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)

Enhancing Planning-Based Adaptation Middleware with Support for Dependability: a Case Study

Romain Rouvoy, Roman Vitenberg and Frank Eliassen

12 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

Enhancing Planning-Based Adaptation Middleware with Support for Dependability: a Case Study

Romain Rouvoy, Roman Vitenberg and Frank Eliassen

Universitetet i Oslo, Department of Informatics,
Networks and Distributed Systems group
P.O.Box 1080 Blindern — 0316 Oslo, Norway
rouvoy@ifi.uio.no, romanvi@ifi.uio.no, frank@ifi.uio.no

Abstract: Recent evolutions of mobile devices have opened up for new opportunities for building advanced mobile applications. In particular, these applications are capable of discovering and exploiting software and hardware resources that are made available in their environment. A possible approach for supporting these ubiquitous interactions consists in adapting the mobile application to reflect the functionalities that are provided by the environment. However, these approaches often fail in offering a sufficient degree of resilience to potential device, network, and software failures, which are particularly frequent in ubiquitous environments. Therefore, the contribution of this paper is to integrate the dependability concern in the process of mobile applications adaptation. In particular, we propose to reflect dependability mechanisms as alternative configurations for a given application. This reflection allows the planning-based adaptation middleware to automatically decide, based on contextual information, to enable the support for dependability or not.

Keywords: Planning-based Adaptation, Component-Based Architecture, Dependability Mechanisms, Self-Adaptation

1 Introduction

The growing interest for highly collaborative applications, such as social networks, and the democratization of mobile devices open up opportunities for new trends of mobile applications. In particular, these applications are able to discover and benefit from software and hardware resources available in their environment. User devices that support such ubiquitous computing systems require continuous software adaptations to face the changes in the user environment (*e.g.*, physical location, network connectivity, energy consumption). However, the state-of-the-art adaptation methods and technologies do not consider possible device, network, and software failures; they may cease being functional in face of such failures [CCKI07].

Although dependability mechanisms provide relevant solutions for building fault-tolerant mobile applications (replication, group communication, etc.) [HHS04], the configuration and the deployment of such mechanisms remains a cumbersome task since it requires to correlate the characteristics of the environment to the attributes of the mechanisms. In a ubiquitous environment, this task becomes critical since the particularities of the environment are not foreseen by definition. Therefore, we propose to enhance an existing adaptation middleware with the capability of selecting, deploying and monitoring dependability mechanisms. In particular, we are

interested in modeling legacy dependability mechanisms in terms of *Quality of Service* (QoS) properties. Thanks to that, we aim at supporting the seamless planning of dependability mechanisms and their respective strategies and thus making easier their integration into mobile applications.

The rest of the paper is organized as follows. Section 2 describes a typical scenario of a social application built atop a planning-based adaptation middleware. It also derives a few salient characteristics of the scenario and illustrates how dependability enhances the application behavior. Section 3 provides background on common adaptation and dependability techniques and solutions. Section 4 outlines our contribution for the seamlessness support of dependable configurations. Section 5 describes and discusses related work, while Section 6 concludes the paper.

2 Motivating Scenario

To further motivate the need for adaptation in dependable computing environments, let us consider the following scenario. Paul recently bought an Internet tablet and installed a version of the *InstantSocial* (IS) application on this mobile device. This application aims at creating virtual communities by instant sharing of multimedia content. In particular, the application offers functionalities, such as picture sharing, instant messaging, and video broadcasting.

Scene 1. The scenario starts with Paul going to a concert of Björk in Paris. Unfortunately, he arrives a bit late and is a bit far away from the stage to see the artists. However, his IS application detects that an ad hoc network is available and that John, who is closer to the stage, broadcasts a video of the performance. The application connects to the broadcasting service advertised by John's mobile device and allows Paul to watch the artists while listening to the music. The video stream is relayed by other mobile devices located in the same place.

