14

Context-aware Service Composition and Change-over Using BPEL Engine and Semantic Web

Yoji Yamato, Yuusuke Nakano and Hiroshi Sunaga
NTT Network Service Systems Laboratories, NTT Corporation
Japan

1. Introduction

With the advancement of IT technology, ubiquitous computing environments (Weiser, 1991) are rapidly becoming a reality where PCs and various other devices are connected to networks. Ubiquitous computing environments are expected to offer context-aware services (Schilit et al., 1994) and customization services. Users' needs change dynamically according to the user context such as location or time. The idea of dynamically composing appropriate service components in the network on the basis of the user context is a promising approach (e.g. (Minami et al., 2001) (Gribble et al., 2001)) to the conventional method of providing services, where service providers prepare services completely in advance.
Our study is on one of the service-composition technologies, and our approach uses dynamic interface resolution using semantic web techniques. We have already examined the methods through prototype implementation (Yamato et al., 2006) (Yamato & Sunaga, 2007). In this paper, we propose a new framework of context-aware service composition and change-over that consists of featured functions of our prototype service composition technology and commercial BPEL (Web Services Business Process Execution Language) (Jordan et al., 2007) engines to achieve a service composition system having commercial level quality at a low cost. We also consider service change-over, which is a problem in using the BPEL engine. We report the implementation of the proposed method, evaluation of its feasibility, multi-vendor compatibility, and processing performance.
This paper is comprised of the following sections. In section 2, we explain BPEL technology and our previous study of service composition technology. Section 3 shows the idea of applying the BPEL product for our service composition execution. In section 4, we implemented the proposed idea and evaluate the effectiveness. Section 5 shows some sample application and section 6 introduces related works. Section 7 concludes this paper.

2. BPEL and our service composition technology

BPEL, an existing service coordinating technology in the B-to-B area, is attracting attention. However, BPEL coordinates multiple web services for the purpose of semipermanent system integration. Therefore, BPEL is not suitable for binding unknown web services according to the user context. BPEL requires a rigid interface description such as the port type names and operation names of web services, so it can only be applied to pre-known web services whose port types and operations exactly match. In other words, BPEL is not flexible in terms of context awareness or user customization.
To solve the problems of BPEL, we use a service template (ST) that describes required service elements (SE) abstractly using semantic metadata, instead of a service flow that describes the rigid interface of individual web services. That makes finding multiple SEs with different interfaces but semantically equivalent functions possible, and the system can select an appropriate SE from candidate SEs according to the user context.
Here, SE is a service component where semantic metadata is assigned to web services or UPnP using OWL-S (Web Ontology Language for Services) (Martin et al., 2004) which is a description of semantic web service technology (e.g., (Paolucci & Sycara., 2004) (Sycara et al., 2003)). OWL-S consists of three parts: Profile, Process model, and Grounding. The Profile has information about properties that describe what capability the service provides. The Process model describes service behavior such as atomic process and its inputs, outputs, preconditions, and effects. The Grounding describes the mapping between abstract processes of OWL-S and actual operations such as WSDL or UPnP docs.
ST is a service scenario described by XML, and the ST grammar is similar to that of BPEL. The control tags of ST are the same as the ones defined by BPEL (e.g., invoke and while, for example). The difference between ST and BPEL is that in ST, SE is specified abstractly using semantic metadata. More precisely, the port type of BPEL is described using a category (a unit of categorization for equivalent SE function), and the operation of BPEL is described using an atomic process of OWL-S.
The flow of service composition is as follows. A user inputs the ST of the desired service into a service composition engine and the engine searches for candidate SEs from the SE-DB using semantic metadata described in the ST. Next, an SE appropriate for the user context is selected from candidate SEs. For appropriate SE selection, a score is marked for each SE by matching 3 elements: a user policy that designates which SE should take precedence, a user context that indicates the user situation, and an SE profile that designates the property of SE. The SE with high scores is automatically selected. Once the SE to be used is selected, the service composition engine converts semantic metadata to the actual interface of SE (WSDL or UPnP) and invokes SE (see, Fig. 1) using the OWL-S Grounding of the selected SE. Please refer to papers (Yamato et al., 2006) (Yamato & Sunaga, 2007) for details.

The following are three key features of our service composition technology:
(1) Interface resolution function
Using the semantic description of ST and OWL relationship, many SEs with a different interface but an equivalent function can be used (e.g., the print function of a printer and a fax machine can be used by the same ST).
(2) Context-aware service selection function

Matching user context, user policy, and OWL-S Profile, a score is assigned to each candidate SE, and using this score, an appropriate SE can be selected automatically (e.g., a printer that is nearest to a user is selected automatically).
(3) Service change-over function
During service execution, when an appropriate SE is found after a change in the user context, the service can be reconstructed by changing to the new appropriate SE while maintaining the service state (e.g., when a user moves, the monitor is automatically changed to the nearest one).

