Components and Aspects Composition Planning for Ubiquitous Adaptive Services

Mourad Alia, Mikaël Beauvois, Yann Davin, Romain Rouvoy, Frank Eliassen

Components and Aspects Composition Planning for Ubiquitous Adaptive Services

Mourad Alia
Department of Informatics,
University of Oslo,
P.O.Box 1080 Blindern,
N-0316 Oslo
Email: mouradal@simula.no

Mikaël Beauvois, Yann Davin
Department of Informatics,
University of Oslo,
P.O.Box 1080 Blindern,
N-0316 Oslo
Email: beauvois, davin@ifi.uio.no

Romain Rouvoy
ADAM project-team
University of Lille 1 – LIFL,
UMR USTL/CNRS 8022,
INRIA Lille – Nord Europe,
F-59650 Villeneuve d'Ascq
Email: romain.rouvoy@inria.fr

Frank Eliassen
Department of Informatics,
University of Oslo,
P.O.Box 1080 Blindern,
N-0316 Oslo
Email: frank@ifi.uio.no

***Abstract*—In ubiquitous environments, resources limitations and fluctuations combined with device mobility requires the dynamic adaptation of mobile applications. This paper reports on an extension of the MUSIC adaptation middleware to support aspect-oriented programming in order to handle cross-cutting adaptations. Basically, this extension specifies an architectural model for defining applications as a composition of aspects and components. The dynamic adaptation of an application in a given context is realised by selecting the appropriate component and aspect implementations using utility functions as a mean of optimising the overall QoS. Our approach and middleware are implemented and tested on top of OSGi framework.**

I. Introduction

The development of mobile and ubiquitous applications raises problems inherent to the heterogeneity and the dynamicity properties that characterizes ubiquitous environments. These challenges have generated the need to consider the dynamic adaptation of the application triggered by different context changes such as resources limitations and fluctuations, mobility and user preferences. Within this setting, the MUSIC project [1], [2], [3] provides a general architecture-based approach for the development and management of self-adaptive applications following the separation of concerns principles. The adaptation reasoning is based on contextual component composition and selection using utility functions as a means to find an optimal configuration to better satisfy mobile users.

The purpose of this paper is to allow the MUSIC middleware to support self-adaptation of cross-cutting concerns such as security, transactions, logging and dependability. This support is often intrusive and requires the developer of the ubiquitous application to merge dynamically both technical and business concerns into the application code. One solution to face this problem is to develop several versions of the application and configures them depending on the adaptation concern requirements. this solution does not scale when the number of concerns increases, meaning that the combination of C concerns for an application typically implies the development of 2^C realizations of this application. Furthermore, adding such concerns into the applications usually introduce trade offs between Quality of Service (QoS) dimensions, such as security level and response time [4] that should be taken into account.

In this setting, we propose a modular approach for dynamically supporting the integration of cross-cutting concerns into self-adaptive ubiquitous applications. This approach combines the strengths of Planning-based Adaptation Middleware [1], [2] and Aspect-Oriented Programming (AOP) [5], [6], [7] for leveraging the development of self-adaptive ubiquitous applications. In particular, AOP offers a modular approach for controlling the effects of code tangling and scattering in an architecture [7] and proposes to isolate crosscutting concerns as aspects, which are woven into different parts of the architecture.

Merging aspects with software components is not a new idea and has been adopted in many projects. However, our approach is different from the others in the sense that our variability model is able to dynamically plan and to select aspects configurations [8], [9], [10], [11], [12]. In our previous contribution, we have introduced initial ideas on how AOP could be of benefit to MUSIC [13] and particularly in case of dependability concern. This paper is a generalisation and a validation of these proposals where the contribution is twofold.

First, we present an adaptation model allowing the selection of the optimal composition of aspects and components implementations depending on context changes. Cross-cutting concerns and their alternative realisations are represented as aspects by isolating them from the business concerns. The integrations of these concerns mechanisms are modeled as specific component compositions, while the alternative configurations are discriminated in terms of QoS properties.

