ADAPTIVE, PLANNING BASED, WEB SERVICE COMPOSITION FOR CONTEXT AWARENESS

Maja Vukovic[*], Peter Robinson[†]

Abstract
The increasing complexity required for context awareness makes such applications difficult to write and adapt. In this paper, we present an architecture for building context aware applications as dynamically composed sequences of calls to fine granularity Web services, addressing the reactive behavior of pervasive environments. Different service compositions of such sequences will result from different contexts such as: devices available, bandwidth, time constraints, location, user requirements and profile. We implement and discuss a specific context aware dynamic service composition problem using the SHOP2 planning system and the BPEL4WS Web service composition technology.

1. Introduction

Traditional computing applications are often static and inflexible. They are designed to run on a specific device, offer a number of predetermined functions to the user and have contextual dependencies embedded in them. Such application models are not suited to operate in a pervasive computing environment, which is characterized by richness of context, by the mobility of users and devices and by the appearance and disappearance of resources over time.

Weiser [10] envisages a pervasive application as means by which a user performs tasks, rather than a collection of computational features. Projects Aura [3], Oxygen [9] and PIMA [1] structure pervasive applications in terms of tasks and their subtasks, which is a software composition problem, allowing for them to dynamically grow and address the changing requirements.

The complexity of a context aware application arises from the sheer number of contextual types it needs to accommodate, as well as the range of values for each context type. To facilitate context awareness we propose a system architecture for building context aware applications as dynamically composed sequences of calls to fine granularity Web services, handled as a planning problem. Contextual changes may trigger further recomposition of the composite service during its execution, causing the application to evolve dynamically.

The remainder of this paper continues as follows. The next section introduces a reference scenario application. Sections 3 presents Web service composition using the Simple Hierarchical Ordered Planner 2(SHOP2) [5]. Section 4 describes the system architecture. We conclude and outline the areas for future work in Section 5.

[*]Computer Laboratory, University of Cambridge, 15 JJ Thompson Ave, Cambridge, UK, maja.vukovic@cl.cam.ac.uk
[†]Computer Laboratory, University of Cambridge, 15 JJ Thompson Ave, Cambridge, UK, peter.robinson@cl.cam.ac.uk

2. Scenario: Mail Replication System

To illustrate how context aware applications can be built as a collection of cooperating services designed to interact with one another, we describe a particular example of a mail replication system. The simplified mail replication process, consists of two subprocesses executed in parallel: *retrieve mail* and *send mail*.

We synthesize a suitable procedure for mail replication dynamically based on user location, activity, computing device and network bandwidth, as summarized in Table. 1. The activity and the location of the user primarily determine the presentation mode of the incoming mail. The network bandwidth and the type of computing device (and consequently its screen size and color depth) affect the mail retrieval and sending. For example, when on the slower network, only the mail headings are initially downloaded, and outgoing mail is compressed.

	Input: Context data				**Output: Expected behavior**	
Case	**Activity**	**Network**	**Device**	**Location**	**Retrieve Mail**	**Send Mail**
1	Walking	GPRS	Smart Phone	Street	Display headers	Compress
2	Driving	GPRS	Embedded	In-vehicle	Read out headers	Compress
3	Not Driving	WLAN	Embedded	In-vehicle	Display adapted mail	No compression
4	Working	LAN	Laptop	At desk	Display full mail	No compression

Table 1. Context and expected application behavior in simplified mail replication system.

3. Web Service Composition using SHOP2

By describing a Web service as a process in terms of inputs, outputs, preconditions and effect, using the metaphor of an action[1], composition can be viewed as a planning problem [4, 11]. There are three main input parts to the planning problem, as Russel at el. [7] outline: inital state, goal state and operator descriptions. A problem definition, which consists of knowledge about system state and the goal to be achieved, and a domain definition, which consists of available actions and their consequences, are used to find a plan.

The proposed system architecture employs SHOP2, a domain independent planner, which uses a hierarchical task network (HTN) to decompose an abstract task into a group of operators that forms a plan implementing the task. Planning progresses as a recursive application of the methods to decompose tasks into subtasks, until the primitive tasks, which can be performed directly using the planning operators, are reached. In the case where the plan later turns out to be infeasible, SHOP2 will backtrack and try other applicable methods.

Figure 1 shows the listing of a sample problem definition for Case 2 and 3 of the scenario, listed in Table 1. The goal is the task "`login_and_replicate_mail`", with input parameters for replication, namely username and password, as well as the type of the device used (e.g. `in_vehicle_inf_sys`). Context data, represented as predicates such as `location in_vehicle` and `activity driving`, forms the description of the initial state.

