Autonomic Composition of Ubiquitous Multimedia Applications in REACHES

Oleg Davidyuk, Ivan Milara Sanchez, Jon Duran Imanol, Jukka J. R. Riekki

Autonomic Composition of Ubiquitous Multimedia Applications in REACHES

Oleg Davidyuk[1], Iván Sánchez[1], Jon Imanol Duran[2] and Jukka Riekki[1]

[1]Dept. of Electrical and Information Engineering, P.O. Box 4500, University of Oulu, 90014, Oulu, Finland
firstname.secondname@ee.oulu.fi

[2]School of Computer Engineering, University of Basque Country, Spain
jiduran001@ikasle.ehu.es

ABSTRACT

In this paper, we describe our work in developing an autonomic system that supports the composition of ubiquitous applications at run-time. The applications are composed and adapted accordingly to user preferences and user-provided criteria. We have designed a proof-of-concept prototype of the system and an example multimedia application. The application is a multimedia player which users can control on a large screen using the mobile phone's UI as the controller. We present a user evaluation of the prototype's feasibility and also determine what feedback and control mechanisms are required by the end-users. We report initial analysis of how user satisfaction and comfort level are affected by the autonomy of the system.

1. INTRODUCTION

Ubiquitous environments have recently become an important application domain for autonomic computing [1, 2]. An autonomic system adapts to resource availability changes, allows user mobility across different environments, and, most importantly, configures and controls applications according to the user's specified high-level goals and criteria. Thus, the users only need to specify a desired task (or an application) to the system which will then perform the task as required. Autonomic computing also provides very promising solutions to deal with composite applications built from a number of components which are physically allocated on multiple computational devices. The system has to dynamically discover relevant ubiquitous resources, compare their properties and choose an optimal application configuration (i.e., a combination of resources and application components) from all the possible options in order to support such applications. Moreover, the system has to be able to adapt composed applications when the context or the user goals change. Autonomic mechanisms are essential when realizing this approach, because users may become too overloaded with manual controlling and configuring tasks, and thus, they may even stop using services offered in the environment, if no autonomic mechanisms are provided. However, designing autonomic mechanisms for ubiquitous environments is challenging: the end-users may feel that the system does not support their actions or the system can go out of control. Therefore, it is important to study how the users experience autonomic mechanisms, how much control should be given to the user, and what kind of user interfaces should be offered for the user to control the autonomic mechanisms.

The main contribution of this paper is the user study which we conducted on a prototype autonomic system used for the composition of ubiquitous applications. The results of the user study have not been reported before. This prototype relies on the REACHES middleware [3] and uses an application allocation algorithm [4] which we described in our earlier publications. The aim of this paper is to evaluate the feasibility of the system's prototype and also to study end-user control mechanisms which the prototype has to use to ensure the user acceptability. Besides, we identify possible situations and physical environments (i.e., locations) where the prototype can be used and we study how user satisfaction and comfort level are affected by how autonomous the system is.

The next section gives an introduction to the related work. A short overview of our system and the prototype for application composition is given in the section 3. We present the user evaluation study in the section 4 before concluding the paper in the last section.

2. RELATED WORK

A number of ubiquitous systems are based on automatic application composition. For example, the system presented in Sousa et al [1] supports the self-adaptation of composite applications on several architectural levels. Sousa's research mainly focuses on expressing application requirements in a framework-independent manner, thus, their work also includes a set of user interfaces which are used to collect und-user requirements and to forward them to the system. In contrast to Sousa's model, our system collects end-user requirements via a physical (RFID-based) interface which we describe in more detail in section 4. Another alternative is the COCOA [5] middleware which focuses on composing task-based applications that are modeled as workflows with QoS properties. COCOA utilizes an ontology-based matching algorithm with QoS support to compose applications. The Galaxy [6] framework enables the interoperability of devices in pervasive environments and supports composite services which control and cooperate with these devices. Galaxy, like COCOA, mainly concentrates on semantic-

MUM'08, December 3-5 2008, Umeå, Sweden

free service composition. The Pervasive Component System (PCOM) [7] uses a set of pluggable algorithms in application composition as PCOM developers argue that in certain situations ubiquitous applications may require changing the algorithm at runtime. The balance between user control and system autonomy is studied in [8] and [9]. However, these related works do not consider application composition. Our work also differs from the examples mentioned above in that our main objective is to assess feasibility and user acceptance by carrying out user evaluation experiments.

