Context-Aware Ubiquitous Service Composition Technology

Yoji Yamato, Yohei Tanaka, and Hiroshi Sunaga
NTT Network Service Systems Laboratories, NTT Corporation,
3-9-11 Midori-cho, Musashino-shi, 180-8585 Japan
yamato.yoji@lab.ntt.co.jp

Abstract. A context-aware ubiquitous service composition technology is proposed. This can be used in both enterprise and consumer communication service environments. Because the context-awareness is the one of the most important factors in ubiquitous computing, it is insufficient in conventional service integration based on rigid interface design. The flexible service composition based on user context is required. Our proposed approach is that a service composition engine discovers suitable service elements from the network based on the user context and binds them dynamically in accordance with a semantic-level service scenario. Evaluation results show effectiveness and sufficient performance of this architecture.

1 Introduction

Ubiquitous computing environments, where not only PCs but also various other devices are connected to networks [1], are expected to offer context-aware services that match user situations and tastes. Because users' needs change dynamically according to the user context such as position or time, an idea to compose appropriate service elements in the network dynamically based on the user context is a promising approach [2][3], as an alternative approach to the conventional way of providing services, where service providers prepare services perfectly in advance.

We have established a service composition technology to achieve this requirement. It provides users with the most appropriate composed service in their context while the service scenario can be described easily by not only professional programmers but also non-professional people [3][4]. We have devised a semantic-level service description and mechanisms for service element discovery and dynamic binding. In the rest of this paper, we describe the concept, proposed architecture, example application, and evaluation results through implementation.

2 The concept of service composition

The purpose of this study is to establish basic technologies that enable non-

Please use the following format when citing this chapter:

Yamato, Y., Tanaka, Y., Sunaga, H., 2006, in International Federation for Information Processing, Volume 205, Research and Practical Issues of Enterprise Information Systems, eds. Tjoa, A.M., Xu, L., Chaudhry, S., (Boston:Springer), pp.51-61.

professional people to compose and customize service easily by using a service composition engine implemented on their terminals such as a PC or cellular phone. Here, in this paper, we assume one prerequisite that service elements previously advertise their service description such as interface to the network.

In the B-to-B area, BPEL (Business Process Execution Language) is attracting attention. However, BPEL coordinates multiple web services for the purpose of semi-permanent system integration, and therefore not suitable for binding unknown web services according to the user context. Because BPEL requires rigid interface description such as the port type names and operation names of web services, it can be applied only to pre-known web services whose port types and operations exactly match. In other words, BPEL is not flexible in terms of context.

To achieve context aware service composition, we introduce a semantic-level service scenario description, called service template (ST). It describes the service flow of basic functions by using metadata instead of defining rigidly the interface of service elements (SEs). This allows a user to select suitable SEs from multiple candidate SEs whose interfaces are different but have equivalent functions (Fig. 1). This eliminates the describing effort of BPEL documents in each situation. One ST can provide composed service in various user contexts by searching appropriate SEs.

When a user starts to compose a service, the service composition engine discovers available SEs using the metadata described in the ST and selects the appropriate SEs from among the candidates based on the user context. Then, the engine acquires their interface information (WSDL, UPnP doc) and invokes the selected SEs. In the process of composing dynamically various SEs based on a given ST, interoperability among SEs is important. To ensure this, we applied OWL-S (Web Ontology Language for Services) [5], one of the semantic web descriptions, to SE description and implement its translation logic in our service composition engine.

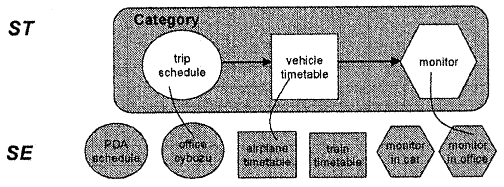

Fig. 1. An image of Service Template

3 Design of the service composition engine

3.1 Description of SE and ST using metadata

Easy composition even by non-professionals has the following requirements:

· STs can be described in a uniform way without being aware of detailed implementation of each SE.

· SEs with equivalent functions can be used equally despite differences in the vocabulary of metadata.

To satisfy these conditions, we decided to use OWL-S for SE description as mentioned above. It consists of three parts: profile, process model, and grounding. The profile has property information describing what capability the service provides;

the process model describes the service behavior such as inputs, outputs, preconditions, and the effects of processes; and the grounding describes the mapping between processes of OWL-S and native operations such as WSDL or UPnP docs.

