Facilitating Dynamic Service Compositions by Adaptable Service Connectors

Gang Li, Institute of Computing Technology, China

Yanbo Han, Institute of Computing Technology, China

Zhuofeng Zhao, Institute of Computing Technology, China

Jing Wang, Institute of Computing Technology and Graduate School of the Chinese Academy of Sciences, China

Roland M. Wagner, Fraunhofer Institute for Software and Systems Engineering, Germany

ABSTRACT

Dynamic changes of services and requirements require service connection relationships adaptable in service compositions. This paper presents an adaptable service connector model and related language and tools. The model presents service connection relationships as an explicit component with which service connections can be reconfigured and changes of services involved in the interaction can be handled; this makes the service connection relationships more adaptive. With the language and tools supporting the model, services can be dynamically chained, which make service-oriented applications adapt to volatile business requirements and dynamic changeable services. The related case study and evaluation are also presented in this paper.

Keywords: *dynamic service composition; service connection; Web services*

INTRODUCTION

As everyone knows, constructing or integrating applications by service compositions is becoming a promising approach for developing distributed applications. However, because the new network environments, like grids, are more and more dynamic and open (Foster, Kesselman, Nick, & Tuecke, 2002; Foster, Kesselman, & Tuecke, 2001), the development, usage, and maintenance of service-oriented applications are facing several challenges.

- **Autonomic evolution of services.** In network environments, services are autonomic. This characteristic usually behaves in different ways. For example, the service providers can independently develop and share their services. The services in an application may come from different providers, and their availabilities can autonomously change (Benatallah, Dumas, Sheng, & Ngu, 2002). At the same time, autonomic services can freely join and quit the networks. In addition, many factors, like new technologies and business model changes and so forth also cause the providers to actively change the functionalities and QoS (such as usability, accessibility, performance, security, reliability and cost, etc.) of services. All of these can affect a service composition and even damage the original service connection relationships.
- **Dynamic changes of user requirements.** The problems caused by requirement changes still exist, and the user requirements are more dynamic in network environments (Zeng, Benatallah, & Ngu, 2001). For example, by sharing resources and coordinating services through a network, different enterprises can compose a virtual enterprise. In this virtual organization, business partners and business logic are dynamically changing, with business evolving. Obviously, the applications are required to be dynamically reconfigured.

When the aforesaid situations emerge, what a user most cares about is whether the applications can still satisfy his requirements without much additional cost. So the services should be dynamically connected, and the compositions should be dynamically reconfigured.

Our goal is to give an adaptable method for facilitating dynamic service compositions. Therefore, we present a service connector model by which a service connection can be designed as an explicit component. With stable interaction interfaces and changeable connection structures, this type of connector facilitates dynamic service compositions by structure reconfiguration. With the languages and tools that support the service connector model, one can not only dynamically chain services but also make service-oriented applications adapt to the changes of services and requirements.

The rest of this paper is organized as follows. First, we discuss the service connection adaptation issues with a case. Second, we present the service connector model. Third, the connection semantic of the connector is given and the related theorem is proved to theoretically clarify that the service connector is adaptable. Fourth, the language and tools that support dynamically chaining services with the service connector are presented. Finally, we present a case study with evaluation and draw our conclusion.

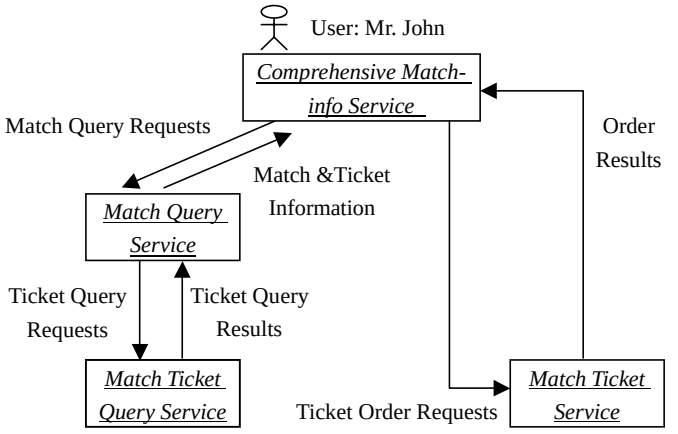
PROBLEM ANALYSIS

In this section, we discuss the service connection adaptation problems through analyzing a case in project FLAME2008[1], which is presented in Figure 1. In this case, users who visit Beijing during the Olympic Games 2008, like Mr. John, want to get comprehensive information about match schedules and tickets. The services involved in the application are listed as follows:

- Match Query Service: Retrieves the match schedules and returns information that users are interested in. It interacts with the Match Ticket Query Service.
- Match Ticket Query Service: Returns match ticket information.
- Match Ticket Service: Orders the tickets if there are remains.
- Comprehensive Match-info Service: Coordinates the previous services to get match schedules and ticket information and order tickets.

When constructing the first prototype with these services, we've met the following problems.

Figure 1. A Case from Project FLAME2008



1. In 2008, some services will be rented from professional service providers. And when a service is unavailable because of changes, such as rent price changes and so forth, other requirement-satisfied service has to dynamically replace the invalid one. However, the first prototype cannot solve this problem well.
2. The Match Query Service retrieves match schedules by match name. To support compound condition retrieving, the service should be updated. In fact, the updating service problem always exists with requirements changing.
3. In 2008, several millions of athletes, coaches, businessmen, spectators, journalists, and other groups of people from all over the world will rush into Beijing in fewer than 20 days. However, the stress testing shows that access bottlenecks exist at some services such as Match Query Service when numerous users simultaneously access the system. If the required services can be dynamically selected from a group of available ones, the access bottlenecks may be avoided or delayed.

To solve the these problems, we need to take the following requirements into consideration:

1. New services can be deployed and added without influencing other parts of the application, so the application can be dynamically reconfigured.
2. The services can be dynamically chained, and service connection relationships can be refreshed to form requirement-satisfied service compositions in a just-in-time way.