Second, we propose a planning-based middleware that embodies this model and provides low level AOP weaving as an additional mechanisms. This is realised by extending the architecture of the MUSIC middleware independently from a particular AOP languages, such as AspectJ [5] and JAC [14]. To assess this model, the middleware is implemented and tested on top of the OSGi framework.

The remainder of the paper is organized as follows. Section II introduces the MUSIC adaptation approach and its concepts. Section III extends the MUSIC models with AOP

support. The implementation of our proposal are explained in section IV before concluding.

II. MUSIC Planning-based Adaptation Approach

The MUSIC project follows an architectural methodology and provides a middleware solution providing support for self-adaptation and context-awareness. An application is designed as a component framework, which defines the functionalities of the application. These components are deployed on resource-constrained nodes.

An application is represented as a *Plan* describing its internal structure at run-time within the middleware. A *Plan* defines a set of *QoS Predictors* which are functions evaluating the QoS properties and resource needs expressed by *QoS Dimensions*. The utility function is defined with these *QoS Predictors*. A *Plan* describes either a *Composite component Realization*, which is a composition of component Realizations or an *Atomic component Realization*. Constraints are predicates over the properties of the constituting components of a composition, which restricts the possible combinations of component realizations. In the rest of the paper, this is referred to the *variability model*.

Planning is the process determining the optimal configuration of component realizations by optimizing the utility function, which is a way of evaluating the overall QoS provided by the application. This process is triggered when the application starts or at run-time when the execution context changes. The adaptation middleware iterates over the plans associated to a given application. For each plan, it resolves the dependencies of the plan and evaluates the utility function according to the current context by computing the QoS Predictors. The utility function is expressed as a weighted sum of dimensional utility functions where the weights express user preferences. A dimensional utility function measures user satisfaction in one property dimension.

III. Aspects and Components Planning Model

The main contribution of this paper is to extend the MUSIC approach with AOP. This contributes to enrich the middleware adaptation expressiveness to handle cross-cutting concerns, such as security, logging, and transactions. In the following, both the MUSIC adaptation model and the middleware architecture are revisited and extended with aspect support.

A. Aspect and Component Variability Model

The challenge with the extension of the variability model to support aspects is twofold. First, the developer should be able to express architectures as a composition of aspects and components. Second, the middleware should be able to reason on such hybrid compositions in order to contextually select the optimal configuration.

Figure 1 depicts the extended adaptation model, which support aspects and components composition. Basically, aspects are associated to components when they are woven. An aspect is considered as any MUSIC component characterized by provided and required properties. The subtle difference with

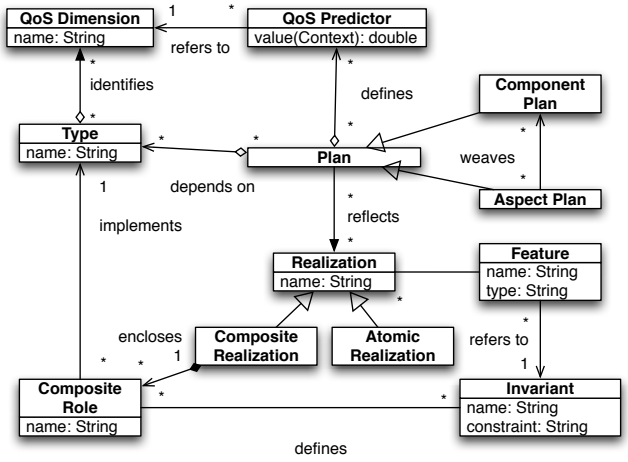

Fig. 1. Variability Model with Aspects Support.

the MUSIC component model is the notion of binding, which is not explicit in the case of aspects since the code is woven directly into components.

A plan could be either of component or aspect type. Like a MUSIC component, an aspect is described by a typed plan. Both aspects and components properties are exploited by the *QoS Predictor* function to derive the aggregated properties values. The realization of an aspect plan is either a *Composite Aspect Plan* or an *Atomic Aspect Plan*.

