An Adaptable Service Connector Model[1]

Gang Li[1], Yanbo Han[1], Zhuofeng Zhao[1], Jianwu Wang[1], and Roland M. Wagner[2]

[1] Software Division, ICT, Chinese Academy of Science, PRC
{ligang, yhan}@ict.ac.cn {zhaozf, wjw}@software.ict.ac.cn
[2] Fraunhofer ISST, Dortmund, Germany
roland.wagner@isst.fhg.de

Abstract. The volatility of network environments requires service connections to adapt to changes of service resources and user requirements. In this paper, we treat service connections as individual components called service connectors and present an adaptable service connector model that adopts a role mechanism to adjust connections between services. A role is an abstraction of services with common functionalities. It offers a changeable connector structure, enables reconfiguration of service interaction and encapsulates changes in interacting participants, making service connections more adaptable.

1 Introduction

Service oriented computing is gaining popularity. In a typical contemporary service-oriented application, service connections are pragmatically implemented using protocols like SOAP. Through this kind of connection, services can be composed into applications. Service composition is regarded as a new approach for developing applications in network environments.

However, service composition still faces serious challenges due to the openness and dynamism of network environment, such as grids [1][2]. Let us take service grids as an example. Firstly, services freely join in or quit from a grid and most services in a grid continue evolving over time. Secondly, user requirements are subject to dynamic changes in a virtual enterprise environment. All these require service connections to be adaptable, so that service interactions can be easily reconfigured and involved services can be changed dynamically, while changes of service resources and user requirements take place. In this paper, we focus our research on how to make a service connection adapt to those changes.

Existing ways that are used to connect services includes control flow based connections [3], data flow based connections [4][5] and hybrid forms of these two types, such as service connection mechanisms in GSFL [6] and BPEL4WS [7]. After setting up an interaction channel between ports of different services through protocols, the validity of control flow based service connection is determined by service states. This type of connection is set up following interaction protocols inhering in services. For it is predefined and fixed, changes of service interactions or requirements tend to invalidate the connection. A data flow based connection links services through data dependencies. With shared data, this type of connection is free from influences caused by service port changes to some extent. However, because of its implicit definition, the structure of data flow based service connections is indistinct, which makes it difficult to adjust the connection. Although in the hybrid from above the two types complement each other, it does

[1] This paper is supported by the Young Scientist Fund of ICT Chinese Academy of Science under Grant No. 20026180-22 and the National Natural Science Foundation of China under Grant No. 60173018.

not contribute much to service connection adaptation. For example, BPLE4WS provides partner link types to describe service connections as partner links, but it does not offer methods for dynamically changing these links.

To make the connection adaptable, the connection structure ought to be changeable. As a semantic concept, role [8][9] provides an organizing mechanism through which the abstraction of services with common functions is derived and marked by role features. With this mechanism, a role offers flexible connection structure, enabling service connection adaptation by reconfiguration. Based on the above rationale, we implement a service connection as an explicit component named service connector and present a role-based service connector model. In this model, a role is used to fulfill the adaptation of a service connection with a stable interaction interface and a changeable connection structure.

The remainder of this paper is divided into 3 sections. Section 2 addresses the role-based service connector model in detail, including its connection structure and interaction protocol. With a case study, section 3 demonstrates the support of the model to making service connection adaptable. Finally, the contribution of our research is concisely summarized in section 4.