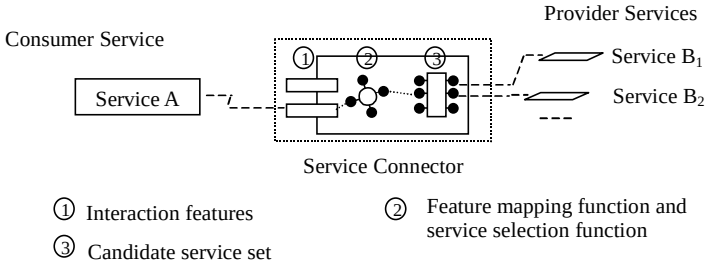
Motivated by the previous challenges from the real world, we present the service connector model.

SERVICE CONNECTOR MODEL

Figure 2 illustrates a service connector through an example in which Service A interacts with others through the service connector. The connector has two kinds of stateless services involved in service interaction. One is named a provider service, such as Service B1, which is requested by other services. And another is named a consumer service, like Service A, which requests something of the provider services. A group of provider services with same function, which compose a candidate service set, is abstracted to a virtual service in the service connector model. In a service composition, the connector indirectly chains the consumer service, and the provider services through the virtual service.

A service connector exposes the virtual service to a consumer service through the connector's interaction features. The interaction features, which provide functions from the

Figure 2. Service connector conception



encapsulated provider services, are relatively stable. But the relationship between the features and the provider services can be modified. So the interaction features, as well as the mapping and selecting relationships between the interaction features and provider services, constitute a changeable connection structure, which facilitates connection adaptation.

By reconfiguring the interaction features, the provider services, or the relations between the features and providers, connection can adapt to the unforeseen changes of services or requirements. For example, when a provider service is invalid, another provider service that implements the same interaction features can dynamically replace the invalid one under the control of the connector. Even the users do not know those changes, and the connection is still available to the users.

The service connector realizes dynamic interactions between services. With unified and stable interaction interfaces and changeable connection structures, the connector provides essential supports for dynamic service compositions.

Definition 1. Service Connector

Given a three-tuple $<Name_c, Features_c, Services_c>$, where $Name_c$ marks the connector name; $Features_c$ is a set of interaction features, $Features_c = \{f_c | f_c = <cn, fn_c, va_c>\}$, where $Features_c \neq \Phi$, *cn* denotes the connector that feature f_c belongs to, fn_c is the name of f_c, va_c = <*inputset, outputset nonfuncattrs*> and it is f_c's argument vector that includes a set of input, output parameters and quantified nonfunctional attributes. The interaction features are interfaced, through which a connector interacts with the outside; $Services_c \neq \Phi$ is a candidate service set that includes a set of provider service references that implement the interaction features.

The three-tuple defines a service connector if and only if it has the following properties:

1. There is a function set, denoted as *Map*, $Map \neq \Phi$. Given $\forall f_c \in Features_c$ then $\exists m \in Map$, $Services_{fc} \subseteq Services_c$, $Services_{fc} \neq \Phi$, $m(f_c) = Services_{fc}$. $\forall s, s' \in Services_{fc}$, *s* and *s*′have the same interaction interfaces specified by f_c.
2. There is another function set, denoted as *Selectors*, $Selectors \neq \Phi$. Consider f_c, $\exists sel \in Selectors$, $s \in Services_{fc}$, $sel(Services_{fc}) = s$.

The function *m* is named as feature mapping function, and the function *sel* is named as service selection function. They are used to match and choose the requirement-satisfied services. Herein, the selection algorithm is presented as Algorithm 1.

Definition 1 precisely and concisely presents a service connector from the structure viewpoint in a formal way, including the $Features_c$, $Service_c$, feature mapping, and service selection functions. By dynamically selecting and configuring those key structure elements, the connection relationships in a service connector are adaptable, which can be deduced with the category theory.

A category consists of object sets and sets of morphisms between objects, which focuses on describing and analyzing relations

Algorithm 1. Service selection algorithm ser_sel (inputSet, outputSet, nonfuncAttrs, candidateServiceSet)

Input: *inputSet, outputSet, candidateServiceSet*;
Output: *selectedService;*

Process:
(1) Set *matchedServiceSet* = | μ
(2) for ∀ *s* ∈ *candidateServiceSet*; {
(3) If ((*s.outputs* ⊇ *outputSet*)&&(*s.inputs* ⊆ *inputSet)*
(4) &&(*satisfied(nonfuncAttrs*)= =True)
(5) *matchedServiceSet = matchedServiceSet + {s};*
(6) }
(7) if (| *matchedServiceSet* | = = 0)
(8) return null; // no matched services
(9) else
(10) ∃ *selectedService*∈ *matchedServiceSet*;
(11) return *selectedService*; //get the selected service

among any type of objects (Goguen, 1991). With categories, relations among services and connectors can be described formally, which offers an approach for analyzing and deducing properties of the service connector without considering implementation details. Herein, the connection semantic of the service connector is presented in categories that take structured sets as objects.

Let *Service* be one of the consumer services involved in the interaction, $Service = <Name_s, Features_s>$, where $Name_s$ is the service name, $Features_s = \{f_s | f_s = <sn, fn_s, va_s>\}$, and f_s is the service feature that describes a service port. Take *Service*s as objects and construct a category *Serv*.

$Serv = <objC_{Serv}, MorC_{Serv}, dom_s, cod_s, \circ>$, where the set $objC_{Ser}$ is the class of objects in *Serv*; given morphism φ: $objC_{Serv} \rightarrow objC_{Serv}$. $E, F \in objC_{Serv}$, $\varphi(E)=F$, φ is specified by the following:

$\delta_1: Name_E \rightarrow Name_F$
$\delta_2: Feature_E \rightarrow Feature_F$

The set $MorC_{Serv}$ is the class of morphisms in *Serv*; namely $MorC_{Serv}$ is the set of connections between *Services*. The morphisms in $MorC_{Serv}$ are defined by φ.

The function dom_s is defined as $dom_s: MorC_{Serv} \rightarrow objC_{Serv}$, where $f \in MorC_{Serv}$, then $dom_s(f)$ denotes the domain of *f*.

The function cod_s is defined as $cod_s: MorC_{Serv} \rightarrow objC_{Serv}$, where $f \in MorC_{Serv}$, then $cod_s(f)$ denotes the codomain of *f*.

The symbol ∘ denotes an operation, which is defined as $\circ: MorC_{Serv} \times MorC_{Serv} \rightarrow MorC_{Serv}$.

