \times 2^{A_S} \rightarrow \{\text{conditional sequence, conditional split, conditional join}\}$. This function defines several control relationships: *conditional sequence* indicates that two abstract software services linked through it are to be invoked successively; *conditional split* means that some of the abstract software services behind this point can be performed concurrently under certain conditions; *conditional join* means that only some of abstract software services before this point need to be finished.

$f_A^2: 2^{A_S} \times 2^{A_M} \rightarrow \{\text{direct, synthesized, decomposed}\}$ defines three types of messages: *direct* means a message is directly transferred from one abstract software service to another, *synthesized* synthesizes multiple messages of several abstract software services into one message and transfers it to an abstract software service, and *decomposed* decomposes one message of an abstract software service into multiple messages and transfers them to several abstract software services.

3) $\tau = (F, Ru)$ is a convergent relation. $F = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ consists of mapping functions between business services and abstract software services, between control logics at the two levels, between service associations and between business information entities and software messages. Ru de-

notes a set of convergent rules that contain auxiliary information needed to implement F .

- $\tau_1 : B_A \xrightarrow{Ru} A_S$ sets up relationships between business activities and abstract software services. Because a business activity is bound with a business service according to B_R^2 and each business service has a definite counterpart (namely an abstract software service) in software-level models according to our convergent rules, this function can map between business activities and abstract software services in a straightforward manner. Here, business services are extracted from abstract software services in terms of the convergent rules and take on the business view of abstract software services. This is the basic convergent point of the business domain and the software domain in CAFISE.

- $\tau_2 : B_R^1 \xrightarrow{Ru} A_R^1$ maps relationships between business activities to control relationships between abstract software services.

- $\tau_3 : B_R^2 \xrightarrow{Ru} A_R^2$ generates the associations between abstract software service declarations and concrete software services based on B_R^2 as well as rules for service abstraction and generalization.

- $\tau_4 : B_R^3 \xrightarrow{Ru} A_R^3$ translates relationships between business activities and business information entities into message relationships between abstract software services. For example, if two business information entities are produced by a business activity X and a business activity Y respectively, and consumed by a business activity Z , we can get the *synthesized* message relationship between the corresponding abstract software service x , abstract software service y and abstract software service z . Here, the abstract software services x , y and z can be identified through τ_1 . The mappings from other relationships of B_R^3 to *direct* and *decomposed* relationships of A_R^3 are similar.

3.2 Constructing Convergent Models with the CAFISE Language






To describe and reify the CAFISE model presented above, the CAFISE language is designed. It is made up of two parts: a graphic business-level language segment and an XML-based software-level language segment. The convergent relation τ is implemented by the CAFISE framework discussed later.

3.2.1 Business-Level Specification

The business-level language segment of CAFISE, called CBL, is designed to be “user-

centric” to enable business people to make on-demand configuration of service grid applications. CBL intends to allow a business expert to transparently examine all dispersed computational resources available to him as business objects and to configure them in an easy and straightforward manner. We introduce some simple graphical symbols for business users to draw his or her personalized pictures of the businesses in concern. Through dragging and configuring these symbols, business users can, to certain extent, program their “applications” quickly according to their personal needs.

Table 1. Graphical Symbols for Business-Level Specification — CBL

Symbol	Description
	Business activity It represents a business activity, which defines what to do at a business step.
	Business service Business services are organized in advance (see Fig.2) and are used to perform business activities. Users can choose and drag them to set up associations with business activities.
	Conditional sequence It links two business activities and denotes that these activities are performed sequentially under certain conditions.
	Conditional split Some of the business activities behind this point can be performed concurrently under certain conditions.
	Conditional join Business activities before this point need to be finished only partially under certain conditions.

The graphical symbols and their descriptions are given in Table 1. Business information entities that business activities consume or produce are specified as attributes of business activities, and thus do not appear as stand-alone icons. A link may be associated with a set of conditions. The condition is defined as attributes of the link and expressed as a Boolean expression.

3.2.2 Software-Level Specification

The software-level language segment of CAFISE, called CSL, is expressed in XML format so that the specifications can be exchanged easily across a wide-area network. Due to the space limitation of the paper, we only present here some key elements of CSL, which are listed in Table 2. Details of a complete definition of the CAFISE language can be found in [17].

