Dynamic Aspect Weaving using a Planning-based Adaptation Middleware

Romain Rouvoy, Mikaël Beauvois and Frank Eliassen
University of Oslo, Dept of Informatics,
P.O. Box 1080 Blindern, N-0316 Oslo
rouvoy@ifi.uio.no, beauvois@ifi.uio.no, frank@ifi.uio.no

ABSTRACT
The growing complexity of applications has enforced the need for separation concern and modularity. These requirements become critical when considering self-adaptive mobile applications, which are capable of optimizing their behavior depending on their context of execution. In our context, the crosscutting concerns of a component-based application are managed by an adaptation middleware. This planning-based adaptation middleware tries to maximize the user satisfaction while minimizing the adaptation cost and the resource consumption. However, the integration of crosscutting concerns (e.g., logging, security) usually leads to a combinatorial explosion of the alternative implementations. Therefore, we propose to integrate aspect-oriented programming principles in our planning-based adaptation middleware to leverage the development and the integration of crosscutting concerns. Thanks to a uniform modeling approach, our adaptation middleware is able to automatically select and configure the aspect(s) to weave into a base component for improving its quality of service.

Keywords
Aspect-oriented Programming, Planning-based Adaptation, Quality of Service

1. INTRODUCTION
Component-Based Software Engineering (CBSE) has been acknowledged as a fruitful technology for implementing mobile applications executing in ubiquitous environments [1]. In particular, the use of CBSE allows the developer to separate application concerns into different components, which can be reused separately to make up new applications. Recently, CBSE has been combined with adaptation mechanisms to support the dynamic selection and configuration of the components that make up an application depending on context information [1, 3, 9]. For example, user devices that support ubiquitous computing systems require continuous software adaptations to face the changes in the user environment (e.g., physical location, network connectivity, energy consumption) [12]. However, the growing complexity of these applications and the increasing need for separation of concerns raise new challenges for the development of such applications. In particular, the combination of possible concerns \( C \) that applies to a given component type usually implies the development of \( 2^C \) component alternatives. Although this number can be reduced by identifying meaningless components combining conflictual concerns, the number of components to consider remains a limiting factor for the developer.

To overcome this limitation, we propose to integrate the principles of Aspect-Oriented Programming (AOP) [8] in the development of software components. By adopting aspects, the developer is able to isolate the implementation of identified concerns from the base component. Then, aspects are combined to the base component to automatically produce new component implementations, thus shifting the development effort to 1 component and \( C \) aspects. In addition, the isolated aspects can, in some cases, be reused and combined with different component implementations.

While the combination of aspects and components has already been addressed in several approaches [4, 11], no solution is actually provided for deciding when and which aspect(s) need to be woven into a base component. Thus, we propose to extend the low-level mechanisms developed in these approaches with the capability to dynamically select and configure the aspect(s) that have to be woven, depending on execution environment. Therefore, the contribution of this paper is to introduce the principles of supporting aspects using a planning-based middleware. To achieve this goal, we identify the Quality of Service (QoS) dimensions of an aspect and we model its impact on the QoS of a base component. Based on this QoS specification, the planning-based middleware is able to select the combination of component implementation and aspects providing the best QoS to the user depending on his current context.

The remainder of this paper is organized as follows. We first introduce the foundations of this work, namely the planning-based adaptation middleware and the principles of AOP (cf. section 2). Then, we motivate the need for AOP technics with a scenario extracted from a business application (cf. section 3). Based on this scenario, we present our support for AOP using a planning-based approach (cf. section 4) and we illustrate it on a case study (cf. section 5). Finally, related works are discussed (cf. section 6) before concluding and exposing our perspectives (cf. section 7).

2. FOUNDATIONS
This section introduces the foundations of our contribution by presenting concepts related to planning-based adaptation middleware (cf. subsection 2.1), and aspect-oriented programming (cf. subsection 2.2).

