Dynamic Planning and Weaving of Dependability Concerns for Self-Adaptive Ubiquitous Services

Romain Rouvoy
University of Oslo,
Dept. of Informatics
P.O.Box 1080 Blindern,
0316 Oslo, Norway
rouvoy@ifi.uio.no

Frank Eliassen
University of Oslo,
Dept. of Informatics
P.O.Box 1080 Blindern,
0316 Oslo, Norway
frank@ifi.uio.no

Mikaël Beauvois
University of Oslo,
Dept. of Informatics
P.O.Box 1080 Blindern,
0316 Oslo, Norway
beauvois@ifi.uio.no

ABSTRACT
Ubiquitous computing and service-oriented computing enable the development of a new trend of applications that can opportunistically interact with services discovered in the surrounding landscape. Although sporadic, this type of interaction requires the deployment of dependable mechanisms to ensure the correct completion of the interactions. However, the integration and configuration of these mechanisms depends not only on the type of service accessed, but also on the surrounding environment. Such a variability requires an extensive effort of the developers to support the alternative mechanisms. Thus, to reduce this effort, we propose to integrate the Aspect-Oriented Programming (AOP) principles into the MUSIC planning-based adaptation middleware in order to dynamically plan and weave dependability concerns into the application depending on the execution context. In particular, this paper introduces our continuous support for AOP, which includes i) a uniform model for describing the dependable application configurations and ii) a modular middleware platform for weaving and configuring the dependability concerns when necessary.

Categories and Subject Descriptors
D.2.13 [Software Engineering]: Reusable Software

Keywords
Component-Based Software Engineering, Aspect-oriented Programming, Planning-based Adaptation, Quality of Service

1. INTRODUCTION
The emergence of service-oriented computing has offered users the opportunity to consume services without knowledge of, expertise with, nor control over the technology infrastructure that supports them. On the other side, ubiquitous computing provides users with a new generation of mobile applications capable of adapting their behavior depending on changes observed in their execution environment. An example of combination of these technologies is self-adaptive mobile applications, which are able to exploit the network connectivity in order to dynamically discover and interact with services available in the surroundings [21]. Supporting such remote interactions requires dependability mechanisms, such as reliable communications, transactions or replication, to ensure the interaction’s reliability. However, this support is often intrusive and requires the developer of the ubiquitous application to merge both technical and business concerns into the application code, even if these remote interactions are sporadic. Moreover, the configuration of the dependability mechanisms are often subjects to trade-offs between Quality of Service (QoS) dimensions, such as consistency, performance, and availability in the case of replication mechanisms [2, 4, 6, 27]. Finally, the configuration of the dependability mechanisms can also be impacted by context changes, e.g. from using an optimistic commit protocol (2PC-PC) to applying a pessimistic one (2PC-PA) when the commit rate decreases [24].

To overcome these problems, the developer usually has to develop several versions of the application services and configures these services depending on the dependability requirements. However, this solution does not scale when the number of concerns increases, meaning that the combination of \( C \) concerns for a service typically implies the development of \( 2^C \) realizations of this service.

In this paper, we propose a modular approach for dynamically supporting the integration of dependability concerns into self-adaptive ubiquitous applications. This approach combines the strengths of Planning-based Adaptation Middleware [3, 8, 16], Service-Oriented Architectures (SOA) [17] and Aspect-Oriented Programming (AOP) [14, 15] 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 [14] and proposes to isolate cross-cutting concerns as aspects, which are woven into different parts of the architecture. Thus, we propose to reflect dependability concerns and their alternative realizations as aspects, which can be dynamically woven into the ubiquitous application depending on context changes. By adopting aspects, the developer is able to isolate the realization of dependability concerns from the business concerns. The integrations of the dependability mechanisms are modeled as specific service compositions, while the alternative configurations are discriminated in terms of QoS properties. These models are exploited by the planning-based adaptation middleware we developed in the context of the MUSIC project [3, 23], which is able to select the composition of services and aspects that satisfies the current execution context and the device capabilities. Then, the selected aspects are woven into the application service to automatically produce new service realizations, thus shifting the development effort from \( 2^C \) services to 1 service and \( C \) aspects.