Figure 2 shows the plans corresponding to the problem definitions listed in Figure 1. Each plan is a list of operators that can be applied to achieve the goal, and their associated costs (by default 1.0).

[1] In the rest of the paper, terms Web service and action (operator in planning terminology) will be treated as synonyms.

```
(defproblem mail_case2 mail_system(
 (activity driving)
 (location in_vehicle)
 (connection_type GPRS)
 (has bandwidth 40200)
 (device_type embedded_system)
 (embedded_system
    in_vehicle_inf_sys)
 (username john)
 (password testpswd)
 (valid_login john testpswd)
 )
((login_and_replicate_mail
    john testpswd
    in_vehicle_inf_sys)))
```

(a) Case 2 SHOP2 problem definition.

```
(defproblem mail_case3 mail_system(
 (not (activity driving))
 (location in_vehicle)
 (connection_type WLAN)
 (has bandwidth 11000000)
 (device_type embedded_system)
 (embedded_system
    in_vehicle_inf_sys)
 (username john)
 (password testpswd)
 (valid_login john testpswd)
 )
((login_and_replicate_mail
    john testpswd
    in_vehicle_inf_sys)))
```

(b) Case 3 SHOP2 problem definition.

Figure 1. Listings of SHOP2 problem definitions for Case 2 and Case 3 of the reference scenario.

```
(!GET_MAIL SERVER1 JOHN) 1.0
(!GET_MAIL SERVER2 JOHN) 1.0
(!GET_MAIL SERVER3 JOHN) 1.0
(!SELECT_MAIL
    #:?LIST_OF_MAILS1789) 1.0
(!DECOMPOSE_MAIL #:?MAIL1807) 1.0
(!SUMMARIZE_MAIL #:?MAIL_HEADER1819
    #:?ATTACHMENT_TYPE1820
    #:?ATTACHMENT_SIZE1821)  1.0
(!TXT_TO_SPEECH
    #:?LIST_OF_MAILS1789) 1.0
(!GET_PENDING_MAIL) 1.0
(!COMPRESS_MAIL M2 70 35.0) 1.0
(!SEND_OFF M2) 1.0)
```

(a) SHOP2 plan for Case 2.

```
(!GET_MAIL SERVER1 JOHN) 1.0
(!GET_MAIL SERVER2 JOHN) 1.0
(!GET_MAIL SERVER3 JOHN) 1.0
(!SELECT_MAIL
    #:?LIST_OF_MAILS1888) 1.0
(!DECOMPOSE_MAIL #:?MAIL1906) 1.0
(!GET_CONVERSION_RULES
    #:?ATTACHMENT_TYPE1919
    #:?FORMATS_SUPPORTED1921) 1.0
(!REASSEMBLE_MAIL
    #:?MAIL_ID1922) 1.0
(!SHOW_INBOX
    #:?LIST_OF_MAILS1888) 1.0
(!GET_PENDING_MAIL) 1.0
(!SEND_OFF M3) 1.0)
```

(b) SHOP2 plan for Case 3.

Figure 2. Resulting SHOP2 plans corresponding to Case 2 and 3 problem definitions shown in Figure 1.

In both cases, the replication process starts with execution of `GET_MAIL` operator triggered by the statement `(!GET_MAIL SERVER1 JOHN)`, where `!` is a prefix for operator, and `SERVER1` and `JOHN` are the input parameters. The retrieve mail subprocess for Case 2, shown in Figure 2(a), is then a sequence of the following operators: getting the mail from three different servers, selecting mails, decomposing them, summarizing and then "presenting" them to user using the text to speech service.

In contrast, the plan for Case 3 shown in Figure 2(b) involves full display of incoming messages, rather than just showing a summary of their headings, given the increase in network bandwidth.

Issues about performance the planning system and its applicability to control complex real-world context-aware systems are also being considered. Our sample problem and domain definition were small compared to the requirements of a real-world scenario, due to assumptions and simplifications made. The domain definition consisted of 3 axioms, 12 operators and 7 methods. Problem definition for each scenario case consisted of 13 facts describing the initial state and context. The number of inferences made by the planner were 56 and 58 for case 2 and 3 respectively. It took SHOP2 0.15 seconds to devise a plan for case 2, and 0.20 seconds for case 3.

4. System Architecture

To address the reactive behavior of pervasive environments we have implemented the system architecture shown in Figure 3. It uses the knowledge about the application domain and current context to devise a composite Web service, by employing the SHOP2 planner for the selection of required services and the sequence of their execution. To express the logic of a composite Web service the framework uses, an XML-based flow composition language, Business Process Execution Language For Web Services (BPEL4WS) [2].