Table 2. Key Elements for Software-Level Specification — CSL

Language elements	Description
\langle AbstractService \rangle	It denotes an abstract software service.
\langle StaticBinding \rangle	It defines the access information of a software service to be bound to an abstract software service, such as a pointer to a WSDL file.
\langle DynamicBinding \rangle	It defines the policy for dynamic service binding.
\langle FunRestrictions \rangle	It describes functional restrictions of input and output messages used for binding a concrete software service at runtime.
\langle NonFunRestrictions \rangle	It describes non-functional properties of services, which can be used to bind a concrete software service at runtime.
\langle ControlLink \rangle	It defines directed links between abstract software services.
\langle ConditionalSequence \rangle	It establishes a sequential relationship between two abstract software services with attribute of conditions.
\langle ConditionalSplit \rangle	It corresponds to a conditional split. Abstract software services behind this link can only be executed partially according to the attribute of conditions.
\langle ConditionalJoin \rangle	It allows a partial join according to the attribute of conditions without synchronizing all previous services.
\langle Message \rangle	It defines messages to be exchanged between abstract software services, including parameters and protocol-related data.
\langle Expression \rangle	There are two types of expressions: Boolean expression and assignment expression. A Boolean expression can be used to define condition attributes of \langle ConditionalSequence \rangle , \langle ConditionalSplit \rangle , and \langle ConditionalJoin \rangle . An assignment expression is used to assign values to a message. (XPath 1.0 ^[19] is used as the expression language.)

The development of CSL benefits from BPEL4WS^[18] that is an executable business process language facilitating Web services composition. In BPEL4WS, the executable processes correspond to composite services that are modeled as a directed graph. BPEL4WS focuses on specifying what composite services do and how they work, and does not yet provide enough support for dynamically reconfiguring the service composition when requirements change.

In CSL, the associations between abstract software services and concrete software services can be defined either in advance or at runtime. With dy-

amic service binding, concrete software services that implement abstract software services can be chosen and composed at run-time. It brings along greater flexibility to choose the most suitable services from multiple candidates and to handle exceptions in invoking remote services. As shown in Table 2, the language element \langle DynamicBinding \rangle is provided for specifying all necessary dynamic binding information. Since service binding policies are relevant to domain knowledge, it is hard to define a universally applicable set of descriptions. An extendable policy description and evaluation mechanism is explicitly introduced and a policy evaluator is provided inside the \langle NonFunRestrictions \rangle element. Through this mechanism, users can define policies by themselves.

After having introduced the CAFISE model and the language thereof, let us summarize the principles of CAFISE modeling as well as the usage of the language. First, users can use the business-level language to build business-level model on their demand. They only need to drag and configure the symbols to define their business requirements using predefined modeling elements and rules. Based on the resulting business-level models, software-level models can be derived in terms of the convergent relation τ with the support of a predefined set of CAFISE rules as well as the CAFISE framework that is to be discussed in the subsequent section. Software-level models are interpreted directly by a core component of the CAFISE framework, called the CAFISE virtual machine to realize the so-called just-in-time configuration or dynamic reconfiguration to meet users' spontaneous requirements.

4 CAFISE Framework

The CAFISE framework is designed to realize our approach and enable on-demand configuration and dynamic reconfiguration of service grid applications. As depicted in Fig.3, the CAFISE framework is composed of a set of core components and front-end tools assisting users to configure and reconfigure service grid applications. The convergent modeling environment offers power users a means to express their requirements from business viewpoint and transform resulting business models to configuration specifications in the CAFISE language. The reconfiguration component takes the responsibility for managing dynamic changes and reconfiguring applications while applications are still in execution. The CAFISE virtual machine provides support for the execution and final

adaptation of service grid applications. In the rest

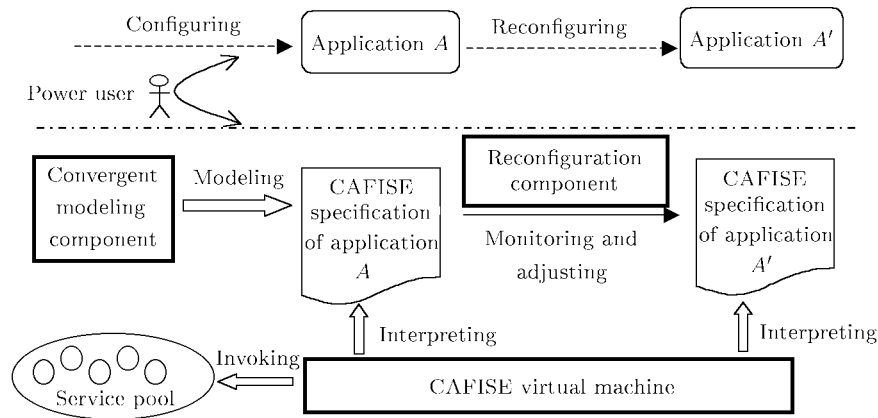


Fig.3. The CAFISE framework.

of this section, we will briefly examine the adaptive configuration process of service grid applications with the help of the CAFISE framework.