The remainder of this paper is organized as follows. We first
introduce the foundations of this work, namely the planning-based adaptation middleware (cf. section 2). Then, we present the extension of this planning-based approach to support AOP principles (cf. section 3), as well as its implementation (cf. section 4). Additionally, we illustrate our proposal on a case study demonstrating the integration of transaction and replication concerns (cf. section 5) and we discuss the complexity and the cost of our proposal (cf. section 6). Finally, related works are discussed (cf. section 7) before concluding (cf. section 8).

2. BACKGROUND ON PLANNING-BASED ADAPTATION

Planning-based adaptation refers to the reconfiguration capability of an application to changing operating conditions by exploiting knowledge about its composition and Quality of Service (QoS) metadata associated to its constituting services [3, 8, 16]. We therefore consider applications that are developed with a QoS-aware service model. The QoS model associated with a ubiquitous application defines all the reasoning dimensions used by the planning-based adaptation middleware to select and deploy the service implementations that contribute to provide the best utility. The utility of an application grows when its constituting services better fulfill user preferences while optimizing device resource consumption. Figure 1 describes the QoS-aware service model we use to describe self-adaptive applications [23]. This model is composed of two parts: a structural model and a reflection model. On the one hand, the structural model describes the structure of a service-oriented application and provides a similar abstraction to the Service Component Architecture (SCA) assembly model [17]. In particular, a Service Type describes not only the service capabilities, but also a set of QoS Dimensions. Provided and required service types, which are causally connected, are grouped as a Service Port. A set of Service Ports can be implemented by an Atomic Realization, a Composite Realization (both deployed locally on demand), or a Service Instance (available remotely). A composite realization encloses a set of stereotyped Roles and Connectors, which implement and bind service ports, respectively. Finally, service roles can define Invariants which, combined with Features associated to service realizations, are used to define constraints on the architecture [13] (e.g., configuration inconsistencies). Additionally, invariants can be declared using the Object Constraint Language (OCL). Thus, inspired by the service-oriented computing principles, our approach considers Everything as a Service (EaaS) and is therefore open to integrate other kinds of services, such as aspects isolating crosscutting concerns (e.g., demarcation or replication policies) as well as the mechanisms supporting them (e.g., transaction or replication services).

On the other hand, the reflection model isolates the adaptation capabilities of an application. In particular, each Service Plan reflects an alternative service realization and its potential explicit dependencies upon other service types (relationship depends on) as well as its implicit dependency upon the hosting platform (e.g., run-time environment type and version), which is reflected by the service realization stereotype. Then, planning refers to the process of selecting the services that make up an application configuration providing 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 run-time if the execution context suddenly changes. When such an adaptation process is triggered for a particular service type, the planning middleware iterates over the service plans associated to the service type. For each service plan, it resolves the plan dependencies—i.e., the required service types and the hosting platform—and invokes the QoS Predictors for evaluating the suitability of the configuration to the current execution context. The predicted QoS properties are input to a normalized utility function that computes the expected utility of a combination of service plans making up an alternative application configuration [3, 8]. Thanks to this approach, the QoS model used by the planning framework can be customized to handle various QoS dimensions (e.g., performance, monetary cost), while the QoS predictors can implement advanced heuristics (e.g., negotiation protocols).

3. DYNAMIC ASPECT PLANNING

The approach we propose hereby consists in exploiting the service meta-model introduced in Figure 1 for modeling aspect-oriented configurations of services. This integration is achieved by i) reflecting an aspect as a service, and ii) reasoning on the aspect QoS to perform dynamic adaptations.