3. APPLICATION COMPOSITION SYSTEM

We assume that ubiquitous applications are built from a set of software components which may reside on physically distributed computational nodes. Each node can host a single component or component groups, however, the node's resource capacities must not be exceeded. Starting and adapting of applications in our system is governed by the three phase application composition process which is illustrated at Figure 1. The first phase of the process starts either explicitly, after the user activated the application via the application's UI, or implicitly, after the Context Management component, which is responsible for managing context information, automatically triggers the application startup. In this phase, the Application Assembly component searches for the available local nodes (that is, nodes in the user's close proximity, e.g. in the same room), or remote nodes (that is, nodes located physically further away from the user) through the Service Discovery component. However, the nodes have to be searched according to their functional and non-functional (i.e., resource) properties. The functional properties of a node denote its ability to provide certain services. The functional properties are descriptions which resemble interface statements in Java. The non-functional properties mainly denote device resource constraints, such as maximum available memory or computational resource capacity. The applications and nodes are specified using similar descriptions in our system. These descriptions are managed by the Service Discovery component whose main purpose is to find matches between discovery requests (coming from the Application Assembly component) and the available descriptions.

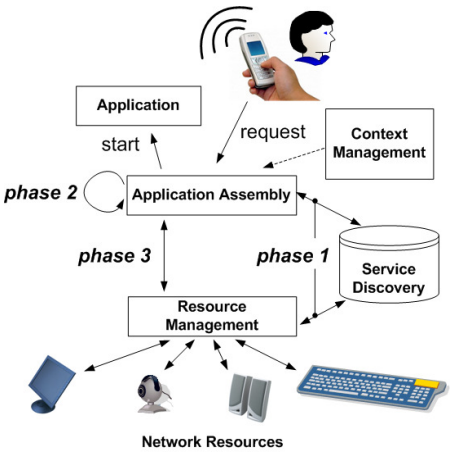


Figure 1. The three phase application composition process.

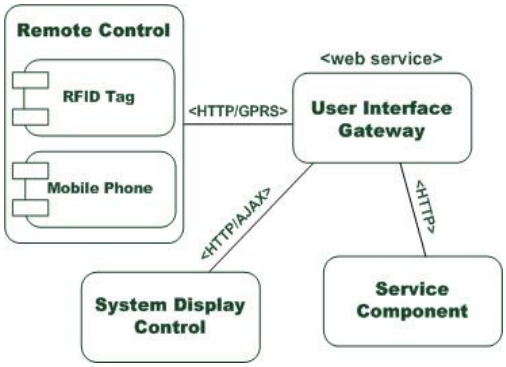


Figure 2. The REACHES architecture.

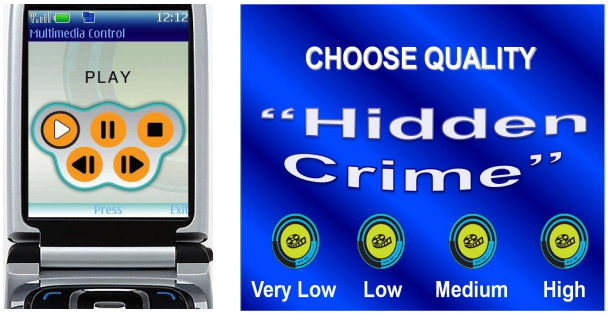
During the second phase of the application composition process the Application Assembly produces the application configuration (i.e., the allocation of components groups onto networking nodes) optimized according to user specified criteria, user needs and the situation in the environment. The Application Assembly component may minimize bandwidth consumption, balance load among the nodes, and meet the various resource requirements imposed by the application components. The resulting application configuration has to meet the functional properties of nodes and application components involved in the configuration. In addition to that, it must not violate the nodes' resource capacities. Our composition system uses a genetic algorithm, which is capable of optimizing application configurations at run-time. The implementation details of the algorithm are presented in [4].

During the third phase of the application composition process the Resource Management component realizes the application configuration by leasing the necessary resources, deploying the components onto them and, finally, configuring the application.

3.1 Proof-of-concept prototype

We realized a proof-of-concept prototype of the application composition system using the REACHES (Remotely Enabling and Controlling Heterogeneous) middleware [3]. It enables the utilization of a mobile terminal UI to control a wide range of ubiquitous applications which can be composed dynamically from pluggable service components. The REACHES architecture (presented in Figure 2) is centralized and consists of four components: Remote Control, User Interface Gateway, System Display Control and Service Components.

The Remote Control UI (RC) consists of a physical user interface (a set of RFID tags) placed in the user's local environment and of a phone equipped with an RFID reader. The RFID tags contain commands which are triggered when the tags are read using the phone. For example, these tags may trigger an application to start, point to a certain resource to be utilized (e.g., an external display), start the adaptation of multimedia content or set user preferences. Both browser and MIDLet-based user interfaces can be used in the mobile phone. The former can be generated and modified at run-time. Predefined user interfaces (created at design time) have to be used in the latter case.



A B

Figure 3. The remote controller GUI (A) and the control panel for choosing quality (B).

