Electronic Communications of the EASST
Volume 11 (2008)

Proceedings of the First International DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS 2008)

"InstantSocial" – Implementing a Distributed Mobile Multi-user Application with Adaptation Middleware

Luís Fraga, Svein Hallsteinsen, and Ulrich Scholz

6 Pages

Guest Editors: Romain Rouvoy, Mauro Caporuscio, Michael Wagner
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

"InstantSocial" – Implementing a Distributed Mobile Multi-user Application with Adaptation Middleware

Luís Fraga[1], Svein Hallsteinsen[2], and Ulrich Scholz[3]

[1]MobiComp
Rua do Parque Poente, Lote 37,
Loteamento das Caldas - Sequeira,
4705-629 Braga, Portugal
lfraga@mobicomp.com

[2]SINTEF ICT
7465 Trondheim,
Norway
svein.hallsteinsen@sintef.no

[3]European Media Laboratory GmbH
Schloß-Wolfsbrunnenweg 33
69118 Heidelberg, Germany
scholzuh@eml.org

Abstract: In this position paper we explore how new capabilities of mobile devices could be used to setup distributed multi-user mobile applications with potentially high interest for end users. We describe an example of such an application by transposing Internet social network trends and principles to a mobile ad hoc environment. Then we present a tentative design and implementation sketch of this application in terms of the MUSIC context-aware adaptation middleware we are currently developing.

Keywords: adaptation middleware, service oriented architectures, social networks

1 Introduction

The widespread use of mobile devices combined with their evolving networking and computational capabilities is a continued source of inspiration for novel applications and use cases. Improved capabilities open new possibilities in the mobile world as devices are evolving from being pure service consumers to becoming able to also act as service providers (*e.g.*, mobile devices are now powerful enough to run ports of Internet webservers such as *Apache httpd* [1]). Despite this, there has been a disappointing lack of compelling mobile applications not depending on central servers, even if it seems that most building blocks are in place. In this paper we explore how these new capabilities could facilitate multi-user applications hosted on the mobile devices carried by the users and cooperating through the exchange of services without requiring support from central servers, and we propose a solution relying on self-adaptation and service level negotiation to balance the load between the devices according to their resources. Our approach resembles aspects of mobile grids [2] but differs in its focus on multi-user data-sharing applications; grids instead aim at the transparent execution of arbitrary applications.

The paper starts with the presentation of a motivating example. Then we briefly present the self-adaptation middleware technology on which we build our solution and explain how our solution exploits this technology. As a preliminary validation we provide a walkthrough of how the proposed solution behaves in typical situations that occur during the use of the example application.

2 Motivating Scenario

To explore the idea of mobile multi-user applications providing typically centralized services, we identified what users carry on their mobile devices: phone contacts and media. Adding to this the premise that users are used to Internet content sharing (in social websites such as

Flickr and *YouTube*), implementing a social media sharing platform with mobile devices could prove to be of interest to users while it is also an interesting technical case study. We named this sharing platform "InstantSocial" and it would be perceived by end users as ordinary simple HTTP/HTML website accessible via a PAN or WLAN. Users would access this site with modern, built-in browsers on mobile devices they are already familiar with. Instead of being based on a central Internet server, this site would be served by a composition of services scattered across nearby devices. As more users participate, the platform becomes more robust/redundant and the number of shared content items increases as well as potentially the overall interest for other users. As soon as a critical mass of users leaves, the site would disappear. Thus, both performance and relevance of this system may potentially increase as more users join in. The following scenario exemplifies how a user could experience the capabilities of such distributed mobile multi-user application.

- Paul is visiting a large rock festival. During a Björk concert, he is not able to take a good shot, others could have done better.
- Back at the tent, Paul is willing to share his pictures with others. He starts the InstantSocial application and his PDA notifies him about the presence of a media sharing group. He happily joins, gives high priority to this application, and a moment later his display shows a *selection* of pictures, each representing a collection of shots matching his interests. He browses through the content, selects the ones he likes, and begins to download.
- When Paul is done with the browsing he lowers the priority of the application so he can listen to some MP3. He still leaves his media (and CPU power) available for others to engage in the InstantSocial platform.
- Some songs later, he decides to save some battery for phone calls and indicates his wish to leave the group. The PDA asks him to wait a few seconds. After getting the OK, he quits.