Aspects Modelling. Figure 2 depicts a modeling example of an aspect-oriented configuration. In practice, we reflect an aspect as a composite role A marked by the stereotype aspect and combined with the application service roles B and C using connectors stereotyped as weave. These associations produce new service realizations integrating the aspect as a policy for using a dependability mechanism reflected by the service role D. This interaction is declared by the binding of the aspect role A to a service role D, which describes that the aspect weaving implies the insertion of a service port D into the service roles B and C. Thus, in a similar way to [19], we enforce separation of concerns by isolating the business logic from the aspect—i.e., the policy to insert into the service—and from the mechanism, which implement the dependability logic. This means that various combinations can be considered for the realizations of roles A and D, which are described as service realizations of service ports pd and pd, respectively.

Aspects Reasoning. The reasoning process consists in selecting and configuring the aspect realizations providing the best utility to the end-user. Since we reflect aspects as services, the reason-
4. IMPLEMENTATION ISSUES

Figure 3 illustrates the architecture of the Adaptation Manager, which supports the planning-based adaptation middleware in order to score the alternative aspect and service realizations. Therefore, the integration of aspects and the associated dependability mechanisms within the reasoning process can be achieved implicitly or explicitly. The implicit reasoning consists in integrating the dependability mechanism and aspect QoS dimensions in the existing QoS predictors associated to the service. This approach seamlessly reflects the impact of aspect weaving on existing QoS dimensions (e.g., a transaction demarcation aspect increases the service reliability by 10%, but reduces its throughput by 15%). As a consequence, the aspect-based configurations can be compared to other configurations without updating the utility function of the application. The explicit reasoning consists in integrating new QoS dimensions in the utility function of the application. These QoS dimensions refer to the QoS model associated to the crosscutting concern (e.g., security, dependability). This approach enables the user to be aware of dependability concerns and possibly to specify its preferences (reflected by the user preference $U_{sec, dep}$), thus influencing the reasoning process and the resulting adaptation configuration.

Thanks to these abstractions, our planning-based adaptation middleware is able to reason about the most suitable configuration to deploy in the current execution context. In particular, the reasoning process will decide whether a dependability mechanism needs to be enabled or not. If it is needed, then the reasoning heuristics will determine which aspect realization has to be selected as well as the best fitting configuration for the mechanism. This also means that, at runtime, the planning-based adaptation middleware can reconfigure the woven aspect(s) and the associated mechanism(s) to tune the weaving and the integration strategies.
form services to satisfy the dependencies. For example, if the Adaptation Manager does not provide a Service Factory and a Service Binder supporting aspect weaving, but these platform services are available as service plans, then the Configuration Executor will be able to deploy the suitable Service Factory and Binder for weaving the selected aspects.

Service plans are typically published to (and discarded from) the Plan Repository by i) applications, ii) service development tools using the interface IPPlanManager, or iii) other plan repositories available remotely, and can thus trigger the planner for re-planning of the application if needed (e.g., the discarded plan was associated to a service instance). The adaptation of a service can also be triggered by context changes detected by the Context Manager [18].

Our current implementation of the adaptation manager is based on the OSGi technology. This choice is motivated by the modularity and the dynamicity of the OSGi model, while existing implementations, such as Conciere [20], exhibit a reasonable memory footprint for resource-constraint devices (80 kB). Furthermore, some existing aspect-oriented frameworks, such as AspectJ [14], are now distributed as OSGi bundles, thus providing support for load-time aspect weaving into OSGI services (e.g., Equinox Aspects [15]). Thus, the implementation of our approach consists in wrapping such an aspect-oriented framework as a Service Factory and a Service Binder in order to make them available to the Adaptation Manager.

5. AOP SUPPORT FOR DEPENDABILITY

This section illustrates the continuous AOP support in our planning-based adaptation middleware. This support includes models for aspect-oriented application configurations as well as middleware services enabling the weaving of aspects into the application services. Thus, we illustrate our approach on the InstantSocial application [9]. This application delivers a distributed platform for finding and retrieving shared multimedia content in an ad hoc manner and tries to keep user interaction models as familiar and friendly as possible. In particular, Figure 4 focuses on the modeling of dependable alternative configurations for the service Content Manager. This service is responsible for storing the content shared via the InstantSocial application.