2 Role-based Service Connector Model

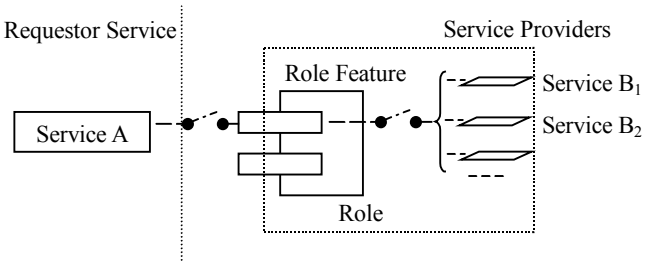

Fig. 1. Sketch Map of Role-based Service Connector Model

Figure 1 illustrates the main idea of the role-based service connector model by an example that Service A interacts with a role. Described by its features, the role is an abstraction of services such as Service B1 and Service B2. As a stable interface for service interaction, role features present functions provided by service providers that interact with the requestor services through the role. Marked by features, functions that a role provides are implemented by service providers that are invisible to requestors. When unanticipated changes cause a modification of a service interaction, the connector can adapt to changes of service interactions or requirements by reconfiguring its role features or service providers. When a service provider involved in an interaction is unavailable, another one with the same role feature can replace it, enhancing the adaptability and reliability of the service connection. A role is a virtual service, and it offers not only a changeable connector structure, but also a unified service interaction interface, providing essential support to connection adaptation in dynamic service composition [14].

Details about the role-based service connector model follow, namely aspects of the connection structure and the interaction protocol.

2.1 Connection Structure

In this section, to present the role-based service connector model precisely and concisely, we give its definition, related semantic and interaction protocol in a formal way. Based on this formalism, we prove that a role-based service connector is adaptable.

Definition 1 Role-based Service Connector

Given a three-tuple $<Name_r, Features_r, Service_r>$, where $Name_r$ marks a role name; $Features_r$ is a set of role features, $Features_r=\{f_r|f_r=<rn,fn_r,va_r>\}$, where rn denotes the role that feature f_r belongs to, fn_r is the name of f_r, va_r is the argument vector of f_r. Role features are an interface with which a role interacts with the environment; $Service_r$ is a set of service references, which are related to features.

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

- There is a function set, denoted as *Map*. Given $\forall f_r \in Features_r$ then $\exists m \in Map$, $S_{sr} \subseteq Service_r$, $m(f_r)=S_{sr}$, and services that belong to set S_{sr} have the same interaction interface marked by f_r.
- There is another function set, denoted as *Selectors*, consider f_r, $\exists sel \in Selectors$, $Ser \in S_{sr}$, $sel(S_{sr})= Ser$.

The function m is named as feature mapping function, and the function sel is named as service selection function.

Thus a service connector presents a configurable service called instead of a requested service. The connector selects an appropriate provided service and maps its parameters to the request. Therefore it is a configurable encapsulation of a service or a group of semantically similar services, abstracted as a role. There are two interaction patterns involved in a role-based service connector model, namely the service-role and role-role patterns. In the first pattern, service providers are packaged by the role, which is depicted by figure 1. It fits the context where service providers are volatile. In the second pattern, both providers and requestors are packaged by roles, which fits the context where both sides are volatile.

The main parts of the model are presented above. Now, we define its connection semantic using category theory to deduce connector properties on adaptation. A category consists of object sets and sets of morphisms between objects, which focuses on describing and analyzing relations among any type of objects [10]. With category, relations among services and roles can be described formally, which offers an approach for analyzing and deducing properties of a role-based service connector in a method independent of implementation details. Hereby it presents the connection semantics in a role-based service connector model with categories that takes structured sets as objects.

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

$Serv=<objC_{Serv}, MorC_{Serv}, dom_s, cod_s, \circ>$, where The set $objC_{Serv}$ 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:

δ_1: $Name_E \rightarrow Name_F$

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

Let *Connector* be the role-based service connector 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 ψ is used to define $MorR_{Conn}$ and describes interactions between roles.

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

$\eta_1: Name_{Ser} \rightarrow Name_{Con}$

$\eta_2: Feature_{Ser} \rightarrow Feature_{Con}$

The connection fulfilled by a role-based service connector is marked as $Connection_R$. $Connection_R=\{<Ser, F_{SR}(Ser)>\} \cup \{<Con, \psi(Con)>\}$.

