

Running Smart Process Based on Goals

Zongmin Shang¹, Haiyang Wang¹, Liqiang Wang²

¹*School of Computer Science and Technology, Shandong University, Jinan, P.R. China
Shangzongmin@mail.sdu.edu.cn, why@sdu.edu.cn*

²*Department of Computer Science, University of Wyoming, Laramie, WY 82071, USA
wang@cs.uwyo.edu*

Abstract

When web services are used to coalesce around the distributed applications, one prominent solution to manage and coordinate web services is the use of process management. Many researches have been done to deal with automatic discovery and composition issues of web services. However, the problem of how to run a process that is composed of distributed services is seldom considered. In this paper, based on our previous work, the Smart Process-based Application Model (SPM), we formalize Smart Process (SP) using algebra CCS, and discuss the benefits. In particular, we propose goal-based algorithms to run Smart Process.

Keywords: Smart Process-based Application Model (SPM); Smart Process (SP); process algebra; CCS; distributed application; Web services

1. Introduction

In recent years, distributed applications over a network (Intranet or Internet) have obtained wide popularity. Many integrated web services paradigms for distributed applications have been developed. Web services can be used for implementing business collaborations across and within corporation boundaries based on XML standards. Web services can be considered services available over a network. Each service has an interface that describes the interaction capabilities and is accessible through standard protocols. They aimed at wild reusability and they are typically designed to interact with other services in order to build distributed applications.

A temporary and dynamic interaction occurs when one service contacts another one in response to an instantaneous need. For example, a travel agent books vacation packages (plane/train/bus tickets, hotels, car rental, excursions, etc.) for customers. This, of course, assumes the use of advanced search tools to enable the automatic discovery of a service matching the need. When web services are used to bind the distributed applications, one prominent solution to manage and coordinate web services is the use of process

management [1]. Web-service-based applications can be considered conglomerates of independent, autonomous services developed by independent parties. According to the current needs, such components are not integrated at design time; instead, they are integrated dynamically at run-time. From a software engineering viewpoint, the construction of new services by the static or dynamic composition of existing services displays new perspectives, which may significantly impact the way how distributed applications will be developed in the future. However, they also raise a number of challenges [2]. For example, web-service-based applications usually have a longer duration and include asynchronous messages passing between partners. When using process management, service binding evolves autonomously, services are coupled highly loosely, and system boundaries are blurred. Due to the message-passing nature of web services interaction, many subtle errors can occur, such as message not received, deadlocks, and incompatible behaviors. These problems are well-known and recurrent in distributed applications, but become even more critical in the open-end world of web services (which is ruled by the longer-term vision of “services used by services” more than by human users), where this interaction should ideally be as transparent and automated as possible.

We proposed a new application model of business processes called Smart Process-based Application Model (SPM) [3]. SPM can analyze users' preferences, compose relevant service resources, and generate a personalized application process to better meet users' individual requirements. A personalized application process is called a Smart Process (SP), which has a short duration and terminates right after execution. SPM is suitable for distributed applications composed of web services, such as the travel agent booking mentioned above. To apply SPM, customers' requirements should be acquired first. After discovering and composing web services, a smart process can be generated and run automatically according to the interactions between system and customers.

Two main problems should be handled before using SPM. One is how to generate a SP automatically according to customers' requirements. The other is how to run the SP correctly when it involves different

heterogeneous services, especially when exceptions occur. The former problem has been discussed in our previous works [3,4], and an approach based on process semantic databases has been proposed to study customers' requirements and generate SP automatically. This paper addresses the later problem. Our approach is to use process algebra Calculus of Communicating Systems (CCS) [5] to formally describe the running behaviors of SP. Process algebras make it easier to formally specify the message exchange between web services and the logic on the specified systems.

This paper is organized as follows. Section 2 introduces the capability description of Smart Process. In section 3, we introduce the process algebra CCS, how to model SP using it, and the algorithm for running SP driven by goals. Section 4 talks related work, section 5 gives the conclusion and future work.

2. Capability Description of Smart Process (SP)

A SP is composed of activities and goals: Activities are the services selected by discovering and composing web services according to customer's requirements when SP is generated. A goal indicates the target to be achieved by the current activity and the conditions to be satisfied by the next activity. Namely, goals denote the end of previous activities and the start of posterior activities.

