eChallenges e-2009 Conference Proceedings
Paul Cunningham and Miriam Cunningham (Eds)
IIMC International Information Management Corporation, 2009
ISBN: 978-1-905824-13-7

Comparing Two Different Architectures for Pervasive Systems from the Viewpoint of Personalisation

Sarah McBURNEY, Elizabeth PAPADOPOULOU, Nick TAYLOR,
M. Howard WILLIAMS, Yussuf ABU SHAABAN
Heriot-Watt University, Riccarton, Edinburgh, UK
Tel: +44 131 4513430, Fax: + 44 131 4513327,
Email: {ceesmm1, ceeep1, nkt, mhw, ya37}@macs.hw.ac.uk

Abstract: Daidalos is a Sixth Framework integrated research project whose aims include the development of a pervasive system that can provide the appropriate infrastructure to support a wide range of personalised context aware services in a way that is easy for the end-user to manage and use. In the course of the project two slightly different architectures were experimented with and this paper compares the two from the viewpoint of personalisation. The first adopted a simple approach in which all functional modules were treated equally while the second followed a layered approach. The latter resulted in a number of problems, including increased complexity, problems with re-composition and with enforcement of privacy, which are described in the paper. In general the simple non-layered approach proved to be the better of the two.

1. Introduction

In 1991 Weiser [1] predicted that the environment around the user will soon be filled with microscopic devices, mobile or stationary, that will aid the user in his/her everyday life. Since then, with the developments in communications and in technologies such as sensors, motes, specks, etc., this is rapidly becoming a reality. This growth is being accompanied by an even larger expansion in the services available to the user, and the result will soon become unmanageable. This is the problem that pervasive computing [2] seeks to address by developing an intelligent environment to support the user and enable him/her to control and manage this situation [3, 4].

In order to hide the complexity of the underlying system from the user, the system needs to take many decisions on behalf of the user. This can only be done if the system knows the needs and priorities of the user. The latter may be represented as a set of user preferences which will, in general, be context-dependent. The term *personalisation* is used to refer to the set of processes used to create, maintain and apply user preferences in decision making. In pervasive systems the scope of such personalisation is quite wide and extends beyond the usual forms of personalisation, e.g. [5]. However, the most significant problem facing the developers of pervasive systems is the creation and maintenance of an adequate set of user preferences for each user.

Early pervasive system developments relied on manual input of user preferences rather than using machine learning techniques to build up these preferences. However, it was soon realized that some form of assistance was needed in gathering and maintaining user preferences. The Adaptive House [6], GAIA [7] and MavHome [8] have all made personalisation an important goal in order to produce effective and acceptable pervasive systems. In these systems gathering preference information is achieved by monitoring the

user's behaviour and using machine learning techniques to identify new user preferences. The Synapse project [9] also makes use of machine learning techniques but combines this with user control to provide more accurate personalisation.

The above systems all employ machine learning techniques that are offline algorithms, which means that there may be a significant delay before the system adapts to changing preferences. On the other hand the pervasive systems developed by projects such as Ubisec [10], Spice [11] and Mobilife [12] make use of online learning algorithms that can respond rapidly to changes in the user's behaviour patterns and update the set of user preferences in real time.

The Sixth Framework integrated project Daidalos[13] aimed to develop an infrastructure that is able to integrate a range of heterogeneous networks and devices and to create a pervasive service platform on top of this, which protects the user from the complexity of the underlying infrastructure while providing personalised and context aware services with minimal user intervention.

The research was divided into two phases with slightly different objectives. In the first phase an architecture was developed using a bottom-up approach and used a simple approach to user preferences without automatic capture. This was implemented and demonstrated using a challenging demonstrator. The second phase built on the experience gained, and, using a top-down scenario-driven approach, it aimed to design a slightly different architecture and implement it, to produce a more challenging demonstration. This included a combination of online and offline learning algorithms together with different approaches to handling user preferences to improve overall effectiveness.

This paper compares the two different approaches from the viewpoint of user preferences and personalisation.

2. Objectives

One of the main objectives within Daidalos was to develop a pervasive system that would provide the necessary functionality to support the mobile user, taking decisions on his/her behalf wherever practical so as to minimise user intervention. As mentioned the research was divided into two phases with slightly different objectives resulting in two different architectures and two different prototype implementations.

The objective of this paper is to compare these two different architectures in the two phases from the viewpoint of personalisation.