After giving connection semantics, we can now prove the following results according to it:

Theorem 1

In interaction where roles are involved, a role-based service connector has the following properties:

- Changes in the service set that correspond to a role feature do not cause changes of the connection;
- When the service set that corresponds to a role feature do not satisfy some requirements of requestors, the connection can adapt to these changes by reconfiguring the role feature and related services.

Proof:

(1) Let *r1* be the role involved in an interaction. According to the definition of *r1*, we conclude: $\forall f_x \in Feature_{r1}, \exists m \in Map_{r1}, m(f_x)=s_{sx}, s_{sx} \subseteq Service_{r1}$, and $\exists sel \in Selectors_{r1}, sel(s_{sx})=Serx, Serx \in s_{sx}$.

When changes take place in s_{sx}, a new function *sel*′ can be constructed, $sel' \in Selectors_{r1}$, such that $sel'(s_{sx})= Serx'$.

∵ *Serx*′ and *Serx* have the same features, and the requestor service interacts with the service that belongs to s_{sx} through the role feature.

∴ There are no changes in the connection.

Proposition (1) follows.

(2) When the service set that corresponds to a role feature does not satisfy some requirements of the requestor services, the interaction interface or the service set has to be changed. If only the service set is to be adjusted, the connection can adapt to those changes according to proposition (1). If the interaction interface is to be changed, it results in a change of the role feature according to **Definition 1**. Then proposition (2) can be proved as follows:

i) when changes take place in a service–role pattern,

let *s*, *r1* be the service and role involved in the interaction, and $f_s \in Feature_s$. According to F_{SR}, we conclude:

$\exists <f_s, f_{r1}>, f_{r1} \in Feature_{r1}$. When $<f_s, f_{r1}>$ changes into $<f_s, f'_{r1}>$, according to the definition of *r1*, $\exists m' \in Map_{r1}$, such that $s_{sx1}' \subseteq Service_{r1}, m'(f'_{r1})=s_{sx1}'$. *m*′ and s_{sx1}' keep the connection available.

ii) when changes take place in role-role pattern,

let *r1*, *r2* be the roles involved in the interaction, and $f_{r1} \in Feature_{r1}$. According to ψ, we conclude:

$\exists <f_{r1}, f_{r2}>, f_{r2} \in Feature_{r2}$. When $<f_{r1}, f_{r2}>$ changes into $<f_{r1}, f'_{r2}>$, according to the definition of *r2*, $\exists m'' \in Map_{r2}$, such that $s_{sx2}' \subseteq Service_{r2}, m''(f'_{r2})=s_{sx2}'$. *m*″ and s_{sx2}' keep the connection available.

According to i) and ii), proposition (2) follows.

The proof is now complete.

Theorem 1 says that a role-based service connector is adaptable. In $Connection_R$, there is a loose coupling between interacting participants, which can be changed by adjusting the service selection

function such as *sel*(). Through role features, services expose a unified interaction interface to requestor services, reducing the influences between services. When unanticipated changes occur, the connection can be reconfigured through changing the involved role features and services. In addition, a role feature can be implemented by several candidates of service providers like services in s_{sx}, which makes the connection adaptable.

2.2 Interaction Protocol

When a role is introduced, changes occur in service interaction patterns to support connection adaptation. As mentioned above, service-role and role-role patterns are the main interaction manners in service compositions where roles are involved. In order to describe them clearly, the interaction protocols are presented in a formal way.

While analyzing security protocols, I. Cervesato etc. used a multiset rewriting formalism, based on linear logic. The existential quantification in it provides a succinct way of choosing new values. Besides that, it has a bounded initialization phase, but allows unboundedly many instances of each protocol participant, making it especially qualified to analyzing finite-length protocols [11]. In this section, we present the interaction protocol of a role-based connector model with this method.

