Self-Adaptive Middleware for Digital Ink Based Applications

Lucas L. Provensi
Federal University of Goiás
Institute of Computing
Goiânia, Brazil
lucas@inf.ufg.br

Fábio M. Costa
Federal University of Goiás
Institute of Computing
Goiânia, Brazil
fmc@inf.ufg.br

Vagner Sacramento
Federal University of Goiás
Institute of Computing
Goiânia, Brazil
vagner@inf.ufg.br

ABSTRACT

Some aspects of the mobile environment, like lower bandwidth and higher error rates, can affect distributed applications that have real-time requirements. In order to maintain the quality of service expected by such applications, a middleware platform can monitor its execution environment and perform dynamic adaptations on its structure and behavior. To do this, the middleware must know which QoS attributes affect the application and act in accordance with policies described in a specific language. In this paper we propose a self-adaptive architecture, based on the Meta-ORB approach, which uses adaptation policies described in the same modeling language used for the definition of middleware configurations. The paper also presents a study on the use of this architecture for applications based on digital ink.

Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems

General Terms

Design

Keywords

Self-adaptive Middleware, Reflective Middleware, Digital Ink, Distributed Multimedia

1. INTRODUCTION

To deal with dynamic changes in the application requirements that may occur in some computational environments, such as the mobile environment, middleware platforms may use reflection techniques to adapt their structure and behavior at runtime. The middleware should also be able to identify changes in the environment that can affect the application, in order to perform adaptations in a transparent way.

Some solutions, such as in [14] and [11], are based on self-adaptive mechanisms that use policies written in a definition language created specifically for that purpose. In this paper, we present a self-adaptive mechanism for the Meta-ORB reflective middleware architecture [10] which, in contrast, is based on policies defined in the platform’s own model. The model used for static instantiation of the Meta-ORB platform is kept accessible at runtime [9], being used by the reflective mechanism to build its self-representation. This model is built according to the platform’s meta-model, which defines the entity types that can be used in the construction of specific platform configurations, such as components, interfaces and bindings. In our proposal, adaptation policies are meta-types defined in the Meta-ORB meta-model. As such, policies can be modeled like the other entities that form a middleware configuration and remain available at runtime, being used by the self-adaptation mechanisms.

This paper discusses how our approach for self-adaptive middleware can be used to handle some communication problems found in real-time applications, with particular interest in those based on digital ink. Digital ink is a recent technology that aims at improving the interaction interface of computing devices, such as tablet PCs, through a special hardware that simulates handwriting. Because of its similarities with media types like audio and video, in our approach we treat digital ink as a media type as well, which brings forth the need for appropriate middleware support.

The paper is structured as follows. Section 2 presents an application scenario based on digital ink and discusses the requirements that lead to the need for self-adaptive middleware. Section 3 discusses how QoS attributes and adaptation policies can be defined in the application model, according to the Meta-ORB meta-model. Section 4 discusses a proposal for the implementation of a middleware infrastructure that can apply adaptations automatically, based on policies defined in the application model. Finally, Section 5 discusses related work and Section 6 presents some concluding remarks and future work.

2. APPLICATION SCENARIO

In this paper we use real-time applications based on digital ink as a scenario to evaluate the proposed infrastructure. Pen-based computing was introduced in order to make the interaction interfaces of computing devices more natural, taking the form of tablet PCs and digital ink applications. The movement of the pen and its contact with the surface of the screen is captured by a digitizer, forming strokes of dig-
digital ink, which are essentially the computational representation of the strokes made with the pen. Several applications can take advantage of this technology, like those already used in classrooms as presentation tools, making them more interactive. For example, in [1] a slideshow application is improved with the support for handwriting over slides in an interactive way.

The application developed to illustrate the use of the proposed middleware infrastructure is a whiteboard, which consists of a shared annotation area, where digital ink is transmitted in real time from one user to the others. Compared with a real whiteboard, the computational whiteboard presents some advantages, such as the direct manipulation of ink for scaling, cropping and movement operations, and the possibility of changing ink properties, such as line color and style. To enable these operations, digital ink must be transmitted in its original form, without converting it to another data type, such as images or a video stream.

However, in this type of application, there are requirements for digital ink communication that must be considered, since the application may be running in a dynamic environment such as a wireless network. Lower bandwidth, higher error rates and more frequent disconnections are some of the problems found in this environment [12]. Considering the whiteboard application, these problems can cause delay in digital ink reception and loss of pieces of the transmitted digital ink.

The delay problem directly affects the coordination of users’ activities. In the whiteboard, for example, the teacher may be talking as he or she writes on the board. If there is a delay in ink reception by the students, the teacher’s talk may become out of sync with what he or she is writing. The loss problem, on the other hand, affects the static representation of digital ink at a given time. Since digital ink packets are basically made of $x$ and $y$ coordinate data, snippets of ink formed by these coordinates are lost. For example, if packets containing certain segments of words are lost, a phrase written on the whiteboard by the teacher may become difficult to understand by the students.