An example of SP is shown in Fig. 1. In the travel agent booking system, a SP is automatically generated by studying customer's requirements. This SP shows that tourism is a sequence of goal start, activity A_1 (by airplane), goal G_1 , activity A_2 (accommodation), goal G_2 , activity A_3 (sight-seeing), goal G_3 , activity A_4 (by airplane), finally goal end.

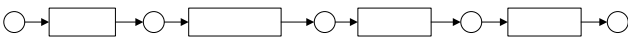


Fig.1. An example of SP

2.1. Goal

When a Smart Process is running, several types of messages will be transmitted among activities or between activity and environment. Activities need input messages in order to execute, and may generate output messages after execution. By abstracting these messages, we give the definition of goal used in the description framework of SP. Goal is defined by a set of attributes, each of which features some aspect of the goals.

Definition 2.1 (Goal) Goal is defined as a tuple,

$$Goal = \langle Attribute, Value \rangle$$

In which, $Attribute = \{attr_1, attr_2, \dots, attr_n\}$ is a finite set of attributes; $Value = \{val_1, val_2, \dots, val_m\}$ is a finite set of attribute values;

The attributes of a goal are classified into static ones and dynamic ones. A static attribute is irrespective of time and its value remains the same throughout the lifecycle. A dynamic attribute may change its value when time advances or exterior conditions change.

Definition 2.2 (Static Attribute, StaAttr) An attribute of a goal is static if its value remains the same during the whole lifetime of the SP.

Definition 2.3 (Dynamic Attribute, DynAttr) An attribute of a goal is dynamic if it takes different values at different stages of the lifecycle.

We define $G[DA] = \{dAttr_1, dAttr_2, \dots, dAttr_n\}$ as the dynamic attributes of goal G , and $G[SA] = \{sAttr_1, sAttr_2, \dots, sAttr_n\}$ as the static attributes of goal G .

Example 2.1 This example lists the description about the goal G_1 in Fig.1.

```

<?xml version="1.0" encoding="UTF-8">
<StaticAttr>
  <destination>airport K, city A
</destination>
  <time>Sat,13:30</time>
  <deskclerk>Bob</deskclerk>
</StaticAttr>
<DynamicAttr>
  <weather></weather>
  <health></health>
</DynamicAttr>
  
```

The static attributes include destination, time and deskclerk. The values of destination and time come from the output messages that are generated when activity A_1 (Flight K)'s execution completes. The value of deskclerk describes the input messages that are needed when activity A_2 (Accommodate in Hotel B) starts to execute. The dynamic attributes include weather and health, whose values are given dynamically during the running of SP.

2.2. Interaction

When a SP is running, the application system can also be viewed as a reactive system. All the SP systems shown in the above example maintain a continuous interaction with their environment. Both the SP and its environment work like parallel processes that communicate one with the other. In addition, unlike with "standard" computing systems, such as a control program, non-termination is a desirable feature of the SP system.

There are two kinds of interactions between activities and environment: interactions among activities, and interactions between activities and goals. The former have autonomous communication, i.e., an activity exchanges messages with other activities in order to fulfill its function. The latter is different, where activities execute in order to achieve goals.

In a SP, interactions between activities and environment contain four required parameters and two optional parameters as in the definition below [6]:

Definition 2.4 (Interaction) Interaction is defined as follows:

$InterAction := \langle content, host, ontoLogy, complement, [reply-to], [reply-with] \rangle$

where, Content expresses interaction content, and has three formats: $content := null | @msg | \#msg$, null denotes that the content of interaction is null, @msg denotes that the msg is just a name, and #msg denotes that the msg is the concrete content of interaction.

Host denotes the activity that executes the interaction behavior. Ontology describes the ontology to which the interaction behavior concept belongs. Match of concepts is meaningful only under the same ontology restriction. Complement is used to express the behaviors that communicate with each other.

The last two parameters are optional: reply-to means that the describing behavior is used for some previous behaviors; reply-with means that the describing behavior is a response to some behavior.