2.1 Planning-based Adaptation Middleware
Planning-based middleware refers to the capability of adapt-
ing an application to changing operating conditions by exploiting knowledge about its composition and Quality of Service (QoS) metadata associated to the application components [3, 1]. 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 Receive, Handle, and Store, which have one, three, and two alternative configurations, respectively.

![Figure 1: Architecture of a Component-based Logger.](image)

As we already mentioned, the evaluation of these alternative configurations 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Capacity of the logger</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cap</td>
<td>0–7</td>
<td></td>
</tr>
<tr>
<td>lat</td>
<td>0–100</td>
<td></td>
</tr>
<tr>
<td>bat</td>
<td>0–100</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Specification of the Logger QoS Model Dimensions.

This utility function minimizes the battery consumption and latency of the configuration, while optimizing its capacity. The user preferences are used to weight each QoS dimension and ensure that the output will be a normalized value. The function is defined as a set of pieces of code (advice) to execute in particular points (pointcuts) of an application. Pointcuts are usually composed of a set of elements of the base code (joinpoints such as, class, method, or control instruction) Aspect weaving is the mechanism that inserts the aspect advices into pointcuts at compile-time, load-time, or run-time. Actually these strategies are several software frameworks implementing AOP concepts, such as AspectJ [8] (compile-time and load-time weaving), or PROSE [10] (run-time weaving).

Although AOP provides abstractions for specifying where and how aspect(s) can be woven into a base program, most of the current approaches are limited in the specification of when and which aspect(s) can be woven at a certain point of time. More importantly, existing approaches fail to specify the impact of aspect weaving on the QoS of an existing application. Thus, we argue that planning-based adaptation middleware can benefit to existing AOP approaches by providing a run-time support for the dynamic selection of aspects. This support is driven by an evaluation of the QoS properties.

### 3. MOTIVATING SCENARIO

This section further motivates the need for supporting AOP in offline properties reference to the development of mobile applications evolving in ubiquitous environments. The scenario we describe is extracted from SATMOTION, a business application assisting an installer in establishing an

![Image](image)
up/down link with a satellite. This application is usually distributed across two locations (cf. Figure 2). The client-side application is deployed on the handheld device of the installer who operates on the installation site, while the server-side is hosted by the control centre. In particular, it provides signal measurement instruments, such as spectrum analyzers or vector analyzers, that receive the signals captured by the antennas. The signal traces are streamed to the mobile device of the installer via Internet.

Depending on the execution environment, SatMotion can operate in different modes. When using the two-way mode, the installer receives the signal traces from the server and can control the analyzer remotely. The one-way mode is activated when cost of accessing network becomes too expensive. In this mode, the installer receives the signal traces from the server, but has no control over the analyzer. Finally, when the network connection becomes unavailable, the application switches to the offline mode. This mode displays to the installer previously recorded signal traces.

The different execution modes are provided as alternative implementations of the SatMotion components (cf. Figure 4).

Table 2: Specification of SatMotion QoS Model Dimensions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc</td>
<td>Accuracy of the signal trace</td>
<td>0–100</td>
</tr>
<tr>
<td>fr0</td>
<td>Freshness of the signal trace</td>
<td>0–1</td>
</tr>
<tr>
<td>ctrl</td>
<td>Control of the analyzer</td>
<td>0–1</td>
</tr>
<tr>
<td>bat</td>
<td>Battery consumption of the application</td>
<td>0–100</td>
</tr>
</tbody>
</table>

The planning-based adaptation middleware is then responsible for selecting the component implementations that provides the best combined utility, while the consistency of configurations is ensured by architectural constraints [7]. However, we observe that the introduction of new concerns (e.g., authentication, logging) leads to a combinatorial explosion of component implementations. This statement becomes critical when considering the other types of security concerns, such as identification, authorization, confidentiality, integrity, non-repudiation,
auditing [5], and their possible implementation alternatives.