Scene 2. The battery of John's device gets depleted so that his device adaptively changes its configuration to prohibit relaying unnecessary data. Fortunately, Paul's IS application detects that Joanne provides a similar video stream, transparently reconnects to Joanne's device, and proceeds with minimal frame loss.

Scene 3. Because Paul really enjoys the concert, he takes some pictures of the event in order to feed his blog when going back home. The photos he takes with the device camera are automatically tagged by the IS applications with his current position using the embedded GPS of the device. The IS application starts looking for other photos taken around him and display them on Paul's device.

Scene 4. Now the concert is over, it's time to go back home and Paul enters the metro station. The IS application detects the network infrastructure provided to the travelers and adapts itself to connect to the available facilities. In particular, the infrastructure provides a service *buddy finder* that helps travelers finding buddies who share common interests. The IS application requests the service to find buddies who share multimedia content related to the concert of Björk that just finished. The service federates multimedia content provided by other devices all around the station and displays it on Paul's device.

Scene 5. The notification service of the metro station broadcasts a message about train delay in the line that Paul was about to take. This announcement causes Paul to take a different transportation path.

Characteristics of the Scenario The scenario we introduce exhibits properties that are representative of nowadays mobile applications:

Moderate to high dynamicity. The application operates in a dynamic environment in which the rate of changes is high and there exists a turnover in the nodes participating in the application. The above scenario illustrates high node mobility and fluctuations in the resource availability.

Large scale. The application may potentially include hundreds and even thousands participating distributed nodes. Realistically, it is quite easy to envision the extension of the above scenario to such scales.

Distribution of nodes and services. The application is based on a fully distributed architecture where nodes can cooperate using a moderately involved communication scheme (as opposed to the star-based architecture wherein all nodes communicate with a single server). Furthermore, we assume that the application knowledge (*e.g.*, the application configurations and available resources) is also distributed among the participating nodes.

Broad diversity of application-required QoS. The application combines many different activities that involve communication between the nodes. We assume that data exchanged between the distributed nodes have different priorities (*e.g.*, logging information [not critical], control information [very critical]), different bandwidth demands (from a few byte long message to intensive multimedia data), and different interaction frequency (*e.g.*, from every 5 seconds to every few days).

Diversity in communication paradigms. The same application may combine different communication types (messaging Vs. streaming) and paradigms (client-server, overlay-based, pub-sub).

Needs for Dependability Although the scenario we exposed proposes usual functionalities of a social application, we foresee that the use of such applications in ubiquitous environments requires the support of dependable mechanisms to ensure an acceptable behavior under volatile connectivity and changing environmental conditions. For example, the entire scenario motivates the need for reliable communication mechanisms. In particular, the video content being delivered by John to Paul should be made highly available to ensure that the quality of the stream will not be critically affected by the number of users watching the video and by the quality of the network. This also means that the content delivery should be made reliable. However, the degree of reliability may depend on a number of factors: type of content, data intensity, update frequency, number of interested consumers, etc. In addition, different contents (photos and videos) may also require varying degrees of freshness and in the case of multi-source updates, policies for conflict resolution (update order errors).

In addition, most of the scenes motivate the need for service discovery. For example, when Paul joins the network infrastructure available in the metro station in Scene 4, its application should discover the services provided by third parties. When fulfilling the requirements of the application, these services should be seamlessly integrated by the mobile device.

Scene 1 emphasizes the demand for dependable cooperation and ad-hoc routing between the devices. It also illustrates the need for reliable synchronization between different streams, such as the audio stream the device receives directly from the technical equipment of the concert settings and the video stream it receives from other cooperating mobile devices. In the general

case, reliable synchronization should be performed between content of different types: video with video, audio with video, text with multimedia, etc.

Scene 2 motivates the need for failure detection, monitoring operational capacity and connectivity, and membership maintenance. These functionalities can obviously prove useful in other scenes as well. Additionally, this scene illustrates the value of transparent takeover mechanisms for the IS application.