An *Atomic Aspect Plan* is an aspect description that has to be woven into a component implementation referred to as an *Atomic Component Plan*. *Atomic Aspect Plans* are usually used when there is only one aspect to be woven.

A *Composite Aspect Plan* defines a set of organized aspects. It refers to the different aspects individual implementations and a possible set of rules that describes the behavior of the weaving process of such aspects. A simple example of such rules is the order in which aspects are executed. Note that weaving many aspects into the same application is not trivial since woven aspects could potentially modify the initial join points [15]. *Composite Aspect Plans* are used when many aspects have to be woven into the same component such as weaving logging and security aspects into the same application.

Finally, as components, architectural constraints—or invariants—can be defined on *Aspect Plans*.

B. Aspect Planning Middleware Architecture

Figure 2 revisits the middleware architecture and introduces aspects components. The main requirement when extending the architecture is to be independent of a particular load time AOP technology. Therefore, we have basically added a new composite called *Aspect Manager*. This component reflects an abstraction of the main services provided by the underlying AOP technology such as AspectJ [5] and JAC [6]. It is composed of three components namely *Aspect Plan Repository*, *Aspect Weaver*, and *Aspect Factory*.

Configuration	Weaving time
Component without weaving	$30ms$
Component with AES-128 weaving	$1,300ms$

TABLE I
ASPECT WEAVING VS. COMPONENT INSTANTIATION PERFORMANCE

The *Aspect Plan Repository* is used to store and to get *aspect Plans* (*i.e.*, aspects implementations). It is called by the *Adaptation Controller* through the *Template Builder* during the planning process in order to get the different aspect implementations. As explained in the previous section, the introduction of aspects does not change the planning process and the existing planning algorithms are still valid.

Once a given plan is selected, the *Configuration Executor* uses the *Aspect Factory* to weave the aspect plans into the components implementations. In case of composite aspect plans, the *Configuration Executor* checks through the associated rules before weaving. The *Aspect Factory* uses the *Aspect Weaver* to manage the aspect weaving process. Typically, the *Aspect Factory* loads component classes and aspects implementations, weaves component classes and aspects, and finally instantiates the generated classes.

IV. IMPLEMENTATION & VALIDATION

To validate our approach, we proceed in two steps. Firstly, we have extended the MUSIC middleware so that it supports aspects planning and weaving. Then, we have implemented and tested a security-based adaptive application on top of the middleware. Because of the lack of place, the secuity-based application is not decribed in this short paper.

The middleware is implemented as a component-based framework using the OSGi technology with respect to the architecture presented in Figure 2. More precisely, the middleware is implemented with Knopflerfish OSGi [16] framework. To support aspects, we use AJDT AspectJ OSGi bundle as provided by the Eclipse project [17]. AspectJ provides the most common aspect programming model. It also supports dynamic aspect weaving, which is stressed by the need to implement the dynamic adaptation. Regarding security, we have used the algorithms library provided by the Java security framework [18]. This framework is architected as a set of providers of security concerns that are basically algorithms and key generators. The framework is supported by JavaME as well as JavaSE.

We have extended the previous version of the middleware by implementing the *Aspect Manager* component (see section III-B). The middleware is then able to reason on AspectJ aspects and MUSIC components compositions and to weave the selected aspects accordingly.

The weaving performance depends on the length of the class byte code. For the same class byte code length, we can see from [19] that AspectJ gives roughly the same performance as the MUSIC middleware as depicted by the first line of table I. This shows that the integration of AspectJ into the MUSIC middleware does not present substantial overhead.

From table I, one can also see that loading and weaving a class with aspect is at least 10 times longer compared to simple component instantiation. More precisely, It is around $1,500ms$ to weave and load the main component class vs $60ms$ for a simple class loading.

A. Analysis

This section discusses the complexity and the cost of integrating AOP into our planning-based adaptation middleware.