The top-level interaction behavior contains two types of performatives [6]: one is INPUT, which indicates activities that acquire information from the environment; the other is OUTPUT, which indicates activities that send out information to the environment. The performatives of INPUT type include: Receive, Get, EDask, Collect, and Subscribe. The performatives of OUTPUT type in the top-level interaction ontology include: Send, Ask, Tell, and Provide.

2.3. Capability Description Framework for Smart Process

We construct a capability description framework for Smart Process—SPCapDes based on goals and interactions. The capability of a Smart Process can be expressed using goals, activities, and interactions involved. SPCapDes is defined as follows:

$SPCapDes := \langle Goal, Activity, Interaction \rangle$

In which Goal is the set of goals before and after the execution of every activities in the Smart Process; Activity is the set of activities of services requested by the customer; and Interaction is the set of interactions happening during the running of Smart Process.

3. Modeling and Running Goal-based Smart Process

Section 2 introduces SPMs and the key aspects of their behaviors. Now we consider how to design an abstract model for a system. In particular, such a model should describe how parallel processes compute independently and interact with one another. It should also be able to describe well-known concurrency phenomena (e.g., deadlock, livelock, starvation and so on). Finally, the model should support the description of

non-determinism, because some dependent issues will occur when abstracting from implementation, e.g., scheduling policies.

We use CCS and axiomatic operational semantics [7] to describe Smart Process and their interaction behaviors. Thus, the behavior properties of the Smart Process can be formally verified. In this section, we first introduce CCS, then propose to use CCS process expressions to model Smart Process, finally give the algorithm of running goal-driven Smart Process.

3.1. Introduction of Process Algebra CCS

The most basic process constructor in CCS is action prefixing. Defining processes in CCS requires a set of action names representing the messages used in the system, for instance {a,b,c...}, or {request, confirm, cancel, notAvailable...}. For example, two processes built using 0 (termination) and action prefixing: a.0 and b.a.0. Intuitively, the former denotes a process termination after executing the action a, and the latter is the one that needs to execute the action b before it can execute like the former. The basic actions in CCS are to receive a message (denoted by ?a or ?request) or to emit a message (denoted by !a or !request). CCS processes allow recursive definitions. For instance, the collection P of CCS expressions is given by the following grammar:

$$P ::= 0 | a.P | P+Q | (P_1 | \dots | P_n) | P \setminus sm$$

$$a = !a | ?a | \tau$$

A process that is terminated is written as 0. A process executes a sequence a.P, where a is an action and P is a process. A process can perform a nondeterministic choice P+Q, which means executes P or Q but not both. A process can be a parallel composition of sub-processes: P₁ | ... | P_n. A process may have restriction, which denoted by P \ sm. P is a process and sm is a set of names, imposing that an emission of m \in sm by one sub-process of P can occur only if another sub-process does a reception of the same message name (synchronization).

Example 3.1 An example of CCS process:

$$S = ?a.b.0 + ?a.?b.0 | !b.!c.0$$

Its derivation is below when offering an action (the process will select the later choice).

$$S \xrightarrow{a} ?b.0 | !b.!c.0 \xrightarrow{\tau} !c.0 \xrightarrow{c} 0$$

According to the expansion theorem and the constraint that processes are guardedly well-defined, each process is equivalent to a sum of guards. Put it another way, a process can be defined in this form [7]:

$$P = \sum a_i.P_i, \quad a_i \in \{!a, ?a, \tau\}$$

3.2. Modeling Smart Process with CCS

Consider a simple travel Smart Process shown in Fig.1. Firstly, the SP receives a request message and

starts to execute the activity A_1 (by Flight K). After normal completion of activity A_1 , the attribute values of goal G_1 are available. Secondly, the activity A_2 (accommodate in hotel B) executes when the messages requested from goal G_1 are satisfied. Then the attribute values of goal G_2 are available after normal completion of activity A_2 . Thirdly, the activity A_3 (Sightseeing) and activity A_4 (by Flight M) execute one by one. Finally, the SP terminates with 0. This Smart Process is represented as follows:

$SP = ?byFlightK. ?accommodateInHotelB. ?sightseeByRouteY.$

$?byFlightM.0$

Its derivation is shown as follows when activities are executing (without exceptions happening):