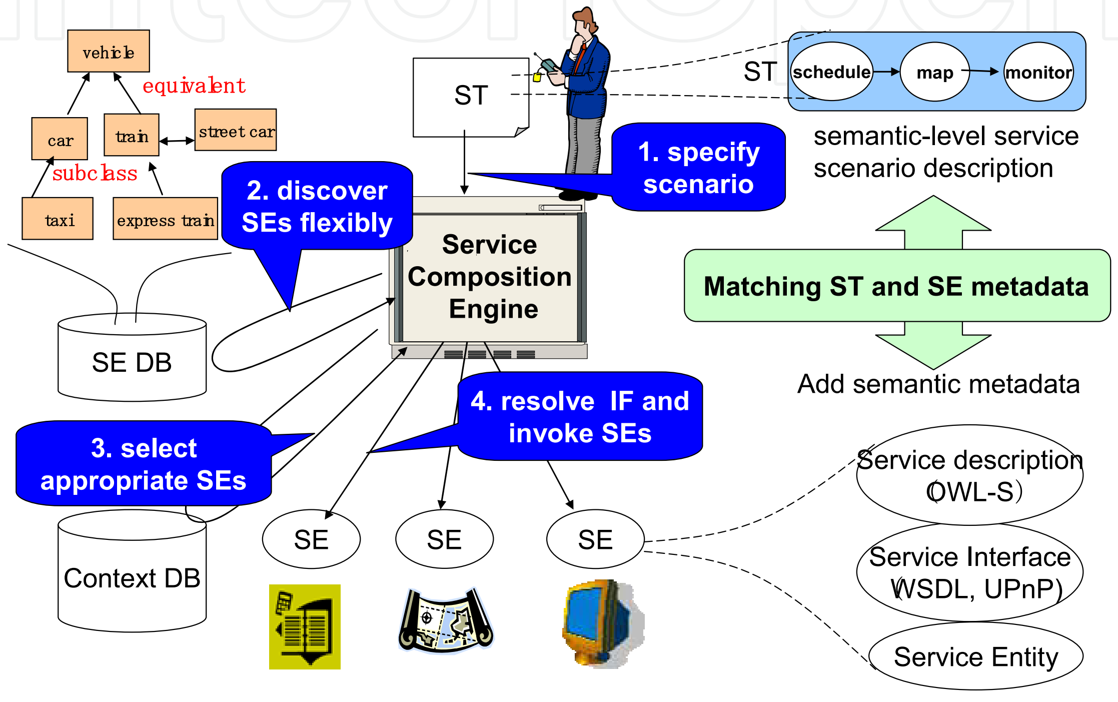

vehicle
equivalent
car
train
street car
subclass
taxi
express train
ST
1. specify scenario
ST
schedule
map
monitor
semantic-level service scenario description
2. discover SEs flexibly
Service Composition Engine
Matching ST and SE metadata
SE DB
Add semantic metadata
4. resolve IF and invoke SEs
3. select appropriate SEs
Service description (OWL-S)
Context DB
SE
SE
SE
Service Interface (WSDL, UPnP)
Service Entity

Fig. 1. Outline of our service composition technology

3. Applying commercial BPEL engine for service composition execution unit

We have already implemented a prototype service composition system with the functions mentioned above (Yamato et al., 2006) (Yamato & Sunaga, 2007). However, to develop this prototype system to commercial-use-level quality, many adjustments such as adding security functions, accommodating web service standard specifications, and tuning of middleware (Java VM, Tomcat, Axis, and RDF parser, for example) are required. These adjustments are ongoing and require a high maintenance cost. Meanwhile, in the market, with the spread of web service technology, there are many market products for the interpretation and execution of the BPEL language (BPEL engine). Therefore, in this study, we evaluate a method for achieving a commercial-level quality of the service composition engine at a low cost by combining featured functions of our prototype system and BPEL engines in the market.
When BPEL and WSDL are deployed, a BPEL engine interprets the BPEL description and executes activities such as the invoking of a web service. Therefore, we propose hybrid

methods where searching for SEs based on ST and selecting an appropriate SE are performed by U-ske (Ubiquitous Service Kernel Emulator) as a preprocess, and execution of an SE including SOAP messaging, for example, is performed by a BPEL engine. Here, BPEL is used, so the target component is limited to web services.
The flow of service composition is as follows. U-ske searches for available SE candidates based on the semantic description of an ST and selects the most appropriate SE using the OWL-S Profile of those candidate SEs and the user context. Next, a BPEL description is dynamically generated using control tags described in ST and WSDL of the selected SE. By deploying the generated BPEL and WSDL on a BPEL engine, the BPEL engine executes the composed service. Here, search and selection units can be reused from a previously developed prototype composition engine while a BPEL generation unit and a BPEL deployment unit need to be newly developed. The BPEL generation unit is universal but the BPEL deployment unit is dependent on the BPEL engine of each vendor.
In this way, featured functions (1)(2) of our service composition technology can be achieved. However, commercial BPEL engines are intended for composition of predefined web services and web service change-over during service execution according to the change of context is out of scope. Therefore, we discuss some methods that correspond to the service change-over function described in (3).

