A Novel Process Network Model for Interacting Context-Aware Web Services

Xiuguo Zhang, Hongbo Liu, and Ajith Abraham, *Senior Member*, *IEEE*

Abstract—Context-aware web services have been attracting significant attention as an important approach for improving the usability of web services. In this paper, we explore a novel approach to model dynamic behaviors of interacting context-aware web services, aiming to effectively process and take advantage of contexts and realize behavior adaptation of web services and further to facilitate the development of context-aware application of web services. We present an interaction model of context-aware web services based on context-aware process network (CAPN), which is a data-flow and channel-based model of cooperative computation. The CAPN is extended to context-aware web service network by introducing a kind of sensor processes, which is used to catch contextual data from external environment. Through modeling the register link's behaviors, we present how a web service can respond to its context changes dynamically. The formal behavior semantics of our model is described by calculus of communicating systems process algebra. The behavior adaptation and context awareness in our model are discussed. An eXtensible Markup Language-formatted service behavior description language named BML4WS is designed to describe behaviors and behavior adaptation of interacting context-aware web services. Finally, an application case is demonstrated to illustrate the proposed model how to adapt context changes and describe service behaviors and their changes.

Index Terms—Context-aware web service, service behavior, CCS process algebra, CAPN, service interaction

1 Introduction

Context-aware web service can help its system to understand situational context and share that context with other services. "Context" mainly refers to the information about clients and their environment that may be used by web services to provide clients with a customized and personalized behavior [1], [2]. Context information includes any additional information, for example, a consumer's name, current location and address, type of client device, and so on, which can be used to improve the behavior of a service in a situation [3], [4], [5], [6]. Without such additional information, the service would be operable as normal but with context information, which is arguable that the service can operate better or more appropriately [7], [8], [9], [10]. Context-awareness is considered as a kind of ability of the application to discover and take advantage of context information. In context-aware application of web services [11], [12], a web service can adapt its operations according to its contexts, that is, service behaviors change as the context changes. To respond to context changes, a context-aware web service has to accommodate for a variety of context types and tune their behaviors dynamically with the changing contexts. Context-aware web services have been attracting significant attention as an important approach to improving the usability of web services [13], [14], [15], [16], [17], [18], [19].

- *X. Zhang and H. Liu are with the School of Information Science and Technology, Dalian Maritime University, Dalian 116026, China. E-mail: {zhangxg, lhb}@dlmu.edu.cn.*
- *A. Abraham is with the Machine Intelligence Research Labs, Seattle, WA 98071, and IT4Innovations, VSB-Technical University of Ostrava, Czech Republic. E-mail: ajith.abraham@ieee.org.*

Manuscript received 9 May 2011; revised 6 Nov. 2011; accepted 23 Jan. 2012; published online 6 Feb. 2012.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSC-2011-05-0042.
Digital Object Identifier no. 10.1109/TSC.2012.6.

In this paper, we focus on the behavior of interacting context-aware web services and automatic behavior adaptation of web services to context changes. We tackle the problem of how contexts are processed and how they affect service behaviors dynamically. A novel approach is proposed to model dynamic behaviors of interacting context-aware web services, aiming to effectively process and take advantage of contexts and realize behavior adaptation of web services and further to facilitate the development of context-aware application of web services. We present an interaction model of context-aware web services based on a formal model, context-aware process network (CAPN) [20], [21], which is a data-flow and channel-based model of cooperative computation. The CAPN is extended to a context-aware web service network by adding a kind of additional sensor processes, which is used to catch contextual data from external environment. In CAPN, the application data and context data are not distinctly separated, while in a context-aware web service network, the separation of application data and context data is enforced by separating respective reading and writing semantics. This separation helps us to realize behavior adaptation of context-aware web service. We use unidirectional register link of CAPN to carry context information of web services. Through modeling the register link's behaviors, we present how the web service's behavior dynamically changes as context changes. Using the CAPN, we aim to reduce unanticipated circumstances and uncooperative cases of interacting web services.

This paper is organized as follows: Related works are presented in Section 2. Section 3 describes the service interaction model of context-aware web service network. Section 4 presents behavior semantics of context-aware web service network. Section 5 introduces the realization of

behavior adaptation and context awareness in context-aware web service network. Section 6 provides an XML-formatted service behavior description language for describing behaviors and behavior adaptation of interacting context-aware web services. Section 7 presents a case to show how the application can adapt to context changes and how to describe service behaviors and their changes. Section 8 draws conclusions and future work.

2 Related Works

2.1 Context-Aware Web Service

Recently, different technologies, approaches, and frameworks for context-aware web services and their application have been researched [1], [4], [22], [23], [24], [25], [26], [27]. A context-aware web service is a smart web service that can understand situational context and can share that context with other services [4], [28]. In this paper, we are mainly concerned about dynamic behaviors of interacting web services. We consider a web service as context aware if it uses context to provide customized and personalized behaviors during its interaction with other services.

The definition of context depends on the purpose of the application. For example, in its broad sense in [29], context is defined as any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant or the interaction between a user and an application, including the user and applications themselves. For web services [5], context is defined as the information characterizing the situation in which the services are being executed. In this paper, we consider a context as any information that can be used to characterize the situation of interactions among web services, including state of interaction environment, state of entities in interaction environment such as user state and web service state.

Context types also differ form situation to situation. For example, Keidl and Kemper [1] presented several context types including the consumer's location, client's devices, information about the consumer invoking the web service, for example, name and e-mail address, connection preferences, which allows to specify properties of the connections to web services. In [29], three categories of context are distinguished: physical (location, time, etc.), computing (terminal form factor, battery life, available bandwidth, accessible computing facilities, etc.), and user context (current activity, schedule, intent, etc.). In this paper, we use three kinds of contexts to characterize the situation of service interactions, including interaction environment context (network bandwidth, operating system, client's devices, etc.), user context (user preference, user location, etc.), and service context (business restriction, service availability, execution time, success rate, etc.).

2.2 Web Service Behavior Modeling

At present, the main methods of modeling web service behaviors are the Petri net-based method [30], [31], the process algebra-based method [32], [33], and the finite-state machine-based method [34]. These methods are formal and commonly used to describe and verify web service behaviors. For example, the Petri net-based behavior modeling method focus on describing behavior process inside web service and is fit to verify correctness of composite service behaviors. Process algebra-based behavior modeling method focus on describing behavior process outside web service and is fit to verify interaction correctness among web services. The finite-state machine-based method describes service behaviors from the view of state transition and message passing, but it is difficult to describe behaviors of sophisticated process using a finite-state machine.

In this paper, we use the CAPN to model dynamic behaviors of interacting context-aware web services. We think that the CAPN is more fit for modeling interacting context-aware web services in several aspects, including its parallelism and communication mechanism, compositional property, executable property, and context-awareness semantics.

2.3 Behavior Adaptation of Context-Aware Web Services

Adaptation is the capability to provide different versions of a service or different presentations of a document, to suite the needs of the user, of the environment, of the equipment, etc., [35]. In this paper, we consider behavior adaptation of a context-aware web service as its capability to provide different versions of customized and personalized behaviors for meeting the needs of context changes.

Chaari and Celentano [36] categorized context adaptation into three categories: services adaptation, content adaptation, and UI adaptation. Here, services refer to the services of the application, content is the exchanged data with the user, and UI is the visualization. They developed a platform that makes the services, data and the user interface of applications adaptable to different context situations.

Jacob and Steglich [25] define the notion of context functions as a statement about the relative influence of a context parameter on the behavior of the respective service. Context functions can be a mathematical function or a regular expression and can be easily adapted to directly affect the context-aware service behavior.

Zhou et al. [26] proposed the concept of context-aware pervasive service composition (CAPSC) and design a CAPSC architecture by taking into account context-aware peer coordination, context-aware process service adaptation, and context-aware utility service adaptation.

Hervás and Bravo [27] exploited semantics to apply context in runtime adaptation. They implement mechanisms to support the dynamic behavior of the users and their surroundings, including techniques to adapt context model to their future needs, to maintain the context information at runtime, and to be interoperable with external context models.

To summarize, different behavior adaptation approaches tackle the problem of behavior adaptation in different manners. In this paper, based on the CAPN model, we present our own context processing and behavior adaptation approaches. Similar to services adaptation in [36], we also provide different behavior versions for a context-aware web service, but we provide a register link based behavior adaptation strategy and a more detailed version selection approach. We adopt context influence functions to denote the influence of contexts on web service. The influence

function in this work is similar to context function in [25] but their context function is not used for behavior version selection of context-aware web service. Also, our behavior adaptation strategy is different from [25] because we provide a unified and formatted influence function for every context to calculate the rank of a behavior version, but Jacob and Steglich [25] do not. Using the CAPN to model context-aware web service interaction, we can present an explicit formal semantic for interacting context-aware web service that facilitates the development of context-aware application of web services.