3. Methodology

The approach used within Daidalos was based on a service oriented approach. In the first phase a simple architecture was investigated for the pervasive system, leading on to the design and implementation of a prototype. The idea behind this was to make a set of assumptions and to design and build a prototype fairly rapidly so that these could be demonstrated and evaluated in the context of a fairly complex demonstrator. In the second phase the aim was to improve on this architecture.

A key component of both implementations was personalisation. This is concerned with the acquisition of user preferences and the application of these to support decision making in various parts of the system (e.g. service selection, network selection, privacy and VID selection, personalisation of individual services, etc.). Different strategies were used to capture these (e.g. direct entry through a GUI, stereotypes, and different forms of automatic learning).

4. Developments – the Two Prototypes

The basic functionality contained in the Phase 1 pervasive system included the following:

(1) Service Management, including Service Discovery, Service Selection and Service Composition.
(2) Session Management.
(3) Personalisation.
(4) Context Management.
(5) Security and Privacy.
(6) Rule Management.

The architecture adopted was one involving six main components as shown in Fig. 1 and described in [13].

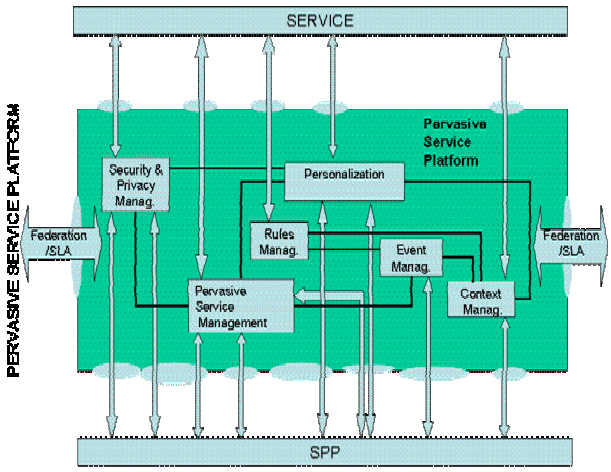

Fig. 1 Basic Architecture of the Pervasive System for Daidalos Phase 1

This architecture was based on a number of assumptions, including the following:
(1) A centralised approach to decision making was adopted, and the Rule Manager was responsible for maintaining and applying the relevant knowledge for different components.
(2) The Personalisation component was responsible for managing and applying user preferences to decisions for both internal services within the pervasive system and external (third party) services.
(3) User preferences were assumed to be supplied by the user via appropriate GUIs provided in the prototype.
(4) The Security and Privacy component adopted a simplistic identity management approach in which the user selected a virtual identity (VID) for the system to use at any stage.

For the prototype implemented in Phase 1, personalisation focused on these areas:
(1) Service selection.
(2) Network selection.
(3) Device selection.
(4) Service personalisation.
(5) Call redirection [14].

By the end of the first phase a prototype had been developed, which was demonstrated using a fairly complex demonstration involving a mobile user in various different situations [15]. From the point of view of personalisation this included the application of user preferences at various points of decision making, including service selection and composition, service personalisation, and call redirection.

However, the prototype had a number of shortcomings, one of which was that the capture of user preferences relied entirely on user input. Since the second phase aimed at improving on this architecture, the following assumptions were made in relation to personalisation.

(1) The management of user preferences was enhanced by the addition of learning. A combination of online and offline learning techniques was used to create new preferences or refine or adapt existing preferences as the user's needs change with time.
(2) The use of stereotypes to create initial preferences was investigated. A stereotype user preference is simply a user preference developed to suit a particular class of users. Thus the user can select the appropriate stereotype and automatically acquire the set of user preferences associated with that stereotype. This is used to provide a starting point which can be refined through the learning processes in (1).
(3) The knowledge formats used to handle user preferences were extended to include Bayesian networks as well as rules.
(4) Identity management was extended in various ways, including the use of user preferences to support privacy policies and automatic VID selection.
(5) The use of user preferences in decisions regarding network selection was extended.

In addition the following general assumptions were made:

(1) The centralised approach to decision taking was replaced by a more distributed one in which decision making occurs in the modules where it is needed. In doing so, rule management was replaced by local knowledge management.
(2) The components were divided into two layers – an essential core layer comprising Service Composition, Event Management and Security and Privacy, and an optional untrusted layer comprising Personalisation (including Learning) and Context Management. It was assumed that core layer components would not depend on so-called optional components.
(3) Pro-active behaviour was to be avoided at all costs, especially any related to personalisation decisions.
(4) There was to be no "hard-wiring" of functionality in the final prototype..