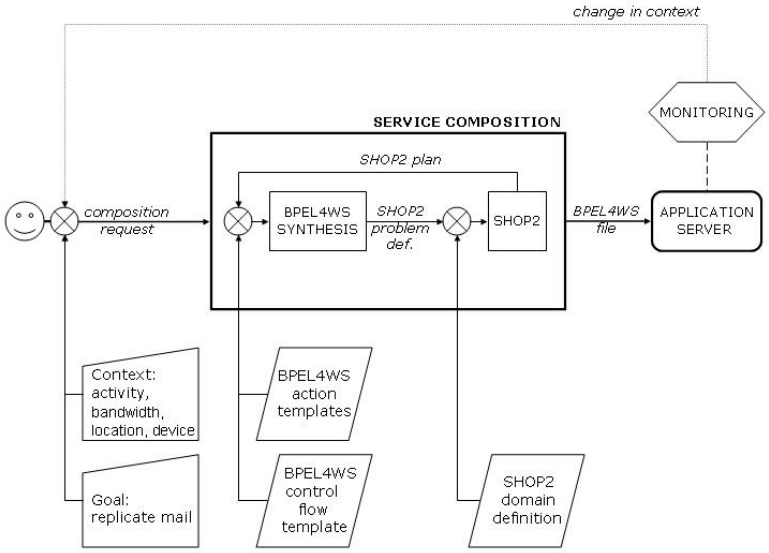
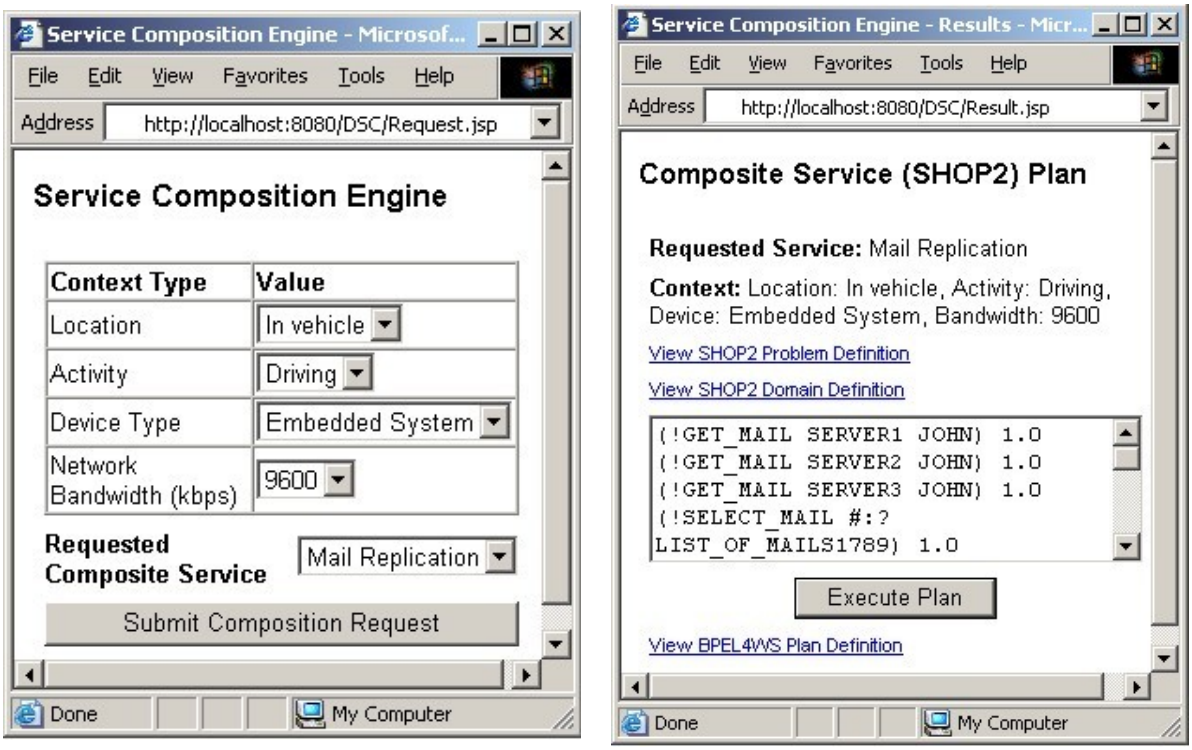


Figure 3. System architecture for context aware, dynamic service composition.