Although this application is simple from a functional point of view, it can also be used in various different contexts such as: conferences, classes, mall communities, traffic jams. All this contexts share the following base requirements:

- As the society is based on a fully distributed architecture where nodes can cooperate using a moderately involved communication scheme (this is unlike the star-based architecture wherein all nodes communicate with a single server), we need mechanisms that allow nodes to discover and compose services hosted on other nodes.
- As applications execute in an evolving environment and are subject to numerous unexpected changes of the execution context, dynamic self-adaptation is a central requirement.
- This society of self-adapting application parts, cooperating through the exchange of services and the negotiation of *Service Level Agreements* (SLA) requires mature *Service-Oriented Architectures* (SOA) composition mechanisms. These SLA negotiations also need to take into account that applications coexist on each participating device with other applications, and that their relative execution priorities inside devices vary dynamically depending on the situation.

3 Introduction to MUSIC

The MUSIC middleware [3,4] provides an execution environment for applications and services which facilitate adaptation to varying context. Based on a model of the adaptation capabilities

and context dependencies of the software entities under its control, it automatically adapts the software to make the best of the current situation. The following features of MUSIC are central in the proposed realization of InstantSocial:

- *Context sensing:* The middleware monitors the relevant context, including user preferences and available resources on the device. It also keeps track of other devices in the vicinity and the services they offer, using standard protocols like SLP and UPnP.
- *Dynamic reconfiguration of applications*: The middleware has an architecture model of the applications running on the device in the form of a composition of cooperating abstract components (roles) which are (re)bound dynamically either to concrete component implementations instantiated locally, or to services available remotely.
- *Adaptation reasoning and decision making:* The adaptation middleware reasons about what is the most suitable configuration in a given situation based on property predictors and utility functions. Property predictors are functions predicting the resource needs and varying properties of the entities they are associated to. Utility functions compare properties (obtained by invoking the appropriate predictor functions) to user needs and resource constraints (obtained from the context sensing subsystem) and computes the utility to the user of a given configuration. When binding to a public service, a service level agreement is established between the provider and the consumer. It is the responsibility of the provider to alert the consumer if the provided service level deviates significantly from the agreed one.

4 InstantSocial design

A partial and simplified MUSIC architecture model for the proposed design is shown in Figure 1. The application is composed of three component types: BP (*browser proxy*), P (*presentation*) and CR (*content repository*). The *content repository* component is responsible to maintain an inventory of available content in all the participating devices and provide access to it. CR instances act both as consumers and providers of the *membership* (ms) service. When a new CR instance is created, it will use the membership service provided by an existing instance to become included in the common distributed content repository, and later it may provide this service to another new instance. CR instances also implement partial replication of content to ensure a certain stability of the federated repository even if participants leave. *Presentation* components monitor the content repository in search for relevant content elements, according to user preferences. They present lists of relevant contents and selected content elements to the BP component in XHTML form through the presentation (pn) service. *Browser proxy* components execute as demons and invoke the built-in browser to present the user interface when InstantSocial is in the foreground.

The architecture model allows three alternative configurations for the application, labeled ISfull, ISmini and ISleech. Examples of possible configurations in compliance with this model are shown in Figure 2. The ISfull configuration instantiates all three components locally and advertises both the presentation service and the membership service for use by other participants. The ISleech configuration only runs the BP component locally and uses a presentation (pn) service provided by another participant. The ISmini configuration instantiates the BP and P components locally, but relies on a content access (ca) service provided by another participant to get access to the content repository. It offers the presentations service for use by others.