To reduce this phenomenon, we propose to apply the principles of AOP to provide a more scalable approach for the integration of crosscutting concerns. In particular, AOP will provide a solution to the problem of concerns tangling of components by separating the business logic from the technical one. In addition, AOP will control concerns scattering effects by isolating concerns distributed across several components as a single aspect. Finally, the planning-based adaptation middleware will provide a mechanism for selecting and deploying aspects in order to optimize the quality of service of an application.

4. PLANNING OF ASPECT WEAVING

This section exposes our first elements for supporting aspects using a planning-based adaptation middleware. In particular, we propose to reflect the QoS model of an aspect using the concept of plan (cf. subsection 4.1), and to reuse composite components for modeling aspect-based configurations (cf. subsection 4.3).

4.1 Specification of Quality of Aspects

Aspects are pieces of code that usually enhance an application to introduce new functionalities. In some other cases, an aspect can also be used to improve an existing functionality. However, in both cases, the weaving of an aspect has an impact on the QoS of the base application. Thus, in order to identify this impact, we propose to extend the definition of an aspect with a QoS model. This QoS model defines the set of QoS dimensions, which are specific to a type of aspect (e.g., logging, security). For example, the QoS model described in Table 1 can be reused to describe a logging aspect.

This QoS model is then associated to aspects using plans. Indeed, while plans were initially designed to reflect components making up an application, we demonstrated in [12] 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 QoS independently of underlying artifacts. Based on this assumption, we believe that plans provide also a suitable abstraction to reflect crosscutting concerns. In particular, plans and the enclosed QoS properties can be reused to model alternative implementations of components.

4.2 Modeling of Aspect-based Configurations

Once such QoS dimensions are specified, we need to describe how, where and when an aspect can be woven into a component. Thus, we propose to reuse the modeling of a composite component and to describe the weaving of an aspect as a particular kind of component binding. In particular, an aspect is reflected as a composite role and combined with other application roles using an aspect binding. For example, the configuration depicted in Figure 5 revisits the one presented in Figure 3 by introducing a new composition Logged Controller that supports the logging of the controller component. The aspect Logging is associated to a role Controller to produce a new implementation supporting logging of the information that are processed. This aspect is also bound to a role Logger to specify that the weaving of the aspect implies the weaving of a port L into the Controller. Thus, in a similar way to [11], we enforce separation of concerns by isolating the aspect Logging—i.e., the pieces of code to insert into the Controller—from the Logger, which implements the logging logic. This means that various implementation alternatives can be considered for the roles Logging and Logger. In particular, the alternative implementations of the role Logging can reflect different AOP technologies to use to weave the concern (e.g., load-time Vs. run-time weaving). The alternative implementations of the role Logger refer to those introduced in Figure 1. In addition, this separation allows our planning middleware to plan separately the integration logic and the business logic to integrate.

![Figure 5: Architecture of a Logging Aspect.](image)

The planning is then driven by the evaluation of the QoS properties that are associated to the implementation alternatives. Thus, in case of aspect-based configurations, we propose to reflect the impact of an aspect on an application role by associating the aspect QoS model with the application QoS model at the composition level. For example, we describe that the freshness (fr) of a logged controller corresponds to the sum of the freshness of the Controller and the Logging aspect divided by the latency (lat) of the Logger (cf. Figure 5). New QoS dimensions can be also added to the composite component specification to reflect the integration of new QoS information. For example, the integration the QoS dimension capacity (cap) to describe that the capacity of the Controller corresponds to the capacity of the contained Logger.

4.3 Evaluating Aspect-based Configurations

The configuration alternatives including crosscutting concerns can be evaluated implicitly or explicitly by the planning-based adaptation middleware.

The implicit evaluation consists in integrating the logging QoS properties in the property predictors associated to the application QoS properties. This approach seamlessly reflects the impact of the logging mechanism on existing QoS properties (e.g., if employed under specific circumstances, the logging mechanism increases the controller battery consumption by 10% and reduces its freshness by 15%). As a consequence, the aspect-based configurations can be compared to other configurations without changing the utility function of the application.