A process is a neutral description of UPnP or WSDL. Therefore, users can describe STs by using processes as semantic metadata for native operations without paying attention to the detailed implementation. Metadata are described in a uniform way so that it is easy to create a graphical user interface (GUI) tool for describing an ST. The way of resolving vocabulary differences by tracing metadata links of OWL, which will be discussed also in 3.2, is within the scope of the semantic web and therefore familiar with OWL-S descriptions.

There have been some studies on using OWL-S; for example, [6] examines techniques for searching for web services. Task [7] manually composes two web services one of whose input matches the other's output; and Ubiquitous Service Finder [8] composes web services with the same input/output structures by binding them one after another. However, the number of web services to be composed is limited and they can only be used for simple service composition. The difference between our study and Task can be summarized as follows: Task tries to compose two services in the above-mentioned fashion in the adjacent area, which we call the bottom-up approach, while in our study a user specifies a desired composed service as an ST and then SEs that match conditions specified by the ST are discovered for the service, which we call the top-down approach. Task has some difficulties: if the number of web services within the area increases, the number of possible combinations increases rapidly, so users can no longer select one easily; and complicated services will not be provided because Task only connects web services manually one by one. On the other hand, our approach makes the selection of SEs easier because each SE is selected from only candidate SEs managed in each category described in the ST. Note that a category is a group of SE with equivalent functions. Furthermore, complicated services can be provided by describing operation tags such as branching, merging, and looping in the ST.

The flow of service composition using OWL-S is expressed as follows. SEs advertise their OWL-S description to the network. The ST specifies required functions by designating the category name, the operation metadata, and information transfer between categories. The operation metadata means a process of OWL-S. In the information transfer between categories, the mapping between the parameters of the preceding category and the parameters of the next or later category is described. The sequence for composing services by the ST is as follows: SEs are searched for using the metadata of a required function described in the ST; an appropriate SE is selected automatically or manually by using profile descriptions of discovered SEs; and the native operation of the SE is invoked using grounding description of the selected SE. An SE description is outlined in Fig. 2.

If an SE having a different vocabulary of operation metadata can provide an equivalent function (for example, one uses “buy” while another uses “purchase”), this will be resolved by vocabulary resolution by tracing links of the metadata management database, which is discussed in the next section. Vocabulary resolution increases the number of convertible SEs and can achieve flexible composition.

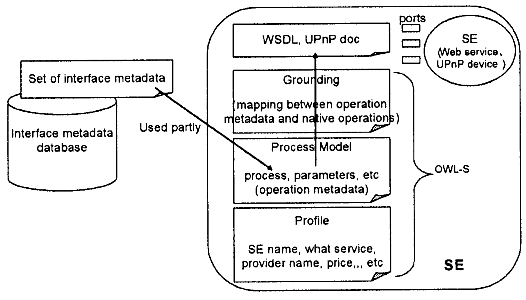
Set of interface metadata
Interface metadata database
Used partly
WSDL, UPnP doc
Grounding
mapping between operation metadata and native operations)
Process Model
process, parameters, etc (operation metadata)
Profile
SE name, what service, provider name, price,,, etc
ports
SE (Web service, UPnP device)
OWL-S
SE

Fig. 2. Using OWL-S to describe an SE

3.2 Management of metadata to resolve the vocabulary differences

Interface metadata must be managed in a way that ST describers, SE providers, and end-users can all use them easily. For it, there are three prerequisites:

· ST describers can easily find and select operation metadata for the ST they want.

· SE providers can add new operation metadata when they implement a new function. In addition, it should be possible to register the equality relationship of metadata with a different vocabulary.

· When a user searches for SEs based on a given ST, it must be able to find all SEs with functions equivalent to the operation metadata described in that ST.

We propose a metadata management method with the features below to meet these prerequisites:

· Operation metadata are grouped into categories based on a particular purpose.

· A new category can be created by adding a new operation metadata to an existing category. A category that has a different vocabulary expression but has equivalent meaning also can be registered.

· Among categories, metadata relationships such as subclass and equality are linked with each other bi-directionally. They are described by the OWL subClassOf property and equivalentClass property.

The management method is shown in Fig. 3. A category has a collection of operation metadata with a particular purpose (for example: video, travel reservation, shopping, etc.). A rectangle shows a category and small characters (e.g., a1, b2, j4) show operation metadata that belongs to a category, and an arrow shows the relation between categories (subClassOf and equivalentClass). For example, suppose that category A has operation metadata a1 and a2. The SE provider creates category C with a new function, adds new operation metadata c3, and registers it as a subclass of the existing category. Category trees will be established by repeating this procedure. Besides the inheritance of categories, equality relationships between metadata with a different vocabulary can be described (category C and K). In Fig. 3, if metadata c3 is described in ST, the composition engine searches by using the link of OWL and finds categories C, E, F, and K.

