A Component-based Meta-Model for Context-Aware, Distributed Adaptation Graphs

Thomas Springer
TU Dresden, Department of Computer Science, Computer Networks Group
thomas.springer@tu-dresden.de

Abstract

In this paper we present a component-based modeling approach for distributed adaptation graphs at architectural level. The graph structure reflects the pipeline-based processing of applications typical for adaptive data processing and transfer in mobile and pervasive environments. Parametrizable components and connectors represent mechanisms for processing application data as well as its mediation between components. Further elements represent contextual information and its mapping to component and connector parameters allowing an explicit definition of adaptation control. Thus, the model enables the explicit modeling of adaptation mechanisms as well as of reconfigurations in form of alternative paths, both controlled by parameter changes. The meta-model provides a certain view and abstraction on adaptive software systems which can be combined with other views to create comprehensive system descriptions. It represents a platform independent model (PIM) not tied to a certain component platform and can be mapped to existing component platforms like J2EE or Microsoft .NET using an MDA-approach.

1. Introduction

Today, mobile and wireless technologies are an integral part of distributed computing environments building up a convergent platform for traditional and innovative services and applications. As a consequence new service and application areas are enabled, but also new challenges for application development are raised regarding to the heterogeneity and limitations of device capabilities, heterogeneity of reliability and performance of network connections, heterogeneous user requirements and computing context, and the frequent changing infrastructure mainly caused by mobility.

Adaptation and Context-Awareness are closely interrelated key concepts for software systems executing in pervasive computing environments. The term adaptation describes the adjustment of a system to specific conditions or changes in its execution environment. The question to ask is: What is adapted to what? asking about object and target of the adaptation. Objects of adaptation from the viewpoint of an application are its processed data, the communication for data exchange and the structure of the application itself (including its functional components, their interconnections and placement). The target of adaptation is the execution environment (i.e. available resources, user information and preferences and context of system usage), characterized by context which represents information about the state and changes of an applications execution environment.

In this paper we present a component-based modeling approach for context-aware, distributed applications supporting the description of the adaptation of application data, communication and structure (i.e. components, their interconnections and placement). Further goals are the integration and flexible combination of adaptation functionality, the explicit description of adaptation control based on context, and platform independence of application models. We introduce a meta-model at architectural level defined using OMG’s Meta-Object Facility (MOF) to describe distributed adaptation graphs. Furthermore, a graphical notion of the modelling elements is defined adopting several elements from UML 2 notion. A graph reflects the pipeline-based structure of applications typically used for adaptive data processing and transfer in distributed multimedia applications. We focus on applications with periodical access to similar multimedia data like email, FTP, and WWW, a class of applications currently adopted for mobile devices.

The paper is structured as follows: In the following section we introduce the MOF-based meta-model. In section 3 we present an example application model to illustrate the feasibility of our approach. The paper closes with an examination of related work, a conclusion and an outlook.

2 Definition of the Meta-Model

The meta-model contains a set of elements we have identified as fundamental for the modeling of adaptive software systems at architectural level based on elements adopted
from architecture description languages (ADL). According to the notion of the OMG our modelling approach is defined at the meta-model level (M2). The definition of the meta-model is based on OMGs Meta-Object Facility (MOF) and the graphical notion of UML. The elements of the meta-model support the modelling of distributed adaptive applications from the view of adaptation. This view can be combined with other views to create comprehensive application models. Within the next subsections the elements of the meta-model are introduced including their graphical representation defined by UML stereotypes. Our modelling approach provides a certain view and abstraction on adaptive applications which can be combined with other views to create comprehensive system descriptions. It represents a platform independent model (PIM) not tied to a certain component platform and can be mapped to existing component platforms like J2EE or Microsoft .NET using an MDA-approach. In the next subsections the main elements of the meta-model and its interrelations are introduced.