The resulting architecture for phase 2 divides the top-level functions into two groups – nominally referred to as the User Experience Group (or Layer) and the Management Group (or Layer), as shown in Fig. 2.

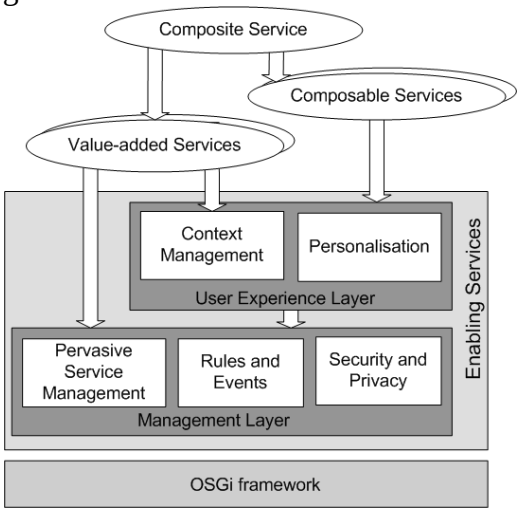

Fig. 2. The Architecture for the Pervasive System in the Second Phase of Daidalos

5. Results – Lessons Learnt

5.1 Effect of Two Layer Approach on Learning and Personalisation

As with other pervasive systems the importance of establishing user preferences was a priority in Daidalos. Although in the first phase simple GUIs were used to set up new user preferences or update existing ones, in the second phase a more elaborate approach was

followed, involving the use of GUIs, stereotypes and both online and offline learning. This combination has the potential to provide significant advantages.

However, the assumption that personalisation (including learning) should be in a separate layer from core system modules (service management and security and privacy) and that the latter should not depend on the former created significant problems. One reason for this was that once control had been handed to a core process it was difficult to return control to the personalisation processes. This made it very difficult to retrieve appropriate information for learning, especially feedback from the user.

This was particularly important for learning preferences for service selection. In this case user preferences are used by personalisation processes to select and rank the best possible services to meet the user's needs. However, once this selection is passed back to the Service Management processes there is no feedback as to what service is finally chosen for the composition (in accordance with the assumption of independence of the core processes). This decision depends on whether services registered are actually available and whether the user intervenes to reject a particular selection for any reason.

Furthermore, this resulted in an increase in complexity of the approach used and the interfaces required to interface with the core components. Initially a clean approach had been adopted and a set of interfaces developed that could be used by all Daidalos Enabling Services and third party services. However, the insistence on independence necessitated a different set of interfaces and approach to be developed for core components. This took considerable additional effort and resulted in an increase in complexity with no obvious benefit.

5.2 Effect on Re-Composition

One of the major aims in Daidalos was that it should compose services in response to a user request and dynamically re-compose these if this becomes necessary. The idea is that as the context of a user changes (for example, as the mobile user moves around) so the user's preferred options may change. This may relate to the choice of network or the particular device used, or even to the service selected. A simple example of this might be that of a telephone call that is switched seamlessly from a user's mobile phone to a car phone when the user gets into his/her car. In such a case the system recognizes that the conditions have changed and that the composition is no longer the best one from the point of view of the user's preferences, and the application can be re-composed to make use of this.

However, this has two aspects to it. The first and most important one is to identify that a re-composition is called for. The second is to perform that re-composition effectively.

Now because of the aforementioned assumptions, two main possibilities were considered for identifying when a re-composition is required:

(1) Program this into the master service for the application. In general, this will be a third party service that may be used by different users with different preferences and, in order to handle this, the master service would need to be able to do two things. Firstly, it must be able to distinguish between a service selection preference outcome and other outcomes such as personalisable parameter outcomes. Secondly, when an identified service selection preference outcome is received, there must be some logic which decides that the outcome translates into the need for a service re-composition. However, this means that considerable additional effort is required on the part of all third party developers to ensure that the master service for every application has the appropriate knowledge for service selection and composition and monitors the appropriate session data in order to be able to take such a decision. This would place an unnecessary burden on the service.

(2) Program this into the user preferences. In this case the user preference rule has to state that if a particular context attribute of the user has a particular value then call for a re-

composition. This clearly violates the assumption about pro-active behaviour on the part of Personalisation. It could also be regarded as a form of hard-wiring, albeit in the user preferences rather than in the main body of code of the pervasive system. since the logic one would expect to decide on this has been replaced by a direct command in a preference rule.