$SP \xrightarrow{?byFlightK} SP_1 \xrightarrow{?accommodateInHotelB} SP_2 \xrightarrow{?sightseeByRouteY} SP_3$
 $\xrightarrow{?byFlightM} 0$

From above we know that goals consist of two types of messages while a SP is running: reception and emission messages from executing activities. The attributes of these messages are either static or dynamic, and have been defined in definition 2.1-2.3. Thus, we can make the complemented definition of Goals like below:

$$G_i = \left\{ \sum_n Input_{A_i}(P_{i+1}) \cup \sum_n Output_{A_i}(P_i) \right\} \quad (1)$$

where n is the total number of activities that the SP involves)

So, we can define SP as follows:

$$SP = \{start, end, \sum_{i=1}^n G_i, \sum_{j=1}^m A_j\} \quad (2)$$

In which start and end are two special goals in SP. Start denotes the initial goal of SP and end denotes the final goal of SP. n is the number of goals of SP, m is the number of all activities of SP.

We can use (2) to model a SP in CCS, Fig.2 shows an example of modeling the SP of Fig.1 in CCS.

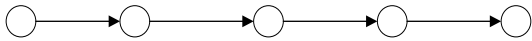


Fig.2. An example of modeling SP in CCS

3.3. Running Goal-Driven Smart Process

A running of a Smart Process is to achieve all goals in the SP, namely, to get or set the values of goals in SP. There are four ways to get the values of all goals when running SP, which we show with the following derivations.

Definition 3.1 (Activity Derivation, ACD) G_1 derives from Activity A_1 if the value of G_1 is got due to the execution of activity A_1 . Writes as $G_1 \xrightarrow{ACD} A_1$;

Definition 3.2 (Goal Derivation, GAD) G_2 derives from goal G_1 if the value of G_2 is got due to the value of goal G_1 . Writes as $G_2 \xrightarrow{GAD} G_1$;

Definition 3.3 (Environment Derivation, EVD) G_1 derives from environment Env if the value of G_1 is got from environment Env . Writes as $G_1 \xrightarrow{EVD} Env$ (e.g., in Fig.2, $Env = \{weather, location, \dots\}$).

Definition 3.4 (Customer Derivation, CUD) G_1 derives from customer Cus if the value of G_1 is got from customer Cus . Writes as $G_1 \xrightarrow{CUD} Cus$ (e.g., in Fig.2, $Cus = \{nationality, age, gender, \dots\}$).

A goal denotes a join-point in SP. The values of goals' attributes denote whether the condition of activity execution is satisfied or not. We define some dependencies between goals as follows:

Definition 3.5 (Start Equivalence, STE) A Goal G_1 is Start equivalence with a goal G_2 (or G_2 is Start equivalence with G_1) if:

$$\sum_{i=1}^n Output_{A_i}(P_k) \in G_1[Attr] \quad \wedge \quad \sum_{i=1}^n Output_{A_i}(P_k) \in G_2[Attr]$$

Shown as: $G_1 \stackrel{STE}{=} G_2$

In which P_k is the evolving process in which the values of goal G_1 will be changed, n is the number of all activities executed in P_k .

Definition 3.6 (End Equivalence, ENE) A goal G_1 is End equivalence with a goal G_2 (or G_2 is End equivalence with G_1) if:

$$\sum_{i=1}^n Input_{A_i}(P_k) \setminus (G_1[Attr] + G_2[Attr]) = \Phi,$$

$$\left(\sum_{i=1}^n Input_{A_i}(P_k) \setminus G_1[Attr] \neq \Phi \quad \wedge \right.$$

$$\left. \sum_{i=1}^n Input_{A_i}(P_k) \setminus G_2[Attr] \neq \Phi \right)$$

Shown as: $G_1 \stackrel{ENE}{=} G_2$

In which, P_k is the evolving process that will execute according to the values of goal G_1 and goal G_2 , n is the number of all activities executed in P_k .

STE and ENE define relationships between goals. For two STE goals, their values are achieved by the same activity. For two ENE goals, their values are needed by the same activity.

When a Smart Process is running, it is necessary to correctly handle exceptions. We propose a new algorithm for this. Before introducing the algorithm, we briefly review the five types of exceptions proposed in our previous work [9].