Let *Connector* be one of the service connectors involved in the interaction, $Connector = <Name_r, Features_r, Service_r>$. In a similar way, take *Connector*s as objects and construct the category *Conn*, $Conn = <objR_{Conn}, MorR_{Conn}, dom_c, cod_c, *>$, where morphism y is used to define $MorR_{Conn}$ and describes interactions between connectors.

Let $Ser \in objC_{Serv}$, $Con \in objR_{Conn}$; constructs function F_{SC}, $F_{SC}: objC_{Serv} \rightarrow objR_{Conn}$, $F_{SC}(Ser) = Con$, F_{SC} is specified by the following:

$h_1: Name_{Ser} \rightarrow Name_{Con}$
$h_2: Feature_{Ser} \rightarrow Feature_{Con}$

So, the connection fulfilled by a service connector can be defined as $Connection_c$. $Connection_c = \{<Ser, F_{SC}(Ser)>\} \cup \{<Con, \psi(Con)>\}$.

Given the connection semantic, we can now prove the following results:

Theorem 1

Given a service connector *c*, the service set $Services_{fx} \subseteq Services_c$, $\exists\, m \in Map_c$, $m(f_x) = Services_{fx}$, and $\exists\, sel \in Selectors_c$, $sel(Services_{fx}) \in Services_{fx}$. In the interactions where the service connector *c* is involved, the service connection has the following properties:

1. If the service set $Services_{fx} \neq \Phi$, adding elements into $Services_{fx}$ does not cause invalid connection;
2. If the service set $Services_{fx} \neq \Phi$ after deleting an element in it, deleting elements in $Services_{fx}$ does not cause invalid connection;
3. If the service set $Services_{fx} \neq \Phi$, replacement of *sel* does not cause invalid connection;
4. If an interaction feature changes, the connection can stay available by reconfiguring the service set $Services_{fx}$.

Proof

1) $\because Services_{fx} \neq \Phi$
$\therefore \exists\, s,\ s \in Services_{fx}$, *s* interacts with the consumer services through interaction feature f_x. Let s' be the new added service, $s' \in Services_{fx}$.
$\because s,\ s' \in Services_{fx}$
$\therefore$ according to the definition of $Services_{fx}$, *s*, s' implement the same interaction feature f_x
$\because f_x$ is not changed
$\therefore$ the addition of s' does not cause invalid connection
$\therefore$ Proposition 1) follows.

2) Let *s* be the deleted service, $s \in Services_{fx}$
$\because$ after deleting, the service set $Services_{fx} \neq \Phi$
$\therefore \exists\, sel(Services_{fx}) = s',\ s' \in Services_{fx}$, s' interacts with the consumer services through the same interaction feature f_x as *s*
$\therefore$ the deletion of *s* does not cause invalid connection
$\therefore$ Proposition 2) follows.

3) Let sel' be the new function that replaces *sel*, $sel' \in Selectors_c$, and $s \in Services_{fx}$, *s* is the service involved the interaction
$\because Service_c \neq \Phi$
$\therefore$ after sel' replacing *sel*, $\exists\, sel'(Services_{fx}) = s',\ s' \in Services_{fx}$, s' interacts with the consumer services through the same interaction feature f_x
$\therefore$ the replacement of *sel* does not cause invalid connection
$\therefore$ Proposition 3) follows.

4) Let *s* be the consumer service that interacts with connector *c*, and f_s is the port through which the consumer service *s* interacts with the connector *c*. According to F_{SC}, we conclude:
$\exists\, \langle f_s, f_x \rangle,\ f_x \in Features_c$, if $\langle f_s, f_x \rangle$ changes into $\langle f_s, f'_x \rangle$, according to the definition of *c*, $\exists\, m' \in Map_c$, such that $Services_{fx}' \subseteq Services_c$, $m'(f'_x) = Services_{fx}'$. So, after f'_x replaces f_x, and $Services_{fx}'$ replaces $Services_{fx}$, m' and $Services_{fx}'$ keep the connection available.
$\therefore$ Proposition 4) follows.

The proof is now completed.

Theorem 1 indicates that a service connector is adaptable. On one side, a service connector can set up loose and flexible connection relationships by dynamically selecting services. On the other side, the connections can be dynamically reconfigured through the following operations.

- Reconfiguring the candidate service set by adding or deleting services;
- Changing service selection policies by replacing service selection function;
- Changing interaction interfaces and composition participants by reconfiguring interaction features or candidate service sets.

DARNEL: A LANGUAGE WITH ADAPTABLE SERVICE CONNECTORS

To depict dynamic service compositions with the service connectors, we present a language named Darnel (**D**yn**a**mic Se**r**vice Composition **D**e**s**cription **L**anguage), which is an XML-based description language that provides constructs for describing the service connector.

Language Constructs

In Darnel, a service composition consists of a set of services, activities, and service connectors. The services, as computational components distributed on the Internet, are basic functional units, and the activities are structural units of an application. There are two kinds of activities: structural activities and simple activities. The structural activities are used to describe control logic of an application. Organized by structural activities, the simple activities act as placeholders where specific service capabilities are needed to meet user requirements. To offer the capabilities, the simple activities can statically or dynamically associate with services by a service connector. Another key element of Darnel is comprised by a set of coordinating activities, named coordination, which provide a way of controlling the structure complexity of a service composition.

In addition, the element of context is included in Darnel to describe environment information related to an application. By context, a service connector defined in Darnel can dynamically select service while environments changing.

Figure 3 presents the outline of a service-oriented application in Darnel with Darnel's abstract syntax.

Compared with other service composition languages such as BPEL4WS (Andrews, Curbera, Dholakia, Goland, Klein, Leymann, Liu, Roller, Smith, Thatte, Trickovic, & Weerawarana, 2003), GSFL (Krishnan, Wagstrom, & Laszewski, 2002), and so forth, Darnel has obvious differences: it has a language construct named as a connector, which directly facilitates the explicit description and the adjustment of dynamic service connection relationships. Figure 4 presents an example of the service connector in Darnel.