(1) Interaction in a service-role pattern

$Ser^0(), R^0()$

$Ser^0() \rightarrow \exists x.Ser^1(x), Con^1(x)$

$R^0(), Con^1(x) \rightarrow \exists y.R^1(x, y), \exists Ser'^0(), Con^2(x, y)$

$Ser'^0(), Con^2(x, y) \rightarrow \exists z.\ Ser'^1(y, z), Con^3(x, y, z)$

$R^1(x, y), Con^3(x, y, z) \rightarrow R^2(x, y, z), Con^4(x, z)$

$Ser^1(x), Con^4(x, z) \rightarrow Ser^2(x, z)$

In service-role pattern, service *Ser* is the requestor service, and $Ser^0()$ denotes that service *Ser* is in initial state 0. Then it produces message x in state 0, sends x to role R transforming into state 1 that keeps message x; $Con^1(x)$ denotes that the connection is in state 1 that keeps message x. After role R receives x at state 0, it produces message y, then with the service selection function, role R chooses a service Ser' as server according to interaction state, and sends y to Ser' transforming into state 1. After processing message y, service Ser' produces massage z that contains the results the client required and sends z to role R. After role R receives z at state 1, it delivers message z to service *Ser* transforming into state 2; service *Ser* receives z at state 1 transforming into state 2.

(2) Interaction in a role-role pattern

$Ser^0(), R^0(), R'^0()$

$Ser^0() \rightarrow \exists x.Ser^1(x), Con^1(x)$

$R^0(), Con^1(x) \rightarrow \exists y.R^1(x, y), Con^2(x, y)$

$R'^0(), Con^2(x, y) \rightarrow \exists z.R'^1(x, y, z), \exists Ser'^0(), Con^3(x, y, z)$

$Ser'^0(), Con^3(x, y, z) \rightarrow \exists w.\ Ser'^1(x, y, z, w), Con^4(x, y, z, w)$

$R'^1(x, y, z), Con^4(x, y, z, w) \rightarrow R'^2(x, y, w), Con^5(x, y, w)$

$R^1(x, y), Con^5(x, y, w) \rightarrow R^2(x, w), Con^6(x, w)$

$Ser^1(x), Con^6(x, w) \rightarrow Ser^2(x, w)$

A role-role pattern is a combination of two service-role patterns. In this pattern, request and return values pass through two roles.

Above protocols describe how to configure the connector automatically. Besides that, they emphasize especially on states of connections and interaction participants. On the one hand, a role supports the dynamic selection of qualified services according to connection states and service states; on the other hand, when the connection is to be reconfigured, these states determine whether the reconfiguration is feasible. During connection reconfiguration, states of interacting participants are saved. After reconfiguring, they are restored to enable service composition to be resumed. It shows that the interaction protocol of a role-based service connector supports dynamic connection and adaptation.

After presenting the model and demonstrating its adaptability in a formal way, we now present a case study in the following section.

3 Exploring the Model with a Case Study

A role-based service connector is more adaptable than others, when changes take place in service resources and/or requirements. In order to strengthen this conclusion and illustrate how to use role-based service connectors, we briefly present a case study of representing the model in XML and applying it in project FLAME2008[2]. Note that the case study focuses on the aspect of connection adaptabilities, other details about the case are beyond this scope.

3.1 A Real Case from FLAME2008

The Olympic Travel Planning application, which is a part of FLAME2008, is to provide pertinent information to those who watch match and tour in Beijing during Olympic Games 2008. Figure 2 presents a requirement segment of the application, which involves the following services.

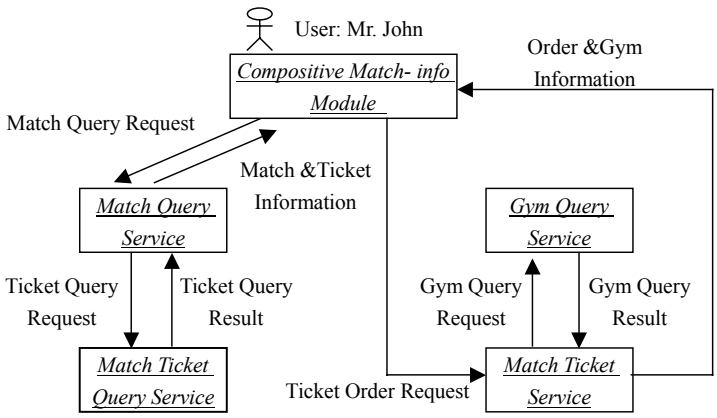

Fig. 2. A Requirement Description Segment of Olympic Travel Planning Application

- *Match Query Service*: retrieving a match schedule, getting information about the match that one wants to watch, and invoking *Match Ticket Query Service*.
- *Match Ticket Query Service*: retrieving match ticket information.