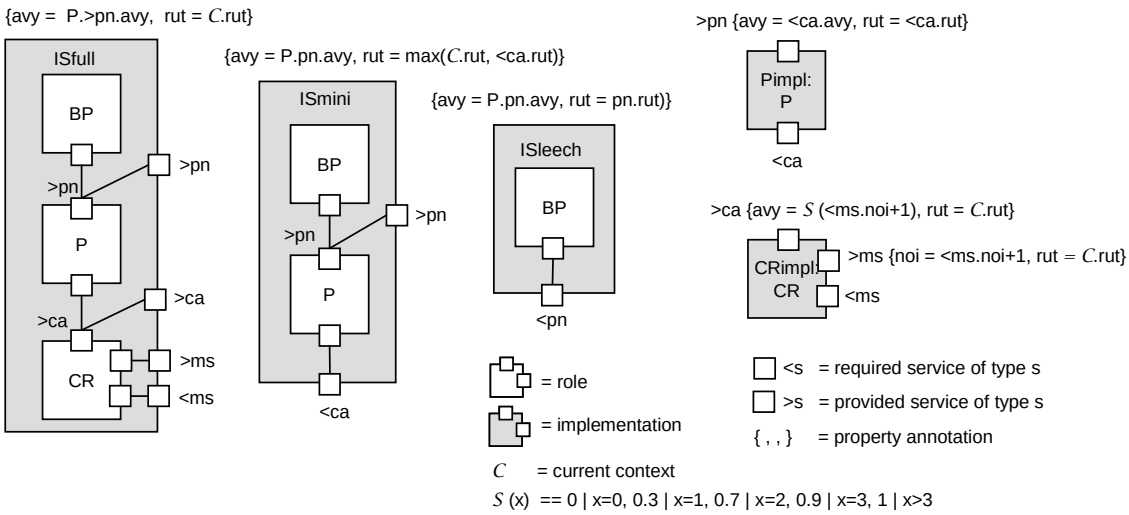

Figure 1. InstantSocial partial architecture model

If the application runs on a device that does not have the resources to run its own content repository and there is no other participant that offers the content access service, there is a fourth possibility not shown in the architecture model. In this case the application is configured with a variant of the *Browser proxy* component which will return to the browser a friendly page informing that there is currently no content repository available.

Please note that we forced the inclusion of the web browser in this design to later explore *mashups* of PAN services with Internet based services and even going from Internet to local websites back and forth seamlessly. For the sake of simplicity, in this paper we do not go into detail on how the presentation component selects relevant contents. In the ongoing implementation, the *Presentation* component has an internal structure with many more variation capabilities, that for example take into account friendship relations and tag/preferences based filtering.

4.1. Adaptation modelling

The annotations in the model provide property predictors needed by the adaptation middleware to reason about what is the most suitable configuration in each situation: There are three properties used in the model, avy (*availability)* which is a measure of the likelihood that content continues to be available, rut (*resource utilisation*), which is a measure of how much of the available resources are utilised, and noi (*number of instances*) which is the number of instances of the content repository component currently constituting the shared repository. Avy and rut are numbers between 1 and 0 while noi is a positive integer. Property predictors, which are associated with ports, predict the properties of the service provided through the port, and property predictors, which are associated with the alternative configurations of the application, predict the properties of the application.

Relevant resources are memory, CPU and network bandwidth. The resource needs of components and compositions must also be specified with property predictors in a similar way, but this is not shown here. Based on the specified resource needs, the middleware computes the resource utilisation on the device and makes it available as context information.

The utility function must be constructed to ensure that a collection of devices each running the self-adapting InstantSocial application will converge towards an overall configuration with sufficient redundancy of components to ensure stable operation when devices leave and a fair distribution of load among the devices taking into account their resources. This is achieved by balancing the availability and the overall resource utilisation as follows,

$$utility = (c_1 * avy + c_2 * (1 - rut)) / (c_1 + c_2)$$

where c_1 and c_2 are constants expressing the relative weights of the availability and the resource utilisation.

An always evolving environment, such as this, demands for the advanced planning mechanism MUSIC provides. Additionally, MUSIC lends itself very well to solving hard problems like these. Note that by declaring an utility function, a developer is instructing the middleware to keep looking for more interesting combinations/compositions of the services that are being made available by nearby devices, with no further need to explicitly hard coding which service composition is better than the other as is usual in other approaches.