A further disadvantage is that the user must code this preference themselves. There is no means for the user to indicate to the system to re-compose and hence for the Learning module to create a preference rule for this. This would have been possible if a closer integration between Service Management and Personalisation had been permitted.

Eventually the assumptions had to be relaxed and re-composition achieved through the user preferences, with a tighter coupling than originally permitted, thereby allowing a degree of learning. This was demonstrated using a simple example in which the system learnt when the user wanted to use a large screen for displaying her lecture material and automatically recomposed the service to use one when one was available.

5.3 Returning Control After Re-composition

Another problem that arises in re-composition is that of re-starting services after a re-composition. When a composition is set up initially, services are started in the normal way. However, after re-composition has taken place, some services need to be re-started in a different way from the normal start-up. For example, when the MMSPUA is started initially, a call of form “callSetup” is used whereas in some cases when it is re-started after a re-composition “partialSessionTransfer” must additionally be called after normal call setup. Once again one would expect a pervasive system to provide some form of support for these system irregularities.

5.4 Using User Preferences to Enhance Privacy

One of the goals in the development of personalisation in the second phase was to make privacy more user friendly. In the first phase the user selected what virtual identity (VID) should be used by the system; in the second phase we wanted the system to determine the VIDs to be used automatically. This is particularly important where different VIDs are used at different stages in the selection, composition and execution of services.

The simplest approach to automatic VID selection involves associating different VIDs with different services and using a simple set of VID selection preferences to choose the appropriate one to use at any stage. However, a more ambitious approach is based around the access that a service needs to various items of personal data belonging to the user (e.g. name, address, credit card details, current location, etc.) and what items he/she is prepared to let a service access. If a service is permitted to access any such item of data, there may be conditions that the user would impose on its use (e.g. that it is not stored at all, that it is held only for the duration of the current session, that it is not passed to any other user or service, etc.). On the basis of this data and the service itself, a VID is selected to run the service.

Thus when a user requests a service, the system must first establish what data the service wants to access, check what the user will permit and if these do not match, attempt to negotiate an agreement with the service that will satisfy the user. To do this the user’s constraints on the use of data are captured in a Privacy Policy and the process of negotiation is referred to as Privacy Policy Negotiation (PPN). This process is similar to that for trust negotiation [16].

The whole process is fairly complex. In order to simplify things from the point of view of the user, two sets of user preferences were developed. The first set relates to data sharing and is used to generate a Privacy Policy for any particular situation. These user preferences

are context dependent so that the Privacy Policy that is generated will depend on the context of the user. An example of such a preference is:

IF location = 'work' AND day = 'weekday' AND LocalTrustLevel(requestor) >0.5 AND GlobalTrustedReputationLevel(service) > 0.7
THEN PrivacyPolicyRule:
Effect: "allow"
Obligations: 1) Data_Retention_Policy < 12 hours
2) Share information with 3rd parties: NO.

In practice the condition part is likely to be more complex, growing with user input and automatic learning of user behaviour. Evaluating this results in a privacy policy that can be used in negotiation.

In addition to the user preferences for data sharing, there is also a set of user preferences for VID selection after PPN.

Both types of user preference can be set up by the user, either through a GUI or potentially through the use of stereotypes, or they can be established through automatic learning. Maintenance is through learning.

However, one is once again faced with the problem that the privacy components are located in the core and the preference management and learning components in the optional layer with the constraint that no dependency should exist between Security and Privacy and this optional layer. As a result the required functionality had to be copied into the core layer for this purpose, thereby resulting in unnecessary duplication of code.

6. Business Benefits

Although business case models were developed within Daidalos, it is not relevant to link these to the developments covered here. The relevant aspect here is the technology description, which is the focus of the final paper. The lessons learnt from this exercise are being used to guide architectural decisions in the Persist project, which aims to develop a pervasive system based on Personal Self-improving Smart Spaces (PSSs).

7. Conclusions

This paper is concerned with some aspects of the problem of creating a pervasive system as experienced in the Daidalos project where two different architectures were tried and evaluated. The architecture in the second phase was based on the following assumptions:
(1) The components were divided into two layers – an essential core layer comprising Service Composition and Security and Privacy, and an optional untrusted layer comprising Personalisation (including Learning) and Context Management.
(2) The Personalisation components were not permitted to handle any pro-active behaviour.
(3) There was to be no "hard-wiring" of functionality for demonstration purposes.