**Modeling Support.** Based on the description of the composition Content Manager, we define a dependable configuration Transactional Content Processor, which extends the service Content Processor with transactional demarcation by weaving an aspect Demarcation Policy into the port cp. This demarcation policy interacts with the Transaction Service for suspending and/or initiating transactions. Possible demarcation policy realizations applied by legacy application servers are Supports, RequiresNew, Required, NotSupported, Mandatory, and Never, while alternative realizations for the transaction service can be the deployment of an embedded transaction service or the interaction with a service already available remotely (cf. Transaction Service Realizations). Additionally, we declare an OCL invariant to specify that only one instance of Transaction Service should be used by the application.

Our modeling approach supports also the expression of aspect-weaving order as illustrated in the composition Transactional Content Repository. The dependencies described between the aspects Logging Policy, Locking Policy, and Resource Handler specify that these aspects should be woven into the port cp.

---

2. The features demonstrated complement those presented in section 3.
of the service Content Repository in the respective order. Then, according to the invariant dependability defined by the Content Manager, this configuration is tagged with a feature of type transaction to impose that this configuration should be combined with Content Processor configurations supporting transactions.

Finally, the composition Replicated Content Repository provides an alternative configuration of the service Content Repository that enables the automatic replication of the content repository. The alternative realizations for the Replication Service are based on group communication or peer-to-peer technologies [4], while the aspect replication policy supports the definition of strong or optimistic replication heuristics [27]. In the case of file-based replication, the availability property can be reflected using the QoS predictor defined in [2]: availability = 1 – (1 – µH)\(^c\) where µH refers to the mean host availability and c represents the number of available replicas (provided by the context middleware). Given that no invariant constrains the combination of the configurations Transactional Content Repository and Replicated Content Repository, these compositions can be combined in an arbitrary way enabling a transactional replication process or not. This decision is driven by the QoS adequacy of each configuration to the current execution context.

**Reasoning Support.** Since our service meta-model is not impacted by the integration of AOP concepts, the planning heuristics we defined for the selection and the configuration of services can be also applied for the selection and the configuration of aspects. Thus, the resolution of configuration alternatives computes all the possible configurations for the applications and filters out the compositions that do not satisfy the architecture invariants (e.g., the transactional content repository and the transaction service do not use the same commit protocol) [13]. Then, the planning heuristics computes and ranks the configurations that maximize the end-user satisfaction. The satisfaction criterion are defined by a utility function, which specifies the priorities for the QoS dimensions associated to the application:

\[
Utility(User_{rel}, Bat_{avail}) = \begin{cases} 
0.0 & \text{if } Bat_{avail} \leq \text{battery} \\
User_{rel} \times \text{reliability} & \text{battery-latency} 
\end{cases}
\]

For example, the utility function defined above, minimize the application latency as well as its battery consumption, while trying to maximizing its reliability. However, when observing the predictors defined in Figure 4, we can notice that the resolution of this problem is not straightforward, but may require to trade-off configurations depending on the execution context. Information related to the execution context is retrieved by the QoS manager via the context manager (here net-lat refers to network latency, while commit-rate refers to application-specific information about the current commit rate) [18]. As an example, although the local component 2PC Transaction Service provides better latency than the remote service 2PC-PA Transaction Service, the opposite applies concerning the battery consumption. On the other side, the aspect 2PC-PA Logging offers a better level of reliability than 2PC Logging. Finally, the latency of the 2PC-PA Transaction Service grows when the commit rate increases, while it is the opposite when considering the 2PC-PA Transaction Service. Thus, the planning heuristics flatten this multi-dimensional problem into a constraint satisfaction problem whose solving is driven by the utility function. Figure 5 visualizes the utility of valid transactional configurations depending on variations of the network latency and the commit rate. Although illustrating only two QoS dimensions, Figure 5 shows the complexity of selecting the best fitting configuration. This decision becomes even more difficult when taking into account additional QoS dimension (e.g., battery consumption), additional service realizations (e.g., local 2PC-PC transaction service), or additional crosscutting concerns (e.g., replication).