[2] FLAME2008 (Project **Fl**exible Sem**a**ntic Service **M**anagement **E**nvironment) is to develop grid service based applications that provide integrated, personalized information services to the public during the Olympic Games 2008. The project is supported by MOST PRC and CAS under Grand No.20012019.

– *Match Ticket Service*: ordering match tickets after making sure that there are remains, and invoking the *Gym Query Service*.
– *Gym Query Service*: retrieving information about traffic, gym location and so on.
– *Compositive Match-info Module*: coordinating above services to get information about match schedules, match tickets, gym and ordering tickets.

In the initial stage of the project, we constructed a prototype by composing services. However, those services and their connections were changing with service resources changing and requirements evolving.

- **Change I**: In the prototype, the response time of *Match Ticket Query Service* was too long to endure, so that the service had to be replaced by a new one.
- **Change II**: Previously, after ordering tickets, audiences wanted to know something about the gym where the match was to be held. So, *Match Ticket Service* interacted with *Gym Query Service*. Now, audiences want to retrieve order results after ordering. To meet the changed requirement, *Match Ticket Service* is to interact with *Order Result Query Service* (This service is a service to be added when requirements change, and it is not shown in figure 2). In this situation, both the interaction interface and services are changed.

3.2 Representations and Applications of the Model

In the initial prototype of FLAME2008, services were connected by control flow based on SOAP messages. To adapt to changes in service providers, the adaptors were modified. However, when the above-mentioned changes occurred, it was very difficult to adjust the application, for the service connections were almost unchangeable. In order to change them, we had to read through the source codes, and modify the processing logic or rebuild the module. It involved many efforts of understanding source code, coding and so on. The modified application was prone to throw exceptions yet.

Hence, we partially adopted and realized role-based service connector in the second prototype, and constructed a supporting tool named CAFISE Framework. Compared with service connections in the first prototype, role-based service connector can be adjusted smoothly with the framework that is depicted in figure 3.

3.2.1 CAFISE Framework

The CAFISE Framework includes a set of essential components and tools assisting to construct and adjust applications. From a business viewpoint, the Convergent Modeling Tool helps designers to present their requirements, and then the requirements are transformed into an executable application specification using XML, which is presented in figure 3 as Specification A. Those specifications describe the coordination among all involved services. While the CAFISE Virtual Machine interprets the specification, the required services of the Service Community are dynamically bound and invoked.