3 Context-Aware Web Service Network: A Service Interaction Model

Service interaction describes how services can communicate with each other at the message level. In this section, we describe an interaction model of context-aware web services. We name this interaction model as a context-aware web service network. Its design idea comes from a data-flow and channel-based model of cooperative computation, i.e., CAPN [20], [21]. Next, we briefly introduce the model of the CAPN.

3.1 CAPN

Kahn process network (KPN) [20] is a model of computation based on data flow and has been widely used in many areas [37], [38], [39]. A strong aspect of KPNs is that they make (task-level) parallelism and communication in an application explicit, which means that they are very suitable for execution on distributed architectures. In KPN, nodes represent computations (concurrent processes), and arcs represent totally ordered sequences of data (commonly called tokens). Concurrent processes communicate with each other using FIFO channels that are unidirectional unbounded queues of data tokens between two processes. Read actions from these FIFOs block until at least one data item (or token) becomes available. The FIFOs have unbounded size, so writing actions are nonblocking. Reading actions from the FIFOs are destructive, which means that a token can only be read once. KPN are determinate, i.e., given an input, the output result of its computation is independent of execution order. This allows us to execute the network in parallel, as well as sequentially. KPNs implement asynchronous communication but synchronous coordination. Geilen and Basten [40] presented an operational semantics for the KPN.

To facilitate asynchronous coordination, van Dijk et al. [21] extended the Kahn model of computations to the CAPN with a simple indeterminate construct, i.e., a unidirectional nonblocking register link (REG), which has destructive and replicative behavior. They defined the behavior of a REG link as follows: Writing to a full register overwrites a previously written value. Reading from a register returns the last received value; a value can be returned multiple times. Because the context information in REG as control information may decide the output of a process, i.e., given an input, the output result of its computation changes as contexts change, so CAPN is *indeterminate*.

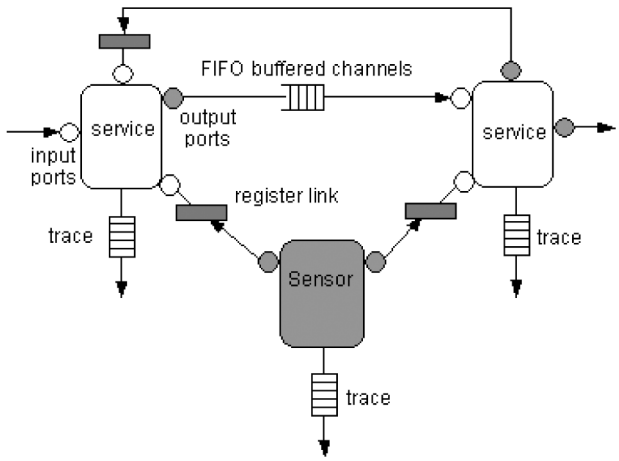


Fig. 1. Context-aware web service network.

3.2 Context-Aware Web Service Network

Several advantages of CAPN make it adequate to model service interactions:

1. parallelism and communication mechanism in CAPNs, which enable distributed service interaction on Internet;
2. CAPNs are compositional, which corresponds to the possibility to build bigger behaviors from small ones;
3. CAPN can be executed, which ensures an executable service interaction environment for actual application; and
4. context-awareness semantics in CAPNs accords to context-awareness semantics of web services.

So, we use the CAPN to model interaction of context-aware web services.

Fig. 1 shows the interaction model of context-aware web services, i.e., context-aware web service network. A web service is modeled as a computing process of the CAPN. Web services communicate with each other using FIFO channels. The main modeling elements of a context-aware web service network are service, sensor, input port (denoted as a small white circle), output port (denoted as a small gray circle), FIFO-buffered channels, register link, and trace. In this model, to simplify the process of context capturing, we add a sensor element that is a kind of additional CAPN processes and only used to get contexts from environment and send them to web services. The normal CAPN process, i.e., computing process is used to deal with application data among web services.

3.2.1 Modeling Web Service and Sensor Service

In this model, we model a web service as a computing process of the CAPN. A web service is described as a service interface that has a set of service operations. A service operation consists of a number of input and output ports through which the input and output parameters of service operations are passed. We model a sensor service as an additional process. For convenience, a sensor service is defined as a kind of specific web services, similar to a web service in its interface, operation, input and output ports, but different from a web service in its behavior. A sensor service can only link to register links and cannot link to FIFO channels, while a web service does not have this limitation.

Fig. 2. Register link between a web service and a sensor service.

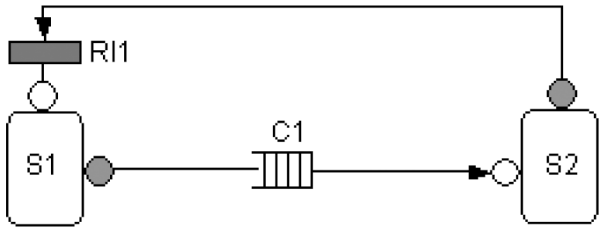


Fig. 3. Register link between two web services.

3.2.2 Modeling Behaviors of FIFO-Buffered Channels

A web service communicates with others using FIFO-buffered channels that are unbounded queues of data tokens between two services. For each channel, there is a single service that produces messages and a single service that consumes messages. Multiple producers or multiple consumers connected to the same channel are not allowed. Data are consumed from these queues in a first-in first-out order. Each service port is linked to at most one other port through a unidirectional unbounded FIFO queue. Services are allowed to read from an input port and are allowed to write to an output port. The read operation is blocking, and the write operation is nonblocking. When tokens are available on every input, the service will execute.

3.2.3 Modeling Behaviors of Register Links

In this model, register link is also a unidirectional nonblocking channel. But the behavior semantic of a register link and a FIFO-buffered channel is very different. In [21], the behavior of a register link is defined as follows:

- Writing to a full register link overwrites a previously written value.
- Reading from a register link returns the last received value.
- A value can be returned multiple times.

In this paper, we only care about the context's influence on web service's behaviors. A register link is used to carry context information between two web services or between a web service and a sensor service. We do not consider the circumstance that a register link is between two sensor services.

The circumstance of a register link between a web service and a sensor service is depicted in Fig. 2.

In this circumstance, context information, which is captured by sensor, including network bandwidth, user preference, and so on, will be sent to a web service for use. When the web service is to execute, the event of reading from its input register link Rl1 will be triggered, to make sure that the latest contexts are read and used by the web service. After execution, the web service may write other context information to its output register link $Rl2$.

As to the circumstance of a register link between two web services, we consider the following case, depicted in Fig. 3.

In this case, $S1$ and $S2$ are two interacting web services. S1 interacts with $S2$ through FIFO-buffered channel $C1$ and a register link $Rl1$. The execution of $S1$ will trigger a writing event to $C1$. When a reading event from $C1$ finishes, $S2$ will execute. Also, the execution of $S2$ will trigger a writing event to $Rl1$, to notify $S1$ whether the interaction is successful. So, the context information in $Rl1$ denotes feedback information from $S2$. From this feedback information, $S1$ will decide whether to adjust its control strategy and reply or not.

3.2.4 Service Interaction Events

In this model, there are five service interaction events, which are reading from an FIFO-buffered channel, writing to an FIFO-buffered channel, reading from a register link, writing to a register link, and executing a web service. We named these five events as $readchannel()$, $writechannel()$, $readreg()$, $writereg()$, and $execute()$. For an interacting web service, its behavior trace is composed of four reading/writing events ($execute()$ is an internal event), the event order lies in the computation logic. A single trace denotes the execution of a single web service and a composite trace of all services represents the unrolled execution of context-aware web service network.

4 Behavior Semantics of a Context-Aware Web Service Network

In this section, based on the works in [40], we describe behavior semantics of context-aware web service network using CCS process algebra [41], [42]. We add context awareness, behavior compatibility, and some other properties to basic semantics of the KPN.

Let C be the set of FIFO channels and for each channel $c \in C$, with a channel alphabet Σ_c. Let R be the set of register links, and for every register link $r \in R$, with a alphabet Λ_r.

Definition 1 (Context-aware web service). *A context-aware web service is a labeled transition system* $LTS = (S; s_0; I_C; O_C; I_R; O_R; Op; Pt; Ctxp; Act; \rightarrow)$, *where*

- S *is the set of service states;*
- $s_0 \in S$ *is the initial service state;*
- $I_C \subseteq C$ *is the set of input channels;*
- $O_C \subseteq C$ *is the set of output channels;*
- $I_R \subseteq R$ *is the set of input register links;*
- $O_R \subseteq R$ *is the set of output register links;*
- Op *is the set of service operations;*
- Pt *is the set of service ports provided by service operations;*
- $Ctxp$ *is the context processor that is a software module used to process contexts inside the web service; and*
- Act *represents the set of actions,* $Act = \{c?a, c!a | c \in I_C \cup O_C, a \in \Sigma_c\} \cup \{r?b, r!b | r \in I_R \cup O_R, b \in \Lambda_r\} \cup \{\tau\}$, *where* $c?a$ *represents a read action from channel* c *and* $c!a$ *represents a write action to channel* c *with*

token a, $r?b$ represents a read action from register link r and $r!b$ represents a write action to register link r with token b, $\{\tau\}$ is the internal or silent action (all other actions); $\rightarrow \subseteq S \times Act \times S$ is a labeled transition system, we write $s_1 \xrightarrow{\alpha} s_2$ if $(S_1; \alpha; s_2) \in \rightarrow$.