![Figure 5: Utility of transactional configurations.](image_url)

**6. DISCUSSIONS**

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

**Deployment Support.** Once the planning process is completed, the application should be reconfigured dynamically to fit the new configuration. This step is realized by the configuration executor service, which deploys each service plan making up the new configuration. The service factory abstraction enables the configuration executor to deploy heterogeneous artifacts [18, 25, 23]. In particular, service factories are available in order to instantiate, import, or weave the described component, service, or aspect, respectively. For example, when the commit rate decreases, the current service 2PC-PC Transaction Service can be replaced by 2PC-PA Transaction Service in coordination with the replacement of the aspect 2PC-PC Logging by 2PC-PA Logging. Later on, if the network becomes unavailable, then the configuration will shift to a configuration based on a local 2PC commit protocol, while the replication mechanism will be automatically disabled.

**Modeling Complexity.** As already mentioned, the integration of AOP principles and dependability mechanisms does not impact our service meta-model. Therefore, the application designer can reuse the same tools and models to describe the structure and the variability of its application. Indeed, aspects are wrapped as services, while the associated pointcuts are reflected as connectors (joinpoints are associated to service ports). By modeling aspect-oriented configurations, the designer reduces the complexity of the application models since crosscutting concerns are not tangled and scattered within the service 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 service QoS prediction. Therefore, the application designer can describe the contribution of an aspect realization as she does for any kind of service realization.

**Reasoning Complexity.** Furthermore, the support of aspects does not increase the complexity of the reasoning process. Nevertheless, our adaptation reasoner implements several optimizations in order
to control the reasoning complexity and reduce the adaptation cost. First, the reasoning process is performed as a background activity that does not suspend the execution of the application. Therefore, applications are only suspended if a reconfiguration needs to be operated. Then, the context plugins triggering the reasoning process are based on Fuzzy Logic [28]. Specifically, fuzzy sets support the description of context situations representing strong changes in the execution environment and thus requiring an adaptation (e.g., network connectivity changes from weak to strong), an improvement over the context regions of QuO [7]. Inspired by hysteresis systems, this approach stabilizes the adaptation process by avoiding systematic reasoning whenever the execution context changes. When initiated, the adaptation process exploits the invariants defined by the application models and the service plan filtering mechanisms provided by the middleware to implement a branch and bound algorithm that discards drastically the unfeasible alternative configurations, as demonstrated in [3, 13]. As part of the evaluation of the feasible configurations, the QoS predicted for each service realization are cached by the adaptation reasoner, thus avoiding extensive computations. Finally, the utility of the selected configuration is also converted into a fuzzy set in order to initiate the reconfiguration process only if a drastic QoS improvement can be perceived by the end-user.

Deployment Cost. During the reconfiguration process, the selected configuration is compared to the deployed one and the differences are scheduled for replacement. Then, we use the service factory and the service binder to implement specific replacement semantics. In particular, we use these abstractions to support the unwrapping of an aspect if this aspect has been woven at run-time. In the case of a load-time weaving, the replacement is implemented as the creation of a new service instance and the migration of the associated internal state. Similarly, depending on the dependability mechanism considered, the configuration parameters can be tuned without stopping the mechanism, thus avoiding an expensive reconfiguration process, while ensuring the service continuity.

Therefore, we believe that the non-intrusive integration of AOP principles and the implementation of severable optimizations within our planning-based adaptation middleware provide an efficient solution for the integration of dependability concerns within self-adaptive ubiquitous applications.

7. RELATED WORK

Dynamic Aspect Weaving. AspectOpenCom [10], Fractal Aspect Component (FAC) [19], and Aspect-Oriented Component Infrastructure (AOCI) [11] are three approaches that aim at reconciling CBSE and AOP. These approaches provide the low-level mechanisms for defining aspects and weaving them into components. However, the support for the automatic weaving of aspects is restricted to Event-Condition-Action (ECA) rules. Depending on context variation, the developer can describe the run-time weaving of a given aspect into a component. Therefore, these approaches i) are limited to interface interception mechanisms ii) are specific to an aspect technology, and iii) do not support continuous QoS optimization and tradeoffs.