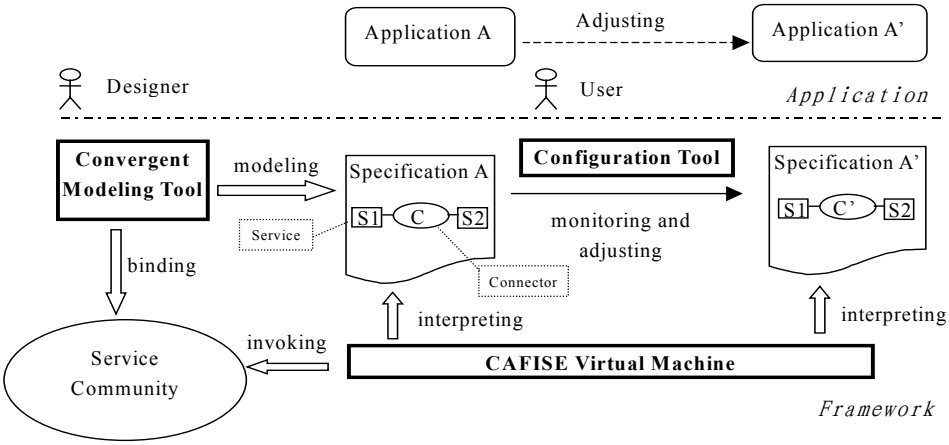

Fig. 3. CAFISE Framework

3.2.2 Reconfiguration of Connectors

The service coordination in an executable specification adopts role-based service connectors. When the aforementioned changes take place, users can reconfigure the role-based connector with the Configuration Tool. More details about the framework are presented in [12]. The following presents an example of a connecter implementation and describe how to use this framework to adjust the connector. Figure 4 illustrates a role named *Ticket_querist* in XML, which is used to connect *match query service* with *match ticket query services* according to the interaction protocol listed in section 2.2. Its feature "*searcher*" describes the ports of the *Match Ticket Query Service01* and *Match Ticket Query Service 02* that are listed as feature implementations by <*Services*>. <*Selector*> specifies the algorithm for selecting the right feature implementation. In figure 4, it uses another service "*MTQSel01*" as service selection function to select the right service involved in the interaction.

```
<Role Name="Ticket_querist " >
  <Feature Name="searcher">
     <Argument Name="Match"
      Type="NameString"
      PassMode="IN"/>
     <Argument Name="Time"
      Type="Timetype"
      PassMode="IN"/>
     <Argument Name="MatchTicket"
      Type="TicketVector"
      PassMode="OUT"/>
  </Feature>
  <Services>
      <DefaultURL>
        www.FLAME08app.com/search/
        Match Ticket Query Service 01
      </DefaultURL>
      <URL>
        www.FLAME08app.com/search/
        Match Ticket Query Service 02
      </URL>
  </Services>
  <Selector>
      <SelectFunction>
        www.FLAME08app.com/select/
        MTQSel01
      </SelectFunction>
  </Selector>
</Role>
```

Fig. 4. The Role Specification of Ticket_querist

```
<Role Name="Ticket_querist " >
  <Feature Name="searcher">
     <Argument Name="Match"
      Type="NameString"
      PassMode="IN"/>
     <Argument Name="Time"
      Type="Timetype"
      PassMode="IN"/>
     <Argument Name="MatchTicket"
      Type="TicketVector"
      PassMode="OUT"/>
  </Feature>
  <Services>
      <DefaultURL>
        www.FLAME08app.com/search/
        Match Ticket Query Service 01
      </DefaultURL>
      <URL>
        www.FLAME08app.com/search/
        Match Ticket Query Service 02
      </URL>
      <URL>
        www.FLAME08app.com/search/
        Match Ticket Query Service New
      </URL>
  </Services>
  <Selector>
      <SelectFunction>
        www.FLAME08app.com/select
        /MTQSel02
      </SelectFunction>
  </Selector>
</Role>
```

Fig. 5. The Modified Specification of Ticket_querist

To adapt to **Change I**, a new service *Match Ticket Query Service New* was designed and registered in the Service Community. The connection was to be changed. The CAFISE Virtual Machine, its

Configuration Tool and role *Ticket_querist* made it easier to reconfigure the service connection. The CAFISE Virtual Machine interpreted the specification of role *Ticket_querist* and collected meta-data of interaction states and the connection structure, supporting the Configuration Tool to modify the connection. The Configuration Tool monitored all interaction states through meta-data collected by the virtual machine. While the number of service requests was reduced to zero, users applied the tool to change the connection meta-data. By modifying meta-data, a new service "*www.FLAME08app.com/search/ Match Ticket Query Service New*" was added, and <*SelectFuction*> was set as a new one "*www.FLAME08app.com/select/MTQSel02*". Thus, the new service co-existed with the old ones, and the connection was changed without effects to the *Match Query Service* in a simple reconfiguration way. The modified specification of *Ticket_querist* is shown in figure 5.