Therefore, it is unnecessary to define uniform interfaces. SE providers can add operation metadata freely. Even if there are only limited categories at the beginning, we can expect that user-friendly categories will expand through natural selection by adding various functions to the original.

When we implemented the proposed management method, we used XML database, and OWL links between categories are described manually. These links can achieve flexible service composition in different vocabularies, but manual description requires efforts. Therefore, we are currently implementing an automatic metadata mapping method using an ontology mapping technology [9]. We expect that this automatic mapping will help the service composition over different companies.

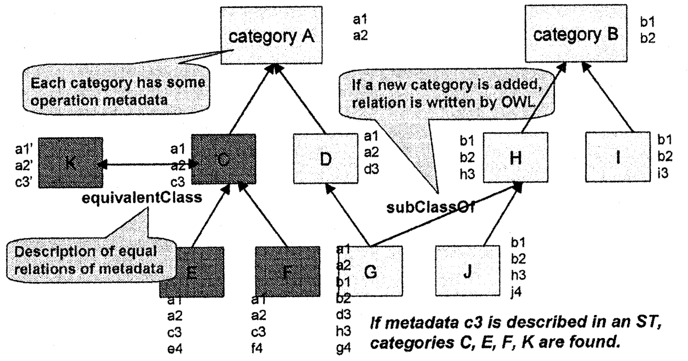
category A
a1
a2
category B
b1
b2
Each category has some operation metadata
If a new category is added, relation is written by OWL
a1'
a2'
c3'
K
a1
a2
c3
C
equivalentClass
D
a1
a2
d3
b1
b2
h3
H
I
b1
b2
i3
subClassOf
Description of equal relations of metadata
E
a1
a2
c3
e4
F
a1
a2
c3
f4
G
a1
a2
b1
b2
d3
h3
g4
J
b1
b2
h3
j4
If metadata c3 is described in an ST, categories C, E, F, K are found.

Fig. 3. Method of managing metadata

3.3 Implementation of the service composition engine

Fig. 4 shows the functional blocks of the composition engine implemented on PCs based on the ideas thus far described. It is implemented in the Java language, and uses jdk1.4.2, tomcat 4.1.31, and axis 1.1 as middleware.

The service search block obtains STs and OWL-S descriptions of SEs by means of a database, UDDI, or JXTA (Peer-to-Peer search technology). When an ST is searched for, the ST name or provider's name, etc., are used as the query keywords. Similarly, when an SE is searched for, the category name and operation metadata described in the ST are used as keywords. Using the metadata management database, OWL-S files with different vocabularies or domains can be obtained.

The resource manage block manages resources such as the ST tags, OWL-S files of SEs, sequence information of ST, and parameter information of SEs, that are used by the composed service.

The service execution block provides functions of tag execution (invocation, looping, switching, etc.), SOAP messaging, and parameter casting. When this block creates SOAP messages according to ST tag information and invokes the operation of a selected SE, the block grounds the operation metadata of the ST to the native operation of the selected SE by using the OWL-S grounding document and WSDL or UPnP doc of that SE. Parameter types casting is also executed in this block.

The event control block receives a synchronous or asynchronous event message from the SE which generates an event message (e.g., the event of a traffic accident).

The service selection block gives a score to each candidate SE by using OWL-S profile documents of the SE, user context information, and user evaluation criteria. An SE of a high score is selected automatically or manually dependently on the user policy. Details of the scoring and selecting SE algorithm are reported in [10].

The control block provides comprehensive control to perform the composition service including acquisition of ST and SE descriptions, SE and ST registration, and execution of the selected ST's tag. If the user policy of SE selection is the manual mode, this block waits for a decision from the user. Also, this block provides parameter inquiry function. This block asks each user for necessary (and unknown) parameters for SEs invocation.

The GUI (user interface) block receives inputs from the user and displays outputs.