Through the convergent modeling environment, power users can design business-level models to express their demand in a just-in-time manner. The graphical tool helps power users to browse the network-based resources made available to them, and to configure their applications by correlating the resources and thus defining the business-level models. The corresponding “executable” software-level models are derived from the resulting business models. As shown in Fig.3, power users can use the convergent modeling component to configure and reconfigure applications, which helps to get the CAFISE-enabled application – A to be modified and evolved to a new application – A’.

The component — CAFISE virtual machine — is responsible for interpreting the software-level models and invoking individual services. In case that dynamic binding mode is specified or exceptions arise during service invocation, the CAFISE virtual machine needs to evaluate the most suitable services from possible candidates. In addition to these main tasks, the CAFISE virtual machine also provides the following functionalities. It helps to maintain a meta-data space containing collections of structural and behavioral meta-data of the “executing” applications. The concepts and techniques of reflection are used to construct and maintain a set of meta objects that are designed for the management of dynamic changes and reconfiguration. In the past years, reflection has been one of the most useful techniques for developing adaptable systems^[20]. Dynamic reconfiguration of ser-

vice grid applications can benefit from the concepts and techniques of reflection, in particular the concepts and protocols of meta objects^[21,22]. A meta object protocol is a supplementary interface to conventional object models. It provides an effective means to incrementally modify the behavior and implementation.

When changes to a business take place while its application is running, a power user can use the reconfiguration component to adjust the application’s configuration to reflect the changes. The reconfiguration component monitors the running status of the application. Each time a power user wants to make a reconfiguration, the component can reproduce the business-level models and present the most state-of-the-art status of the running application, and allow the power user to make legal modifications through related meta object protocols. We reify two aspects of the business-level models of CAFISE, the structural aspect and the behavioral aspect, to build two sorts of corresponding meta objects, namely structural meta objects and behavioral meta objects respectively. A structural meta object captures topological information through extracting the relation among business activities. A behavioral meta object captures how business activities act through extracting relationships between business activities and business services, and relationships between business activities and business information entities. At the same time, a set of operations is defined based on these two sorts of meta objects to support dynamic application reconfiguration. The operations defined on the structural meta objects are used to insert and delete business activities, business information

entities as well as business services. The operations defined on the behavioral meta objects are used to adjust relationships between business activities and the corresponding business information entities or business services. Through these meta objects and the operations, a basic ability for dynamically reconfiguring service grid applications is obtained.

5 Case Study with an Application Scenario

The CAFISE approach has been applied within the context of a real project called FLAME2008. FLAME2008 (A Flexible Semantic Web Service Management Environment for the Olympic Games Beijing 2008) is a project under the Chinese Digital Olympiad Framework initiative supported by China's Ministry of Science and Technology and Chinese Academy of Sciences. The project aims at developing a service-oriented application platform, on which an effective information system providing integrated, personalized information services to the public can be based. In such a large-scale information system, many applications providing services are involved and there are numerous individual requirements from various users. It requires the information system to adapt to changing requirements by quickly (re)configuring different service resources. To illustrate how the CAFISE approach works, we use the following simplified scenario: Mr. John Bull is a sport reporter and will come to Beijing for interview during the Olympic Games 2008. While watching matches, making interviews and going sightseeing, he can seek help and acquire personalized services from this information system. Based on his demand, the information system composes dispersed services over the Internet to provide one-step information services to him. With the support of the CAFISE approach and its supporting mechanisms, John can construct his personal application in a just-in-time manner through directly composing network-based services on his own demand.

5.1 Just-in-Time Configuration of Service Grid Applications

John can switch his role as a power user or as an end user. For example, as a power user, he may configure spontaneously his personalized application supporting his daily schedule. As an end user he can use the application he builds. To build the personal application for his travel planning, he defines several business activities and their

relationships. The business activities are named as *order airline ticket*, *query match ticket*, and *order match ticket* respectively. Then, he can check what kinds of business services are available, choose and then drag business services to the corresponding business activities to complete the specification. The business services are abstracted and organized based on related software services over the Internet. The assemblies of business services are organized as service communities^[16] according to access rights, preferences, and other organization principles. In this simplified scenario, the following business services from registered software services may be useful for him: *airline ticket booking*, *Olympic information inquiry service* and *Olympic ticket application service*. John arranges the business activities in a certain order (using the mechanisms defined in CBL), chooses suitable business services and associates the chosen business services with the business activities he defined. As such, he defines a simple business-level CAFISE model.