Figure 3. Outline of a service-oriented application in Darnel

```
application = version
      context
      coordination +
coordination = co-name
      sub_context ?
      simple activity+ | structured activity*
      [coordination]
simple activity = activity-name
      service static binding | connector
connector = con-name
      feature +
      selection function
      candidate service set
feature = feature-name
      input*
      output*
candidate service set = set-name
      candidate service+
```

As Figure 4 shows, the connector construct materializes the service connector model on the language level, which consists of features, selection function, and candidate service set. The features are language constructs to materialize the interaction features of the service connector model, and they are described by the tags <Feature> … </Feature>. As the language construct for describing provider services specified by a feature, the candidate service set, which is notated by tags <CandidateServices> … </CandidateServices>, includes a set of services that interact with the consumer services through a feature. Marked by tags <SelectionFunction>… </SelectionFunction>, the selection function specifies the service selection policies and implements the feature mapping and service selection functions, with which a service connector dynamically selects a requirement-satisfied service from the candidate service set and connects it with the consumer services.

Figure 4. The service connector in Darnel

```
<Connector Name="schedulingCon" >
  <Feature Name=" reqScheduling ">
    ……
  </Feature>
  <CandidateServices>
      ……
      <Service>
        NewSchedulingService
      </Service>
  </CandidateServices>
  <SelectionFunction>
      ……
  </SelectionFunction>
</Connector>
<SimpleActivity name="scheduling"
    connector="schedulingCon">
```

Figure 5. The PartnerLink in BPEL4WS

```
<PartnerLinkType
    name="schedulingLT">
  <Role name="schedulingService">
    <PortType name="
        schedulingPT"/>
  </Role>
</PartnerLinkType>
<PartnerLink name="scheduling"
  partnerLinkType="schedulingLT"
  partnerRole="schedulingService
"/>
<Invoke partnerLink="scheduling"
  portType="schedulingPT"
  operation="requestProductionSc
heduling"
  inputVariable="PO">
</Invoke>
```

Dynamic Features of Darnel

With Darnel, the service connection, which can be dynamically changed through altering feature specifications, reconfiguring selection function, and adjusting the candidate service set, is manifested as follows.

1. The connector in Darnel wraps the provider service changes in an interaction. That means the consumer services connect to a virtual service. So, the provider services can join or quit the coordination without affecting all other interaction participants.

Comparing with other service composition languages, such as BPEL4WS, Darnel can define a more flexible service-oriented application. In BPEL4WS, a service connection relationship is defined with a PartnerLink. When a service is invoked, the PartnerLink is used in an Invoke activity. Figure 5 shows a connection for invoking a "scheduling service" defined in BPEL4WS.

In BPEL4WS, although one can explicitly describe service connection with PartnerLink and separate connecting services from using services, there are still some shortages. For example, as shown in Figure 5, the PartnerLink "scheduling" is an instance of PartnerLinkType "schedulingLT" that is defined according to the PortType in WSDL file of "scheduling service." In this connection, the Invoke activity also has to specify its PortType. By this way, the service connection is fixed virtually. When the "scheduling service" needs to be substituted, the PortTypes in "schedulingLT" have to be changed, and the counterpart in the Invoke activity, which calls the service, has to be changed, too.

Figure 4 shows the service connector "schedulingCon" in Darnel, which invokes the same "scheduling service," To substitute the "scheduling service," one can add the new service into the candidate service set of the connector and delete the old one.

2. With the service selection function, it is easy to select requirement-satisfied service from provider services. And through reconfiguring the service selection function and the candidate service set, service connections can adapt to dynamically changeable requirements.

When current connected provider service cannot meet changed requirements, one can add new services by reconfiguring the candidate service set; in addition, by reconfiguring the service selection function, one can alter service connection relationships or set up new connections to make the service composition adapt to new requirements. Those languages, like BPEL4WS, BPML, WSCL and so forth, do not offer enough supports for defining dynamic service connection relationships. Once the connections are set up, it is difficult to alter them. Taking the "scheduling service" as an example, it is troublesome to change the service compositions in BPEL4WS when a service with higher qualities is required to substitute the old one. However, if the composition is defined in Darnel, one can easily reconfigure the

service selection function to choose and connect the higher-quality one.

With the connector construct, Darnel provides notation for describing dynamic service compositions. With supporting toolkit, service compositions in Darnel can be dynamically built and adjusted during run-time.

TOOLS FOR DYNAMIC SERVICE COMPOSITIONS

This section presents the supporting tool CAFISE Toolkit (Han, Zhao, Li, Xing, Lv, Wang, Xiong, & Liu, 2003), with which a service-oriented application in Darnel can not only run effectively but also adapt to the changes of services or requirements automatically or semi-automatically.

CAFISE Toolkit

CAFISE Toolkit includes a visual business-end programming workshop, an execution environment, a monitoring and adjusting tool, a service community, and a context console. Figure 6 presents a snapshot of the toolkit. With the programming workshop and services in the service community, one can chain the services in a business-level visual language, VINCA (Han, Geng Hui, Li, Xiong, Li, Holtkamp, Gartmann, Wagner, & Weissenberg, 2003). Then the service compositions are converged into an executable service-oriented application in Darnel. While the application is running in the execution environment, the monitoring and adjusting tool can be used to illustrate the application states and dynamically adjust the application.

The white and central section of Figure 7 shows the key parts related to the service connector in CAFISE Toolkit.

In Figure 7, acting as an execution engine, the Resolver interprets and executes the application in Darnel. While the connector elements are interpreted, their runtime objects are created, according to connector properties of features, service selection functions, candidate service sets, and so forth. Then properties of the runtime objects are passed to the connector controller in order to construct a service connector at runtime (Li, Wang, Wang, Han, Zhao, Wagner, & Hu, 2003).

The connector controller not only controls the dynamic construction of calling service stubs but also maps the interaction features into candidate services. In addition, according to the runtime object properties, the connector controller calls the service selection function specified in the Darnel specifications, selecting a requirement-satisfied candidate service and associating it with the consumer service.

After the connector controller dynamically chains services, the service access proxy binds parameters and calls the selected service through the calling service stub. After getting results, the service access proxy passes them to the Resolver, which returns the result to the consumer services. Tthen the connector controller releases the service connector.

Adaptation Features

With its supports to service connector, the CAFISE Toolkit makes the applications more adaptable to the changes of services and requirements. The dynamic service composition establishment and reconfiguration features in CAFISE Toolkit manifest this point.