In Scene 3, the pictures concert fans take should be replicated for the sake of high-availability and fault-tolerance through redundancy. Since it is impossible to store all data on every device, data-sharing should be attained through partial and content-dependent replication.

3 Foundations

This section introduces the foundations of our contribution by presenting concepts related to planning-based adaptation middleware (cf. Subsection 3.1), and dependability mechanisms (cf. Subsection 3.2).

3.1 Planning-based Adaptation Middleware

Planning-based middleware refers to the capability of adapting an application to changing operating conditions by exploiting knowledge about its composition and *Quality of Service* (QoS) metadata associated to the application components [FHS+06, AHPE07]. We therefore consider applications that are developed with a QoS-aware component model. The QoS model associated with a ubiquitous application defines all the reasoning dimensions used by the planning-based middleware to select and deploy the component implementations that contribute to provide the best utility. The utility of an application grows when its constituting components better fulfill user preferences while optimizing device resource consumption.

For example, in Figure 1, we illustrate the modeling of a self-adaptive logger component using a QoS-aware component model. The configuration of this logger can be optimized dynamically in order to maximize the satisfaction of the user. In particular, the component Logger Composition is made of three components Receiver, Handler, and Storage, which have one, three, and two alternative configurations, respectively.

As we already mentioned, the evaluation of these alternative configuration is driven by the QoS properties associated to the application. For example, Table 1 reflects possible property dimensions for describing the logger QoS. These properties values are then specified and associated to components using notes in the application model. In case of composite components, the value is a function combining the QoS properties of contained components. In case of atomic components, the value usually refers to a primitive value.

Then, *planning* refers to the process of selecting components that make up an application variant that provides the best possible utility to the end user. This process can be triggered during several steps of the application life cycle, such as during the deployment of the application or at runtime if the execution context suddenly changes. The parts of the application that are considered during planning are called *variation points*. These correspond to functionalities (type of behavior) defined in the component frameworks modeling the application. Thus, each vari-

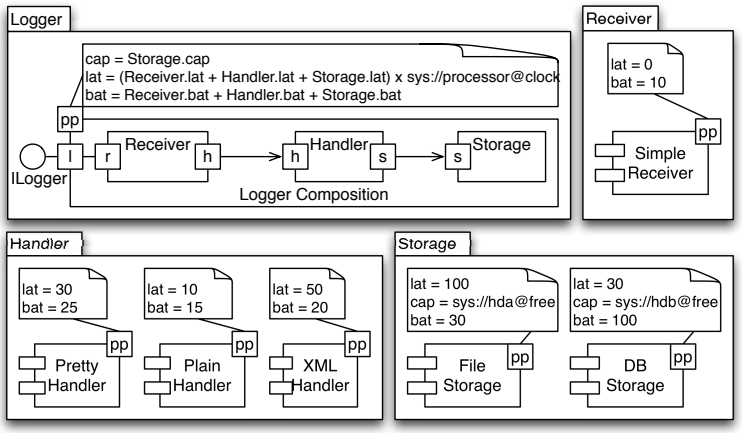

Figure 1: The Model of a Component-based Logger.

Table 1: The Specification of the Logger QoS Property Dimensions.

Property Name	Description	Value Range
cap	Capacity of the logger	0–?
lat	Latency of the logging request	0–100
bat	Battery consumption of the logger	0–100

ation point identifies a functionality of the application that can be implemented differently. In addition, each component implementation suitable for a variation point is reified as a *plan* by the planning-based middleware. A plan mainly consists of a structure that reflects the type of the component implementation and the QoS properties associated to the services it provides. In particular, the plan exhibits both *requested properties* (*e.g.*, memory consumption, network bandwidth, network connectivity) and *offered properties* (*e.g.*, request throughput, latency, result accuracy) referring to the QoS model of the application. To estimate the offered properties of a plan, the planning-based middleware relies on *property predictors*. The property predictors are used to predict the offered properties of a component implementation as a function using the required properties and the current execution context as parameters. The predictors can also take into account the state of the component implementation associated to the plan—*i.e.*, described, deployed, or running—to refine the prediction. The QoS model used by the planning framework can be customized to handle new QoS dimensions (*e.g.*, monetary cost), while the property predictors can be configured to support complex heuristics (*e.g.*, QoS negotiation protocols). The predicted properties are input to a normalized *utility function* that computes the expected utility of a composition of plans making up an alternative application configuration [BHRE07]. For example, the utility function used to evaluate the logger configuration can be configured as the following:

$$Utility(\text{Logger}) = \frac{User_{bat}}{norm(\text{bat})} + \frac{User_{lat}}{norm(\text{lat})} + User_{cap} \times norm(\text{cap})$$

This utility function minimizes the battery consumption and latency of the configuration, while maximizing its capacity. The user preferences ($User_{xxx}$) are used to weight each QoS dimension and ensure that the output will be a normalized value. The function $norm(\text{xxx})$ is used to normalize the property prediction to a scalar value between 0 and 1. The planning-based middleware compares the expected utility of all alternate application configurations, and finally selects the one that provides the highest value.

Figure 2 illustrates the architecture of the MUSIC middleware, which supports the planning-based adaptation. The component Planner supports the *planning procedure* by operating a generic reasoning heuristics that exploits metadata included in the available plans. In particular, the plans are composed based on their type compatibility to describe alternative application configurations. Then, the heuristics ranks the application configurations by evaluating their utility with regards to the application objectives. This evaluation is achieved by computing the offered properties using the property predictors associated to each plan contained in the selected application configuration and retrieved from the component Plan Repository.

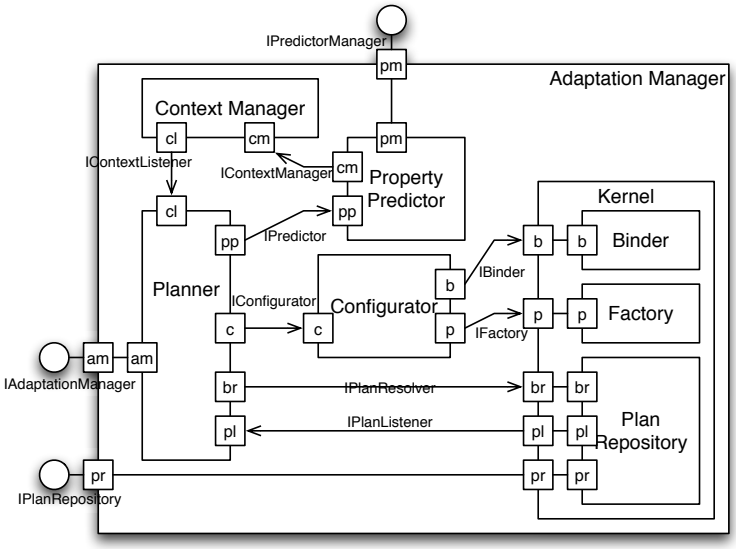

Figure 2: The Architecture of the MUSIC planning-based adaptation middleware.

The component Plan Repository provides an interface IPlanResolver for the Planner to retrieve plans associated with a given component type during planning. The Planner may request plans that are compatible with a given variation point, at which point the Plan Repository will search for matching component types. Any additional metadata on the required component type will help the Plan Repository to exclude plans and filter the search space [LSO+07, BHRE07]. Plans are typically published to (and discarded from) the Plan Repository by applications and component development tools using the interface IPlanRepository, and can thus trigger the Planner for re-planning of the application if needed (*e.g.*, the discarded plan was associated to a running component).