The visualized view of business-level model built with the CAFISE modeling environment is shown in the right corner at the bottom of Fig.4. After John's requirements are captured in the intuitive way, it is the CAFISE framework's job to derive an "executable" software model. The CAFISE framework, which implements the convergent relation τ and embodies the basic convergent rules, translates the resulted business-level model into a software-level CAFISE specification. The CAFISE virtual machine can then interpret the specification when an end user, either John or someone else, calls the application.



Fig.4. The power user interface reflecting the business-level view of the scenario.

5.2 Dynamic Reconfiguration of Service Grid Applications against Business Changes

While using the application he built, John finds that if he does not know the detailed event schedule of match, he cannot inquire match tickets conveniently. If he knows the match schedule before querying match ticket, this problem can be solved. So, he may want to adjust the application according to his new demand at runtime. With CAFISE, John, as a power user, can reconfigure his application dynamically. He may insert and configure a new business activity (*get match schedule*, for example), before the business activity *query match ticket*, with the reconfiguration tool. Among possible candidates offering schedule information, the most suitable business service can be chosen according to a flexible scheme for service selection to fulfill this business activity. Fig.5 illustrates the adjustment process of business-level model. Then, the software-level specification is altered automatically to reflect the changes. The CAFISE framework will check and verify the validity of the changes. Finally, the CAFISE virtual machine will continue interpreting and executing the changed specification.



Fig.5. An example of business-level model adjustment.

6 Concluding Remarks

Since a service-grid-based application is subject to just-in-time construction and changes, how to make a service-grid-based application user-configurable becomes a challenging issue. Based on an integrated model that relates some key elements of a business-level design issues and their counterparts in the software domain in convergent

way, CAFISE promotes the metaphor of user-end programming and enables demand-driven and user-centric configuration of service grid applications. The development of CAFISE is guided by a number of real-world scenarios to highlight its practical usefulness. The approach and the CAFISE framework are used in the FLAME2008 project to allow a business expert to transparently examine all scattered resources that are accessible and available to him or her and to configure and reconfigure them in an easy and straightforward manner.

In FLAME2008, we focus on the issues of service composition and dynamic service binding in the software-level CAFISE model. As a future work, we will extent the mechanisms to cover service deployment and service-level interactions. Also, the convergent relation between business-level models and the software-level models in FLAME2008 is straightforward to some extent. At present, the convergent rule set *Ru* mentioned in Section 3 takes into consideration some predefined business rules and some basic relationships between business-level models and software-level models in a fixed manner. Constructing and leveraging a fully-fledged and easily maintainable rule set are one of the most important goals of our undergoing research work. The CAFISE approach is based on a grid-oriented resource organization. Though we did not discuss issues of resource organization in details in the paper, methodological and tool supports for the service community mentioned earlier are among the key research themes of our research group.

References

- [1] Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. Morgan-Kaufman Publishers, July 1998.
- [2] Foster I, Kesselman C, Tuecke S. The anatomy of the grid: Enabling scalable virtual organizations. *The International Journal of Supercomputer Applications*, 2001, 15(3): 200–222.
- [3] Reinefeld A, Schintke F. Concepts and technologies for a worldwide grid infrastructure. In *Euro-Par 2002 Parallel Processing*, Lecture Notes in Computer Science 2400, Springer, 2002, pp.62–71.
- [4] Weissman J B, Lee B. The service grid: Supporting scalable heterogeneous services in wide-area networks. In *Proc. Symp. Applications and the Internet*, San Diego, CA, January 2001, pp.95–104.
- [5] Kramer J. Configuration programming — A framework for the development of distributable systems. In *Proc. IEEE International Conference on Computer Systems and Software Engineering (COMPEURO 90)*, Tel-Aviv, Israel, May 1990, pp.374–384.
- [6] Lim A S. Abstraction and composition techniques for