Method 1: Service change-over is out of the scope of this U-ske and BPEL hybrid system.
Method 2: According to the change of context, U-ske generates a new BPEL description and has it re-executed by a BPEL engine. The state of service (parameter value or the advance state of process) of the old BPEL is obtained from interfaces of each vendor BPEL engine. The state is copied to a new BPEL, and the new BPEL is restarted by U-ske.
Method 3: Re-execution of a new BPEL is performed in the same manner as described in Method 2. During execution, the BPEL engine writes the process state to U-ske periodically according to the description of BPEL, and when the SE change-over occurs, a new BPEL is executed by copying the state maintained in U-ske.
Method 4: SE changing is performed by a representative SE, and the BPEL engine invokes a fixed representative SE. The representative SE selects, changes, and invokes an appropriate SE according to the change in user context.

	Method 0 (full development of composition engine)	Market BPEL engine + U-ske			
		Method 1	Method 2	Method 3	Method 4
feasibility	Good	Good	Bad	Fair	Good
feature function sufficiency level	Good	Bad	Good	Good	Good
commercial function sufficiency level	Bad	Good	Good	Good	Good
development cost	Bad	Good	Fair	Fair	Fair
maintenance cost	Bad	Fair	Fair	Fair	Fair
low vendor dependency	Good	Fair	Bad	Fair	Fair
memory usage	Fair	Good	Good	Fair	Fair
process performance	Good	Good	Fair	Bad	Fair

Fig. 2. Evaluation table of five methods obtained through case study

In Method 2, the method of obtaining the execution state is vendor dependent, and that method is difficult in terms of feasibility. Method 3 leads to a lot of wasted processes because the execution state needs to be written in U-ske periodically. Method 4 has some difficulty to describe because the representative SE needs to be described in ST, but Method 4 is easy to implement. These Methods together with Method 0 (previous work -fully developed service composition engine by ourselves) are compared in terms of the following evaluation criteria: feasibility, feature function sufficiency level, commercial function sufficiency level, development cost, maintenance cost, low vendor dependency, resource usage, and process performance. As a result of case studies, Method 4 seems acceptable in terms of feasibility and feature function sufficiency level (see Fig. 2). On the basis of these case studies, we adopt Method 4 to achieve a service change-over function.

4. Implementation and evaluation of proposed method

So far, we have studied a context-aware service composition using a BPEL engine and U-ske. We adopted Method 4, which uses a representative SE to achieve service change-over. This method has the following features.
Using the BPEL engine as an execution unit, service composition and change-over can be performed according to the user context.
U-ske generates the standard BPEL1.1 language, so many BPEL engines that are compatible with BPEL 1.1 can be used (multi-vendor compatibility).
SE search, selection, BPEL generation, and deployment are performed before BPEL execution, so the overhead is large compared to the fixed BPEL execution.

Consequently, these features were evaluated through the implementation. Evaluation points were as follows.

1) Verification of feasibility of context-aware function: The feasibility of whether SE can be selected and composed according to user context needs to be confirmed and whether SE can be changed-over according to the change of user context during execution.
2) Implementation cost of vendor-dependent unit: From the viewpoint of multiple vendor compatibility, the cost of implementation of a vendor-dependent unit needs to be low.
3) Total performance of the system: Total performance of the system needs to be evaluated, SE search, SE select, BPEL generation, BPEL deploy, and BPEL execution.

4.1 Verification of feasibility of context-aware function

Outline of operation of Method 4 is shown in Fig. 3. First, the ST of the desired service is input into a SE search/selection unit. According to the method [5][6], the SE search/selection unit searches for and selects an SE appropriate for the user context and transfers information of the ST and selected SEs to the BPEL generation unit. The BPEL generation unit generates a BPEL description using this information. Then, the generated BPEL is deployed on a BPEL engine through a BPEL deployment unit. The BPEL engine executes the generated BPEL service.
During service execution, a representative SE is used for the SE change-over. The representative SE is generated as an inner resource of U-ske at the same time that the BPEL is generated from the ST. The grammatical difference between ST and BPEL is that in the ST, the invoked web service is specified abstractly, and there is an original control tag "search" that searches for an SE during execution. Other control tags are the same as those of BPEL. Therefore, the mapping from ST to BPEL is performed in the manner shown in Fig. 4. The "search" tag is mapped to the representative SE in the BPEL. The BPEL invokes a representative SE, which is inner resource of U-ske, and the representative SE changes and invokes an appropriate SE using the SE search/selection unit. The representative SE can be automatically generated from OWL-S Process model.
According to this design, the proposed method was implemented by Java language using an open source BPEL engine, Active BPEL2.0. Using this implemented system, the sample ST (Fig. 3), which shows the hotel map on the nearest monitor, is examined. We confirmed that an appropriate BPEL description could be generated for each user, and changing the SE to the nearest monitor SE according to the change in the user position during execution was performed. In this way, our system demonstrates the feasibility of context-aware composition and changing, which was impossible using the ordinary BPEL engine.