The User Interface Gateway (UIG) connects users and service components. It synchronizes different subsystems by processing events from the RC and dispatching them to appropriate service components. In addition, the UIG provides the Service Discovery functionality.

The System Display Control (SDC) connects external displays to the REACHES server. After the UIG registers a display, the browser, which is hosted on a computer connected to the display, loads scripts (REACHeS client) that enable communication between the SDC and the browser. The REACHES architecture does not require deploying any other software on the display side. When an application is started, the system assigns one or more services and resources to it. Then, the service components send events to the displays via the SDC, which dispatches each event to the corresponding browser to perform the requested update to the user interface. Service components can send events to the mobile terminal as well.

The Service Components perform application specific functions and are allocated onto remote computation nodes. The Service Components communicate with the other elements of the REACHES architecture via the UIG.

3.2 Ubiquitous Multimedia Application

We integrated a multimedia player application into REACHES. The application is based on Flash and supports various types of multimedia content. The player allows rendering of files, supports streaming and accepts dynamic playlists.

The application controller UI is shown in Figure 3, A. The multimedia player is controlled by a Service Component which manages the multimedia content. The SC receives commands (generated by the user) from the Remote Control, interprets them and dispatches the commands to the external display which embeds the multimedia player and updates the display with the action required. The URL specifying details of the playlist file can be stored as a parameter in an RFID tag. This data is delivered to the SC when the application is started. Figure 3, B shows the RFID control panel used in our user experience tests.

4. EXPERIMENTAL EVALUATION

We evaluated the feasibility of the application composition prototype via user evaluation in a real environment. Our main goal was to assess user attitudes towards autonomy of the system and analyze the relationship between user satisfaction and the extent of control provided by the system. The user study also helped us to understand how users interact with the prototype in a real situation.

The test subjects were 10 students and research staff members who represented people interested in using new technologies. They were given a task to watch a video file using the multimedia player application (shown at Figure 3, A). The users started the application by touching an RFID tag on a control panel (shown at Figure 5, B) where each tag was associated with a certain video quality. We used four video quality levels which represented user's fidelity requirements: very low (240x180 pixels), low (480x360 pixels), medium (768x576 pixels) and high (1024x768 pixels). After a user chose a quality level the system then determined an optimal configuration of the application according to the available resources. After the application was composed and started, the user could watch the video and control playback using his mobile phone's UI. Users could also change the quality of the video during playback. We used two types of network resources: PCs connected to monitors and three media servers. The PCs provided audio/video capabilities and the media servers hosted the replicated video files. The properties of the network resources included processor CPU speed, bandwidth, memory and screen size. The screen sizes varied from 11 inches (a laptop) to 100 inches (a video projector). The bandwidth levels varied from very slow (GPRS), to medium (WLAN) and fast (broadband). The CPU and memory levels were set according to capacities of the PCs. In addition, each media server only hosted two copies of video files locally, so that different server instances had to be used even during the same test session. Although, all the resources were available, only two configurations of resources (i.e., a combination of a PC with monitor and a media server) could satisfy the application and user requirements simultaneously. All the other configurations were inappropriate, because the playback was either too slow (due to network or CPU speed), or the display size was too small or the chosen nodes did not have the required video files available. For example, if a user chose medium or high quality video files, they were only played on the PC which had a fast (broadband) connection to the server hosting the video file's replica. Also, to satisfy the application requirements, the system had to maximize the screen size, minimize the number of hardware nodes used to allocate the application and minimize the overall bandwidth consumed by the application. We conducted personal interviews and used anonymous questionnaires to collect feedback from our testers. Next, we discuss the user study results.

Feasibility. All the participants found the demonstrated prototype feasible. When asked them to describe a typical situation where they would use our system, they suggested public places such as shopping malls, business centers, and airports. We, then, identified two important characteristics of these environments: first, such places are populated with a large number of ubiquitous resources and, secondly, users are not familiar with resources in these places and, therefore, they have problems comparing the resource properties. Our test subjects also remarked that our system required less cognitive effort to start the application because they did not explicitly need to choose or compare resources. Yet, the users were concerned that the system's autonomy breached their privacy in some application scenarios. Privacy is an important factor to consider, because it may prevent users using the system in real-life.