- reconfiguration of large-scale complex applications. In *Proc. the 3rd International Conference on Configurable Distributed Systems*, Annapolis, Maryland, USA, May 1996, pp.186–193.
- [7] Oreizy P, Gorlick M, Taylor R N *et al.* An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 1999, 14(3): pp.54–62.
- [8] Oreizy P, Taylor R N. On the role of software architectures in runtime system reconfiguration. In *Proc. the 4th Int. Conf. Configurable Distributed Systems*, IEEE Computer Society Press, Annapolis, Maryland, USA, May 1998, pp.61–70.
- [9] Shrivastava S K, Wheeler S M. Architectural support for dynamic reconfiguration of large scale distributed applications. In *the 4th Int. Conf. Configurable Distributed Systems (CDS'98)*, Annapolis, Maryland, USA, May 4–6, 1998, pp.10–17.
- [10] Warren I, Sommerville I. A model for dynamic configuration which preserves application integrity. In *Proc. the 3rd Int. Conf. Configurable Distributed Systems*, IEEE Computer Society Press, Annapolis, Maryland, USA, May 1996, pp.81–88.
- [11] Taylor D. *Business Engineering with Object Technology*. John Wiley & Sons, 1995.
- [12] Hubert R. *Convergent Architecture: Building Model-Driven J2EE Systems with UML*. New York: John Wiley & Sons, 2002.
- [13] Singh M P. Physics of service composition. *IEEE Internet Computing*, May & June 2001, pp.6–7.
- [14] Kiciman E, Melloul L L, Fox A. Towards zero-code service composition. In *Proc. the Eighth Workshop on Hot Topics in Operating Systems (HotOS VIII)*, Germany, 2001, p.172.
- [15] Casati F, Ilnicki S, Jin L *et al.* Adaptive and Dynamic Service Composition in eFlow. HP Labs Technical Report, HPL-200039, Software Technology Laboratory, Palo Alto, CA, March 2000.
- [16] Benatallah B, Dumas M, Sheng Q Z, Ngu A H H. Declarative composition and peer-to-peer provisioning of dynamic web services. In *Proc. the Int. IEEE Conf. Data Engineering*, San Jose, USA, February 2002, pp.297–308.
- [17] CAFISE group. CAFISE Language Specification. Technical Report, Software Division, ICT of CAS, 2002.
- [18] BPEL4WS (Business Process Execution Language for Web Services), Version 1.1. <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>, 2003.
- [19] XPath 1.0. <http://www.w3.org/TR/1999/REC-xpath-19991116>, 1999.
- [20] Maes P. Concepts and experiments in computation reflection. *ACM SIGPLAN Notices*, Dec. 1987, 22(12): 147–155.
- [21] Kickzales G, Rivieres J, Bobrow D G. *The Art of the Metaobject Protocol*. MIT Press, Cambridge, Massachusetts, 1991.
- [22] Edmond D, Hofstede A T. Achieving workflow adaptability by means of reflection. In *Proc. the ACM Conf. Computer Supported Cooperative Work (Workshop on Adaptive Workflow Systems) CSCW'98*, Seattle, Nov. 1998, available at <http://ccs.mit.edu/klein/cscw98/>.
- HAN YanBo** is a professor of the Institute of Computing Technology, the Chinese Academy of Sciences. He holds the Ph.D. degree received from the Technical University of Berlin, Germany. His current research interests are middleware and software integration technologies, service-oriented computing, and software engineering of Internet-based applications.
- ZHAO ZhuoFeng** is a Ph.D. candidate at the Institute of Computing Technology, the Chinese Academy of Sciences. His research interests are service composition, service-oriented application and workflow technologies.
- LI Gang** received his Ph.D. degree in computer science from the Beijing University of Aeronautics and Astronautics. He is a research assistant of ICT, the Chinese Academy of Sciences. His research interests are service-oriented grid computing, adaptive software architecture and software evolution.
- XING DongShan** received his Ph.D. degree in computer theory and application from Xi'an Jiaotong University. He is a post-doc research assistant at ICT, the Chinese Academy of Sciences. His research interests are web mining and software evolution.
- LV QingZhong** is a Ph.D. candidate at the Institute of Computer Engineering, Beijing University of Aeronautics and Astronautics. His research interests are software engineering, knowledge representation and ontology, and semantics enabled web services.
- WANG JianWu** is a Ph.D. candidate at the Institute of Computing Technology, the Chinese Academy of Sciences. His research interests are service-oriented application, workflow technologies and service grids.
- XIONG JinHua** is an associate professor of the Institute of Computing Technology, the Chinese Academy of Sciences. His current research interests are software integration technologies, service-oriented computing, and business-modeling technologies.
- LIU Hao** is a Ph.D. candidate at the Institute of Computing Technology, the Chinese Academy of Sciences. His research interests are software integration, software architecture and workflow technologies.