The request for the composite service, consisting of context data and goal, is manually acquired from the user interface as shown in Figure 4(a). The framework then uses the BPEL4WS control flow template to determine SHOP2 subgoals. These subgoals are branches of the BPEL4WS control structure, which need to be populated with the sequences of Web services. System generates SHOP2 problem definition file, such as the one shown in Figure 1, from the composition request. The domain definition file has been pre-compiled and includes operator descriptions. Both problem and domain definition files are passed to the SHOP2 planning system via JSHOP [6], a Java interface to SHOP2.



(a) User front-end for composition request

(b) Plan for composition request in Figure 4(a)

Figure 4. User interface for service composition request and display of the result.

The resulting SHOP2 plans, such as the ones in Figure 2 are then semi-automatically transformed into BPEL4WS schemata. Each operator in SHOP2 plan is mapped to the corresponding BPEL4WS activity definition. The resulting BPEL4WS plan, describing the composite Web service, is then deployed and executed in the IBM Business Process Execution Language for Web Services Java^{TM} Run Time (BPWS4J) [8], running on an application server. At present, the user manually initiates execution of the composite service, as shown in Figure 4(b).

In summary, we have tackled a number of principal challenges by using SHOP2 and BPEL4WS for Web service composition to facilitate context awareness. A method for meaningful goal-decomposition, similar to the one employed in HTN-based planners, will be the key enabler for the successful composition of Web services. Planning systems assume the full knowledge of the domain, which is not feasible and realistic in the case of context aware applications, therefore a mechanism for expressing non-determinism is essential. In contrast to BPEL4WS, SHOP2 does not support concurrent activities. Finally, a proper method is needed for mapping the Web service descriptions into the planning domain, consequently formally separating the concerns from the underlying planning technology.

5. Conclusions and Future Work

In this paper, we have presented our system architecture for constructing context aware applications based on the notion of dynamic Web service composition, to address the increasing complexity that context awareness requires. Contextual changes may trigger further recomposition during the execution of the services, causing the application to evolve dynamically.

The framework employs the SHOP2 planning technology, BPEL4WS and BPWS4J for composition, definition and execution of the composite service, respectively.

The flawless planning and execution in SHOP2 and TLPlan arises from the assumptions that the state of the world (i.e. application domain) is always accessible, static and deterministic. In addition, all the method descriptions are complete and correct–they precisely describe all the possible consequences. The real world and the pervasive operating environment are characterized by information incompleteness (e.g. world may be inaccessible) and incorrectness (e.g. world may not match the model of the planner). Design of the context aware applications therefore requires a mechanism to express non-determinism.

Our future work will involve investigating applicability of a non-HTN planner, devising techniques for run-time recomposition, and addressing scalability issues. We will be composing a more complex and context-rich application to validate the framework.

Acknowledgements

We are grateful to Carl Binding, Yigal Hoffner and Jana Koehler for providing valuable feedback and their guidance in this work. Maja Vukovic would like to thank IBM Zurich Research Laboratory for supporting the funding of her PhD and sponsoring this work.

References

[1] Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremy Sussman, and Deborra Zukowski. Challenges: An Application Model for Pervasive Computing. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 266–274. ACM Press, 2000.

[2] F. Curbera, T. Andrews, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language For Web Services, version1.1. White paper, http://www-106.ibm.com/developerworks/webservices/ library/ws-bpel/, 2003.

[3] David Garlan, Daniel P. Siewiorek, Asim Smailagic, and Peter Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31, 2002.

[4] S. McIlraith and T. Son. Adapting Golog for Composition of Semantic Web Services. In *Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, Toulouse, France, April 2002.

[5] D. Nau, H. Munoz-Avila, Y. Cao, A. Lotem, and S. Mitchell. Total-Order Planning with Partially Ordered Subtasks. Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, pages 425–430. Morgan Kaufmann, San Francisco, 2001.

[6] Dana S. Nau, Yue Cao, Tsz-Au Chiu, Okhtay Ilghami, Ugur Kuter, Steve Mitchell, and J. William Murdock. JSHOP, a Java interface to SHOP2. Website, http://www.cs.umd.edu/projects/shop/download.html.

[7] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, NJ, 1995.

[8] IBM. The IBM Business Process Execution Language for Web Services Java^{TM} Run Time (BPWS4J). Website, http://www.alphaworks.ibm.com/tech/bpws4j.

[9] MIT. Project Oxygen - Pervasive, Human-Centered Computing. Website, http://oxygen.lcs.mit.edu/.

[10] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):66–75, January 1991.

[11] Dan Wu, Evren Sirin, James Hendler, Dana Nau, and Bijan Parsia. Automatic Web Services Composition Using SHOP2. In *13th International Conference on Automated Planning & Scheduling. Workshop on Planning for Web Services.*, Trento, Italy, June 2003.