Some important lessons learnt from this were:
(1) The layered approach increased the complexity of interfaces and code for the Personalisation components and restricted functionality.
(2) The assumptions created significant problems for re-composition and eventually had to be relaxed.
(3) The layered approach resulted in significant replication of code when privacy was extended with user preferences to make it more user-friendly.

These lessons have helped in designing an architecture for another pervasive system focusing on Personal Smart Spaces, called PERSIST.

Acknowledgements

This work was supported in part by the European Commission under the FP6 programme (Daidalos project) which the authors gratefully acknowledge. The authors also wish to thank all colleagues in the Daidalos project without whom this paper would not have been possible. The ideas are also contributing to the PERSIST project, which is being funded by the European Commission under the FP7 programme. Apart from funding these two projects, the European Commission has no responsibility for the content of this paper.

References

[1] M. Weiser, The computer for the 21st century, *Scientific American, 265*(3), 1991, pp. 94-104.
[2] M. Satyanarayanan, Pervasive computing: vision and challenges, *IEEE PCM, 8*(4), 2001, pp. 10-17.
[3] J. Sun, Mobile ad hoc networking: an essential technology for pervasive computing,. In *Proc. Int Conf on Info-tech & Info-net*, Beijing, China, 2001, pp. 316-321.
[4] A. Zaslavsky, Adaptability and interfaces: key to efficient pervasive computing,. In *NSF Workshop on Context-Aware Mobile Database Management*, Providence, Rhode Island, 2002, pp. 24-25.
[5] D. Pacey, E. Dempster, M.H. Williams, A. Cawsey, D. Marwick & L. MacKinnon, A Toolkit for Creating Personalized Presentations. In *Proc. IEEE/WIC International Conf on Web Intelligence*, Halifax, Canada, 2003, IEEE Computer Society Press, pp. 550-553.
[6] M. C. Mozer, Lessons from an Adaptive House. In D. Cook & R. Das (Eds.), Smart Environments: Technologies, protocols and applications, 2004, pp. 273-294.
[7] B. D. Ziebart, D. Roth, R. H. Campbell & A. K. Dey, Learning Automation Policies for Pervasive Computing Environments. In *Proc. 2nd Int. Conf. on Autonomic Computing (ICAC '05)*, 2005, pp. 193-203.
[8] M. G. Youngblood, L. B. Holder & D. J. Cook, Managing Adaptive Versatile Environments. In *Proc. 3rd IEEE Int. Conf. on Pervasive Computing and Communications (PerCom '05)*, 2005, pp. 351-360.
[9] H. K. Y. Si, A Stochastic Approach for Creating Context-Aware Services on Context Histories in Smart Home. In *Proc. ECHISE2005*, Pervasive 2005, 2005, pp. 37-41.
[10] J. Groppe & W. Mueller, Profile Management Technology for Smart Customizations in Private Home Applications. In *Proc 16th Int. Workshop on Database and Expert Systems Applications (DEXA'05)*, 2005, pp. 226-230.
[11] C. Cordier, F. Carrez, H. Van Kranenburg, C. Licciardi, J. Van der Meer, A. Spedalieri, J. P. Le Rouzic & J. Zoric, Addressing the Challenges of Beyond 3G Service Delivery: the SPICE Service Platform. In *Proc. Workshop on Applications and Services in Wireless Networks (ASWN '06)*, 2006.
[12] M. Strutterer, O. Coutand, O. Droegehorn & K. David, Managing and Delivering Context-Dependent User Preferences in Ubiquitous Computing Environments. In *Proc. Int. Symp. on Applications and the Internet Workshops (SAINTW '07)*, 2007.
[13] M. H. Williams, N. K. Taylor, I. Roussaki, P. Robertson, B. Farshchian & K. Doolin, Developing a Pervasive System for a Mobile Environment. In *Proc. eChallenges 2006 – Exploiting the Knowledge Economy*, Barcelona, Spain, 2006, IOS Press, pp. 1695 – 1702.
[14] Y.Yang & M. H. Williams, Adaptation of Content in Personalized Redirection of Communication, *Int. J. Business Data Communications and Networking*, *1*(4), 2005, pp. 51-63.
[15] M. Angermann, S. McBurney, C. Kuhmuench , F. Mahon, J. Mitic, P. Robertson & J. Whitmore, Integrating and Demonstrating Pervasiveness in a Scenario Driven Approach. In *Proc. eChallenges 2006*, Barcelona, Spain, 2006, IOS Press, pp. 1703-1710.
[16] T. Yu, M. Winslett, & K. E. Seamons, Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur. 6*, 1 (2003), pp. 1-42.