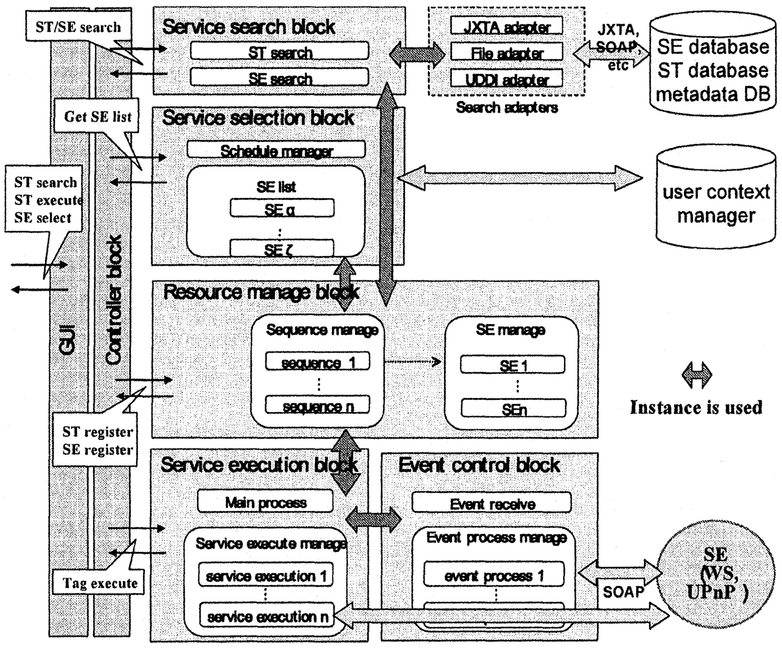

Fig. 4. Function blocks of service composition engine

4 Example Application

An example application using the service composition engine, "business trip supporter" (Fig. 5), was demonstrated on Interop2005 [11]. This service reduces routine work when a user makes a business trip. Wherever he is, when the time for him to leave comes, a device containing an alarm function near him reminds him of the departure time (according to the event of PDA scheduler), and trip information (maps, train timetable, weather, etc) 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 thanks to the ST and the SE selection 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, a pet robot is related to sound device by OWL subClassOf property, so the engine discovers the pet robot as a substitute of an alarm, using metadata tree. The latter allows a user to change an ST easily because he can describe the ST without knowing rigid interface definitions of SEs. For example, a user can easily add the weather information using the GUI ST editor.

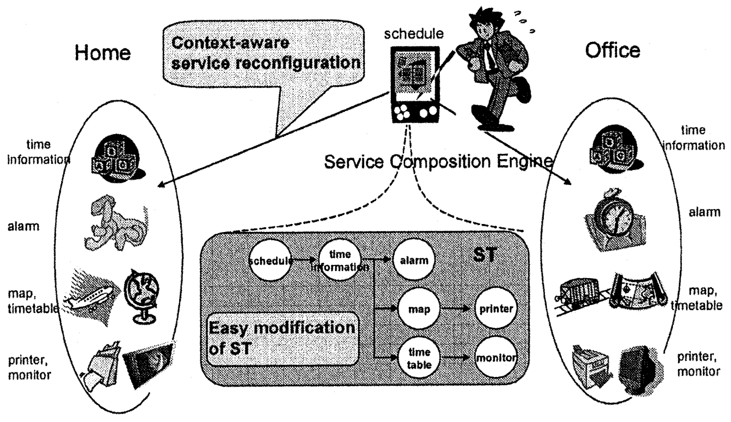

Fig. 5. Overview of business trip supporter

5 Performance measurement of the service composition engine

We have thus far explained the design concept of the composition engine and an example application. Since rich computer resources are indispensable for semantic web handling, usually high-performance servers are used to manage enormous amounts of information. However, our service composition technology aims at ad hoc service composition by using available or appropriate SEs in various user situations. Therefore, we suppose that the number of candidate SEs is not so large, and that the resources of PC or PDA are sufficient for the composition engine. We evaluated the performance of the composition engine implemented on a PC to examine if its performance is sufficient for the assumed usage areas, where an ST has about ten categories and each category has about ten candidate SEs.

5.1 Measurement of service composition

We measured the processing time and memory usage for service composition five times. The graphs in this section show the average of five measurements.

The measurement environment was as follows.

Five IBM Netvista PCs (CPU: Pentium4 2.53GHz, RAM: 512 MB, OS: Windows XP SP2) were connected to the same hub with a 100BASE-TX Ethernet cable.

- PC1 is the service composition engine. In the SE selection step, three evaluation equations are calculated and each score is summed to select appropriate SEs.

- PC2 is the metadata management database. In it, XML database, EsTerra, manages the metadata and OWL links. In this measurement, the number of OWL link steps when users search is only 1. For example, in Fig. 3, if we look at category C, the categories within one step from C, i.e., A, C, E, F, and K, are the search range.