On one hand, the CAFISE Toolkit can dynamically select and connect services by supporting the dynamic service selection function. When service changes make the connection unavailable, a new service that satisfies the consumer service can replace the changed one, and the new connection is dynamically recovered.

In the CAFISE Toolkit, the default service selection function adopts dynamic service selection policy based on context. The CAFISE Toolkit's context console, which connects with sensors that gather dynamic context item value, offers context information to the connector controller. The default service selection function uses context item values as parameters to select requirement-satisfied services. For example, when a user moves out of the valid scope of the connected service, the service is invalidated.

Figure 6. Snapshot of CAFISE Toolkit

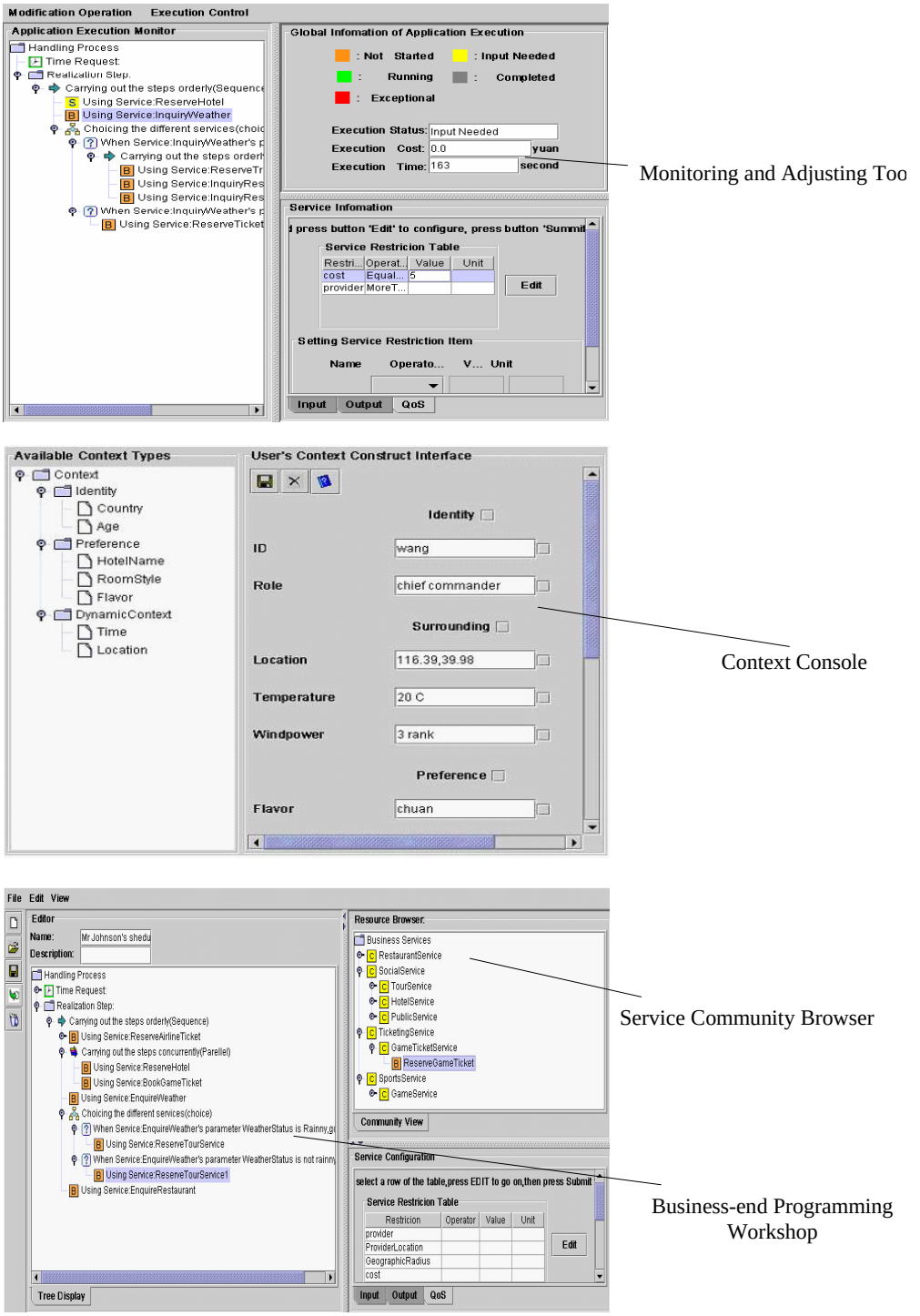
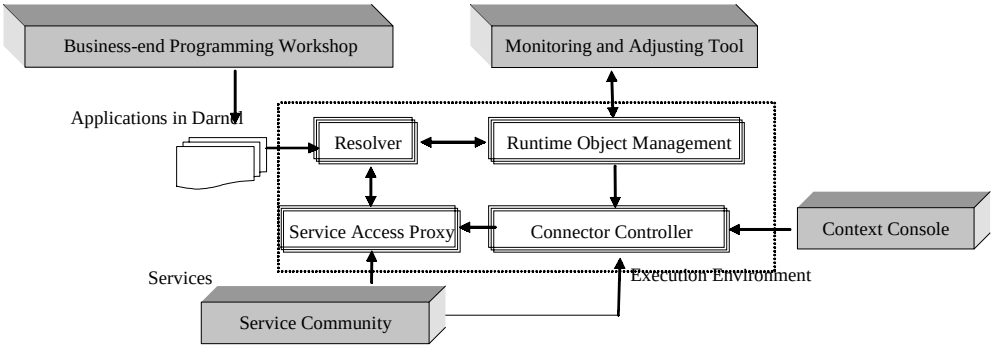


Figure 7. Supports to the service connector in CAFISE Toolkit



The service selection function will choose a valid and nearest service based on user's current location. All this process is transparent to users; users have no idea of the invalidation and replacement of the services.

The context-based dynamic service selection is a kind of realization of service selection function, which can be reconfigured and replaced by other ones.

Through this mechanism, the connection relationships can be dynamically established at runtime and the application structure can be changed without altering the service connections. Furthermore, the application can behave according to different users and different execution environments, adapting to changes.

On the other hand, the service connector has reconfigurable explicit structure. And with CAFISE Toolkit's monitoring and adjusting tool, the connector can be reconfigured, which enables service composition co-evolution while requirements change (Wang, Han, Wang, & Li, 2004).