2.1 Modeling application data

Application data is mediated within adaptation graphs in the form of data objects encapsulated into abstract data containers. A data container consists of a unique identifier, attributes for the name, size and type of data and the data itself. Furthermore, meta-information for adapting and mediating data objects can be attached in form of annotations. Annotations are typed values defined by a unique identifier, data type, unit, default value and constraint. To distinguish the type of data objects and basic data types, we use the notion object type to specify the type of data objects. The modeling of object types is based on the approaches of MIME and [8]. We use a composite type specification forming a type hierarchy with the levels type, subtype and encoding. Object types can be denoted in the form $t_o = \{\text{type}/\text{subtype}/\text{encoding}\}$ (e. g. $t_o = \{\text{audio}/\text{waveform}/\text{wav}\}$). To define the type of a data object encapsulated by a data container all three levels are mandatory. Further information about data objects (e. g. audio sample rate or image resolution) can be explicitly represented by annotations. The type systems allows the declaration of types of data objects independent from any platform or programming language dependent type system which can be mapped to any platform dependent type system. Furthermore, new types of data objects can be easily introduced by extending the type hierarchy.

2.2 Defining adaptation graphs

For modeling the structure of adaptation graphs basic elements from architecture description languages, namely components, connectors, ports and roles, are adopted. A configuration, i.e. a complete adaptation graph, consists of at least 2 components interconnected by 1 connector. Components represent processing operations including mechanisms for adapting data objects and network transfer. Connectors mediate data objects between components and represent communication protocols (e. g. TCP or UDP) as well as mechanisms for splitting up, parallelizing and sequentializing communication paths.

Ports and roles describe interfaces of components and connectors which are limited to the exchange of data objects. What types of data objects can be exchanged is defined by their signature. Because data objects and streams of data objects can be merged and decomposed within an adaptation graph, a signature is defined by a set of object types, defining that any data object of one of the object type element of the set can be exchanged by the port or role. A signature containing an empty set of object types defines an unbound port or role. Unbound ports and roles are bound by the composition with corresponding ports and roles. A signature can be denoted by $\text{Set}\{\text{objecttype}_1, ..., \text{objecttype}_n\}$.

Ports represent the components interfaces to interact with their environment. For adaptation graphs these interactions are limited to sending or receiving data objects of the types defined by the signature of the port. According to the direction of data flow from or to the component the port types Import ($I$) and Outport ($O$) can be distinguished. This limitation enables to abstract from concrete operations of component interfaces and therefore from a concrete implementation of data exchange. This ensures the platform independence of the meta-model and increases the reusability of adaptation mechanisms and the flexibility of their combination significantly. During model tranformation ports and roles can be mapped to concrete, platform-dependent interfaces and communication mechanisms (e.g. operations for message, request/response or stream-oriented communication). A port belongs to a certain component and is defined by a name (which is unique within the namespace of the component), a type (i.e. Import or Outport), and a signature; denoted by $\text{portname} = (\text{type}, \text{signature})$.

Each component has at least one port. A component containing Outports only is a data source, a component containing Imports only a data sink. Only data sources can create data objects within the adaptation graph. These data objects have to be explicitly removed by data sinks, no data container can be lost within the graph. Moreover, processing and storage components can be distinguished. A processing component contains at least one Import and one Outport and processes each incoming data object immediately and outputs it before processing the next one. Storage components store incoming data objects in a storage managed by the component. In a second step, data objects are taken from the storage and sent via an Outport. Both component
types contain at least one Import and one Outport.

For each port of a component an **Interception Point** can be defined. Interception Points related to Inports define points of preprocessing and interception points related to Outports define points of postprocessing. At these points meta-data can be captured from data containers and annotations can be read or written to data containers. A component represents a processing operation within the adaptation graph. All elements of a component are interrelated with this processing operation. Especially, a component can encapsulate a certain adaptation operation which can be parameterized by component parameters, increasing its transparancy and reusability.

**Roles** are the complement elements of ports. They represent the interaction interfaces of a connector. Similar to ports, roles are limited to the exchange of data objects abstracting from concrete interface operations. In correspondence to Inports and Outports, Sender roles \((S)\) represent sender of data objects via the connector and Receiver roles \((R)\) represent receiver of data object sent via the connector. What data objects can be exchanged is defined by the signature of a role. Initially, the signature of a connectors role is defined as "unbound", representing the semantics of a connector to mediate any type of data object. Signatures of roles are defined during the binding to a certain port. A role belongs to a certain connector and is defined by a name (which is unique within the namespace of the connector), a type (i.e. Import or Outport), and a signature; denoted by \(\text{rolename} = (\text{type}, \text{signature})\).

**Connectors** have at least two roles. They receive data objects via Outports of components connected to Sender roles and mediate them via Receiver Roles to Inports of other components. Especially, no data objects are lost within a connector, i.e. each data object received by a Sender role is mediated to at least one Receiver role. Connectors also can contain processing operations related to the mediation of data objects. These operations are also parameterizable via connector parameters.