Figure 6 presents the role connected match ticket service and gym query services that are the service providers in interaction. With **Change II**, *Match Ticket Service* changed to connect with *Order Result Query Service*. Adaptation of this connection required adjusting the role's <*Feature*>, <*Services*> and <*Selector*> parts. Using CAFISE Framework, users smoothly changed the specification of *Gym_querist* to a new one that is presented in figure 7.

```
<Role Name="Gym_querist" >
  <Feature Name="searcher">
     <Argument Name="Match"
      Type="NameString"
      PassMode="IN"/>
     <Argument Name="Time"
      Type="Timetype"
      PassMode="IN"/>
     <Argument Name="Gym_info"
      Type="GymVector" –
      PassMode="OUT"/>
  </Feature>
  <Services>
     <DefaultURL>
       www.FLAME08app.com/search/
       Gym_Query_Service_01
     </DefaultURL>
     <URL>
       www.FLAME08app.com/search/
       Gym_Query_Service_02
     </URL>
  </Services>
  <Selector>
     <SelectFunction>
       www.FLAME08app.com/select
       /Gym_selector01
     </SelectFunction>
  </Selector>
</Role>
```

Fig. 6. The Role Specification of Gym_querist

```
<Role Name="Gym_querist" >
  <Feature Name="result_searcher">
     <Argument Name="Order_Number"
      Type="String"
      PassMode="IN"/>
     <Argument Name="Result_info"
      Type="ResultVector" –
      PassMode="OUT"/>
  </Feature>
  <Services>
     <DefaultURL>
       www.FLAME08app.com/search/
       Order_Result_Query_Service_01
     </DefaultURL>
     <URL>
       www.FLAME08app.com/search/
       Order_Result_Query_Service_02
     </URL>
  </Services>
  <Selector>
     <SelectFunction>
       www.FLAME08app.com/select
       /Order_resultSel
     </SelectFunction>
  </Selector>
</Role>
```

Fig. 7. The Modified Specification of Gym_querist

The above case study demonstrates that a role-based service connector enables service connection adaptation smoothly. Moreover it shows that <*Feature*> and <*Services*> separates services from features and makes it feasible to modify the interaction interface and its implementation independantly. By changing the <*SelectFunction*>, the relation between interaction interface and services can be adjusted.

Table 1 lists the comparison of service connection adaptations in those two prototypes of the case. It says that these connections can be adjusted, and the adjustments all impact application behaviors with effects to semantics. However, adaptations of control flow based service connections and service adaptors involve more efforts of executants obviously. Role-based service connector can be reconfigured at runtime. And changes of the connection are incremental, which means that a new service can be incorporated into the service composition while old ones co-exist [13]. With the CAFISE Framework, the adaptation process is semi-automatic, and change impacts can be controlled. The comparison shows that the role-based service connector model provides more support to connection adaptation.

Table 1. Properties of Service Connection Adaptations in the Case

	Control flow based service connections	Service adaptors	Role-based service connectors
Adaptation executants	programmers	programmers	users
Adaptation way	by modifying source codes	by modifying source codes or customizing	by reconfiguring
Degree of automation	non-automation	non-automation	semi-automation
Adaptation time	at non-runtime	at non-runtime	at both runtime and non-runtime
Incremental changes	no	no	yes
Effects to semantics	yes	yes	yes
Change impacts control	no	no	yes

3.3 Evaluation Based on the Case Study

The case study demonstrates how to use a role-based service connector model to make service connections adaptable. In addition, it also shows that a role-based service connector has advantages in improving connection adaptability at the following aspects:

- **Communication Stability**

  Communication is the essential function of a role-based service connector, which takes charge of data exchange between services involved in an interaction. Through features, roles expose an interaction interface, and allow requestors to invoke services. Role features offer a unified interface for service interaction, improving communication stability from the view of connection structure.