The explicit evaluation 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., the capacity of the logger). For example, the utility function below extends the one defined in section 3 and specifies that the capacity of controller should be maximized (therefore enforcing the
need for logging):

\[ Utility(SatMotion) = U_{sec_{acc}} \times norm(acc) + U_{sec_{ctrl}} \times norm(ctrl) + \frac{U_{sec_{bat}}}{norm(bat)} + U_{sec_{cap}} \times norm(cap) \]

This approach enables the user to specify its preferences with regards to the crosscutting concern (reflected by the user preference \( U_{sec_{cap}} \)).

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 process will decide whether the logging mechanism needs to be enabled or not. If the logging is needed, then it will determine which aspect technology has to be selected as well as the best fitting configuration for the logger. This also means that, at runtime, the planning-based adaptation middleware can reconfigure the logging mechanism to tune the weaving and the logging strategies.

5. CASE STUDY: CRYPTOGRAPHY

This section illustrates the design of a security concern as an aspect and demonstrates how this concern is dynamically adapted according to the scenario.

When assisting in the alignment of a satellite antenna, the SATMOTION application may detect that the signal traces are too sensitive for being transmitted as raw content via Internet. Thus, when the device is connected to an untrusted network, SATMOTION has to switch to an encrypted execution mode to ensure the privacy of information. Therefore, we propose to integrate this communication mode using AOP. The idea is to develop and model aspects, whose integration will be planned by our adaptation middleware.

5.1 Implementing Cryptography as Aspects

The development of aspect is based on existing AOP technology, such as AspectJ [8], or PROSE [10]. For example, Listing 1 and Listing 2 describe implementations of decryption and encryption aspects in AspectJ, respectively.

Listing 1: Decryption Aspect in AspectJ

```java
public abstract aspect DecryptionAspect {
    public abstract pointcut decryptOps(SignalTrace t); 1
    private Cipher decoder; 2
    public void setCipher(Cipher c) { decoder = c; } 3
    public void around(SignalTrace t) { decryptOps(t) { 5
        proceed(decoder.decrypt(t)); //Decrypt 6
    } } 7
}
```

Listing 2: Encryption Aspect in AspectJ

```java
public abstract aspect EncryptionAspect { 1
    public abstract pointcut encryptOps(Operation o); 2
    private Cipher encoder; 3
    public void setCipher(Cipher c) { encoder = c; } 4
    public void around(Operation o) { encryptOps(o) { 5
        proceed(encoder.encrypt(o)); //Encrypt 6
    } } 7
}
```

These aspects are used to decrypt (resp. encrypt) the signal traces (resp. the control operations). They also introduce into the component a reference to the cipher algorithm used to decode/encode transmitted data. Various implementation of cipher algorithms are actually available, such as Advanced Encryption Standard (AES), Data Encryption Standard (DES), or RSA. These strategies mostly differ by in terms of privacy degree and processing cost [6]. Thus, we propose to reflect this crosscutting concerns as well as the encryption strategies as configuration alternatives for the component Controller.

5.2 Specifying Cryptography QoS

First, we propose we identify the QoS model associated to the cryptography concern. This model is described in Table 3. Properties \( \text{deg} \), \( \text{cost} \), and \( \text{bat} \) reflects offered properties, while property \( \text{conf} \) is requested. We assume that the requested properties are provided by a context middleware. As this QoS model is an extension, default values are provided (using the square brackets) if the property is not mentioned explicitly.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>deg</td>
<td>Privacy degree</td>
<td>(Unsecured) 0–10 (Secured) [0]</td>
</tr>
<tr>
<td>cost</td>
<td>Processing cost</td>
<td>(Low) 0–10 (High) [0]</td>
</tr>
<tr>
<td>bat</td>
<td>Battery consumption</td>
<td>(Weak) 0–100 (Strong) [0]</td>
</tr>
<tr>
<td>conf</td>
<td>Confidence in the network</td>
<td>(Untrusted) 0–10 (Trusted) [0]</td>
</tr>
</tbody>
</table>

5.3 Modeling Encrypted Configurations

Then we model aspects as new configuration alternatives for the component Controller (cf. Figure 6). In particular, we specify that the aspect Decrypt applies to components Controller exhibiting a port AS—i.e., components used in modes one-way and two-way—while the aspect Encrypt applies to components declaring a port AS (used in the mode two-way). The component Cipher reflects the alternative algorithms that can be used to encrypt/decrypt data. The QoS of each algorithm is specified as note for each implementation and integrated into the property predictors of the composite configurations.