The *reconfiguration process* is handled by the component Configurator and consists of taking the set of plans selected by the component Planner and reconfiguring the application. Before deploying the application configuration selected by the reasoning engine, the component Configurator brings the current application into a quiescence state, by suspending the execution of its contained components. Then, if the component described by a plan is in the running or deployed state, the associated component instance is configured for the variation point and connected to other components using the component Binder. If the component is in the described state, then the component should be preliminary instantiated and deployed by the component Factory using the component implementation description associated to the plan. The result of the reconfiguration (*e.g.*, reference of the deployed instance) is automatically reflected into the selected plans. Thus, the MUSIC planning-based middleware offers a modular and extensible approach for adapting applications built with various types of component models. In particular, the concept of plan can be derived to support heterogeneous artifacts and their associated states. Furthermore, the components Factory and Binder provide sufficient abstractions for supporting different dependability mechanisms (*e.g.*, replication, load-balancing, group communication).

3.2 Dependability Mechanisms

Dependability covers a wide range of mechanisms, and each of these mechanisms exhibit strategies and parameters that usually need to be set up based on application-specific and context-dependent information. For example, we observed that each of the following dependable mechanisms can be configured in different ways.

Replication mechanisms are used to replicate parts of system state in order to ensure fault tolerance and high availability. However, there exist a large variety of replication strategies (*e.g.*, passive/active replication) and each of these strategies supports several configuration parameters (*e.g.*, replication factor, update propagation scheme and frequency, replication degree);

Load-balancing mechanisms rely on different scheduling strategies to dispatch the incoming requests to services that are expected to be highly available. The goal of these mechanisms is to ensure that the system latency is not severely affected by the load. However, scheduling-based approaches can use different heuristics to dispatch the requests (*e.g.*, round-robin, load-aware) and these heuristics behaves differently depending on the type of load introduced into the system;

Group communication mechanisms provide a paradigm for reliable many-to-many communication with advanced properties with regards to reliable delivery, ordering, and synchronization of messages with membership changes. The possible strategies implemented by these mechanisms are related, *e.g.*, to the underlying multicast mechanism employed, buffering and congestion handling, as well as the required synchronization guarantees between delivery of application messages and membership change notifications;

Reliable communication mechanisms support techniques for ensuring reliable message delivery. Such mechanisms are especially important for mobile networks and have to support dy-

namic reconfiguration to react to unstable situations. An example is Mobile IP, which supports network handovers. Other examples in this category include message redundancy and concurrent paths. All these mechanisms provide different quality of service with respect to reliability and costs and their configuration can vary depending on the environmental conditions.

Transaction mechanisms ensure the consistency of system states. Each transaction must succeed or fail as a complete unit; it cannot remain in an intermediate state. However, the performance of the services supporting the transactions can be impacted by the evolution of the execution environment [RSM06]. Thus, in a ubiquitous environment, the configuration of transaction properties (*e.g.*, commit protocol, locking strategy) can be planned by the adaptation middleware depending on the context information.

Discovery mechanisms provide a support for detecting new operational nodes in the system. Once again, these services are influenced by different strategies that are related to the type of the underlying network (*e.g.*, infrastructure, ad hoc) as well as the supported technology (Web Services, OSGi, etc.). Besides, some protocols define lookup strategy (proactive or reactive) as well as lookup frequency. Due to the dynamic nature of the environment, these strategies and their related parameters may require to be reconfigured on the fly. For example, the lookup frequency can be increased when in an indoor environment depending on the available battery of the mobile device.

4 Dynamic Planning of Dependability

This section exposes our first elements for supporting dependability using a planning-based adaptation middleware. In particular, we propose to reflect the characteristics dependable mechanisms using plans (cf. Subsection 4.1), and to use composite components to model dependable configurations (cf. Subsection 4.2).

4.1 Specification of Quality of Dependability

It is important to observe that for each of these mechanisms, ensuring high dependability of service compositions implies that the planning heuristics take the corresponding dependability parameters into account. This also ought to be reflected in the QoS model, which should include new QoS dimensions (number of faults, cost, etc.) and incorporate them into the utility function.