Control. All the participants reported that they wanted more control over the system's choices in order to feel comfortable. For example, the participants suggested that the system should show feedback, clarifying its choices, on the mobile terminal UI or even on the displays in the environment. Other users desired to confirm the system's choices manually before the system requests the resources and deploys the service components. Thus our key finding is that the system feedback is an important factor which increases the feeling of control. Riekki et al [10] have also studied this issue. We also observed that the users felt more in control when they were just aware of the fact that they were able to change (or cancel) the system's choices, if they wanted to. So, the testers suggested that the system should recommend several application configurations instead of choosing the best one automatically. These recommended configurations have to meet the user criteria and have to be ordered by their fitness value ("the best" configuration - first), so that the user can choose another one if "the best" recommended configuration is not desirable. We found the idea potential, as the system can then automatically adjust the selection criteria according to past user preferences and current context information. However, a further problem is introduced if the users want to choose from multiple application configurations, because the users need graphical interfaces capable of visualizing the available resources and application configurations on a small-sized terminal screen. We also observed whether or not the system behavior matched the user expectations. Although, the majority of the testers indicated that the system behaved as they anticipated, two participants had expected different behavior. That is, they assumed that if they opted for a low video quality level (see control panel at Figure 3, B) the system should then choose an application configuration with a small-sized wall display. However, the system played low quality video on a large display, thus resulting in visible noise caused by video compression. The two testers were not satisfied with the system's choice and preferred to choose a smaller display manually. Thus, we discovered that if additional control mechanisms are provided, the user may tolerate system's choices that do not exactly meet his expectations.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a user study in which we evaluated feasibility and user's attitudes towards the system for composing ubiquitous applications. We have implemented a proof-of-concept prototype which is capable of automatic composition of applications accordingly to user-preferences and user-provided criteria. We evaluated the prototype on 10 users. Whilst admitting that the user group reproduced only a small sample of the population, we found the initial results promising because they indicated that technical users experienced our prototype well. More extensive experiments (e.g., in different contexts and with different user groups) are necessary to study the feasibility of our approach thoroughly.

This user study showed that there is a relationship between user satisfaction and the extent of control provided by the system. That is, we found out that the user's feeling of control was the most important factor which dominated over the other ones. We also found out that even if the system is meant to be autonomous, it nevertheless has to provide optional control mechanisms to correct system's behavior when it doesn't match the user's expectations. We recommend using multiple control mechanisms. E.g., our test subjects required explicit manual control for applications where privacy is a crucial issue. But, they preferred autonomy support when using an application for entertainment. These findings are similar to ones reported by Hardian et al [8]. Thus the most interesting direction of our future research is to study how to balance user control and system's autonomy in order to increase user's comfort level, feeling of control while at the same time minimizing the cognitive load. To address this, we are planning on designing a set of interactive control mechanisms, each of them providing different degree of user control and system's autonomy at run-time. We will compare these mechanisms in order to clarify factors affecting the user's comfort level and feeling of control.

6. REFERENCES

[1] J. Sousa et al, "Task-based Adaptation for Ubiquitous Computing," IEEE Trans. on Systems, Man, and Cybernetics, Special Issue on Engineering Autonomic Systems, Vol. 36 (3), pp. 328-340, 2006.

[2] J. Kephart and D. Chess, "The Vision of Autonomic Computing", Computer, Vol. 36 (1), pp. 41-50. April 2003.

[3] Sánchez, I., Cortés, M., & Riekki, J. (2007) Controlling Multimedia Players using NFC Enabled mobile phones. In Proc. of 6th Int. Conference on Mobile and Ubiquitous Multimedia, Oulu, Finland, December 12-14 2007.

[4] O. Davidyuk, I. Selek, J. I. Duran and J. Riekki, Algorithms for Composing Pervasive Applications, Int. Journal of Software Engineering and Its Applications, Vol. 2, No. 2, April, 2008, pp. 71-94.

[5] S. Ben Mokhtar, N. Georgantas, and V. Issarny, "COCOA: COnversation-based Service Composition in PervAsive Computing Environments with QoS Support", Journal of Systems and Software, Vol. 80 (12), 2007.

[6] J. Nakazawa, J. Yura and H. Tokuda, "Galaxy: a service shaping approach for addressing the hidden service problem," In Proc. of the 2nd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, pp. 35-39, 11-12 May, 2004.

[7] M. Handte, K. Herrmann, G. Schiele and C. Becker, "Supporting Pluggable Configuration Algorithms in PCOM" In Proc. of Int. Workshop on Pervasive Computing and Communications, pp. 472 – 476, 19-23 March, 2007.

[8] B. Hardian, J. Indulska and K. Henricksen, "Exposing Contextual Information for Balancing Software Autonomy and User Control in Context-Aware Systems", Workshop on Context-Aware Pervasive Communities: Infrastructures, Services and Applications, Sydney, May, 2008.

[9] M. Vastenburg, D. Keyson, and H. de Ridder, "Measuring User Experiences of Prototypical Autonomous Products in a Simulated Home Environment", HCI (2) 2007, pp. 998-1007.

[10] J. Riekki, T. Salminen and I. Alakärppä, "Requesting Pervasive Services by Touching RFID Tags", IEEE Pervasive Computing, Vol. 5 (1), pp. 40-46, 2006.