Aspects & Quality of Service. Quality Objects (QuO) [7] provides a middleware platform supporting the execution of distributed critical applications. In particular, QuO provides the ability to specify, monitor, and control aspects of the QoS in an application, and to adapt to changing levels of QoS at runtime. A major component of the QuO framework is a set of Quality Description Languages (QDL) for specifying possible QoS states, the system resources and mechanisms for measuring and providing QoS, and behavior for adapting to changes in QoS. QDL descriptions describe QoS policies and behaviors to apply whenever the system moves from a context region to another. Although QuO maximizes the application QoS depending changes in the execution context, it is not capable of operating coordinated reconfiguration of crosscutting concerns and business services as our approach proposes. Furthermore, the rule-based definition of context regions in QuO can result in unstable behaviors when the context is close to the region boundaries, contrary to our approach based on fuzzy regions and hysteresis transitions.

Planning-based Adaptation. QUAMobile [16] and Mobility and ADaptation enABling Middleware (MADAM) [11] use CBSE and mirror-based reflection for modeling functional and non-functional concerns. In these approaches, the integration of crosscutting concerns usually leads to a combinatorial explosion of configuration alternatives. Nevertheless, QUAMOBILE uses aspect-oriented modeling techniques to separate the QoS model from the application logic, enabling the definition of method call interceptors via the concept of connectors. Although these connectors are suitable for implementing some crosscutting concerns (e.g., encrypted communications), their use is restricted to design-time and thus do not support run-time AOP mechanisms, such as AspectJ [14]. Our approach provides a continuous integration of AOP principles from design-time to run-time, including the support for different AOP weaving frameworks.

Architecture & Dependability. Several architectural approaches for dependable systems have been proposed in the literature [5, 12]. Although these approaches reflect the dependability concerns (exception handling) within the application architecture to tolerate erroneous behaviors, they offer a statically defined support for dependability. The approach we propose provides a more generalized support for adaptive dependability and crosscutting concerns based on architecture reconfigurations.

Adaptive Dependability. Context-Aware Transaction Service (CATE) [24] is a component-based transaction services that supports the dynamic reconfiguration of the 2PC protocol. This reconfiguration is driven by the objective of minimizing the overall latency of the service depending on the monitored commit rate. However, as this decision heuristics is embedded within CATE, it makes difficult i) to synchronize the selection of the 2PC protocol with other concerns and ii) to include additional QoS dimensions in the decision process, such as the network latency. Similarly, several replication mechanisms, such as Total Recall [2], MEADS [6], AQuA [22], or MDFIT [26], emphasize the need for balancing the QoS properties of replication mechanisms depending on the environmental conditions. They also claim that the task of availability management should be automated to leverage the system manager, who is not able to properly manage the trade-offs involved. However, the proposed adaptive mechanisms remain isolated within the respective dependability frameworks, therefore limiting the coordination of the replication mechanism configurations with other concerns involved in the application. The approach we propose consists in externalizing the adaptation logic from the mechanism in order to describe and operate the adaptation of both application and dependability layers in a coordinated fashion.

8. 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 ben-
efits from AOP by controlling the combinatorial explosion of service realizations needed to be developed when a growing number of crosscutting concerns is taken into account during the development of an application. This limitation is resolved by modeling and planning separately the business concerns, the aspect policies, and the crosscutting 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. More specifically, we apply this approach to the integration of dependability concerns into service-oriented applications. The configuration of these dependability mechanisms often requires to trade-off QoS dimensions depending on the application requirements and the execution context. By externalizing the decision-making process from the dependability mechanism, we can organize the adaptation of the whole system in a coordinated fashion. In particular, we demonstrate the benefits of our approach on the integration of transaction and replication concerns into the InstantSocial ubiquitous application.

9. REFERENCES