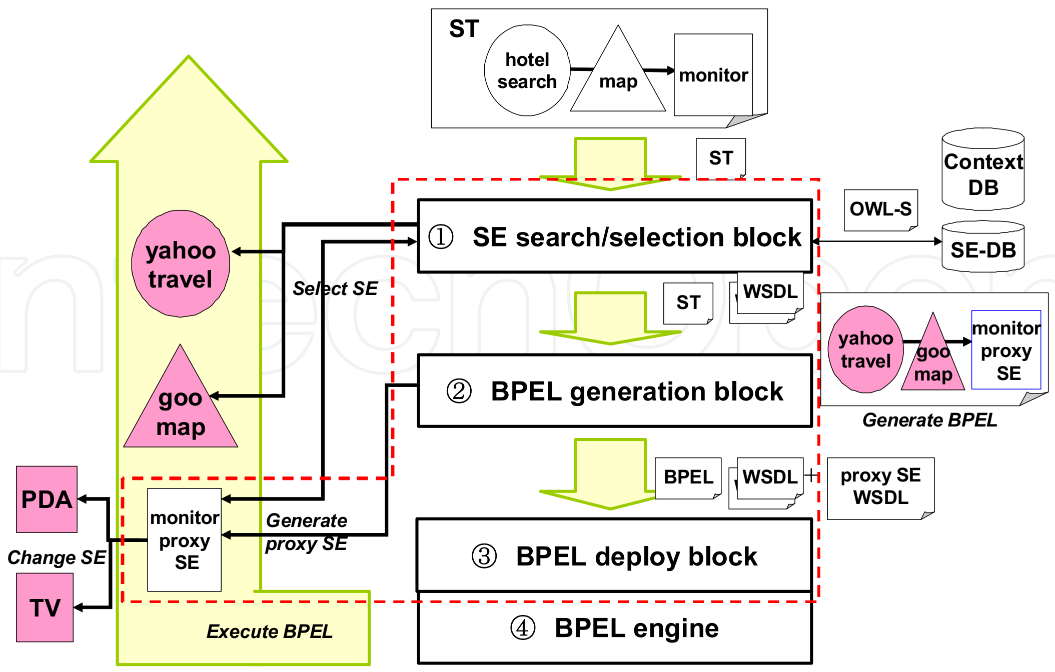


Fig. 3. Outline of service composition system using U-ske and BPEL engine. (U-ske function is inside dotted line)

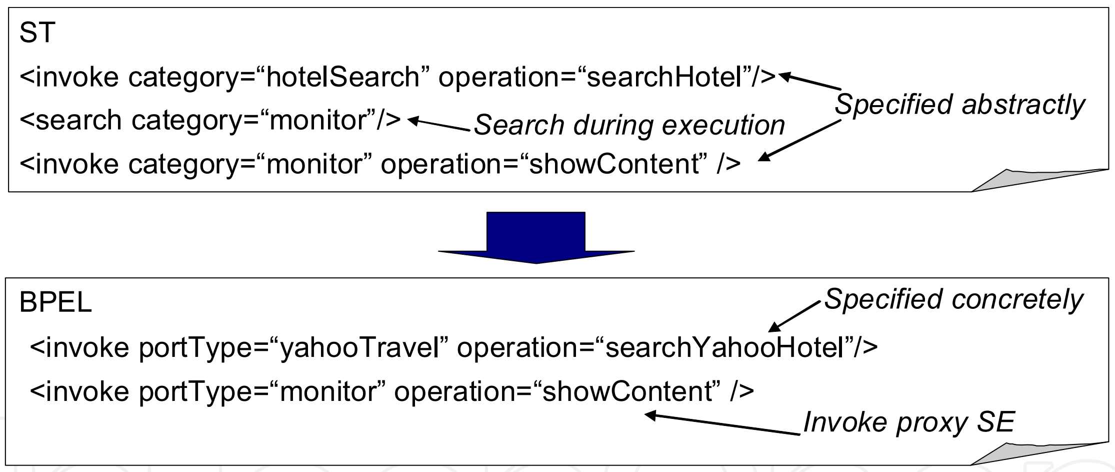

```
ST
<invoke category="hotelSearch" operation="searchHotel"/>
<search category="monitor"/>
<invoke category="monitor" operation="showContent" />
```
```
BPEL
 <invoke portType="yahooTravel" operation="searchYahooHotel"/>
 <invoke portType="monitor" operation="showContent" />
```

Fig. 4. Transforming example from ST to BPEL

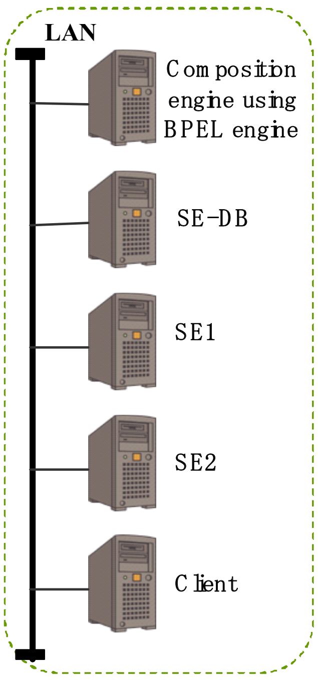
4.2 Implementation cost of vendor dependent unit

There are many BPEL engines in the market, so an appropriate product needs to be selected according to the user policy (e.g., users wishing to use SAML for authentication can select a BPEL engine on which a WS-Security SAML token profile is implemented). In this method, a BPEL description is generated from an ST through a BPEL generation unit and then deployed and executed on a BPEL engine. Therefore, the only vendor-dependent unit is a BPEL deployment/un-deployment unit to deploy/un-deploy a BPEL description on a BPEL engine.