According to the number of roles and the mediation operation, several kinds of connectors can be distinguished. A simple 1:1 connection is described by a Sequence connector which contains one Sender and one Receiver role. A Demultiplex connector contains one Sender and multiple Receiver roles. It clones incoming data objects and mediates them to exactly one of the Receiver roles, i.e. it can split up a single data stream into several data streams. The Multiplex connector contains more than one Sender and one Receiver role. It synchronizes the incoming data objects and mediates them to the Receiver role. The Multicast connector has one Sender and several Receiver roles. It clones incoming data objects and mediates them to all defined Receiver ports. A Typeswitch connector also contains one Sender and more than one Receiver roles, but mediates incoming data objects to exactly one Receiver role based on the object type. Different to the Demultiplex connector and the other connectors, the Receiver Roles of the Typeswitch connector have different signatures.

Based on these connector types which can all be parameterized, alternative and parallel pathes can be modeled which can represent structure changes of the adaptation graph. For instance, two alternative pathes which differ in only one component represent the exchange of these two components. The condition for choosing one of the two pathes defines the condition for the reconfiguration of the application, too. Thus, connectors and their parameterization enable the explicit definition of variations of the application structure and the conditions for reconfiguration.

Both elements, components and connectors, can contain parameters; i.e. they are derived from the element "Parametrizable Element". A **Parameter** is a typed value denoted by \(\text{parametername} = (\text{type}, \text{unit}, \text{defaultvalue}, \text{constraint})\). Each Parameter
belongs to a certain Parametrizable Element and the parameter name has to be unique within its namespace.

To interconnect components ports and connectors roles a Connection element is defined in the meta-model. The definition ensures that Inports can only be bound to Receiver roles and Outports can only be bound to Sender roles with compatible signatures. A signature \( \sigma_1 \) is type compatible with \( \sigma_2 \) if \( \sigma_1 \subseteq \sigma_2 \). Moreover, a set of rules for ensuring well formed adaptation graphs are defined, e.g. that all ports and roles within a graph have to be bound and for each parameter a value mapping has to be defined.

2.3 Modeling context and meta-information for adaptation control

Context represents information about the state and changes of an applications execution environment. In the meta model we abstract from the gathering of contextual information as well as from context fusion. Any context information is represented by an abstract Sensor element, which models the access to this contextual information assuming the availability of a context service providing all the required information. A Sensor is defined by a unique sensor name, a description, a data type, an optional constraint statement, a unit, a default value, and its behaviour; denoted by \( \text{sensorname} = (\text{description}, \text{type}, \text{unit}, \text{defaultvalue}, \text{constraint}, \text{behaviour}) \) (e.g. \( \text{CS}_1 = (\text{device.display.x}, \text{Integer}, \text{"pixel"}, 1280, " > 0", \text{passive}) \)). The behaviour of a Sensor can be either active, passive or constant. According to the defined behaviour, the sensor value is gathered by pull, push, or by definition.

The Sensor values representing context have to be mapped to component and connector parameters to control the adaptation of the graph. Usually a one to one mapping between context information and parameters do not exist. Instead complex mapping functions are required, especially if the parameterization of several components and connectors depends on the same Sensors. Mapping Functions enable the modelling of the interrelations between context and parameters. The element Mapping Functon is derived from Sensor as well as from Parameterizable Element. Thus, the result of a Mapping Function can be used as Sensor value, i.e. as input for other Mapping Functions as well as for the definition of parameters. This enables a stepwise calculation of parameter settings based on context. Especially, standard mappings can be defined and reused. Furthermore, interrelations between context and parameter settings have to explicitly defined by application developers and can be visualized to increase the transparancy of adaptation control.

Additionally to context, meta-information about the data objects and the processing state of the adaptation graph can be used to control adaptation. This information can also be explicitly modeled based on the elements Meta Data, Annotation and Interception Point. The element Meta Data represents information available at a certain Interception Point. This can be information about the data object currently passing the Interception Point (i.e. the resolution of an image) or component internal information (i.e. elements in the queue of a Queing component). Meta Data as well as Sensor are derived from the element Typed Value. Thus, Meta Data and Sensors can be used to set parameters and as input of Mapping Functions.