- PC3 is the SE/ST database. PostgreSQL 8.0 manages ST, WSDL and OWL-S.

- PC4, PC5 are the servers to provide SEs. All SE are web services each of which has only one method that returns the value of the formula by mod (arg + L)*N/M.

The patterns and sections for service composition measurement were as follows.

· Measurement patterns

Number of categories described in one ST: 5, 10, 20, 40, 70, 100
Number of candidate SEs in each category: 1, 3, 10, 30, 100

Here, the operation tags described in the ST were only "invoke" (to invoke the SE's method) and "copy" (to copy the SE's parameters to another SE's parameters) and they were used once for each category. The SE with the highest score in the category was automatically (not manually) selected in SE selection.

· Measurement sections:

Startup of the composition engine, ST search, SE search, SE selection, and composed service execution.

Figs. 6 (a), (b) and Figs. 7(a), (b) show logarithmic charts of the processing time taken by the composition engine for SE search, SE selection, composed service execution, and total processing time of the five sections (vertical axis) versus the number of candidate SEs in each category (horizontal axis). For the composition engine startup and ST search, both processing times were constant (less than 500 ms). Similarly, Figs. 8 (a) and (b) show the memory usage for SE search and SE selection versus the numbers of candidate SEs in each category.

· **SE search (Fig. 6(a) and Fig. 8(a))**

For SE search, we found that as the number of candidate SEs in each category and the number of categories in one ST increased, the processing time and memory usage also increased. The reason for this is that the composition engine parsed the files of OWL-S service, profile, process, grounding, and WSDL to check the validity of the SE. Therefore, we are planning to improve this implementation to reduce the number of parsed files; for example, the grounding and WSDL files are parsed after SE selection because they are used only for SE invocation. We also found that very long processing time was required when there were many SEs. To solve this, we suppose that it is effective to narrow down the number of candidate SEs before parsing all OWL-S of SEs, by searching with context information such as position.

· **SE selection (Fig. 6(b) and Fig. 8(b))**

For SE selection, we also found that as the number of candidate SEs in each category and the number of categories in one ST increased, the processing time and memory usage increased. To solve this problem, it is also effective to narrow down the number of candidate SEs before scoring SEs, which is the same as in the SE search case. The increase in time and memory was affected by the number of items for criteria and the evaluation equations for scoring. Therefore, fewer items for the criteria and simpler equations are required to reduce the processing time. However, it should be noted that this is a trade-off with appropriate SE selection processing.

· **Execution of composed service (Fig. 7(a))**

Logically, the chart should be flat regardless of the number of candidate SEs in a category. However, when the number of candidate SEs increased, the chart showed fluctuations. This suggests that the increase affects memory control and instance control to some degree. For the processing time, however, it was so small that there was nothing to do with performances because the average time for one web service invocation was around 20 ms, the average for one parameter copy was around 50 ms, and only less than one second was required in the case of ten categories in one ST.

- **Total processing time (Fig. 7 (b))**

The total processing time was proportional to the power of 0.7–0.9 of the number of candidate SEs in each category. This is because the time for SE search and SE selection was dominant, and these times were almost proportional to the number of candidate SEs in each category and the number of categories described in an ST. We assumed that the usage area was the case where there are about 10 categories in an ST and about 10 candidate SEs in a category. In this case, the most appropriate SE was selected among 10 available SEs in various situations. In that case, only 7 seconds were required for processing, which is small enough to be feasible.