While plans were initially designed to reflect components making up an application, we demonstrated in [REF+08] that plans can also reflect other kind of artifacts, such as *Service-Oriented Architectures* (SOA). In particular, we proposed a component-based architecture for discovering, negotiating, and planning application configurations supporting SOA. More generally, we observed that plans provide a suitable abstraction for reasoning on quality of service independently of underlying artifacts. Based on this assumption, we believe that plans can also be use to reflect dependability concerns. In particular, plans and the enclosed QoS properties can be reused to model a dependable mechanism and its characteristics. To attain this goal, we intend to characterize existing dependable mechanisms (*e.g.*, active/passive replication, group

communication, resilient routing protocols at the level of both networking and application overlay networks) in terms of QoS properties. For example, Table 2 identifies the QoS model that can be used to reflect the QoS of a replication mechanism, namely the *factor*, *degree*, *frequency*, and *reliability* of replication. As this QoS model is an extension, default values are provided (using the square brackets) if the property is not mentioned explicitly.

Table 2: The Specification of the Replication QoS Property Dimensions.

Property Name	Description	Value Range
fac	Factor of replication	0–? [0]
deg	Degree of replication	(None) 0–1 (Full) [0]
fre	Frequency of replication	(Never) 0–1 (Always) [0]
rel	Reliability of replication	(Weak) 0–? (Strong) [0]

4.2 Modeling of Dependable Configurations

Once such QoS properties are specified, the role associated with a dependability mechanism can be combined with other application roles in the architecture using composition plans (cf. Figure 3). In particular, the QoS properties can be associated to the configuration either *implicitly* or *explicitly* by the modeling approach.

The implicit modeling approach consists in integrating the dependability QoS properties in the property predictors associated to the application QoS properties. This approach reflects seamlessly the impact of the dependability mechanism on existing QoS properties (*e.g.*, if employed under specific circumstances, the active replication mechanism increases the logger capacity by 10% and reduces its throughput by 15%). For example, the configuration depicted in Figure 3 extends the one presented in Figure 1 by introducing a new configuration Dependable Store that supports the replication of the logger storage component. The type Store is connected to the type Replicate using stereotype dependable to specify that the connector is not a component binding. The QoS properties associated to this dependable configuration use the properties of the dependable mechanisms to reflect how the dependable mechanism impact the QoS of the configuration. Then, available replication mechanism strategies are described as alternative component implementations for the type Replicate, while their respective parameters are reflected as QoS properties.

The explicit modeling approach consists in integrating the dependability QoS properties in the function used to compute the utility of the application configuration. For example, the utility function below extends the one defined in Subsection 3.1 and specifies that the reliability of replication should be enforced:

$$Utility(\text{Logger}) = User_{cap} \times norm(\text{cap}) + \frac{User_{bat}}{norm(\text{bat})} + \frac{User_{lat}}{norm(\text{lat})} + User_{rel} \times norm(\text{rel})$$

This approach enables the user to specify its preferences with regards to the reliability concern (reflected by the user preference $User_{rel}$).

Thanks to this abstraction, the planning-based adaptation middleware is able to reason about the most suitable configuration to deploy in the current context. In particular, the reasoning

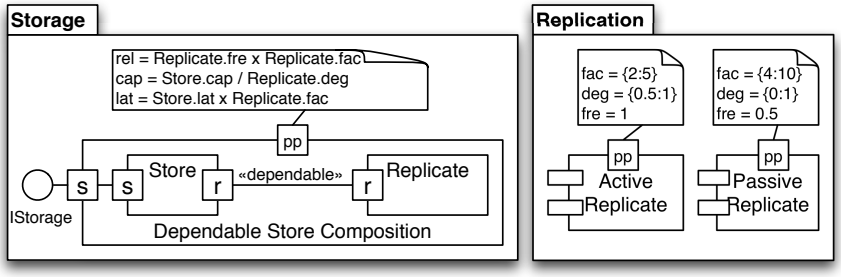

Figure 3: The Model of a Dependable Storage Configuration.

process will decide whether the replication mechanism needs to be enabled or not, thus avoiding potential loop feedback problems. If the replication is needed, then it will determine which strategy has to be selected and the best fitting value for its different attributes. This means that, at runtime, the planning-based adaptation middleware can reconfigure the dependable mechanism to tune the replication attributes or change the strategy.