Definition 2 (Context-aware web service network). *A context-aware web service network is a tuple $(P; N_C; N_R; I_C; O_C; I_R; O_R; Act; \{LTSp | p \in P\})$, where*

- *P is the set of web services;*
- *$N_C \in C$ is the set of internal channels (between two services);*
- *$N_R \in R$ is the set of internal register links (between two services);*
- *$I_C \in C$ is the set of input channels;*
- *$O_C \in C$ is the set of output channels;*
- *$I_R \in R$ is the set of input register links;*
- *$O_R \in R$ is the set of output register links;*
- *Act represents the set of actions, $Act = \{c?a, c!a \mid c \in I_C \cup O_C \cup N_C, a \in \Sigma_c\} \cup \{r?b, r!b \mid r \in I_R \cup O_R \cup N_R, b \in \Lambda_r\} \cup \{\tau\}$, where $c?a, c!a, r?b, r!b, \{\tau\}$ have the same meaning as that in Definition 1; and*
- *$LTS_p = (S_p; s_{p_0}; I_{C_p}; O_{C_p}; I_{R_p}; O_{R_p}; Op_p; Pt_p; Ctxp_p; Act_p; \rightarrow)$ denotes the labeled transition system of $p \in P$.*

Definition 3 (Configuration). *A configuration of a context-aware web service network is a 3-tuple (ψ, η, ζ), where*

- *$\psi : P \rightarrow S = \cup_{p \in P} S_p$ is a service state function that maps every service $p \in P$ to a local state $s \in S_p$, here S_p is the state set of service p;*
- *$\eta : C \rightarrow \Sigma^*$ is a channel state function that maps every channel $c \in C$ to a finite string $\eta(c)$ on Σ_c, here Σ_c is the corresponding channel alphabet of c, Σ denotes the union of all channel alphabet, and Σ^* denotes the set of all finite (and infinite) strings over Σ; and*
- *$\zeta : R \rightarrow \Lambda^*$ is a register link state function that maps every register link $r \in R$ to a finite string $\psi(r)$ on Λ_r, here Λ_r is the corresponding register link alphabet of r, Λ denotes the union of all register link alphabet, and Λ^* denotes the set of all finite (and infinite) strings over Λ.*

In a context-aware web service network, there exists an initial configuration $(\psi_0, \eta_0, \zeta_0)$. Here, ψ_0 maps every service $p \in P$ to its initial state s_{p_0}, η_0 maps every channel $c \in C$ to an empty string, and ζ_0 maps every register link $r \in R$ to an empty string.

Definition 4 (Behavior semantics of channel and register link). *The reading and writing semantics of FIFO channel and register link are defined as follows:*

1. *A service reads from a channel:*

$$\frac{\psi(p) \xrightarrow{c?a} s, \eta(c) = a\sigma, c \in C}{(\psi, \eta) \xrightarrow{c?a} (\psi[s/p], \eta[\sigma/c])}. \quad (1)$$

2. *A service writes to a channel:*

$$\frac{\psi(p) \xrightarrow{c!a} s, \eta(c) = \sigma, c \in C}{(\psi, \eta) \xrightarrow{c!a} (\psi[s/p], \eta[a\sigma/c])}. \quad (2)$$

Here, $\sigma \in \Sigma^*$ and $\psi[s/p]$ is the function with the same domain as ψ except that $\psi(p) = s$. $\eta[\sigma/c])$ is the function with the same domain as η except that $\eta(c) = \sigma$.

3. *A service reads from a register link:*

$$\frac{\psi(p) \xrightarrow{r?b} s, \zeta(r) = b\rho, r \in R}{(\psi, \zeta) \xrightarrow{r?R} (\psi[s/p], \zeta[b\rho/r])}. \quad (3)$$

Here, we assume that b is the last received value in register link r. If b has not been rewritten by a writing event, it is still in a register link for further reading. And, $\rho \in \Lambda^*$, $\psi[s/p]$ is the function with the same domain as ψ except that $\psi(p) = s$, $\zeta[\rho/r]$ is the function with the same domain as ζ except that $\zeta(r) = \rho$.

4. *A service writes to a register link:*

$$\frac{\psi(p) \xrightarrow{r!b} s, \zeta(r) = \rho, r \in R}{(\psi, \zeta) \xrightarrow{r!R} (\psi[s/p], \zeta[b\rho/r])}. \quad (4)$$

Here, we assume register link r is not full. When r is full, let b_0 is the last token in register link r, the above expression will be

$$\frac{\psi(p) \xrightarrow{r!b} s, \zeta(r) = \rho b_0, r \in R}{(\psi, \zeta) \xrightarrow{r!R} (\psi[s/p], \zeta[b\rho/r])}. \quad (5)$$

That is, after b is written to the full register link r, the last token b_0 in a register link r has been rewritten.

Definition 5 (Execution of context-aware web service network). *A labeled transition system LTS is defined as a tuple $(Z; z_0; N_C; N_R; I_C; O_C; I_R; O_R; Act; \rightarrow)$, where*

- *Z is the set of all configurations;*
- *$z_0 \in Z$ is the initial configuration;*
- *$N_C \subseteq C$ is the set of internal channels (between two services);*
- *$N_R \subseteq R$ is the set of internal register links (between two services);*
- *$I_C \subseteq C$ is the set of input channels;*
- *$O_C \subseteq C$ is the set of output channels;*
- *$I_R \subseteq R$ is the set of input register links;*
- *$O_R \subseteq R$ is the set of output register links;*
- *Act represents the set of actions, the meaning is the same as that in Definition 2; and*
- *$\rightarrow \subseteq Z \times Act \times Z$ is a labeled transition relation, we write $z_1 \xrightarrow{\alpha} z_2$ if $(z_1; \alpha; z_2) \in \longrightarrow$.*

An execution of a context-aware web service network is a sequence $k = z_0 \xrightarrow{\alpha_0} z_1 \xrightarrow{\alpha_1} z_2 \xrightarrow{\alpha_2} \cdots z_i \xrightarrow{\alpha_i} \cdots$ of configurations, $z_i \in Z$ and action $\alpha_i Act$, such that $z_i \xrightarrow{\alpha_i} z_{i+1}$ for all $i \geq 0$. Every execution step $z_i \xrightarrow{\alpha_i} z_{i+1}$ denotes a synchronization point of interacting web services in a context-aware web service network.

Definition 6 (Behavior trace and composite behavior trace). *In an execution of a context-aware web service network, as described in Definition 5, we define a* behavior trace *of an interacting context-aware web service p as a finite action sequence $\lambda_p = \alpha_{p1}\alpha_{p2}\cdots\alpha_{pi}\cdots\alpha_{pn}$, where $\alpha_{pi} \in Act$, Act represents the set of reading or writing actions, the meaning is the same as that in Definition 2, $1 \le i \le n$, $n \in N$, N is the set of positive integers. We define the finite sequence $\lambda = \alpha_0\alpha_1\cdots\alpha_i\cdots$ in execution $k = z_0 \xrightarrow{\alpha_0} z_1 \xrightarrow{\alpha_1} z_2 \xrightarrow{\alpha_2} \cdots z_i \xrightarrow{\alpha_i} \cdots$ as a composite behavior trace of this network.* Composite behavior trace *represents the unrolled execution of this network. We use $trace(p)$ to denote behavior trace of an interacting context-aware web service p.*

Behavior traces can be used to analyze and verify the behaviors of interacting web services. Behavior traces as behavior description of a web service can serve as inputs to related verification tools (e.g., CWB-NC [42]), where the service behavior represented in the description can be analyzed and verified.

In a context-aware web service network, we consider contexts in register links as a kind of controlling information to adjust internal control flow of a web service, not as the parameters of a web service. Only when a web service is to execute, it reads contexts from register links. So, if we do not consider the influence of contexts, this network becomes corresponding KPN. Next, we define behavior compatibility of two interacting web services, we only consider writing/reading actions on internal FIFO channels between this two interacting web services.

Let Q be the set of web services, $q_0 \in Q$, $q_i \in Q$, where $1 \le i \le n$, $n \in N$, N is set of positive integers.

Definition 7 (Weak transitions). *Let $q \in Q$, $q' \in Q$, τ is transition label for action that is not externally visible, if*

- $q \Rightarrow^{\varepsilon} q'$ *iff* $q = q_0 = \rightarrow^{\tau} q_1 \rightarrow^{\tau} \cdots \rightarrow^{\tau} q_i \rightarrow^{\tau} \cdots \rightarrow^{\tau} q_n = q'$, $n \ge 0$;
- $q \Rightarrow^{\tau} q'$ *iff* $q \Rightarrow^{\varepsilon} q'$;
- $q \Rightarrow^{\alpha} q'$ *iff* $q \Rightarrow^{\varepsilon} q_1 \rightarrow^{\alpha} q_2 \Rightarrow^{\varepsilon} q' (\alpha \ne \tau)$,

then $q \Rightarrow^{\alpha} q'$ is a weak transition.