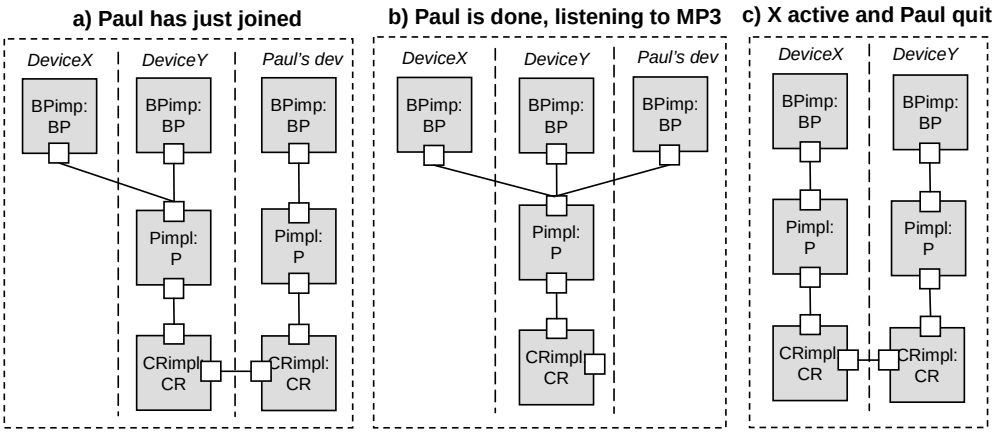

Figure 2 Configurations of InstantSocial

4.2. Informal Demonstration

Assuming the mapping onto the MUSIC technology outlined above, we have achieved a system of collaborating self-adapting peers that forms dynamically, where adaptation is coordinated through the SLA negotiation protocol. Below we try to justify this claim by explaining how the system behaves throughout the scenario presented in Section 2.

1. When Paul launches the application on his device, there is already a virtual website with two other devices *X* and *Y*. *Y* is configured with the ISfull configuration running all components locally, while *X* is a "leech" running only the remote proxy locally and relying on the presentation service of *Y* Paul is not running any other application on his device, so InstantSocial has all resources to itself. In this situation the adaptation middleware configures Paul's device as an ISfull configuration. This configuration gives a higher *availability* and therefore higher utility than other possible configurations, because the content is now replicated on two devices. The situation after Paul has joined is shown in Figure 2a.
2. After Paul has found the pictures he wants and starts listening to music, the priority of InstantSocial drops and it can not use as much memory as before. Therefore the

middleware reconfigures it to the ISleech configuration. As a result, X and Y will notice a drop in *availability*.

3. In the meantime X has completed his resource demanding task and is focusing on Instant Social and therefore has more resources for it. Together with the drop in availability, this causes his adaptation middleware to reconfigure to the ISfull configuration. This brings availability and hence his utility back up again, and now he has enough resources for this configuration.
4. When a user hosting components serving other users leaves the community, other Instant Social platforms detect the disappearance of the service and adapt. Either they find an alternative service provider or instantiate the necessary component locally.

5 Conclusions and future work

In this paper we have introduced "InstantSocial" as an example of a value-added service provided by a distributed, multi-user, mobile application. This application delivers a platform for finding and retrieving shared content in an ad hoc manner and tries to keep user interaction models as familiar and friendly as possible. We designed it using a well known interaction model — *i.e.*, a browser accessing a social website — to make sure we keep differences to other environments at a minimum. Furthermore, its base concept can potentially be applied in a variety of different scenarios ranging from football stadiums to personal home sites.

We have proposed an implementation as a collection of self adapting individual applications collaborating through the exchange of services where each member applications chooses between instantiating components locally or using services provided by another member. The proposed implementation is based on the MUSIC adaptation middleware where this choice is handled by the middleware directed by a utility function constructed to ensure sufficient replication of data for stable operation when members join and leave dynamically, and to fair distribution of load among member devices according to their capabilities.

As future work we plan to realise and evaluate the proposed implementation and to evaluate and compare it with a series of other technologies that could also enable this example application, in particular ongoing efforts on *mobile grid* and the possibility that InstantSocial could be implemented as a presentation service layer on top of pure P2P networks.

Acknowledgements:
This work was funded by the European Commission under the contract number 035166 for the MUSIC project. The authors would like to thank Romain Rouvoy for his helpful comments.

References

1. Mobile Web Server. http://research.nokia.com/research/projects/mobile-web-server/
2. S. Isaiadis and V. Getov, "Evaluation of Dynamic Clustering Architecture for Utilising Mobile Resources", in IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN), Innsbruck, Austria, pp. 117-125, 2008
3. R. Rouvoy, F. Eliassen, J. Floch, S. Hallsteinsen, and E. Stav. "Composing Components and Services using a Planning-based Adaptation Middleware". International Symposium on Software Composition (SC), vol. 4954, pp. 52–67, Springer, LNCS, 2008
4. J. Floch, S. Hallsteinsen, E. Stav, E. Eliassen, K. Lund, and E. Gjørven: Using Architecture Models for Runtime Adaptability. IEEE Software, 23, 2 (2006), 62-70.