When this function was implemented on Active BPEL, the number of program code lines was approximately 800. In addition, we confirmed that the function could be implemented on the BPEL Process Manager of Oracle at approximately the same size. This indicates that in the proposed method, the interface is based on standard technologies such as BPEL and WSDL, so the service composition featured function can be implemented on various BPEL engines with the development of a small-scale vendor-dependent unit. Here, the BPEL generation function generates BPEL1.1, so this method cannot use the vendor-dependent BPEL extension.

4.3 Evaluation of total performance of proposed service composition system

In this method, performance deteriorates compared to normal use of BPEL because steps of BPEL generation and deployment are needed. Therefore, the performance of this method needs to be evaluated through a performance measurement. Follows are measurement conditions, and Fig. 5 shows the performance measurement environment.



	Hardware	Software
Composition engine using BPEL engine	Dell Precision650 CPU: Xeon 3.2 GHz RAM: 3.0 GB NIC: Broadcom Gigabit ethernet	Windows XP SP2 JDK 1.4.2_13 Tomcat 4.1.31 Active BPEL2.0 U-ske program
SE-DB	HP Proliant ML310 CPU: Pentium4 2.53GHz RAM: 2.25GB NIC: Realtek RTL8139 Family PCI Fast Ethernet	Windows XP SP2 JDK 1.4.2_13 PostgreSQL 8.0
SE1, SE2	HP Proliant ML310 CPU: Pentium4 2.53GHz RAM: 2.25GB NIC: Realtek RTL8139 Family PCI Fast Ethernet	Windows XP SP2 JDK 1.4.2_13 Tomcat 4.1.31 Axis 1.2 sample Web service
Client (JMeter)	IBM Netvista CPU: Pentium4 2.53GHz RAM: 512MB NIC: Intel(R) PRO/100 VE NetworkConnection	Windows XP SP2 JDK 1.4.2_13 JMeter 2.0.1

Fig. 5. Performance measurement environment

Measured items:
・Throughput (CPU utilization and memory usage as the number of ST processes per hour is changed)
・The number of concurrent ST executions (CPU utilization and processing time of each processing section as the number of concurrent ST executions is changed)
Measurement setting:
・Candidate SEs matching OWL-S Process model are found from one SE-DB.
・SE is automatically selected according to its marked score (In this measurement, the physically nearest SE gains a high score and is selected automatically).
・Five sections are measured: SE search, SE select, BPEL generation, BPEL deployment, and BPEL execution.
Measurement conditions:

・An ST consists of five categories and there are ten candidate SEs for each category.
・An ST invokes each SE once and copies one parameter to the next SE (The process of copying a string variable of the former SE return value to the next SE argument is repeated five times).
・Response time of SE is fixed at 0.1 s.

As a result of the measurements, throughput of service execution by the above-mentioned ST was 3600 ST executions/hour with CPU utilization of 50% (Fig. 6 (a)). When the throughput exceeded 3600 ST executions/hour, BPEL deployment failed frequently while there was some extra capacity of CPU resources. This was due to Active BPEL. Therefore, the limit of throughput is about 1 ST per second. The measurement during concurrent ST executions demonstrates that BPEL deployment and BPEL execution require much processing time (Fig. 7). BPEL deployment in particular required a heavy CPU load because a grammatical validity check needed to be performed. To check the influence of BPEL deployment, the throughput was measured when the BPEL was generated from the ST for the first time, and the generated BPEL was executed repeatedly for each user. In this case, when CPU utilization was 50%, throughput was 44,000 BPEL executions/hour, and the throughput maximum was 52,000 BPEL executions/hour (Fig. 6 (b)). That is more than ten times the number of executions in the case of deploying BPEL every time. This value is nearly equivalent to the throughput value of a normal usage of the BPEL engine with the same environment. For the memory usage, the value increased to the heap size of JVM, and after that, the value became steady. Meanwhile, the variation of CPU utilization was large as shown in Fig. 6 due to the influence of garbage collection.
In the concurrent ST executions, CPU utilization was 100% when 50 STs were executed. Therefore, assuming commercial operation, the load needs to be distributed to multiple engines using a load balancer so that no more than 20 processes of BPEL generation and deployment occur concurrently. When a generated BPEL description is used repeatedly, the maximum number of concurrent ST is higher than 50, of course.
According to the performance measurement result, we found that concurrent processes with frequent BPEL deployment is a difficult task in terms of CPU processing load. However, we also found that sufficient performance is obtained by customizing, generating, and deploying BPEL descriptions according to the user for the first time only, using the generated BPEL descriptions repeatedly and changing SE, which is frequently changed by the representative SE.
In the areas where customization and context-awareness are required, our method can compose the service for each user. In these areas, there are several possible selections. For example, when the service is reconstructed frequently according to the user context, load distribution is needed, so the composition engine is implemented on each home gateway. In addition, when the service, once customized by a user, is used repeatedly, the composition engine is implemented on a network server to provide a service for many public users.