- **Structure Expansibility**

  A role-based service connector is extensible in structure aspect. Service resources and user requirements are various and mutable. Unavoidably, service connections have to co-evolve with changes. A role-based service connector provides an extensible cadre composed of <*Feature*>, <*Services*> and <*Selector*>, which enables the connector to be extended and reconfigured according to changes.

- **Connection Adaptability**

  The role-based service connector model provides essential support to connection adaptation. With the extensible cadre, it can be modified and reconfigured. Besides that, it can accommodate changes of connection to some extent through encapsulating changes in service providers. And the connection can be adapted at run time, for role-based service connector can dynamically switch service provider in the way of modifying parameters to change connection structure at run time.

4 Conclusions

Service-oriented application development in network environments meets large challenges due to open and dynamic features of the environments, which requires service connections to be adaptable.

In this paper, a role-based service connector model is presented to solve the problem. With role features, a role-based service connector offers a changeable service connection structure, which makes

connections more adaptable: by modifying the feature and related service references, the connection can be reconfigured. In addition, adjustments in a role-based connector are limited to some modifications; changes of interaction partners do not influence each other. So that, the connector enhances the flexibility of service coordination.

Through the case study of project FLAME2008, we conclude that the role-based service connector model has advantages in adaptation of service connection. Besides that, the following work should be done.

- To reduce side effects of adaptation, a run-time model of the connector should be offered to monitoring status of service connection;
- The application of the model in Web service chaining, such as BPEL4WS, is to be considered in further work.

Acknowledgements

When we wrote the paper, Dr. Agnes Voisard gave some good suggestions; Dipl.-Inf. Norbert Weissenberg corrected the writings. And Dipl.-Inf. Rdiger Gartmann gave generous helps on paper presentation. We are grateful to them for their helps.

References

1. I. Foster, C. Kesselman, J. Nick, S. Tuecke: Grid Services For Distributed System Integration. Computer. vol. 35, no.6 (2002) 37-46
2. I. Foster, C. Kesselman, S. Tuecke: The anatomy of the grid: Enabling scalable virtual organizations. The International Journal of Supercomputer Applications. vol.15 no.3(2001) 200-222
3. F. Casati, S. Ilnicki, J. LiJie, S. Ming-Chien: An Open, Flexible, and Configurable System for E-Service Composition. The Second International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, Milpitas, USA, June 2000
4. E. Kiciman, A. Fox: Using Dynamic Mediation to Integrate COTS Entities in a Ubiquitous Computing Environment. The Second International Symposium on Handheld and Ubiquitous Computing, Bristol, UK, September 2000
5. Emre Kiciman etc: Position Summary: Towards Zero-code Service Composition. The Eighth Workshop in Hot Topics in Operating Systems, Oberbayern, Germany, May 2001
6. S. Krishnan1, P. Wagstrom1, G. Laszewski: GSFL: A Workflow Framework for Grid Services. http://www-unix.globus.org/cog/projects/workflow/, July 2002
7. T. Andrews, F. Curbera etc.: Business Process Execution Language for Web Services Version 1.1. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, May 2003
8. F. Steimann: On the Representation of Roles in Object-Oriented and Conceptual Modeling. Data & Knowledge Engineering. vol. 35, no.1(2000)83-106
9. B. Kristensen: Object-oriented Modeling with Roles. The 2nd International Conference on Object-Oriented Information Systems, Dublin, Ireland, 1995
10. J. Goguen: A Categorical Manifesto. Mathematical Structures in Computer Science. vol. 1, no. 1(1991)49-67
11. Cervesato etc.: A Meta-notation for Protocol Analysis. The 12th IEEE Computer Security Foundations Workshop, Mordano, Italy, June 1999

12. Y. Han, Z. Zhao, G. Li etc.: CAFISE: An Approach to Enabling Adaptive Service Configuration of Service Grid Applications. Journal of Computer Science and Technology. vol. 18, no.4(2003) 484-494
13. G. Li: Adaptive software architecture and Adaptive software architecture development. Ph.D. Dissertation, Beijing University of Aeronautics and Astronautics. 2002