Definition 8 (Observational equivalence). *Let $S \subseteq Q \times Q$. The relation S is a weak bisimulation relation if whenever $q_1 S q_2$, then*

- *if $q_1 \rightarrow^{\alpha} q_1'$, for some q_2', implies $q_2 \Rightarrow^{\alpha} q_2'$, and $q_1' S q_2'$;*
- *if $q_2 \Rightarrow^{\alpha} q_2'$, for some q_1' implies $q_1 \Rightarrow^{\alpha} q_1'$, and $q_1' S q_2'$;*

then q_1 and q_2 is observationally equivalent, or weakly bisimulation equivalent, that is, for an external observer, it is not possible to distinguish the behavior of q_1 and q_2.

Definition 9 (Behavior compatibility). *Two interacting web services p and q are behavior compatible, denoted as $compatible(p, q)$, if they have opposite behavior trace (only including reading/writing actions on FIFO channels between this two web services), i.e., p is observationally equivalent to q.*

For example, "!" denotes writing action, $writingP(p)$ denotes the writing token sequence of service p, and $readingQ(q)$ denotes the reading token sequence of service q. Let $trace(p) = !Req.Receive.nil$, $trace(q) = Req.!Receive.nil$, it is clear that $writingP(p) = readingQ(q)$ and $writingQ(q) = readingP(p)$ when p and q execute a reading/writing action, respectively, that is, $p \xrightarrow{!Req} p'$, $q \xrightarrow{Req} q'$, then, $p' = Receive.nil$, $q' = !Receive.nil$. It is clear $writingP(p') = readingQ(q')$ and $writingQ(q') = readingP(p')$ when p' and q' execute a reading/writing action, respectively, that is, $p' \xrightarrow{Receive} nil$, $q' \xrightarrow{!Receive} nil$, p' and q' will be nil. It's clear that behaviors of p and q are opposite. So, they are behavior compatible.

When a context-aware web service network is executing, deadlock may be occurred. In [43], Parks distinguishes between two types of deadlocks. "True" deadlocks are those described in Kahn's model, that is, all processes are blocked on reads and all channels are empty, then the computation has terminated. "Artificial" deadlocks are caused by bounded channel capacities. Since channel capacities are not restricted in Kahn's model, artificial deadlock is impossible.

In a context-aware web service network, a deadlock can only be "True" deadlock. Since both FIFO-buffered channels and register link have unbounded size.

Theorem 1 (Undecidable). *The problem of whether a context-aware web service network will terminate is undecidable.*

Proof. KPN can be thought of as a set of Turing machines connected by one-way tapes. It is impossible to decide whether an arbitrary Turing machine program will halt [43], [44]. So, it is undecidable if a KPN will terminate. This implies the fact that it is undecidable whether a context-aware web service network will terminate. □

Let $I = I_C \cup I_R$, $O = O_C \cup O_R$, in an execution k of this network; we define an input function $in : I \rightarrow \Sigma^* \cup \Lambda^*$ that maps every input channel to a finite (and infinite) strings on Σ^* and every input register link to a finite (and infinite) strings on Λ^*. We define a function $out : O \rightarrow \Sigma^* \cup \Lambda^*$ that maps every output channel to a finite (and infinite) strings on Σ^* and every output register link to a finite (and infinite) strings on Λ^*. For some $c \in I$, we use $k?c$ to denote the input consumed on c and $k?I$ to denote the input consumed by the network in execution k. For some $o \in O$, we use $k!o$ to denote the output on o, and $k!O$ denotes the output of the network in execution k.

Definition 10 (Completeness). *An execution k of a context-aware web service network with an input $in : I \rightarrow \Sigma^* \cup \Lambda^*$ is complete, denoted as $complete(k)$, iff the execution of corresponding KPN is complete [43], i.e.,*

- *$\forall p \in P$, P is the set of web services, $\exists n \in N$, N is the set of positive integers, such that $0 \le |trace(p)| \le n$; and*
- *$\forall o \in O$, $(k!o)/R = out(o)/R$, $(k!o)/R$ denotes the output of the context-aware web service network in execution (taking out those outputs produced by actions involving register links set).*

That is, in the execution k of corresponding KPN, every behavior trace of web service has a finite number of actions, and none of the output can be extended again.

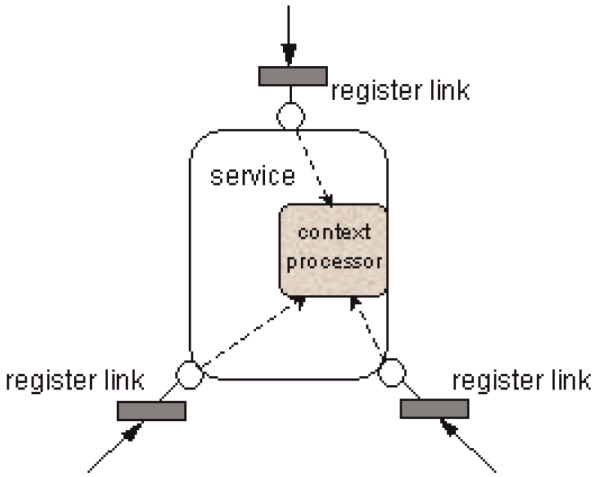


Fig. 4. Context processor of a web service.

Definition 11 (Effectiveness). *Here, we add the support of register links to the definition of maximality and effectiveness of an execution of KPN in [40] and define effectiveness of an execution of a context-aware web service network. An execution k of a context-aware web service network with an input $in : I \rightarrow \Sigma^* \cup \Lambda^*$ is effective, denoted as $effective(k)$, iff*

- *at the last configuration, only read actions are possible, and all inputs of input $in : I \rightarrow \Sigma^* \cup \Lambda^*$ have been consumed entirely (maximality in [40], here, we consider not only FIFO channels, but also register links), i.e., if $n \in N$ is the length of the execution $z_n \overset{\alpha}{\rightarrow}$, then $\alpha = c?a$ and $k?c = in(c)$ for some $c \in I_C \cup I_R$ and $a \in \Sigma_c \cup \Lambda_r$.*
- *every token produced in internal channels and internal register links are ultimately consumed (effectiveness in [40], here, we consider not only FIFO channels, but also register links), i.e., $k?N_C = k!N_C$ and $k?N_R = k!N_R$, where N_C is the set of internal channels and N_R is the set of internal register links.*

Theorem 2 (Comp^2). *In an execution k of a context-aware web service network, if $effective(k)$, then $complete(k)$ and $compatible(p, q)$, for some $p, q \in P$, P is the set of web services in this network.*

Proof.

1. We first prove $complete(k)$ can satisfy. An execution k of a context-aware web service network is effective states that two conditions in Definition 11 are satisfied. The first condition of Definition 11 implies that at the last configuration, execution k finishes, and only a new execution with a new input can execute, that is, $\forall p \in P$, we assume that $k = z_0 \overset{\alpha_0}{\rightarrow} z_1 \overset{\alpha_1}{\rightarrow} \cdots z_i \cdots \overset{\alpha_i}{\rightarrow} z_j$, then $trace(p) = \alpha_1\alpha_2 \cdots \alpha_j$, there must $\exists n \in N, j \leq n, |trace(p)| = j$. This satisfies the first condition of completeness definition in Definition 10. Also, at the last configuration, the fact that execution k finishes implies that none of the output can be extended again, that is, $\forall o \in O$, $(k!o)/R = out(o)/R$. This satisfies the second condition of completeness definition in Definition 10.
2. We prove that $compatible(p, q)$ can satisfy. $\forall p, q \in P$, p and q is two interacting web services, let $trace(p) = \alpha_1\alpha_2 \cdots \alpha_j, trace(q) = \beta_1\beta_2 \cdots \beta_j$. According to behavior compatibility in Definition 9, we only need to consider writing/reading actions on internal channels between this two services. For an internal channel, writing action precedes its opposite reading action. The second condition of Definition 11 implies that if $\alpha_1 = c!a$ is a writing action in $trace(p)$, there must exist β_i in $trace(q)$ is the opposite reading action of α_1. Because a channel is a FIFO queue, the token a in channel c must be read first in $trace(q)$. So, β_i must be the first action in $trace(q)$, that is, $\beta_i = \beta_1$. For the same reason, β_2 is the opposite action of α_2, β_i is the opposite action of α_i, and so on, finally, p and q produce the opposite behavior trace. This satisfies the behavior compatibility in Definition 9, that is, $compatible(p, q)$, is satisfied. □

5 Behavior Adaptation of Web Services

Context-awareness allows a web service to adapt to changes of their contexts. To realize this goal, we define a context processor for every context-aware web service. A *context processor* is a software module used to collect and process every context input to a context-aware web service. According to the basic semantic of a context-aware web service network, when tokens are available on every input, the web service will execute. Now, we consider the influence of contexts on this web service. When a web service is to be executed, the $execute()$ event will trigger its context processor to work in advance, as depicted in Fig. 4.