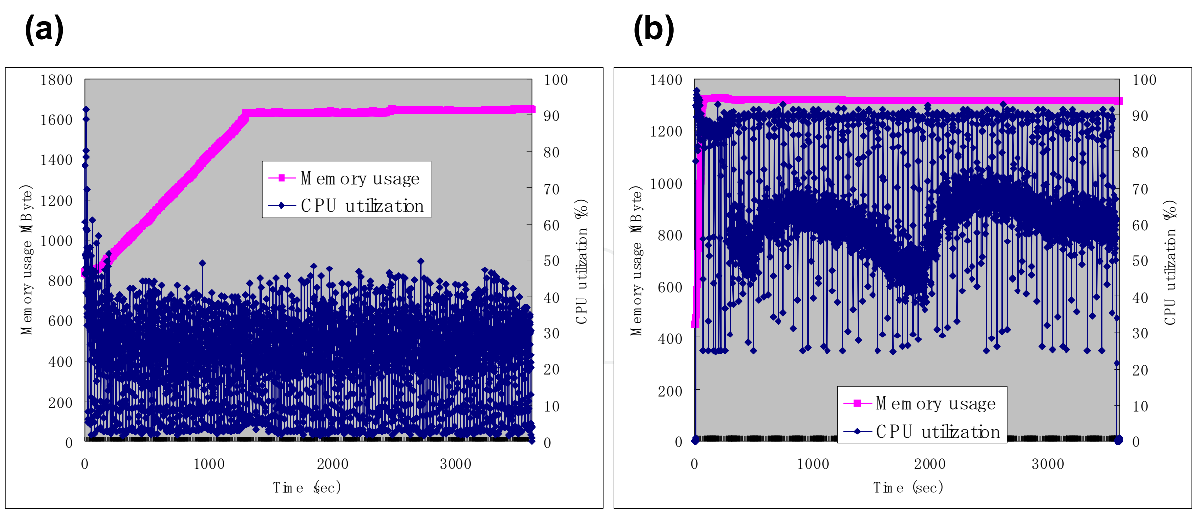


Fig. 6. (a) Memory usage and CPU utilization when the ST is used 3600 times per hour (when BPEL generation occurs every time).
(b) Memory usage and CPU utilization when the ST is used 52,000 times per hour (when the generated BPEL is used repeatedly).