1) Modeling Complexity: : As already mentioned, the integration of AOP principles and dependability mechanisms does not impact our variability model (see section III-A). Therefore, the application designer can reuse the same tools and models to describe the structure and the variability of its application. By modeling aspect-oriented configurations, the designer reduces the complexity of the application models since cross-cutting concerns are not tangled and scattered within the components realizations, but clearly isolated as composite realizations. The complexity we introduce can be related to the description of aspect QoS. However, the added complexity is relatively small since the aspect QoS prediction is expressed in a similar fashion to the component QoS prediction. Therefore, the application designer can describe the contribution of an aspect realization as she does for any kind of component realisation.

2) Deployment Cost: : During the reconfiguration process, the selected configuration is compared to the deployed one and the differences are scheduled to implement the reconfiguration. In the case of a load-time weaving, the replacement is implemented as the creation of a new component instance and the migration of the associated internal state. However, one have seen from our experiment that this loading time introduces a non negligible overhead to the adaptation process. This has to be taken into account in case of time-sensitive adaptations.

3) Reasoning Complexity: : The support of aspects does not increase the complexity of the reasoning process and the selection problem stills combinatorial. To overcome this issue, our adaptation reasoner implements several optimizations in order to control the reasoning complexity and reduce the adaptation cost. Many algorithms and heuristics have been proposed varying from Bruteforce, Greedy to more evaluated algorithms [20].

V. CONCLUSION

This paper has introduced the extension of an existing planning-based adaptation middleware for the support of Aspect-Oriented Programming (AOP) principles. Our adaptation middleware benefits from AOP by dynamically planning aspects. This limitation is resolved by modeling and planning separately the business concerns, the aspect policies, and the cross-cutting concerns. Furthermore, AOP benefits from our planning-based adaptation middleware by supporting the dynamic selection and configuration of the aspect policies depending on contextual information variation. The middleware architecture has been extended with minimal modifications independently from a particular AOP technology. Our approach is implemented and tested on top of the OSGi framework.

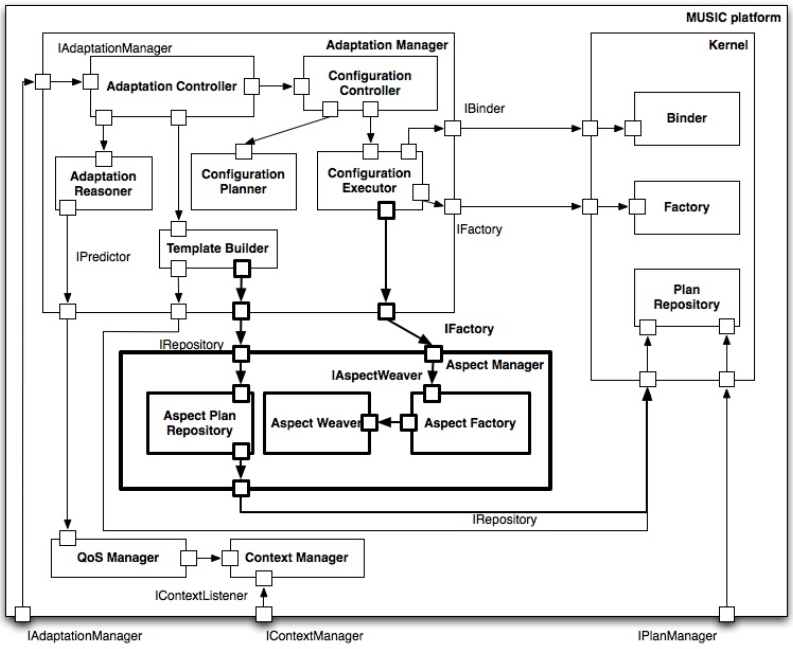

Fig. 2. Extended Middleware Architecture.

VI. Acknowledgements

We would like to thank our partners of the MUSIC project for valuable comments. This work was partly funded by the European Commission through the project MUSIC (EU IST 035166). involved partners.

References

[1] The MUSIC project, "http://www.ist-music.eu."

[2] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven, "Using architecture models for runtime adaptability," *IEEE Softw.*, vol. 23, no. 2, pp. 62–70, 2006.