The context processor reads from each register link in terms of its reading semantic and puts them into a context file. Then, the context processor is responsible for managing and updating this file. A web service may contain lots of context types and their values. Also, different web service contains different contexts. So, we model context information as an XML-formatted file that provides a unified description format for different contexts. The platform independent feature of XML can easily tackle the heterogeneity of different contexts. A web service uses such context information to adjust its internal control flow as well as content and format of its replies.

The context file of a web service is composed of elements of context type. A context type contains a context value and a context weight. The basic format of a context file is listed in Table 1. Table 1 contains three context types that are user preference, service availability, and network bandwidth.

During the execution of a web service, it responds to each context in context file. The basic context processing process is as follows:

1. *Parse* context file using XML parser.
2. *For each* context in context file, *calculate* weight or influence on this web service.
3. *Adjust* internal control flow and reply of this web service according to all contexts.

We use three kinds of contexts to characterize the situation of service interactions, including interaction

TABLE 1
A Context File of a Web Service

```
<ContextFile xmlns="http://ticket.com/context">
  <ServiceIdentifier>TicketBooking</ServiceIdentifier>
   <ContextType ID=http://ticket.com/context:UserPreference>
    <Context name="AircraftType">Boeing737</Context>
    <Weight>0.6</Weight>
  </ContextType>
   <ContextType ID=http://ticket.com/context:ServiceAvailability>
    <Context name="ServiceAvailability">0.93</Context>
    <Weight>0.2</Weight>
  </ContextType>
   <ContextType ID=http://ticket.com/context:NetworkBandwidth>
    <Context name="NetworkBandwidth">4M</Context>
    <Weight>0.2</Weight>
  </ContextType>
</ContextFile>
```

environment context (network bandwidth, operating system, client's devices, etc.), user context (user preference, user location, etc.), and service context (business restriction, service availability, execution time, success rate, etc.). In actual application, we can obtain these context values with various approaches. For example, network bandwidth can be obtained through related network bandwidth monitoring software, service execution time can be obtained through web log, user preference can be expressed by user when interacting with web service, service availability can be achieved by calculating the average available time of a web service in the past period of time, and so on.

Then, how do we calculate the influence of contexts on a web service? We know that a context-aware web service has to accommodate for a variety of context types. So, we need to calculate influences on a web service in different manner according to the characteristics of each context type. For example, network bandwidth may influence the execution of a web service; in case of a narrow bandwidth, the web service may terminate; in case of a normal bandwidth, the web service can output the expected result. User preference context may influence the output of web service, and we can use this context to filter the output of the web service, thus getting a new output meeting the user preference. In addition, the value type of each context is also different, some is integer, some is string, and so on, so we need to convert them to a uniform format for convenience of calculations.

We define an influence function $f(C)$ for every context type, denoting the influence of context C on a web service, whose value type is numeric and ranges from 0 to 100 or others according to the actual application. We also assign a weight w whose value is ranging from 0 to 1 for each context C, denoting the influence degree of this context type.

As discussed above, for a context-aware web service, behavior adaptation is its capability to provide different versions of behaviors to meet the needs of context changes. We define different behavior versions for a context-aware web service according to the influences of different context types and their values. In this paper, we assume that the influence function and weight are the same in different behavior versions for a certain context type. So, we only need to differentiate context types and their values in different behavior versions. We use an XML-formatted file to describe different behavior versions of a context-aware web service. The following is an example:

```
<behaviorVersion1>
   <UserPreference AircraftType="Boeing737"/>
   <ServiceAvailability>0.9</ServiceAvailability>
   <NetworkBandwidth>4M</NetworkBandwidth>
</behaviorVersion1>
<behaviorVersion2>
   <UserPreference AircraftType="Boeing747"/>
   <ServiceAvailability>0.8</ServiceAvailability>
   <NetworkBandwidth>2M</NetworkBandwidth>
</behaviorVersion2>
.....
```

Assuming that a web service has n behavior versions denoted as $s_1, s_2, \ldots, s_n$, then we use the following formula to calculate the predefined rank for each version (one version with a unique predefined rank):

$$Rank = \sum_{i=1}^{n} f_i(C_i) * w_i,$$

where $f_i(C_i)$ is the influence function of context C_i, w_i is the weight of context C_i, $1 \leq i \leq n$, n is a positive integer. When the current context values of a web service are obtained, we can calculate the rank of current context situation using the formula above. From the predefined behavior versions, we can select an appropriate one whose predefined rank is equal or closest to the calculated one.

In an actual application, we can adjust internal control of a web service by adjusting its behavior versions. If there exists a large discrepancy between the rank of current context situation and all predefined ranks, we need to adjust the behavior versions of this web service statically or dynamically to make sure an appropriate behavior version can be selected. For example, we may add a new behavior version when one or more new context types are added. We may remove a behavior version when related context types are not useful. We may also modify a behavior version by modifying related context types and their values or modifying related context influence functions and their weights. We may define a context influence function and specify its weight for a context with experiential values at the very beginning, and then adjust them dynamically and iteratively according to the reply of web service.

6 An XML-Formatted Service Behavior Description Language

Based on the basic semantics of context-aware web service network, we design an XML-formatted service behavior description language named behavior modeling language for web service (BML4WS) to describe behaviors and behavior adaptation of interacting context-aware web services. Because contexts in register links are not the parameters of a web service only as a kind of controlling information to adjust internal control flow of a web service, if we do not consider the influence of contexts, this network becomes corresponding KPN. According to KPN semantics, we can execute the KPN in parallel, as well as sequentially. So, we consider web services in a context-aware web service network as sequential processes in BML4WS.

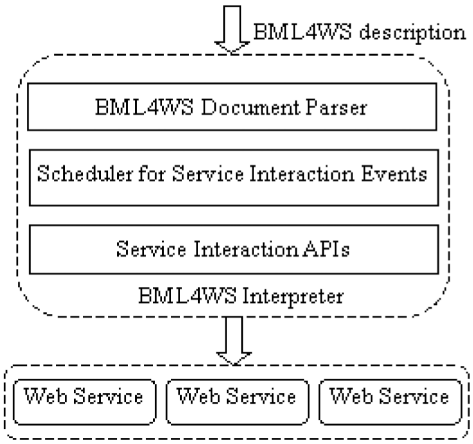


Fig. 5. Architecture of BML4WS interpreter.

6.1 Structure of BML4WS

The basic structure of this description language is as follows:

```
<bml:process>
  <bml:service name="service name">
    <!-describing service operations ->
    <bml:operation name="operation name">
     <!-describing input ports of service operation ->
       <bml:input>
        <bml:port name="port name" type="type
        name"/>
          ......
       </bml:input>
       <!-describing output ports of service operation ->
       <bml:output>
         <bml:port name="port name" type="type
         name"/>
         ......
         </bml:output>
      ......
    </bml:operation>
  </bml:service>
<bml:service name="service name">·····</bml:service>
......
  <!- defining channels ->
  <bml:channel>channel name</bml:channel>
   ......
  <!- defining register links ->
   <bml:reg>register link name</bml:reg>
     ......
<!-describing behavior interaction ->
<bml:composition>
  <!-describing sequence interactions ->
  <bml:sequence>
   <!-reading from a channel ->
     <bml:readchannel channel="channel name"
     service="service name" operation="operation
     name" port="port name"/>
  ......
   <!-executing a service operation ->
   <!-At this moment, context processor of this web service will
   read from each register link and form context file, then
   context file will be processed by this web service - >
   <bml:readreg reg="register name" service="service
    name" operation="operation name" port="port
    name"/>
    ......
      <bml:execute operation="operation name"
      service="service name"/>
       <!-writing to a channel ->
      <bml:writechannel channel="channel name"
       service="service name" operation="operation
       name" port="port name"/>
       ......
      <!-writing to a register link ->
      <bml:writereg reg="register name" service=
       "service name" operation="operation name"
        port="port name"/>
        ......
       </bml:sequence>
        ......
       <bml:loop> ····· </bml:loop>
        ......
       </bml:composition>
<bml:process>
```

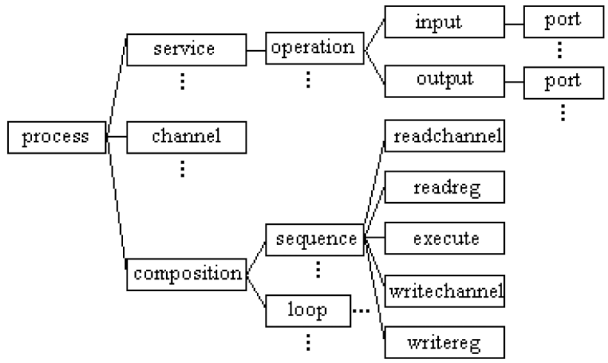


Fig. 6. DOM tree of BML4WS file.