Definition 3.7 (Exception in SP, ESP) There are five types of exception in SP:

$ESP = \{SU, DE, ET, RV, AC\}$,

Service Unavailability (SU), Deadline Expiry (DE), External Trigger (ET), Rationality Violation (RV).

Algorithm 3.1 Adjust goal-driven SP when an exception SU takes place.

Input: $SP = \{start, end, \sum_{i=1}^n G_i, \sum_{j=1}^m A_j\}$, $\{su\}$, A_k

Output: $SP' = \{start, end, \sum_{i=1}^{n'} G_i, \sum_{j=1}^{m'} A_j\}$, state = {interrupt, continue}.

Start:
 A_k Send({su}, $G = \{G_i | Output_{A_k}(P_i) \in G_i\}$)
 A_k Ask(msg)
 A_k Receive($G', repA_k = \{A_{k_1}, A_{k_2}, \dots, A_{k_m}\}$)
 // the goals of SP don't change, and new activities have been received from environment.
 if ($G' = G$) and ($repA_k \neq \emptyset$) then
 if exist $A_{k_j} \in repA_k$ and
 ($Input_{A_k}(P_i) = Input_{A_{k_j}}(P_j)$) and ($Output_{A_k}(P_i) = Output_{A_{k_j}}(P_j)$)
 then
 $SP' = \{start, end, \sum_{i=1}^n G_i, (\sum_{j=1}^m A_j \setminus A_k) \cup A_{k_j}\}$
 state = {continue}
 return (SP' , state)
 else $SP' = SP$ State = {interrupt}
 return (SP' , state)
 endif
 endif
 // some goals G_i will be gotten rid of from G
 if ($G' = G \setminus \{G_i\}$) then
 if $(\forall (G_i, G_m) | G_i \Rightarrow G_i \wedge G_m \Rightarrow A_k,$
 $\exists A_{k_j} | G_i \Rightarrow A_{k_j} \wedge G_m \Rightarrow A_{k_j} (A_{k_j} \in repA_k))$ then
 $SP' = \{start, end, G', (\sum_{j=1}^m A_j \setminus A_k) \cup A_{k_j}\}$
 state = {continue}
 return (SP' , state)
 else
 $SP' = SP$ state = {interrupt}
 return (SP' , state)
 end if
 endif
 // some goals G_i in G will be replaced with G'_i
 if ($G' = G \setminus \{G_i\} \cup \{G'_i\}$) then
 if $(\forall G_i | G_i \Rightarrow G'_i$ where $(G_i \Rightarrow G'_i)$) and
 $\exists (A_{k_j} \in repA_k) | \forall (G_i = G'_i) \wedge \forall (G_s = G'_s) \wedge \forall (G_t \Rightarrow A_{k_j})$
 where $(G_i = G'_i \wedge G_s = G'_s \wedge G_t \Rightarrow A_k)$
 then $SP' = \{start, end, G', (\sum_{j=1}^m A_j \setminus A_k) \cup A_{k_j}\}$
 state = {continue}
 return (SP' , state)
 else $SP' = SP$ state = {interrupt}
 return (SP' , state)
 endif
 endif