Like in other works [2], we treat digital ink as a first-class data type and as a new media type as well, because of its similarity with other media types, such as audio and video [17]. In real-time applications these media types present some tolerance to delay and loss. A small delay in the ink reception, as well as low loss rates, may not be perceived by users. However, when the delay and losses reach a certain level, the user may lose the sense of real-time and may not correctly understand the media content. Digital ink also has its peculiarities, such as the static view of the strokes, which allows the recovery of segments lost during transmission, which is not possible with audio and video streams.

The problems of delay and loss cannot be evaluated prior to execution of the application, because they can vary over time. Therefore, continuous monitoring of network quality is necessary to to identify them. Once such problems are identified, the application must change its behavior in order to adjust itself to the new network characteristics. For this, the application must inspect and adapt its internal structure at runtime [4]. In order to enable this, we propose extensions for self-adaptation in the Meta-ORB platform. Such extensions are discussed in the next sections, using the whiteboard as an application example.

3. DEFINING THE APPLICATION MODEL

Meta-ORB [8] is a reflective middleware architecture based on the Open ORB approach [3]. It combines reflection and metamodeling techniques, which allow the creation of customized middleware configurations that can be dynamically adapted. The metamodeling technique provides a type system for the definition of entities that form a middleware configuration. The reflection technique divides the architecture in a base-level, which contains the middleware functionality, and a meta-level, which provides the reification of the base-level features. Both techniques are based on the same kind of meta-information. This meta-information is the model that represents the middleware configuration both statically and at runtime.

A statically created model can be used to instantiate configurations, which are formed basically by components that implement the middleware features and can interact with each other through well-defined interfaces. The interaction between components is defined in the model as well, in terms of binding objects. This model remains available at runtime, being used by the meta-components, which form the meta-level, to build the representation of the base-level. In our proposal, we introduce in the model the possibility of defining adaptation policies, which are associated with bindings. With this extended model, specialized middleware configurations can be instantiated, which apply the policies according to QoS constraints defined in the component’s interfaces connected by the binding.

The whiteboard application, for example, can be modeled in terms of components, interfaces and bindings. Figure 1 illustrates the relationship between the runtime entities, the application model and the constructs of the platform metamodel. The meta-model defines the constructs that can be used for modeling middleware configurations, which, for the application in question, are primitive components, stream interfaces with their media flows, and binding objects. The model in turn is formed by instances of the meta-model constructs. For example, the primitive component WhiteBoard has two stream interfaces, InkConsumerInterface and InkProducerInterface, which can be linked through a binding of the type InkBinding. Specialized factories can instantiate the runtime entities (concrete components and bindings) using the model. They are used to instantiate WhiteBoard components on several tablet PCs in a wireless network. The factories then create a distributed binding object to allow interaction among the remote components.

From the moment a configuration is instantiated, it is subject to the reflection mechanisms of the platform. A meta-component can be created to reify the binding of Figure 1, for example. This meta-component (meta-level) builds a self-representation, causally connected to the runtime entities (base-level), using the same model that the factories use to instantiate the configuration. The meta-component can manipulate such self-representation and the changes made on it are reflected in the structure of the base-level binding.

So far, the whiteboard application has only the basic functionality for digital ink communication, and can use the Meta-ORB reflective infrastructure to carry out dynamic adaptations. As seen in Section 2, this is not enough for this kind of application, as there are QoS requirements that should be considered. The middleware must be able to ana-
lyze the application environment to determine whether these requirements are being satisfied. First of all, the middleware should know these requirements. For that reason QoS constraints can be defined in the application model. During the instantiation of a configuration, a specialized factory can check the model for QoS constraints and apply a monitoring strategy.

As an example, Figure 2 shows the definition of QoS constraints as part of the application model, built in accordance with the Meta-ORB meta-model. In the figure, the digital ink flow that arrives at the stream interface of a component is constrained by QoS annotations, which are defined in the model as expected values for certain attributes, such as the maximum delay and maximum loss percentage. Special monitors can be created to detect violations on the QoS attributes of the flow during the application execution. These monitors may load different implementations for each type of monitored attribute. For the QoS attributes of interest, one of the implementation alternatives is the use of interceptors on the interfaces, which feed monitors with information about the interactions on those points.

QoS constraints can thus be defined in the model and middleware configurations can be created to monitor violations in the interaction between components. As a result of these violations, the middleware must undertake a series of procedures to adapt its structure and behaviour to the new scenario. In our approach, these procedures are described in the model itself, as adaptation policies associated with a binding, which are applied as a result of violations in one or more QoS attributes. Middleware configurations can thus include the necessary infrastructure for QoS monitoring and treatment, together with the interaction infrastructure for remote components.