[3] M. Alia, S. Hallsteinsen, N. Paspallis, and F. Eliassen, "Managing distributed adaptation of mobile applications," in *7th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)*, ser. LNCS 4531, K. R. Jadwiga Indulska, Ed. Springer Verlag, 2007, pp. 104–118.

[4] M. Alia and M. Lacoste, "A qos and security adaptation model for autonomic pervasive systems," in *COMPSAC '08: Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 943–948.

[5] The AspectJ project, "http://www.eclipse.org/aspectj/."

[6] JAC - A framework for Aspect-Oriented Programming in Java, "http://jac.ow2.org/."

[7] G. Kiczales and M. Mezini, "Aspect-oriented programming and modular reasoning," in *ICSE '05: Proceedings of the 27th international conference on Software engineering*. New York, NY, USA: ACM, 2005, pp. 49–58.

[8] P. K. Sharma, J. P. Loyall, G. T. Heineman, R. E. Schantz, R. Shapiro, and G. Duzan, "Component-Based Dynamic QoS Adaptations in Distributed Real-Time and Embedded Systems," in *6th Int. OTM Conference on Distributed Objects and Applications (DOA)*, ser. LNCS, vol. 3291. Larnaca, Cyprus: Springer, Oct. 2004, pp. 1208–1224.

[9] G. Söldner, R. Kapitza, and S. Schober, "Dynamic weaving of components in a distributed environment," in *Companion '08: Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*. New York, NY, USA: ACM, 2008, pp. 95–96.

[10] N. Pessemier, L. Seinturier, T. Coupaye, and L. Duchien, "A Model for Developing Component-Based and Aspect-Oriented Systems," in *5th Int. Symposium on Software Composition (SC)*, ser. LNCS, vol. 4089. Vienna, Austria: Springer, Mar. 2006, pp. 259–274.

[11] P. Grace, B. Lagaisse, E. Truyen, and W. Joosen, "A Reflective Framework for Fine-Grained Adaptation of Aspect-Oriented Compositions," in *7th Int. Symposium on Software Composition (SC)*, ser. LNCS, vol. 4954. Budapest, Hungary: Springer, Mar. 2008, pp. 215–230.

[12] S. A. Lundesgaard, A. Solberg, J. Oldevik, R. B. France, J. Ø. Aagedal, and F. Eliassen, "Construction and Execution of Adaptable Applications Using an Aspect-Oriented and Model Driven Approach," in *7th IFIP WG 6.1 Int. Conference on Distributed Applications and Interoperable Systems (DAIS)*, ser. LNCS, vol. 4531. Paphos, Cyprus: Springer, Jun. 2007, pp. 76–89.

[13] R. Rouvoy, F. Eliassen, and M. Beauvois, "Dynamic planning and weaving of dependability concerns for self-adaptive ubiquitous services," in *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*. New York, NY, USA: ACM, 2009, pp. 1021–1028.

[14] R. Pawlak, L. Seinturier, L. Duchien, and G. Florin, "JAC: A Flexible Framework for AOP in Java," in *3rd Int. Conference on Metalevel Architectures and Separation of Crosscutting Concerns (Reflection)*, ser. LNCS, vol. 2192. Kyoto, Japan: Springer, Sep. 2001, pp. 1–24.

[15] R. Lopez-Herrejon, D. Batory, and C. Lengauer, "A disciplined approach to aspect composition," in *PEPM '06: Proceedings of the 2006 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*. New York, NY, USA: ACM, 2006, pp. 68–77.

[16] Knopflerfish - The open source OSGi Service platform, "http://www.knopflerfish.org/."

[17] AJDT: AspectJ Development Tools - The Eclipse AspectJ plugin, "http://www.eclipse.org/ajdt."

[18] T. J. S. Framework, "http://java.sun.com/javase/technologies/security/."

[19] E. Hilsdale and J. Hugunin, "Advice weaving in aspectj," in *AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development*. New York, NY, USA: ACM, 2004, pp. 26–35.

[20] G. Brataas and S. Hallsteinsen, "Scalability of decision models for dynamic product lines," 2007.