// the goal G_i in G will be divided into serial goals
 G_{i1}, G_{i2}
 if ($G' = G \setminus \{G_i\} \cup \{G_{i1}, G_{i2}\}$)
 if $(\forall (G_i \Rightarrow G_{i1})$ where $(G_i \Rightarrow G_{i1})$ and
 $(\exists (A_{k_j} \in repA_k) | \forall (G_i = G_{i1})$ where $(G_i = G_{i1})$) and
 $(\forall (G_i = G_{i1})$ where $(G_i = G_{i1} \wedge Act \in \sum_{j=1}^m A_j))$ then
 $SP' = \{start, end, G', (\sum_{j=1}^m A_j \setminus A_k) \cup A_{k_j}\}$
 state = {continue}
 return (SP' , state)
 else $SP' = SP$ state = {interrupt}
 return (SP' , state)
 endif
 endif
 // the goal G_i in G will be divided into parallel goals
 $G_{i1} | G_{i2}$
 if ($G' = G \setminus \{G_i\} \cup \{G_{i1} | G_{i2}\}$)
 if $(\forall (G_i \Rightarrow G_{i1} \wedge G_i \Rightarrow G_{i2})$ where $(G_i \Rightarrow G_{i1})$ and
 $(\exists (A_{k_j} \in repA_k) | \forall (G_i = G_{i1} \wedge G_i = G_{i2})$ where $(G_i = G_{i1})$) and
 $(\forall (G_i = G_{i1} \wedge G_i = G_{i2})$ where $(G_i = G_{i1} \wedge Act \in \sum_{j=1}^m A_j))$
 then $SP' = \{start, end, G', (\sum_{j=1}^m A_j \setminus A_k) \cup A_{k_j}\}$
 state = {continue}
 return (SP' , state)
 else $SP' = SP$ state = {interrupt}
 return (SP' , state)
 endif
 endif \square
 Similarly, like algorithm 3.1, we can give algorithms when other exceptions (DE, ET, RV, AC) take place. The other algorithms should be intuitive to the user based on algorithm 3.1.
Algorithm 3.2 Running SP driven by goals
Input: a SP $\{start, end, \sum_{i=1}^n G_i, \sum_{j=1}^m A_j\}$
Output: {finish, interrupt}
Start:
 Output= Φ ;
 While Output \neq {finish} or Output \neq {interrupt} do
 Run SP.Act; // $Act \in \sum_{j=1}^m A_j$
 Select algorithm 3.1 when exception EU takes place, and select the relevant algorithms when other exceptions (DE, ET, RV, AC) take place.
 If state={interrupt} then Output={interrupt}
 else If SP.end then Output={finish}
 endif

end while
return Output □

4. Related Work

It has indeed already been argued by other authors [10] that web services and their interaction are best described using process description languages. Many previous works utilize process algebra to describe and analyze the composition of web services[2,7,8]. There are a large number of proposals in the literature describing and analyzing compatibility of web services using CCS. In [2], Salaun et al. uses CCS to describe and reason web services, and demonstrates a case study of efficient conformance verification between the design requirements and the selected web services. In [7], Liu et al. uses CCS to describe dynamic behaviors of web services and discuss its compatibility at different levels. Many other works[11,12,13] also propose to compose of web services using process algebra. However, no previous work deals with how to run the composed web services.

In [8], Brogi et al. present formalization of web services choreography proposal WSCI, and provide reasoning mechanism and tools to check compatibility of web services. External and internal choices are differentiated when modeling control flows based on the composition of web services. When web services are not compatible, specification of adaptors will be generated to enable communication of incompatible web services. While different levels of compatibility and substitutability are not discussed, Hou et al. give a web services capability description framework based on the environment ontology in [6]. In the same paper, Hou et al. proposes the formalization of web services interaction with Pi calculus and depicts web services capability in two aspects: the operable environment and the changing environment resulting from behaviors of web services. The method to describe the capability of web services based on ontology in [6] is the same as the capability description of Smart Process in this paper.

In some cases, web services may be operated in isolated form, this is the reason that many traditional approaches use workflow technology to integrate web services. The Workflow Management Systems (WFMS) have become a fundamental building block in system organization and have been advanced significantly in recent years. Adaptive workflow systems [14] allow dynamic adaptation to the changing environment as they provide mechanisms to seamlessly integrate exception handling into workflow descriptions. These approaches lack practical aspects, such as the participation of autonomous, heterogeneous legacy systems, and the increased interactions among business activities. Some systems employ event-based model [15] to support exceptions and asynchronous behaviors during workflow execution by enhancing database technology

and extending transaction management. The limitation of these approaches is that they can not be applied to complex or unstructured process environments.

In [16], Russell et al. investigates what issues may lead to exceptions during workflow execution, as well as the various description ways. In that paper, a classification framework is also proposed to handle workflow exceptions in form of patterns. A Collaborative Framework for Exception Handling in Business Process Execution is proposed in [9], where several responding strategies to handle exceptions during the execution of business processes are also described. In this paper, we mainly focus on exception handling during the execution of a SP bound with loose web services. Unlike traditional WFMS, exception handling in SP requires much interaction between activities and the environment.