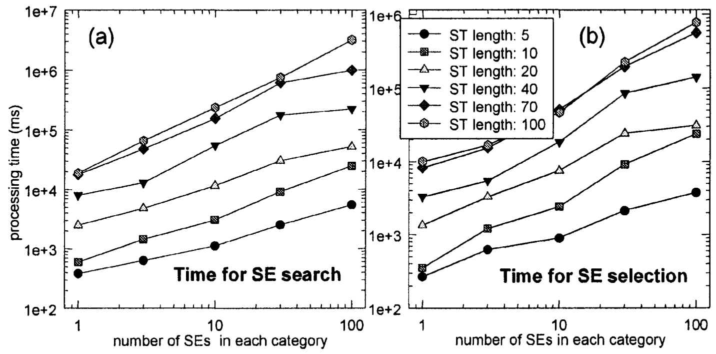

Fig. 6. (a) Time for SE search, (b) Time for SE selection

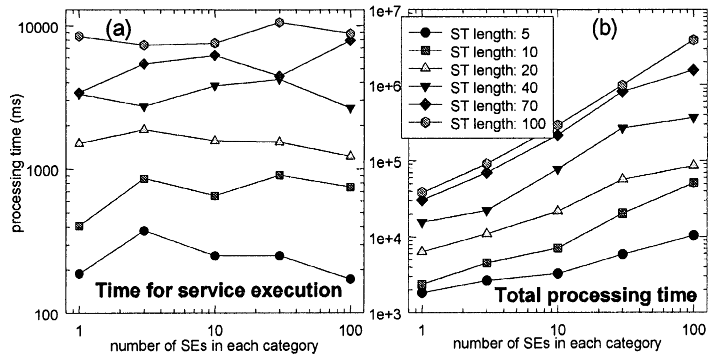

Fig. 7. (a) Time for service execution (b) Total process time versus number of SEs

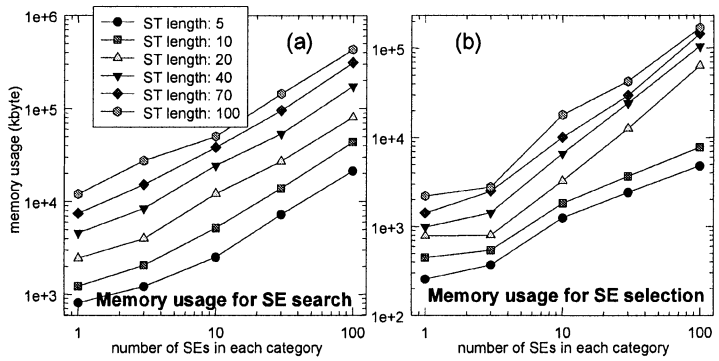

Fig. 8. (a) Memory usage for SE search, (b) Memory usage for SE selection

5.2 Discussion

When there were many candidate SEs for SE search and SE selection, the processing time was rather long. However, in the assumed usage area (about 10 categories in one ST and about 10 candidate SEs in each category) where available and appropriate devices or web services fitting well with user context were composed, it took only 7 seconds for processing time and the memory usage was only 7 Mbytes. This result shows sufficient feasibility in usage area. To enhance the performance of composition, the ideas below should be considered.

- Discover bottlenecks of searching and selection, and make modifications (e.g., modify the parsing OWL-S files)
- Narrow down the number of candidate SEs before scoring (e.g., searching for SEs using the user context, such as position).

6 Conclusions

In this paper, we described the design and implementation of a service composition engine to provide context-aware services in ubiquitous computing environments. We achieved some sample applications using our service composition technologies. We also measured the performance of the service composition engine and found that the performances of service composition in the assumed usage areas were sufficient. In the future, we intend to reduce bottlenecks in the service composition processing found in the measurement when there were many candidate SEs. We are now planning conducting actual experiments of the composition engine in a shopping mall of Aomori Japan in February 2006 to identify feasibility and other problems. In this paper, we explained the service composition engine mainly for PCs, but we have also developed a compact composition engine implemented on cellular phones [12].

References

1. M. Weiser, Ubiquitous Computing, *Communications Of ACM* 36(7), 75-84 (1993).
2. S. Gribble, et al., The Ninja Architecture for Robust Internet-Scale Systems and Services, Special Issue of Computer Networks on Pervasive Computing, 2000.
3. M. Takemoto, et al., A Service-Composition and Service-Emergence Framework for

Ubiquitous-Computing Environments, SAINT2004, pp. 313-318, Jan. 2004.
4. T. Oh-ishi, et al., Service-Composition Technology, Networks, June 2004.
5. OWL-S web site, http://www.daml.org/services
6. M. Paolucci, et al., Autonomous Semantic Web Services, *IEEE Internet Computing* 34-41, October 2003.
7. R. Masuoka, et al., "Ontology-Enabled Pervasive Computing Applications," *IEEE Intelligent Systems* **18**(5), 68-72, 2003.
8. T. Kawamura, et al., Ubiquitous Service Finder, Activities on Semantic Web Technologies in Japan (WWW 2005), 2005.
9. M. Nakatsuji, et al., Proposal and Verification of Flexible Interface Mapping Technique for Automatic System Cooperation based on Semantics, Web Intelligence, September 2005.
10. Y. Yamato, et al., Development of the Service Composition Engine Equipped with the Proposal Function of Suitable Service Elements, Technical Report of IEICE NS2004-225, March 2005
11. Interop web site, http://www.interop.jp/call.html
12. Y. Yamato, et al., Composition Engine on Cellular Phone, FIT2005 LL-001, September 2005.