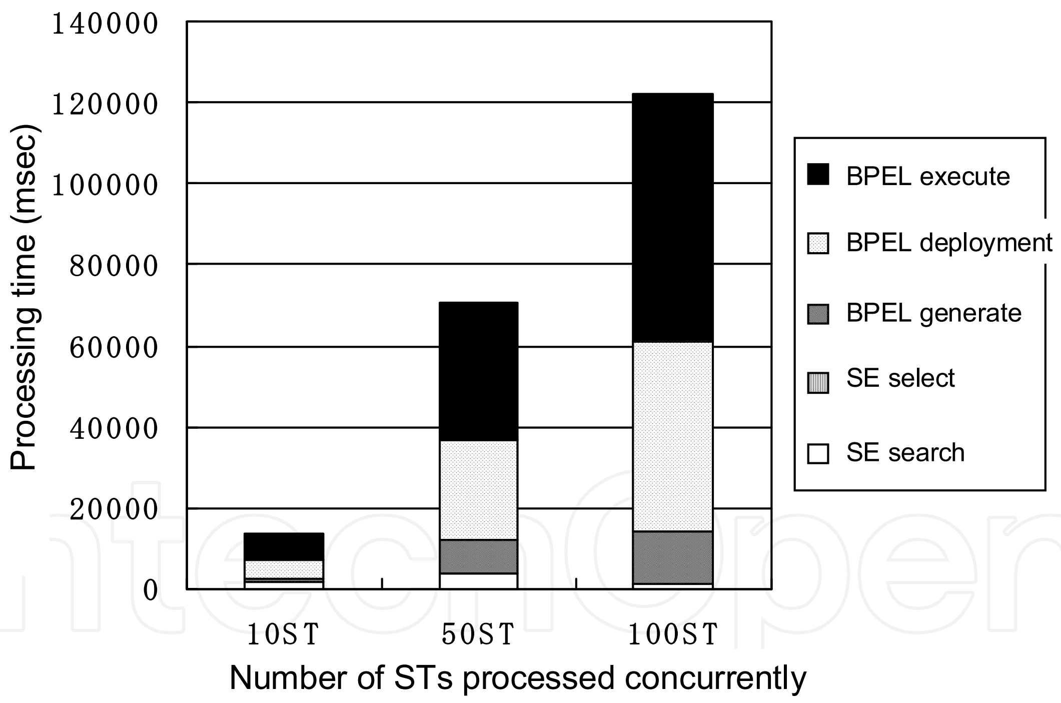


Fig. 7. The processing time sections during concurrent ST execution

5. Sample Application

An example application using the service composition technology, "business trip supporter" (see Fig. 8), was achieved. This service reduces routine work when a user makes a business trip. Wherever the user is, when the time to leave comes, a device containing an alarm function nearby reminds the user of the departure time (according to the event in a PDA scheduler), and trip information (maps, train timetable, weather, for example) is provided to the nearest printer or monitor.

The main features of this service are context-awareness and customization. The former means that an appropriate device is selected automatically according to the user's location without paying attention to differences in device interfaces due to the ST and SE change-over functions. In the office, a speaker may be invoked, while a pet robot with a sound device may be invoked at home because there is no other sound device at home. In the metadata DB, the pet robot is related to the sound device by the OWL subClassOf property, so the engine discovers the pet robot as a substitute for an alarm using metadata links. The customization allows a user to change an ST easily because the user can describe the ST without knowing rigid interface definitions of SEs. For example, a user can easily add weather information using the GUI ST editor.