6.2 BML4WS Interpreter

To evaluate and validate the service behavior modeling approach presented in this paper, we design a BML4WS interpreter to implement behavior interactions among web services. The architecture of BML4WS interpreter is described in Fig. 5. BML4WS interpreter is composed of three main components:

- *BML4WS document parser* is used to load and parser a BML4WS service behavior description file; the document object model (DOM) tree of BML4WS file is depicted in Fig. 6. The output is composite behavior trace of the whole system represented by this description file. Algorithm 1 shows the generation algorithm of composite behavior trace.
- *Scheduler for service interaction events* is used to schedule and execute every service interaction event in composite behavior trace. Algorithm 2 shows the scheduling algorithm of the composite behavior trace.
- *Service interaction APIs* is the Java implementation of five service interaction events that are $readchannel()$, $writechannel()$, $readreg()$, $writereg()$, and $execute()$.

Algorithm 1. BuildCompositeTrace(BML4WSFile)
Inputs: BML4WSFile: a BML4WS service behavior description file
Outputs: actionList: a composite behavior trace of the system represented by BML4WSFile

```
01  ArrayList<ActionType> actionList =
01      new ArrayList();
02  ActionType action=null;
03  for all element in BML4WSFile do
04    if(element.getName.equals("composition")) then
05      for all subelement of element do
06        action= buildAction(subelement);
06            //add into actionList
07        actionList.add(action);
08    end for
09  endif
10  end for
11  return actionList
```

BuildAction(element)
Inputs: an element of BML4WS Document
Outputs: action type: one of readchannel, writechannel, readreg, writereg and execute

```
12  elname=element.getName;
13  if (elname.equals("readchannel")) then
14    ReadChannel readChannel =
14      new ReadChannel();
15    return readChannel;
16  elseif(elname.equals("writechannel")) then
17    WriteChannel writeChannel =
17      new WriteChannel();
18    return writeChannel;
19  elseif(elname.equals("readreg")) then
20    ReadReg readReg = new ReadReg();
21    return readReg;
22  elseif(elname.equals("writereg")) then
23    WriteReg writeReg = new WriteReg();
24    return writeReg;
25  elseif(elname.equals("execute")) then
26    Execute execute = new Execute();
27    return execute;
28  endif
```

Algorithm 2. ScheduleInteractionEvents(actionList)
Inputs: actionList: a composite behavior trace of the system represented by BML4WSFile
Outputs: output of FIFO channels and register links

```
01  for all event e in actionList do
02   if(e.getName().equals("readchannel")) then
03      invoking readchannel();
04   elseif(e.getName().equals("writechannel")) then
05      invoking writechannel();
06   elseif(e.getName().equals("readreg")) then
07      invoking readreg();
08   elseif(e.getName().equals("writereg")) then
09      invoking writereg();
10   elseif(e.getName().equals("execute")) then
11      invoking execute();
12   endif
13  end for
```

We implement five service interaction events using Java remote method invocation and Java stream programming technologies.

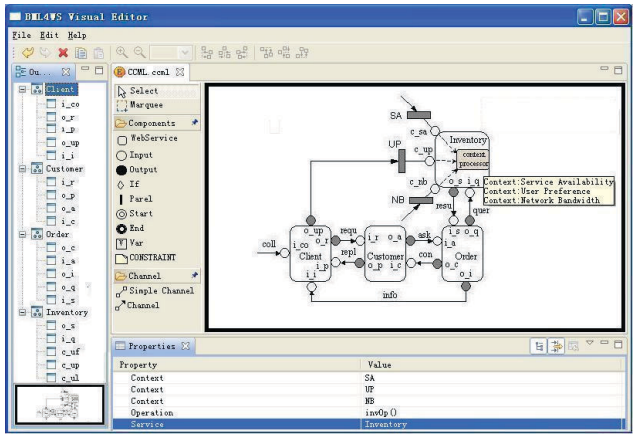


Fig. 7. A context-aware ticket booking system.

7 Case Study

7.1 A Context-Aware Ticket Booking System

We design a context-aware ticket booking system, which is composed of four interacting web services, i.e., Client service, Customer service, Order service, and Inventory service. We use visual BML4WS editor implemented by ourselves to design the context-aware ticket booking system, as shown in Fig. 7.

The basic business process of this ticket booking system is as follows:

1. A Client service collects booking requests from users through channel "coll" and form a FIFO queue of request messages.
2. The Client service reads from channel "coll" and sends a request message through channel "requ" to Customer service for booking tickets. A request message represents a ticket booking request that is an XML-formatted file, describing three booking parameters that are departure station, destination station, and date.
3. The Customer service receives a request message and sends message through channel "ask" to Order service to ask creating order.
4. The Order service sends request to Inventory service through channel "quer" for querying the ticket stock.
5. The Inventory service retrieves and sends the ticket stock result to Order service through channel "resu."
6. If the tickets have been sold out, the order will not be created, Order service will send a rejection message to Customer service through channel "con," and Customer service sends the rejection message to Client service through channel "repl."
7. If the tickets is available, the order and booking result will be created, Order service sends booking result (order information) to Client service through channel "info."
8. Return to 2 to deal with another ticket booking request.
9. Return to 1 to deal with another queue of booking request messages.

In this application, if we do not consider the influence of context information on every service, the Order service will

send user an XML file that includes all ticket booking results according to the booking parameters.

Now, we consider the influence of contexts on this system. Many context information can be usable in this ticket booking system, including user preference, user feedback, service availability, user location, network bandwidth, and so on. Assuming that we only concern about three types of context information, which are user preference (user context, UP for short), service availability (service context, SA for short), and network bandwidth (interaction environment context, NB for short). These three types of contexts will be used by Inventory service when ticket stock are retrieved, as depicted in Fig. 7.

In this case, user preference context comes from the current user's input, service availability context is obtained by calculating the average available time of this web service in the past period of time (second), and network bandwidth context comes from related network bandwidth monitoring software. User preference context will influence output result of the web service, network bandwidth, and service availability context will influence the execution efficiency of the web service.

To unify and standardize different value type of contexts, for every context, we define an influence function $f : C \rightarrow N$ that maps a context value $x \in C$ on a score n, where C is a set of context values, $n \in N$, N is the set of values ranging from 0 to 100. We define the following context influence function experientially for every context:

For user preference context, $f_{up} = 100$, for every $c \in C$, C is the set {"Boeing737," "Boeing747,"...}.

For service availability context, $f_{sa} = 100 * c$, $0 \leq c \leq 1$.

For network bandwidth context,

$$f_{nb} = \begin{cases} 100, \ c >= \ 4 \\ 80, \ 1 \leq c < 4 \\ 60, \ 0 < c < 1 \\ \ \ 0, \ c = 0. \end{cases}$$

We also need to specify a weight for every context. According to the importance degree of every context, we specify the experiential weights as 0.6, 0.2, 0.2 for user preference, service availability, and network bandwidth. In actual situation, we may dynamically adjust these influence functions and related weights according to the reply of web service. Using the rank calculation formula in Section 5, we can calculate the predefined ranks for some frequently used behavior versions, for example, a rank of a main behavior version may be $\mathrm{Rank} = 100 * 0.6 + 0.2 * 90 + 0.2 * 80 = 94$. In current situation, for user preference context, the current user can decide the actual context value through user interface provided by Client service. For example, the user may select an expected aircraft type, such as Boeing737, by clicking a CheckBox item through user interface provided by Client service. The real-time context value is passed to Inventory service through register link UP. For network bandwidth and service availability contexts, they are collected dynamically by context processor of Inventory service from their own register link. Then, context value and its weight are stored into a context file. Using all context values and their weights, the rank of behavior version under current situation can be calculated. According to the calculated rank, the context processor will select a matched behavior version from the predefined behavior versions whose rank is equal or closest to the calculated one to execute.