![Figure 6: Modeling of Encrypted Configurations.](image-url)
As the consequence, when the mobile device communicates using the WiFi network connection. The network is untrusted and the device has enough resources to run complex encryption algorithms. Thus, the aspects AspectJ Encryption and AspectJ Decryption, as well as the cipher RSA Cipher are selected by the planning middleware and woven into the component Twoway Controller (cf. Figure 3). When the application loses the WiFi connectivity, the device switches to a GPRS trusted network. In this case, the aspects are unselected and the planning middleware uses the component Twoway Controller. When the device battery becomes too low for the using network, the planning middleware unweaves the authentication aspect from the component UI (not presented in this paper due to the page limitation).

6. RELATED WORK

**Quality Objects** (QuO) [13] is a middleware framework supporting adaptive distributed object applications that can specify their QoS requirements as QoS contracts. QuO allows an application developer to design application behavior using IDL (Interface Description Language) and QoS contracts, alternate implementations, and adaptation strategies using QDL (Quality Description Language) that describes the expected regions of QoS and the transitions for changing levels of service provided by objects, SDL (Structure Description Language) that describes alternative behaviors for objects, bindings and connection strategies, and RDL (Resource Description Language) that describes available system resources and their status. QuO’s code generators use AOP mechanisms to weave these descriptors together into a single application. However, the adaptation mechanism relies on static conditions described in the QoS contracts, which may lead to non-optimal configurations. Finally, designing multiple crosscutting concerns can cause conflicts between QoS contracts.

**QuA** [9] and MADAM [1] use CBSE and mirror-based reflection to express functional and non-functional concerns, respectively. This means that the integration of crosscutting concerns usually leads to a combinatorial explosion of configuration alternatives. In some cases, these planning-based adaptation middleware, which is restricted to compositional adaptation, can support AOP principles partially. In particular, QuA 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), they do not support other AOP mechanisms, such as the transformation of legacy components. Furthermore, these connectors are often application-specific, which limits their reusability.

**FRACtAL Aspect Component** (FAC) [11] and ASPECTOPEN-COM [4] are two approaches that aim at reconciling CBSE and AOP. These approaches provide the core mechanisms for defining aspects and weaving them into components. However, the support for automatic weaving of aspects is restricted to Event-Condition-Action (ECA) mechanisms. Depending on context variation, the developer can describe the run-time weaving of a particular aspect into the component. This means that these approaches i) are limited to interface interception mechanisms ii) are specific to an aspect technology, and iii) do not support continuous QoS optimization.

7. CONCLUSION & PERSPECTIVES

This paper has introduced the extension of planning-based adaptation middleware for the support of Aspect-Oriented Programming (AOP) principles. Our adaptation middleware benefits from AOP by controlling the combinatorial explosion of implementation alternatives, which appears when a growing number of crosscutting concerns is taken into account by a component-based system. This limitation is resolved by modeling and planning separately the business concern, the aspect mechanism, and the crosscutting concern. Our planning-based adaptation middleware benefit also to AOP by supporting the dynamic selection and configuration of the aspect mechanism and the associated policies depending on contextual information variation.

As a preliminary validation of our approach, the paper also explained our planning-based adaptation middleware handles a use case scenario in which the mobile device of a satellite installer can dynamically switch to secured configurations to improve the utility of its SATMotion application.

In our future work, the presented planning framework 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). Furthermore, we are also interested in investigating the identification and the detection of compatible/conflictual aspects using the architectural constraints supported by our modeling approach [7].

8. REFERENCES