With the monitoring and adjusting tool, one can modify the attributes of a service connector runtime object, such as candidate service set, service selection function, or features. For example, the addition or deletion of provider services can be realized by adjusting candidate service set; service selection policies can be replaced by reconfiguring service selection functions; when provider services cannot satisfy their consumer partner, the connector features can be reconfigured and new candidate services can be added. Through the previous reconfigurations, service connector can be dynamically changed to establish new service compositions to satisfy changed requirements.

Advantages Over Existing Work

Based on UDDI's capabilities, like registering and retrieving Web services and so forth, the CAFISE Toolkit's service community organizes and presents candidate services at the business level, where semantic and business constraints are added. Beyond the Web service discovery function of UDDI, the service connector dynamically abstracts a set of candidate services into a virtual service and exposes the virtual service to consumer services while discovering and connecting the concrete candidate Web services in a dynamic and transparent way. The CAFISE Toolkit facilitates dynamically chaining services with the service connector, which makes the applications developed with CAFISE Toolkit more dynamic and flexible. Currently, there are many tools supporting service compositions or development of service-oriented applications, like CB-Sec framework, Self-Serv, eFlow, SWORD, and so forth. By benefiting from those tools at some aspects, the CAFISE Toolkit contributes to flexible and dynamic service compositions with its own advantages.

Combined with context-aware computing and agent technology, the CB-SeC framework provides a dynamic service selection and composition framework based on context

(Mostéfaoui, 2003). It can match and recommend services by requested service features, time, and user contexts. And its dynamic service selection embodies flexible service compositions. Different from the CB-SeC framework, the CAFISE Toolkit can not only dynamically select services by user contexts but also dynamically replace involved services, change service selection policies, and reconfigure service connections.

The Self-Serv (Benatallah, Sheng, & Dumas, M., 2003) also focuses on dynamic aspects of service compositions and presents a concept of service container, which uses a scoring service consisting of a set of multi-attribute utility functions such as the service selection function, to select the specific service that will run. Our service connector presents an open and scalable connection structure with which one can not only use the Self-Serv style multi-attribute utility functions as the service selection functions but also benefit from other research related to matching methods (such as those presented in Noia, Sciascio, Donini, Mongiello, Bari, & David, 2003; Veit, Müller, Schneider, & Fiehn, 2001; Zhang & Zhang, 2002) and add new types of selection policies into the service connector. In addition, the service connector can be explicitly described, and its interface can be reconfigured. So, our work is more scalable and open, although it does not support peer-to-peer orchestration.

There are some researches similar to the service selection. In their researches on multi-agent systems, Sycara, Klusch, Widoff, and Lu (1999) propose an agent capability description language, LARKS. To select agents, the LARKS agent matchmaking process uses five different filters, which include context matching, profile comparison, similarity matching, signature matching, and constraints matching (Sycara et al., 1999). Zhang and Zhang present an agent selecting method based on agent's behavior history in accomplishing similar tasks (2002). Tang, Chao, et al. also develop an agent-based architecture and present a broker agent for matchmaking (Tang, Feng, & Li, 2001). Tommaso and Di Noia et al. define the process of selecting possible matches between demands and supplies in an electronic marketplace as matchmaking and present a facilitator that supports semantic-based matchmaking (Noia et al., 2003). Zhang, Chang, and Chao present their research results on service relationships and service discovery (2002). Veit et al. design a configurable framework for agent matchmaking in electronic marketplaces, which provides an extensible library of matchmaking functions (2001). Compared with these researches from domains of agent-based systems or e-marketplace systems, our work focuses on dynamic service composition and presents an adaptable service connector model and related service composition languages and supporting tools. When one designs the service selection function of the connector, he or she can be enlightened by these researches on matchmaking.

HP's eFlow provides features and constructs to support dynamic service compositions, which include dynamic service discovery, multi-service nodes, and dynamic service node creation (Casati, Ilnicki, Jin, Krishnamoorthy, & Shan, 2000). At the aspect of the dynamic service discovery, the service selection rules are defined for the service nodes in eFlow, which is interpreted and computed by a service broker. Users can replace the default broker with a user-tailored service broker that best meets their requirements. Aiming at the issue of dynamic modification, eFlow presents two kinds of service composition modifications: ad-hoc and bulk changes. eFlow is a remarkable work in the field of dynamic service composition. Benefiting from its ideas of application dynamic modification and modification security, we concentrate our research on the basic element of service compositions — service connection and contributing to dynamic service compositions. Through the reconfigurable service connection structure, one can reconfigure the candidate service sets, service selection function, or features, which results in dynamically changing service compositions with lower cost.

There are some outstanding researches on business level service composition and service discovery. The work in (Zhang, Li, Chao,

& Chang, 2003) aims at supporting service composition from business perspectives. It provides a Business Process Outsourcing Language (BPOL) to capture business requirements. Then the business-level elements described in BPOL can be mapped into Web services flow to achieve on-demand Web services composition. During the mapping process, Business Explorer for Web services (BE4WS) (Zhang, Chao, Chang, & Li, 2001) is used to search business and service information in one or more UDDI registry. At the same time, in order to fulfill the business processes, Web service clusters are provided, which can be provided by more than one provider. Web service clusters can also be looked at as a connector that connects to some concrete Web services. Compared to our work, this research focused on business-level service composition.

The Web Services Invocation Framework (WSIF) is an open source project of Apache, which provides a simple Java API for abstract invocations of Web services and enables dynamic invocation and client-unaware service upgrade (Apache Software Foundation, 2003). Based on WSIF, the CAFISE Toolkit implements the service access proxy, which supports not only dynamic service invocation but also dynamically selecting and chaining services. Through CAFISE Toolkit, the service compositions can be dynamically reconfigured.

Stanford University developed the toolkit SWORD for service-oriented application development, which facilitates service developers by rapidly constructing new services with existed services (Ponnekanti & Fox, 2002). The SWORD realizes a service execution model to enable the effective execution of service compositions, in which services are connected with data flow.

Moreover, some companies and research institutes developed tools for BPEL4WS, like IBM's BPEL4WS execution engine, Collaxa's BPEL4WS server, Vitria's BusinessWare and so forth. By benefiting from their strong points at the aspects of construction, execution, and monitoring of service compositions, we contribute an aspect of service compositions' adaptability to the changes of requirements and services.