7.2 Describing Service Interaction Using BML4WS

In this section, we use BML4WS to describe behaviors of interacting context-aware web services. The corresponding BML4WS file can be exported from visual BML4WS editor. Here, we only describe interaction between Order service and Inventory service.

```
<bml:process>
<!- Describing Order service and Inventory service->
<bml:service name="Order">
  <bml:operation name="ordResuOp">
    <bml:input>
      <bml:port name="$i_s$" type="xml"/>
    </bml:input>
    <bml:output>
      <bml:port name="$o_c$" type="xml"/>
      <bml:port name="$o_i$" type="xml"/>
    </bml:output>
  </bml:operation>
</bml:service>
<bml:service name="Inventory">
  <bml:operation name="invOp">
    <bml:input>
      <bml:port name="$i_q$" type="xml"/>
    </bml:input>
    <bml:output>
      <bml:port name="$o_s$" type="xml"/>
    </bml:output>
  </bml:operation>
</bml:service>
<!- defining channels and register links ->
<bml:channel>quer</bml:channel>
<bml:channel>resu</bml:channel>
<bml:channel>info</bml:channel>
<bml:reg>SA</bml:reg>
<bml:reg>UP</bml:reg>
<bml:reg>NB</bml:reg>
<!- Describing interaction between Order service and
Inventory service ->
  <bml:composition>
   <bml:loop>
    <bml:sequence>
    <!-here, we omitted the description of writing to
    channel quer ->
    <!-reading from channel quer->
    <bml:readchannel channel="quer"
    service="Inventory" operation="invOp" port="$i_q$"/>
  <!-next, service operation invOp will execute. At this moment,
    context processor of this web service will read from each
    register link and form context file, then context file will be
    processed by this web service ->
    <bml:readreg reg="SA" service="Inventory"
     operation="invOp" port="$c_sa$"/>
    <bml:readreg channel="UP" service="Inventory"
     operation="invOp" port="$c_up$"/>
    <bml:readreg channel="NB" service="Inventory"
```

```
      operation="invOp" port="c_n b"/>
        <bml:execute operation="invOp"
        service="Inventory"/>
        <bml:writechannel channel="resu" service="
        Inventory" operation="invOp" port="o_s"/>
      <bml:readchannel channel="resu" service="Order"
       operation="ordResuOp" port="i_c"/>
      <!-executing a service operation ordResuOp ->
           <bml:execute operation="ordResuOp"
            service="Order"/>
      <!-writing to channel info->
      <bml:writechannel channel="info" service="Order"
       operation="ordResuOp" port="o_i"/>
      </bml:sequence>
    </bml:loop>
  </bml:composition>
</bml:process>
```

7.3 Execution and Analysis of a Ticket Booking System

When BML4WS document of the ticket booking system is exported, we can use BML4WS interpreter to interpret and execute it. We can also use related definitions and theorems presented in Section 4 to analyze and verify the properties of this system. For example, we assume an effective execution of a ticket booking system with an input $in = [x_1, x_2, x_3]$, x_1, x_2, and x_3 denote three XML-formatted ticket booking requests.

From the BML4WS description, we can extract composite behavior trace of the system using Algorithm 1. From composite behavior trace of the system, we can extract single behavior trace for every web service. If we only process a booking request, the behavior traces of Order service and Inventory service are as follows:

$$trace(Order) = ask.!quer.resu.(!con + !info)$$
$$trace(Inventory) = quer.SA.UP.NB.!resu.$$

From Theorem 2, we know that an effective execution implies a complete execution, and any two interacting web services in this system are behavior compatible. In this complete execution of the ticket booking system, three requests must be consumed entirely, and every behavior trace has a finite number of actions, as follows:

$$trace(Order) = ask.!quer.resu.(!con + !info).ask$$
$$.!quer.resu.(!con + !info).ask.!quer.resu.(!con + !info)$$
$$trace(Inventory) = quer.SA.UP.NB.!resu.quer.$$
$$SA.UP.NB.!resu.quer.SA.UP.NB.!resu.$$

Next, we verify whether two interacting web services are behavior compatible. We only need to consider writing/reading actions on internal FIFO channels between this two interacting web services. So, the trace of Order service and Inventory service becomes

$$trace(Order) = !quer.resu.!quer.resu.!quer.resu$$
$$trace(Inventory) = quer.!resuquer.!resuquer.!resu.$$

It is evident that $trace(Order)$ and $trace(Inventory)$ have opposite behavior trace. So, in the complete execution of ticket booking system, Order service and Inventory service are behavior compatible.

In fact, in an actual execution of this system, using related definition, it is easy for us to verify whether this execution is effective or complete, and whether two interacting web services are behavior compatible.

8 Conclusions and Future Works

In this paper, a novel process network approach was proposed to model dynamic behaviors of interacting context-aware web services, aiming to effectively process and take advantage of contexts and realize behavior adaptation of web services. We presented the interaction model of context-aware web services based on a formal model, CAPN. The CAPN was extended to context-aware web service network by adding a kind of sensor processes, which is used to catch contextual data from external environment. Through modeling the register link's behaviors, we present how a web service can respond to its context changes dynamically. Its explicit formal behavior semantic was provided to implement context awareness and behavior adaptation of context-aware web service. We designed an XML-formatted service behavior description language named BML4WS to describe behaviors and behavior adaptation of interacting context-aware web services. Our model was demonstrated in a practical case, i.e., ticket booking system. The results illustrated that the proposed approach is available and flexible.

Our future work is to make an attempt to implement context-aware web service network for more web service applications. As discussed above, context-aware web service network is based on the CAPN and is adequate to model service interactions in several aspects, including its parallelism and communication mechanism, compositional property, executable property, and context-awareness semantics. Other formal models, such as Petri net and process algebra, lack these advantages.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant Nos. 60873054, 60973013, 61073056, 61073134, and 51179020), the Fundamental Research Funds for the Central Universities (Grant No. 2011JC006), and the Dalian Science and Technology Fund (Grant No. 2010J21DW006).

References

[1] M. Keidl and A. Kemper, "Towards Context-Aware Adaptable Web Services," *Proc. 13th Int'l World Wide Web Conf.*, pp. 55-65, May 2004.

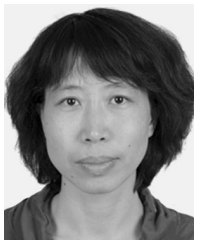
[2] A. Abraham, J. Chung, and S. Han, "Web Services: Recent Advances and Applications," *J. Digital Information Management,* vol. 4, no. 1, pp. 1-3, 2006.

[3] F. Gandon and N. Sadeh, "Semantic Web Technologies to Reconcile Privacy and Context Awareness," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 1, no. 3, pp. 241-260, 2004.

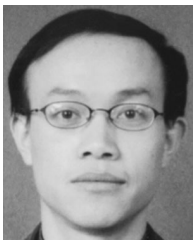
[4] H. Truong and S. Dustdar, "A Survey on Context-Aware Web Service Systems," *Int'l J. Web Information Systems,* vol. 5, no. 1, pp. 5-31, 2009.

[5] L. Li, D. Liu, and A. Bouguettaya, "Semantic Based Aspect-Oriented Programming for Context-Aware Web Service Composition," *Information Systems,* vol. 36, no. 3, pp. 551-564, 2011.