At the middleware level, supporting these dependable mechanisms requires to extend the architecture of the component Adaptation Manager (depicted in Figure 2) in order to interpret dependable configurations. This implies that the alternative implementation of components Factory and Binder have to be developed to support the configuration and the deployment of a dependable mechanisms, respectively. Thus, for each dependable mechanism supported by the MUSIC middleware, a pair of Factory and Binder components is integrated into the architecture and connected to the component Configurator in order to instantiate the dependable mechanism and configure the selected strategy, respectively.

5 Related Work

A vast body of work has been devoted to dependability techniques, such as replication and group communication [Bir05] as well as transactions [GR93, BHG87]. However, most of the research on dependability did not target adaptive and autonomous systems: most existing technologies require manual configuration and assume that the universe of nodes is static. Furthermore, scalability is known to be a general issue for dependability mechanisms.

Only a handful of works consider adapting dependability mechanisms by taking the context into account. [KIBW99] and [FSS03] consider adaptive fault-tolerance for component-based models. According to their claims, componentization may facilitate implementation of schemes for fault-tolerance and load-balancing.

[HHS04] presents a *Quality of Service* (QoS) architecture that allows flexible combinations of dependability attributes such as reliability, timeliness, and security to be enforced on a per-service request basis. In addition to components that implement the underlying dependability techniques, the architecture includes policy components that evaluate a request's requirements and dynamically determine an appropriate execution strategy.

The AQuA project [RBC+03] has developed an architecture that allows distributed applica-

tions to request and obtain a desired level of availability using the *Quality Objects* (QuO) framework. The architecture includes a dependability manager that attempts to meet the requested availability levels by configuring the system in response to outside requests and changes in system resources due to faults.

While the research has been focusing on developing dependable techniques [Bir05, GR93, BHG87] and making them adaptable [KIBW99, FSS03, RBC+03], the contribution introduced in this paper proposes an flexible adaptation framework for selecting, configuring, and monitoring dependable mechanisms. The selection process is driven by the correlation of the QoS properties reflected by a given dependable mechanism with the current context information monitored by the associated context middleware. This approach allows the developer to specify the dependable mechanisms that can be integrated into the application as configuration alternatives, while the adaptation framework decides dynamically which configuration should be applied in a given execution context.

6 Conclusion & Perspectives

This paper introduced the motivations for generalizing the support of dependable mechanisms in mobile applications. In particular, the expansion of ubiquitous environments and the growing role of mobile devices open up new opportunities for these mobile applications. Although these new applications are capable of reconfiguring themselves to face changes in their environment, they are still missing missing dependable features to reasonably tolerate possible device, network, and software failures.

Therefore, we proposed to extend an existing planning-based adaptation middleware with the capacity of reasoning about dependable configurations. By reflecting dependability as a new QoS dimension, the developer can model application configuration that are resilient to faults. These configurations are selected and configured automatically by the planning-based adaptation middleware depending on the current execution context of the user.

As a matter of perspectives, we plan to experiment the support of various state-of-the-art dependable mechanisms in order to identify a comprehensive QoS model dedicated to dependability issues. This support will be realized as part of the MUSIC project. The framework will be validated using real world pilot applications of the industrial partners of the MUSIC project (http://www.ist-music.eu).

Acknowledgements: The authors thank the partners of the MUSIC project and reviewers of the CAMPUS workshop for their valuable comments. The authors would also like to thank Luís Fraga from MobiComp for suggesting the InstantSocial scenario. This work is partly funded by the European Commission through the project MUSIC (EU IST 035166).