Figure 3 illustrates, in a simplified form, the meta-model extensions supporting adaptation policies. Following the principles of the Meta-ORB meta-model, the definition of an adaptation policy is a container that contains a series of adaptation rules. A policy is defined for a binding (to address QoS aspects of the binding) and is associated with one or more QoS attributes (defined in the interfaces that interact through the binding), which are the triggers to adapt the interaction infrastructure. An adaptation rule, in turn, determines the operations of the binding’s meta-component that should be called, together with its parameters, in response to QoS violations.

In the Meta-ORB meta-model, the definition of a binding may contain a set of internal bindings. Each of the bindings in turn may define their own adaptation policies. For example, an outermost binding containing two internal bindings, one to handle video and the other to handle a digital ink stream, may contain policies to address common issues of these media types, and each one of the internal bindings may contain policies to address particular aspects of each media. Despite the advantages offered by the composition of bindings, however, a more appropriate way to define composition of policies still has to be studied.

Figure 3 also shows a model created according to the meta-model. This model has an adaptation policy to handle the binding that conveys the digital ink flow. This policy is associated with the delay and loss attributes of that flow and has a series of rules to adapt the binding. These rules may define, for example, calls to the meta-component operations for the replacement or addition of components in the binding, such as to introduce the compression of digital ink in order to reduce the size of the packets sent through the network.

The model shown in Figure 3 provides the necessary information for carrying out adaptations based on QoS violations in a interaction between remote components. Specialized components, which implement a protocol based on the policy definitions described in the model, can be instantiated during the binding creation to manage the adaptations. In the next section, a concrete implementation of a Meta-ORB prototype will be described to demonstrate the extensions discussed in this section.

4. MIDDLEWARE IMPLEMENTATION

The Meta-ORB platform was first prototyped in Python [7], as a reference implementation, and later in Java (including J2ME) [8], called MetaORB4Java, for deployment on portable devices with limited computing resources. For this work, a prototype in C#.NET is currently under development. This prototype is intended for deployment on mobile devices without major resources limitations and that
are suitable for specific APIs, such as Tablet PCs and digital ink APIs. The prototype is also interoperable with MetaORB4Java because it uses the same protocols and the same Java implementation of its remote services: the naming service and the type repository.

The type repository is one of the main services of the platform. It contains the Meta-ORB meta-model representation, implemented as a global repository that facilitates the definition and retrieval of the entities (types) that form the models. In its latest implementation, the type repository uses a meta-model defined using the metamodeling tool EMF (Eclipse Modeling Framework) [5]. This tool enables automatic generation of the repository implementation with basic accessor methods and graphic plugins for the Eclipse platform, which facilitate the definition of types. This generated repository was extended with web service capabilities, allowing its access by remote instances of the platform. More details on this EMF based implementation can be found in [9].

Instances of the platform containing the basic middleware features, which are called capsules, can access the repository web service to obtain and manipulate types. As the result of a search, types retrieved from the repository, which are runtime EMF EObjects, are serialized in a XMI format and sent to the capsule that originated the search. In the Java prototype, the XMI received in this way can be directly converted into EObjects. In the C#.NET prototype, the language facilities to treat XML were used to interpret the types. As seen in the previous section, types are used by specialized factories as blueprints to build middleware configurations and by meta-components to build the self-representation of the base-level.

In our proposal we extended the Meta-ORB meta-model to support adaptation policies as a new meta-type that can be defined and stored in the repository. Policies contain meta-information that defines how the middleware must proceed in order to automatically adapt itself. To evaluate and apply policies, a specialized component was designed for the management of adaptations. The adaptation manager acts on a binding, together with a component specialized in QoS monitoring, and must implement a protocol to analyze policies and determine when they should be applied to adapt the interaction as a result of violations in the QoS attributes.

Figure 4 illustrates the operation of the adaptation manager and the QoS monitor components in a implementation that is under development. In the figure, a binding was registered with the adaptation manager of the capsule by the factory that created it. All that the manager needs is the binding’s unique name (registered in the naming service) and its type identifier. Using the type identifier, the manager obtains the definition of the binding from the type repository and identifies the interfaces of the binding that have flows with QoS attributes. With the unique name of the binding, the manager creates a meta-component that will later be used to reify the binding.

To decide when to perform an adaptation, the manager
uses a component of the capsule that is specialized in QoS monitoring, informing the attributes of interest and the binding that should be monitored. The monitor, as well as the factories and the adaptation manager, are generic components that can have different implementations. A simplified implementation for the digital ink flow, as an example, uses the interception meta-space (one of the partitions of the meta-level) to monitor package delay and loss. The interceptors are inserted in the interfaces linked by the binding, collecting information about the flow and sending it to the QoS monitor. Other, more elaborate, implementations, which use more precise context information, still need to be studied.