[6] R. Hervas and J. Bravo, "Towards the Ubiquitous Visualization: Adaptive User-Interfaces Based on the Semantic Web," *Interacting with Computers,* vol. 23, no. 1, pp. 40-56, 2011.
[7] R. Bhatti, E. Bertino, and A. Ghafoor, "A Trust-Based Context-Aware Access Control Model for Web-Services," *Distributed and Parallel Databases,* vol. 18, no. 1, pp. 83-105, 2005.
[8] M. Mrissa, C. Ghedira, D. Benslimane, Z. Maamar, F. Rosenberg, and S. Dustdar, "A Context-Based Mediation Approach to Compose Semantic Web Services," *ACM Trans. Internet Technology,* vol. 8, no. 1, article 4, 2007.
[9] J. Simoes and S. Wahle, "The Future of Services in Next Generation Networks," *IEEE Potentials,* vol. 30, no. 1, pp. 24-29, Jan./Feb. 2011.
[10] Z. Chen, K. Ma, A. Abraham, B. Yang, and R. Sun, "An Executable Business Model for Generic Web Applications," *Proc. Int'l Conf. Computer Information Systems and Industrial Management Applications,* pp. 573-577, 2010.
[11] T. Tiropanis, H. Davis, D. Millard, and M. Weal, "Semantic Technologies for Learning and Teaching in the Web 2.0 Era," *IEEE Intelligent Systems,* vol. 24, no. 6, pp. 49-53, Nov./Dec. 2009.
[12] L. Zhang, "Services Design and Optimization," *IEEE Trans. Services Computing,* vol. 2, no. 2, article 93, Apr.-June 2009.
[13] M. Brambilla, S. Ceri, F. Facca, I. Celino, D. Cerizza, and E. Valle, "Model-Driven Design and Development of Semantic Web Service Applications," *ACM Trans. Internet Technology,* vol. 8, no. 1, pp. 3-32, 2007.
[14] I. Elgedawy, Z. Tari, and J. Thom, "Correctness-Aware High-Level Functional Matching Approaches for Semantic Web Services," *ACM Trans. Web,* vol. 2, no. 2, article 12, 2008.
[15] K. Scott and R. Benlamri, "Context-Aware Services for Smart Learning Spaces," *IEEE Trans. Learning Technologies,* vol. 3, no. 3, pp. 214-227, July-Sept. 2010.
[16] S.-Y. Hwang, E.-P. Lim, C.-H. Lee, and C.-H. Chen, "Dynamic Web Service Selection for Reliable Web Service Composition," *IEEE Trans. Services Computing,* vol. 1, no. 2, pp. 104-116, Apr.-June 2008.
[17] A. Segev and E. Toch, "Context-Based Matching and Ranking of Web Services for Composition," *IEEE Trans. Services Computing,* vol. 2, no. 3, pp. 210-222, July-Sept. 2009.
[18] A. Staikopoulos, O. Cliffe, R. Popescu, J. Padget, and S. Clarke, "Template-Based Adaptation of Semantic Web Services with Model-Driven Engineering," *IEEE Trans. Services Computing,* vol. 3, no. 2, pp. 116-130, Apr.-June 2010.
[19] D. Vieira, C. Melo, A. Bezerra, Y. Ghamri-Doudane, and N. da Fonseca, "LatinCon01—A Content-Oriented Web Cache Policy under P2P Video Distribution Systems," *IEEE Latin Am. Trans.,* vol. 8, no. 4, pp. 349-357, Aug. 2010.
[20] G. Kahn, "The Semantics of a Simple Language for Parallel Programming," *Information Processing,* J.L. Rosenfeld, ed., pp. 471-475, North Holland, Aug. 1974.
[21] H. van Dijk, H. Sips, and E. Deprettere, "Context-Aware Process Networks," *Proc. IEEE Int'l Conf. Application-Specific Systems, Architectures, and Processors,* pp. 6-16, 2003.
[22] D. Zhovtobryukh, "Context-Aware Web Service Composition," PhD dissertation, Univ. of Jyvaskyla, 2006.
[23] D. Kulkarni and A. Tripathi, "A Framework for Programming Robust Context-Aware Applications," *IEEE Trans. Software Eng.,* vol. 36, no. 2, pp. 184-197, Mar./Apr. 2010.
[24] M. Sama, S. Elbaum, F. Raimondi, D. Rosenblum, and Z. Wang, "Context-Aware Adaptive Applications: Fault Patterns and Their Automated Identification," *IEEE Trans. Software Eng.,* vol. 36, no. 5, pp. 644-661, Sept./Oct. 2010.
[25] C. Jacob and S. Steglich, "Modeling Dynamic Service Behavior Using Context Functions," *Proc. First Int'l Conf. Networked Digital Technologies,* pp. 88-93, 2009.
[26] J. Zhou, E. Gilman, J. Palola, J. Riekki, M. Ylianttila, and J. Sun, "Context-Aware Pervasive Service Composition and Its Implementation," *Personal and Ubiquitous Computing,* vol. 15, no. 3, pp. 291-303, 2011.
[27] R. Hervás and J. Bravo, "COIVA: Context-Aware and Ontology-Powered Information Visualization Architecture," *Software: Practice and Experience,* vol. 41, no. 4, pp. 403-426, 2011.
[28] A.T. Manes, "Enabling Open, Interoperable, and Smart Web Services—The Need for Shared Context," *Proc. W3C Web Services Workshop,* http://www.w3.org/2001/03/WSWS-popa/paper29, 2001.
[29] A.K. Dey, "Understanding and Using Context," *Personal and Ubiquitous Computing,* vol. 5, no. 1, pp. 4-7, 2001.
[30] X. Li, Y. Fan, Q.Z. Sheng, Z. Maamar, and H. Zhu, "A Petri Net Approach to Analyzing Behavioral Compatibility and Similarity of Web Services," *IEEE Trans. Systems, Man and Cybernetics, Part A: Systems and Humans,* vol. 41, no. 3, pp. 510-521, May 2011.
[31] C. Yan, C. Jiang, and Q. Li, "The Composition and Analysis of Web Service Based on Petri Net," *Computer Science,* vol. 34, no. 2, pp. 100-103, 2007.
[32] J. Liao, H. Tan, and J. Liu, "Describing and Verifying Web Service Using Pi-Calculus," *Chinese J. Computers,* vol. 28, no. 4, pp. 635-642, 2005.
[33] X. Gu and Z. Lu, "A Formal Model for BPEL4WS Description of Web Service Composition," *Wuhan Univ. J. Natural Sciences,* vol. 11, no. 5, pp. 1311-1319, 2006.
[34] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "LTSA-WS: A Tool for Model-Based Verification of Web Service Compositions and Choreography," *Proc. 28th Int'l Conf. Software Eng.,* pp. 771-774, May 2006.
[35] T. Chaari and A. Celentano, "Design of Context-Aware Applications Based on Web Services," Technical Report RR-2004-033, INSA, Lyon, Sept. 2004.
[36] F.L.T. Chaari and A. Celentano, "Adaptation in Context-Aware Pervasive Information Systems: The SECAS Project," *Int'l J. Pervasive Computing and Comm.,* vol. 3, pp. 400-425, 2007.
[37] Y. Dou, G. Wu, J. Xu, and X. Zhou, "A Coarse-Grained Reconfigurable Computing Architecture with Loop Self-Pipelining," *Science in China Series F: Information Sciences,* vol. 52, no. 4, pp. 575-587, 2009.
[38] D. Webb and A. Wendelborn, "The PAGIS Grid Application Environment," *Proc. Int'l Conf. Computational Science,* pp. 692-693, 2003.
[39] E. De Kock, W. Smits, P. van der Wolf, J. Brunel, W. Kruijtzer, P. Lieverse, K. Vissers, and G. Essink, "YAPI: Application Modeling for Signal Processing Systems," *Proc. 37th Ann. Design Automation Conf.,* pp. 402-405, 2000.
[40] M. Geilen and T. Basten, "Requirements on the Execution of Kahn Process Networks," *Proc. 12th European Conf. Programming,* pp. 319-334, 2003.
[41] R. Milner, *Communication and Concurrency.* Prentice-Hall, 1989.
[42] R. Cleaveland, T. Li, and S. Sims, *The Concurrency Workbench of the New Century.* Stony Brook Univ., 2000.
[43] T. Parks, "Bounded Scheduling of Process Networks," PhD dissertation, Electrical Eng. and Computer Sciences Dept., Univ. of California, Berkeley, 1995.
[44] J. Buck and E. Lee, "Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model," *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing,* pp. 429-432, 1993.



Xiuguo Zhang received the PhD degree in computer science from Dalian Maritime University, China. She is an associate professor at the School of Information Science and Technology, Dalian Maritime University. Her research interests include web services and software engineering. She has authored more than 40 refereed journal/conference papers.



Hongbo Liu received the PhD degree in computer science from the Dalian University of Technology. He is a professor at the School of Information Science and Technology, Dalian Maritime University, China, with an affiliate appointment in the Computer Science and Biomedical Engineering Division at the Dalian University of Technology. His research interests include system modeling and optimization involving soft computing, nature-inspired computing, swarm intelligence, multiagent systems, fuzzy inference systems, rough set, and probabilistic model. His application areas include neuroinformatics, cognitive computing, computational grids, web services, machine learning, and data mining. He has authored more than 60 refereed journal/conference papers and book chapters, and some of the works have also won best paper awards at international conferences and received several citations. Some of the articles are available in ScienceDirect's Top 25 Hottest Articles. He co-organized the 2010 China-Japan Joint Conference on Computational Geometry, Graphs and Applications (CGGA '10) in Dalian, China. In 2011, he was the general chair of the Third IEEE International Conference on Soft Computing and Pattern Recognition (SoCPaR '11) in Dalian, China.

Ajith Abraham received the MS degree from Nanyang Technological University, Singapore, and the PhD degree in computer science from Monash University, Melbourne, Australia. He is currently the director of the Machine Intelligence Research Labs, Scientific Network for Innovation and Research Excellence, which has members from more than 85 countries. He has worldwide academic experience with formal appointments in at Monash University, Australia; Oklahoma State University; Chung-Ang University, Seoul, Korea; Jinan University, China; Rovira i Virgili University, Tarragona, Spain; Dalian Maritime University, China; Yonsei University, Seoul, Korea; the Open University of Catalonia, Barcelona, Spain; the National Institute of Applied Sciences, Lyon, France; and the Norwegian University of Science and Technology, Trondheim. He servers or has served on the editorial boards of more than 50 international journals and has also guest edited 40 special issues on various topics. He has published more than 800 publications, and some of the works have also won best paper awards at international conferences. His research and development experience includes more than 20 years in the industry and academia. He works in a multidisciplinary environment involving machine intelligence, network security, various aspects of networks, e-commerce, web intelligence, web services, computational grids, data mining, and their applications to various real-world problems. He has given more than 50 plenary lectures and conference tutorials in these areas. He is the chair of the IEEE Systems, Man, and Cybernetics Society Technical Committee on Soft Computing and a distinguished lecturer of the IEEE Computer Society, representing Europe. He is a senior member of the IEEE, the IEEE Computer Society, the Institution of Engineering and Technology (United Kingdom), and the Institution of Engineers Australia. He is the founder of several IEEE-sponsored annual conferences, which are now annual events, including Hybrid Intelligent Systems, Intelligent Systems Design and Applications, Information Assurance and Security, Next Generation Web Services Practices, Computational Aspects of Social Networks, Soft Computing and Pattern Recognition, and Nature and Biologically Inspired Computing.