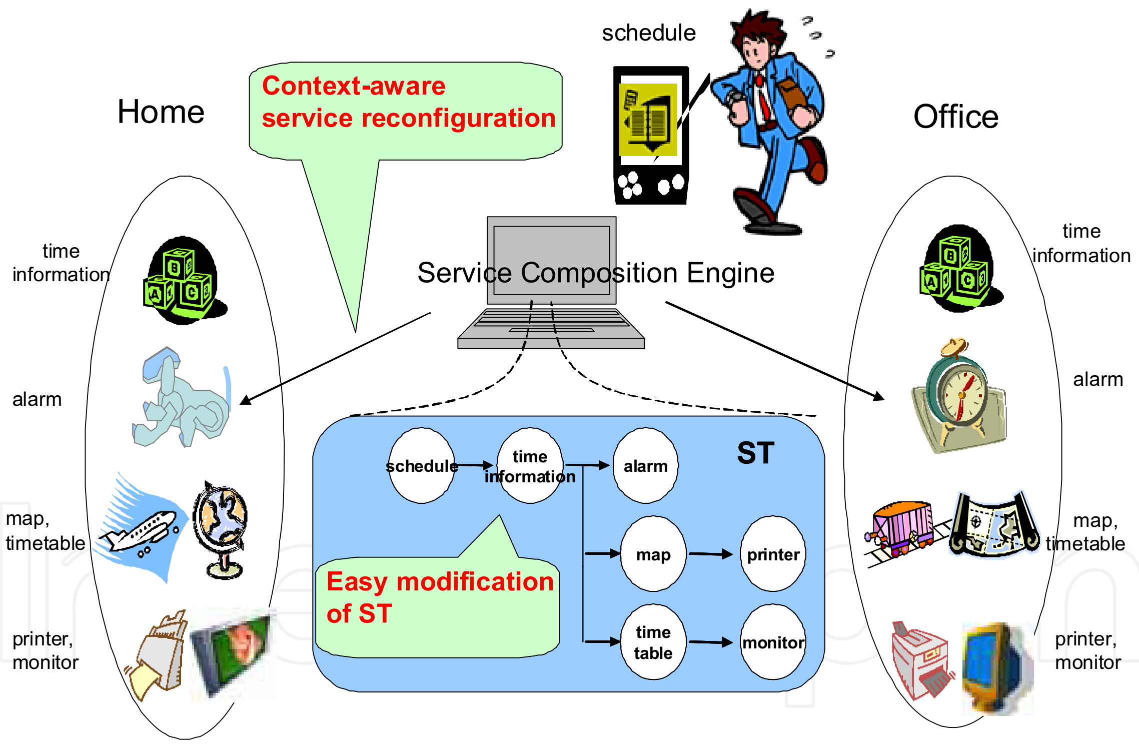


Fig. 8. Sample application of business trip supporter

6. Related works

In various related studies, multiple service components are composed on a network and composite services are provided. In particular, BPEL (Jordan et al., 2007), STONE (Minami et al., 2001) and Ninja (Gribble et al., 2001) have similar targets as this study. Those technologies compose service components based on service flow descriptions such as BPEL document, service graph, or Path, just as our ST does. BPEL is strongly dependent on WSDL, so changing WS is difficult. STONE names each component's input and output using an original naming system and binds components where the output of one component matches the input of another component. However, such original naming makes the system less extensible. The flexible composition techniques of our technology make it superior to other technologies. Standard semantic metadata is assigned to an SE, and SEs are searched with metadata using OWL links. Then, the SE is invoked after being converted to an actual interface using a grounding description.
Similar to our study, TaskComputing (Masuoka et al., 2003) and Ubiquitous Service Finder (Kawamura et al., 2005) also try to compose service components using Semantic Web technology. TaskComputing converts all devices and objects into services and describes their properties using OWL-S. In TaskComputing, a thing corresponding to our "SE" is called a "service," and possible combinations of services, in which service outputs match inputs of the next service, are proposed from all services that are available in the user area. Then, a user selects a desired combination manually to invoke a composite service. Ubiquitous Service Finder also composes service components in a sequence by invoking WSs in which one's output matches the next one's input. TaskComputing targets a bottom-up approach by focusing on each service and searching for the next service that matches. On the other hand, our study targets a top-down service composition where a user specifies a desired composite service as an ST, and the system finds and composes appropriate SEs. Therefore, our study and TaskComputing are complementary studies. However, TaskComputing has two problems. 1) A user's manual selection might be difficult because the number of candidate combinations increase enormously when the number of available services in the user area increases, and 2) complex combinations of services including branches and loops are hardly feasible because services are linked one-by-one in sequence. As a user's desired components are specified by a category in the ST description, our technology makes such selection easier because the user only selects appropriate SEs in each category. Available control tags of an ST are the same as those of BPEL, and of course, execution of a complex service using branches and loops is possible.
Our previous work (Yamato and Sunaga, 2007) is discussed service composition technology using semantic web techniqus and implemented the system by ourselves. This work uses commercial BPEL engine, therefore users can select the best BPEL engine based on their usage. U-ske unit of this work enables ordinal BPEL engines to user customizable service composition systems with small cost.

7. Conclusion

We presented a context-aware service-composition system using U-ske (Ubiquitous Service Kernel Emulator) and the BPEL engine to achieve commercial-level quality at a low cost. U-ske searches for and selects appropriate service components according to the user context, generates a BPEL description, and deploys that description in the BPEL engine. The BPEL engine invokes each web service based on the deployed BPEL description. We also presented a method for service change-over, which was a problem of using the BPEL engine. We have implemented the method and evaluated its feasibility, multi-vendor compatibility, and processing performance. The evaluation results demonstrate the effectiveness of our method. For the future, we will conduct a detailed evaluation of practical applications of customized and context-aware services.

8. References

Weiser, M. (1991) .The Computer for the 21st Century, *Scientific America*, Vol.265, No.3, (Sep. 1991) pp.99-104

Schilit, B.; Adams, N. & Want, R. (1994). Context-Aware Computing Applications, *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications,* pp.85-90, Dec. 1994.

Minami, M.; Morikawa, H. & Aoyama, T. (2001). The Design of Service Synthesizer on the Net Workshop on Highly Distributed System, *Invited Presentation of IEEE Symposium on Applications and the Internet (SAINT2001)*, Jan. 2001.

Gribble, S.; Welsh, M.; Behren, R.; Brewer, E.; Culler, D.; Borisov, N.; Czerwinski, S.; Gummadi, R.; Hill, J.; Joseph, A.; Katz, R.; Mao, Z.; Ross, S. & Zhao, B. (2001). The Ninja Architecture for Robust Internet-Scale Systems and Services, *IEEE Computer Networks Special Issue on Pervasive Computing,* Vol.35, No.4 (Mar. 2001) pp.473-497.

Yamato, Y.; Tanaka, Y. & Sunaga, H. (2006). Context-aware Ubiquitous Service Composition Technology, *Proceedings of IFIP International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS2006)*, pp.51-61, Apr. 2006.

Yamato, Y & Sunaga, H. (2007). Context-Aware Service Composition and Component Change-over using Semantic Web Techniques, *Proceedings of IEEE 2007 International Conference on Web Services (ICWS 2007)*, pp.687-694, July 2007.

Jordan, D.; Evdemon, J. et al. (2007). Web Services Business Process Execution Language Version 2.0, *OASIS Standard,* http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, Apr. 2007.

Martin, D. et al. (2004). OWL-S: Semantic Markup for Web Services, *W3C Member Submission* , http://www.w3.org/Submission/OWL-S/, Nov. 2004.

Paolucci, M., & Sycara, K. (2003). Autonomous Semantic Web Services, *IEEE Internet Computing*, Vol.7, No.5, (Sep. 2003), pp34-41.

Sycara, K.; Paolucci, M.; Anolekar, A. & Srinivasan, N. (2003). Automated discovery, interaction and composition of semantic web services, *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol.1, No.1, (Dec. 2003), pp.27-46.

Masuoka, R.; Parsia, B.; Labrou, Y. & Sirin, E. (2003). Ontology-Enabled Pervasive Computing Applications, *IEEE Inteligent Systems*, Vol.18, No.5, (Sep. 2003) pp.68-72.

Kawamura, T.; Ueno, K.; Nagano, S.; Hasegawa, T. & Ohsuga, A. (2005). Ubiquitous Service Finder - Discovery of Services semantically derived from metadata in Ubiquitous Computing, *Proceedings of 4th International Semantic Web Conference (ISWC 2005)*, pp.902-915, Nov. 2005.