The adaptation manager receives a compilation of data gathered by the QoS monitor, according to a specific time interval. The manager then verifies whether QoS attributes are being violated and, if so, what policies can be applied. If there are conflicting policies, with rules for adapting the same target or to handle the same QoS attribute, the manager chooses the policy to apply according to some predefined priority.

Since the violation of a QoS attribute may be temporary, the manager waits, during a tolerance period, before applying the policy. If the violation is maintained during that period, the policy is applied, using the meta-component to reify the binding and applying the operation and the parameters defined in the policies. When the meta-component finishes the binding adaptation, the manager enters a period of experimentation, in order to determine whether the adaptation obtained its desired effect. During that period, if the monitored QoS level has improved, then the manager keeps the adaptation. From that moment, policies conflicting with the policy applied are discarded by the manager.

However, if at any time (during the experimentation period), the adaptation fails to maintain the expected QoS level, the manager calls the meta-component for its reversion, and the policy in question receives a lower priority, giving opportunity for the application of other policies. It is interesting to note that the parameters used by the manager, such as the initial priority, the tolerance period, the period of experience and the target of the rules, are specified directly in the definition of the policy obtained from the type repository.

5. RELATED WORKS

Despite the breadth of this work, we believe that its major contributions, in its present stage, are in two points: The first is the treatment of digital ink as a media type. While not the main focus of this paper, a preliminary study was presented as a motivation for our approach. Digital ink technology has been used in various applications, such as in [6, 18, 1]. Some of them work with the transmission of the ink through a network, but none adequately address the QoS issues involved in its communication. Like other proposals [17, 19], we consider digital ink as a new media type, so its transmission in real time applications requires special middleware treatment to maintain the QoS level expected by the user.

The second point is the definition of policies as part of the application model. There are well-established reflective middleware approaches, such as Open ORB [3] (in which the Meta-ORB was based) and Dynamic TAO [15]. In these approaches, the decision to adapt the middleware and the policies used for this are defined locally, in each of the nodes, usually in an ad hoc fashion. In our approach, the management of adaptations is made globally, following policies that can affect all the nodes of a binding.

There are several works that propose self-adaptive middleware architectures, such as in [13] and [14]. The first presents an extension of the Open ORB approach with mechanisms for safe, valid and distributed adaptation. The second presents an open framework for dynamic adaptation of services. In common, these works use policies, described with a specific language, to make adaptations as a function of changes in the application execution context. In contrast, although using similar mechanisms, in our work policies and QoS annotations are described in the same language used to describe the application model and are stored in the platform’s type repository alike.

In our approach, policies are specified in a simplified form, as they were designed to meet only the adaptations that we consider necessary for distributed multimedia applications, especially those based in digital ink. The Quality Objects (QuO) [20] approach defines a set of languages for describing a number of application aspects, such as QoS specification, monitoring an adaptation. Other works, such as [16] and [11], present a language with a syntax that allows the construction of more elaborate policies. However, in our approach the metamodel that defines the policies can be easily extended to accommodate the needs of other kinds of application. Moreover, unlike the other approaches, the type repository can be used for definition, storage, distribution and versioning of policies, without the need of an extra middleware service for doing that.

6. CONCLUSIONS AND FUTURE WORK

This paper presents our approach for middleware self-adaptation based on extensions of the Meta-ORB metamodel for the support to adaptation policies, together with
an infrastructure designed to apply these policies. We believe that this is a more natural approach to represent adaptation policies. Besides, tools generated with the EMF type repository can be employed to facilitate the modeling of policies.

We have particular interest in applying this approach to address the QoS requirements (found in the mobile environment) of real-time applications based in digital ink, since this area is still underexplored. QoS parameters for such applications must be studied, to more precisely determine the QoS level expected by the users. We also expect to apply this work in a real application scenario. One of the possibilities being studied is the use of the proposed architecture to address communication aspects of collaborative systems based on digital ink.

The work presented in this paper is still in progress, since the prototype with the described features is under development and its behavior has not been properly evaluated. The current implementation of the C#.NET prototype has only basic functionality for the instantiation of capsules and communication with the type repository and naming service. Other elements of the described architecture are still under development, such as the component and binding factories, the adaptation manager and the QoS monitor. Furthermore, adaption policies to handle other kinds of application are yet to be studied to validate the proposal, which may raise the need for more elaborate strategies for context monitoring, as well as a more elaborate syntax and semantics to express adaptation policies and policy composition.

7. ACKNOWLEDGMENTS

We thank CAPES/Brazil and the HP Technology for Teaching Higher Education Grant 2007 for supporting the development of this work.

8. REFERENCES