CASE STUDY

In this section, we briefly present a case study with the same case discussed in the Problem Analysis section. Note that the case study focuses on the aspect of service connection adaptabilities; other details about the case are beyond this scope.

In the aforementioned first prototype related to the case, we used WSDL and UDDI to discover Web services and used SOAP to connect and invoke the services. To add new services, update services, or alter service connections, the source codes had to be read through and the composition logic had to be modified. Extremely, some modules had to be rebuilt. That was heavy work. What's more, exceptions were likely to occur in the modified application.

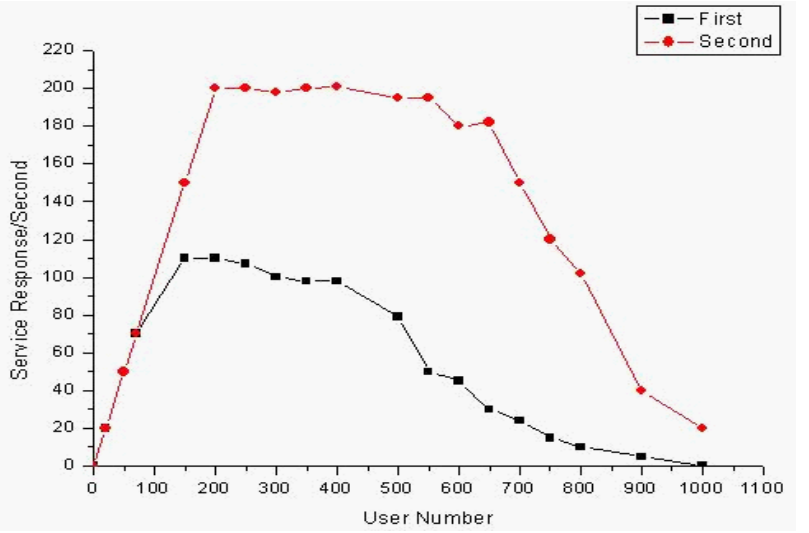
Dynamic Service Composition in FLAME2008

In the second prototype of FLAME2008, the Match Query Service connected to Comprehensive Match-info Service through a connector named Match_querist. Match_querist keeps the Match Query Service a candidate service that realized Match_querist's feature "retrieve" and used service selection function MQSel01 to dynamically choose the available candidate services, according to the user context, such as time, location, and sport preferences.

While the Match Query Service was to be replaced, the deployed new ones (like *Gym Match Query Service01, Gym Match Query Service02, and so on*) were conveniently added to Match_querist's candidate service set, by using CAFISE Toolkit's service community and monitoring and adjusting tool.

With the previous reconfiguration, the Comprehensive Match-info Service dynamically connected the available *Gym Match Query Service01* and so forth, instead of the Match Query Service. By interacting only with Match_querist's features, Comprehensive Match-info Service was not influenced by these changes. When Comprehensive Match-info Service sent a request for match schedules to Match_querist, the available service was dy-

Figure 8. The stress test results of getting match schedules



namically selected and called, which reduces the access congestion.

Results and Evaluations

In the second prototype, we registered 50 services like *Gym Match Query Service01* in the service community, which were used as candidate services in the connector Match_querist. The stress test of getting match schedules was done in two prototypes. Figure 8 outlines the test results when the number of simulation users increases.

Comparing with the first prototype, the second one can keep a higher service response limitation of about 200 times per second. This result shows that with more service resources offered by the candidate service set, the access bottleneck is reduced by balancing the service access requests and service resources with dynamic service connection.

Besides the project FLAME2008, the service connector is also used in service roaming (Gartmann, Holtkamp, Weissenberg, & Li, 2005) and the project AMGrid[2] (CAFISE, 2003), which targets virtual enterprises among different manufacturing companies. The experiences from real-world projects show that with the model, language and tools presented in this paper, a service-oriented application has the following advantages:

- **Context-awareness.** With the context console and service selection function, the application can detect location, time, or preference changes, and choose the requirement-satisfied service to be invoked. In the same way, the application can become aware of other type context changes and act accordingly.
- **Self-configuring.** The services in an application are dynamically selected and connected by the service connectors. New services can be plugged and played without influencing other application components. The application configuration can dynamically change according to the changes of requirements and services. So, the application can be dynamically constructed and self-managed.

As complements of autonomic capabilities, like context-awareness and self-configuration, semi-autonomic capabilities are offered through the monitoring and adjusting tool of CAFISE Toolkit, which obviously reduces the costs and complexities of reconfiguring service

compositions. These autonomic or semi-autonomic capabilities make the applications adapt to the changes of requirements and services, and facilitate the dynamic service composition.

CONCLUSION

The open and dynamic network environments bring new challenges to service-oriented applications. How to dynamically compose services is a key issue. Our research contributes to this issue in the following aspects:

- The service connector presents an adaptable connection structure. Through the connector, a consumer service can dynamically connect to a provider service. And the service compositions can be reconfigured dynamically.
- The Darnel language and CAFISE Toolkit, which support the dynamically reconfigurable service connectors, facilitate dynamic service composition.

In addition, the experiences in project FLAME2008, service roaming, and project AMGrid show that one can conveniently implement dynamic service composition with the service connectors presented in this paper.

Our current work mainly focuses on composition stateless web services; the research on state service compositions is to be done in the future, when how the business state is related to a service will be considered.

ACKNOWLEDGMENT

Our research is supported by National Natural Science Foundation of China. We thank them for their generous help.

REFERENCES

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., & Weerawarana, S. (2003). *Business process execution language for Web services version 1.1*. Retrieved September 2005 from http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

Apache Software Foundation. (2003). *WSIF overview*. Retrieved October 2004 from http://ws.apache.org/wsif/overview.html

Benatallah, B., Dumas, M., Sheng, Q. Z., & Ngu, A. (2002, February 26-March 1). Declarative composition and peer-to-peer provisioning of dynamic Web services. In *Proceedings of the 18th International Conference on Data Engineering* (pp. 297-308), San Jose, CA.

Benatallah, B., Sheng, Q. Z., & Dumas, M. (2003, January/February). The self-serv environment for Web services composition. *IEEE Internet Computing*, 40-47.