5. Conclusion and future work

Web services are an emerging and promising area involving important technological challenges, such as how to describe web services correctly, compose them adequately and/or automatically, and how discover suitable web services to work out a given problem. In this paper, based on the application model SPM, we propose a new method using CCS to model a SP and design new algorithms to execute SP.

The goal-driven semantic description of SP formalizes activity's behaviors to ensure the correctness of the SPM. Our approach to model SP has the following distinct advantages in comparison with the previous work. Firstly, a top level goal description is constructed to capture the semantics of the SP capability, and a top level interaction description is given to denote the interaction between activities and the environment, i.e., changes of goals caused by interaction behaviors may affect the execution of SP. Secondly, new algorithms are designed based on the formal SP model to run SP driven by goals. Finally, CCS process expressions are used to characterize the behaviors of SP. Five types of exceptions are identified to run SP automatically and accurately.

For the future work, we plan to validate the compatibility and substitutability of the executions of SP using bisimulation, extend the verification and validation of SPM, and establish compensating activities to recover those exceptional activities.

Acknowledgement

This work is supported by (NSFC National Natural Science Foundation of China) under Grant No. 60673130.

References

- [1] D.S. Zhang, "Web services composition for process management in E-business". *Journal of Computer Information Systems*, 2004, 45(2), 83-91.
- [2] G.Salaun, L.Bordeaux, M.Schaerf, "Describing and Reasoning on Web Services using process algebra", *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, 2004, pp. 43-51.
- [3] Z. Shang, H. Wang, "Acquiring users' requirements based on process semantic database under smart process application model", *Journal on Communications (China)*, 2006, 27(11), 73-77.
- [4] Z. Shang, L. Cui, H. Wang, "Automatic generation of process based on process semantic database with smart process application model". *Computer Science (China)*, 2006, 33(11B), 428-431.
- [5] R. Milner, "Communication and Concurrency", Prentice-Hall, 1989.
- [6] L. Hou, Z. Jin, B. Wu, "Modeling and verifying Web services driven by requirements: An ontology-based approach", *Science in China Series F: Information Sciences*, 2006, 49(6), 792-820.
- [7] F. Liu, L. Zhang, Y. Shi, L. Lin, B. Shi, "Formal Analysis of Compatibility of web service via ccs", *International Conference on Next Generation Web Services Practices (NWeSP'05)*, 2005, pp. 143-148.
- [8] A. Brogi, C. Canal, E. Pimentel, A. Vallecillo, "Formalizing Web Service Choreographies", *Electronic Notes in Theoretical Computer Science*, 2004, 105(12), 73-94.
- [9] Z. Shang, L. Cui, H. Wang, "A Collaborative Framework for Exception Handling in Business Process Execution", *11th International Conference on Computer Supported Cooperative Work in Design(CSCWD 2007)*, Melbourne, Australia, 2007, pp. 914-919.
- [10] L. G. Meredith, S. Bjorg, "Contracts and Types", *Communications of the ACM*, 2003, 46(10), 41-47.
- [11] J. Cámara, C. Canal, J. Cubo, A. Vallecillo, "Formalizing WSBPEL Business Processes Using Process Algebra", *Electronic Notes in Theoretical Computer Science*, 2006, 154 (1), 159-173.
- [12] A. Ferrara, "Web Services: A Process Algebra Approach", *Proceedings of the 2nd International Conference on Service Oriented Computing*, 2004, pp. 242-251.
- [13] J. Liao, H. Tan, J. Liu, "Describing and Verifying Web Service Using Pi-Calculus", *Chinese Journal of Computers*, 2005, 28 (4), 635-644.
- [14] S. Rinderle, M. Reichert, P. Dadam, "Flexible support of team processes by adaptive workflow systems", *Distributed and Parallel Databases*, 2004, 16 (1), 91-116.
- [15] A. Basu, A. Kumar, "Research commentary: Workflow management issues in e-business", *Information Systems Research*, 2002, 13 (1), 1-14.
- [16] N. Russell, Wil.M.P Van der Aalst, Arthur H. M. ter Hofstede, "Workflow exception patterns", *CAiSE 2006*, Springer-Verlag Berlin, 2006, pp. 288-302.