Bibliography

[AHPE07] M. Alia, S. O. Hallsteinsen, N. Paspallis, F. Eliassen. Managing Distributed Adaptation of Mobile Applications. In *7th IFIP WG 6.1 Int'l Conference on Distributed Applications and In-*

teroperable Systems (DAIS). Lecture Notes in Computer Science 4531, pp. 104–118. Springer, Paphos, Cyprus, June 2007.

[BHG87] P. A. Bernstein, V. Hadzilacos, N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA, 1987.

[BHRE07] G. Brataas, S. O. Hallsteinsen, R. Rouvoy, F. Eliassen. Scalability of Decision Models for Dynamic Product Lines. In *Int'l SPLC Workshop on Dynamic Software Product Line (DSPL)*. P. 10. Kyoto, Japan, Sept. 2007.

[Bir05] K. P. Birman. *Reliable Distributed Systems: Technologies, Web Services, and Applications*. Springer, Secaucus, NJ, USA, Mar. 2005.

[CCKI07] M. Cinque, D. Cotroneo, Z. Kalbarczyk, R. K. Iyer. How Do Mobile Phones Fail? A Failure Data Analysis of Symbian OS Smart Phones. In *37th Annual IEEE/IFIP Int'l Conference on Dependable Systems and Networks (DSN)*. Pp. 585–594. IEEE Computer Society, Edinburgh, UK, June 2007.

[FHS+06] J. Floch, S. O. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, E. Gjørven. Using Architecture Models for Runtime Adaptability. *IEEE Software* 23(2):62–70, Mar./Apr. 2006.

[FSS03] F. Favarim, F. Siqueira, J. da Silva Fraga. Adaptive Fault-Tolerant CORBA Components. In *Workshops of the Int'l Middleware Conference*. Pp. 144–148. PUC-Rio, 2003.

[GR93] J. Gray, A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufman Publishers, 1993.

[HHS04] J. He, M. A. Hiltunen, R. D. Schlichting. Customizing Dependability Attributes for Mobile Service Platforms. In *34th Int'l Conference on Dependable Systems and Networks (DSN)*. Pp. 617–626. IEEE Computer Society, Florence, Italy, June/July 2004.

[KIBW99] Z. T. Kalbarczyk, R. K. Iyer, S. Bagchi, K. Whisnant. Chameleon: A Software Infrastructure for Adaptive Fault Tolerance. *IEEE Transactions on Parallel and Distributed Systems* 10(6):560–579, 1999.

[LSO+07] S. A. Lundesgaard, A. Solberg, J. Oldevik, R. B. France, J. Ø. Aagedal, F. Eliassen. Construction and Execution of Adaptable Applications Using an Aspect-Oriented and Model Driven Approach. In *7th IFIP WG 6.1 Int'l Conference on Distributed Applications and Interoperable Systems (DAIS)*. Lecture Notes in Computer Science 4531, pp. 76–89. Springer, Paphos, Cyprus, June 2007.

[RBC+03] J. Ren, D. E. Bakken, T. Courtney, M. Cukier, D. A. Karr, P. Rubel, C. Sabnis, W. H. Sanders, R. E. Schantz, M. Seri. AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects. *IEEE Transactions on Computers* 52(1):31–50, 2003.

[REF+08] R. Rouvoy, F. Eliassen, J. Floch, S. Hallsteinsen, E. Stav. Composing Components and Services using a Planning-based Adaptation Middleware. In *7th Int'l Symposium on Software Composition (SC)*. Lecture Notes in Computer Science 4954, pp. 52–67. Springer, Budapest, Hungary, Mar. 2008.

[RSM06] R. Rouvoy, P. Serrano-Alvarado, P. Merle. Towards Context-Aware Transaction Services. In *6th IFIP WG 6.1 Int'l Conference on Distributed Applications and Interoperable Systems (DAIS)*. Lecture Notes in Computer Science 4025, pp. 272–288. Springer, Bologna, Italy, June 2006.