CAFISE group. (2003). AMGrid project technical report. Software Division, ICT, CAS.

Casati, F., Ilnicki, S., Jin, L. J., Krishnamoorthy, V., & Shan, M. C. (2000). *Adaptive and dynamic service composition in eFlow* (HP Technical Report HPL-2000-39). HP Labs.

Foster, I., Kesselman, C., Nick, J., & Tuecke, S. (2002), Grid services for distributed system integration. *Computer, 35*(6), 37-46.

Foster, I., Kesselman, C., & Tuecke, S. (2001). The anatomy of the grid: enabling scalable virtual organizations. *The International Journal of Supercomputer Applications, 15*(3), 200-222.

Gartmann, R., Holtkamp, B., Weissenberg, N., & Li, G. (2005, July). Service roaming in mobile applications. In *Proceedings of IEEE SCC2005*, Orlando, FL.

Goguen, J. (1991). A categorical manifesto. *Mathematical Structures in Computer Science, 1*(1), 49-67.

Han, Y. B., Geng Hui, Li, H. F., Xiong, J. H., Li, G., Holtkamp, B., Gartmann, R., Wagner, R., & Weissenberg, N. (2003, December). VINCA — a visual and personalized business-level composition language for chaining Web-based services. In *Proceedings of The First International Conference on Service-Oriented Computing* (pp. 165-177), Trento, Italy.

Han, Y. B., Zhao, Z. F., Li, G., Xing, D. S., Lv, Q. Z., Wang, J. W., Xiong, J. H., & Liu, H. (2003). CAFISE: an approach to enabling adaptive service configuration of service grid appli-

cations. *Journal of Computer Science and Technology, 18(*4), 484-494.

Krishnan, S., Wagstrom, P., & Laszewski, G. (2002). *GSFL: A workflow framework for grid services*. Retrieved October 2005 from http://www-unix.globus.org/cog/projects/workflow/

Li, G., Wang, J.W., Wang, J., Han, Y.B., Zhao, Z.F., Wagner, M.R., & Hu, H. (2003 December). MASON: a model for adapting service-oriented grid applications. In *Proceedings of GCC2003* (pp. 99-107), Shanghai, China.

Mostéfaoui, S. K. (2003 August). Towards a context-oriented services discovery and composition framework. In *Proceedings of the workshop on AI Moves to IA: Workshop on Artificial Intelligence, Information Access, and Mobile Computing* (pp. 34-40), Acapulco, Mexico.

Noia, T. D., Sciascio, E. D., Donini, F. M., Mongiello, M., Bari, P. D., & David, V. R. (2003, May). A system for principled matchmaking in an electronic marketplace. In *Proceedings of the 12th International Conference on World Wide Web* (pp. 321-330), Budapest, Hungary.

Ponnekanti, S. R., & Fox, A. (2002). SWORD: *A developer toolkit for Web service composition*. Retrieved October 2005 from http://www2002.org/CDROM/alternate/786/

Sycara, K., Klusch, M., Widoff S., & Lu, J. G. (1999). Dynamic service matchmaking among agents in open information environments. *ACM SIGMOD Record, 28*(1), 47 - 53.

Tang, C., Feng, S., & Li, D. X. (2001, December). An agent-based architecture in geographical information System. In *Proceedings of the American Control Conference 2001* (pp. 912-917), Arlington, VA.

Veit, D., Müller, J. P., Schneider, M., & Fiehn, B. (2001, May). Matchmaking for autonomous agents in electronic marketplaces. In *Proceedings of the Fifth International Conference on Autonomous Agents* (pp. 65-66), Montreal, Canada.

Wang, J. W., Han, Y. B., Wang, J., & Li, G. (2004, November). An approach to dynamically reconfiguring service-oriented applications from a business perspective. In *Proceedings of AWCC2004* (pp. 357-368), Zhenjiang, China.

Zeng, L., Benatallah, B., & Ngu, A. (2001, September). On-demand business to business integration. In *Proceedings of the 9th International Conference on Cooperative Information Systems* (pp. 403-417), Trento, Italy.

Zhang, L. J., Chang, H., & Chao, T. (2002, June). Web services relationships binding for dynamic e-business integration. In *Proceedings of International Conference on Internet Computing* (pp. 561-570), Las Vegas, NV.

Zhang, L. J., Chao, T., Chang, H., & Li, H. (2001). *Business explorer for Web services*. Retrieved September 2005 from http://www.alphaworks.ibm.com/tech/be4ws

Zhang, L. J., Li, B., Chao, T., & Chang, H. (2003, October). On demand Web services-based business process composition. In *Proceedings of 2003 IEEE International Conference on Systems, Man & Cybernetics*, Washington.

Zhang, Z., & Zhang, C. (2002, July). An improvement to matchmaking algorithm for middle agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems* (pp. 1340-1347), Bologna, Italy.

ENDNOTES

1 FLAME2008 (Project **Fl**exible Sem**a**ntic Service **M**anagement **E**nvironment) is to develop service-oriented applications that provide integrated, personalized information services to the public during the Olympic Games 2008. The joint project is funded by the Chinese Ministry of Science and Technology (MOST Grand No.20012019) and the German Ministry of Education and Research (BMBF Grant No. 01AK055).

2 It is funded by High-tech Research and Development Program of China-863 Program (Grant No.2003AA414330).

Gang Li received his PhD in computer science from the University of Aeronautics and Astronautics, China. He is an associate professor of the Institute of Computing Technology, Chinese Academy of Science. His research interests includes service-oriented computing, adaptive software architecture, and WLAN-based network.

Yanbo Han is a professor of the Institute of Computing Technology, Chinese Academy of Sciences. He holds a PhD 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.

Zhuofeng Zhao received his PhD from the Institute of Computing Technology, Chinese Academy of Sciences. His research interests are service composition, service-oriented application, and workflow technologies.

Jing Wang is a PhD candidate at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests are service composition, service-oriented application, and workflow technologies.

Roland M. Wagner received his PhD from the Technical University of Berlin, Germany. He joined the Fraunhofer Institute for Software and System Engineering (Fraunhofer ISST) in Berlin/Dortmund in August 2000. In 2005, he got a research position at the University of Münster. His research interests are Web pricing and ordering service and spatial data infrastructures.