Annotations represent meta-information available at a certain Interception Point which is attached to data objects. Using annotations, interrelated components within a graph can exchange information (e.g. for parametrization) in the direction of data flow. This can for instance be information about a complex data structure shared by a Decomposition and a Composition component. For instance, the Decomposition component can annotate each resulting data object with structure information allowing the Composition component to restore this structure. Annotations can be written or read at Interaction Points. Again, the annotated information as well as the components writing and reading these annotations can be explicitly modeled based on the meta model.

2.4 Composite graph structures

The hierarchical composition of components and connectors enables the description of the application structure at different levels of detail. Complex components and connectors can be represented as a simple composite element hiding the details and enabling the development of large applications. The other way around, components and connectors can be decomposed to explicitly model their internal structure and behavior. The meta model allows the definition of composite components and connectors based on the mapping of elements between two levels of hierarchy. Thus, ports, roles, parameters and the elements for modeling adaptation control can be mapped from a graph structure.
to a composite element, similar to the concept of "Representation Maps" described in ACME [4].

3 Example application model

Figure 1 illustrates a sample adaptation graph for the modelling of an adaptive email application. The figure is based on the graphical representation defined for the elements of the meta model using UML stereotypes. The graph consists of a Mail Server (i.e. the data source) and a Mail Client (i.e. the data sink) component which exchange email messages. The Mail Server component is extended by a component for requesting and pro-actively sending email messages to the Outport O_{1.1}. Moreover, the Mail Client component can request messages from the mail server by sending request messages via Outport O_{5.1}. These messages can request for instance all new messages on the server but also selected messages according to rules encoded in the request (e.g. all messages from sender X or all messages not older than Y days).

Email messages sent via Outport O_{1.1} are mediated to Inport I_{2.1} of the Headerfilter component. This component analyses the header of the email messages according to a set of filter statements defined via parameter P_{2.1}. This parameter is of type Set(string)\(^1\) A single string represents a statement of the form "header: value" The parameter value \( P_{2.1} = \text{Set}(["from: thomas.springer@tu-dresden.de", "subject: project"]) \) defines, that only messages with the given sender address or the subject "project" pass the filter, i.e. are mediated to Outport O_{2.1}. All other messages are send to Outport O_{2.2} and are explicitly removed from the adaptation graph by the Reject component acting as a data sink. The interpretation of the value of parameter \( P_{2.1} \) is specific to the implementation of the Headerfilter component. For instance, the defined filter statements could be interpreted as preclusive filters. Similarly, rules for spam filtering could be defined for the component.

The data objects passing the filter are mediated to the Inport I_{3.1} and are adapted according to the parameterization of the Message Adaptation component. The resulting email messages are sent to the mail client via the Disconnected Operation component, i.e. via ports O_{3.1}, I_{4.1}, O_{4.1}, and I_{5.1}. The Message Adaptation and Disconnected Operation components are composite components and contain complex adaptation graphs.

Figure 2 depicts a sample graph for the decomposition and composition of email messages, part of the composite component Message Adaptation of the graph shown in figure 1. The Decomposition component decomposes incoming messages into message headers, body and attachments and mediates these parts to different Outports (i.e. \( O_{1.1} = \text{headers}, O_{2.1} = \text{body}, \) and \( O_{3.1} = \text{attachments} \)). The headers and attachments are sent as a sequence of data objects to the Outports. Each resulting message part is also annotated with composition information (i.e. ObjectId and SubId) at the postprocessing Interception Point of the ports \( O_{1.1}, O_{2.1}, \) and \( O_{3.1} \). Moreover, a data object containing the structure description is created with the according ObjectId. The Outports of the Decomposition component are interconnected with the Sender roles of a Multiplex connector which merges the different object streams to one stream mediated to the Inport I_{3.1} of the Composition component. To include further adaptation operations, the according Output ports of the Decomposition component have to be interconnected with appropriate adaptation components (e.g. for filtering attachments according to a maximum threshold size or downscaling image attachments). After the sequentialization of the data objects within the Multiplex connector, the data objects are collected within the Composition component based on the same ObjectId taken from the annotation. If all data objects belonging to the same ObjectId are arrived at the component, the structure of the email message is reconstructed based on the structure description data object. Thus, at the postprocessing Interception Point of the Decomposition component the annotation "composition information" is written and at the preprocessing Interception Point the annotation is read.

Based on the Disconnected Operation component the protocol used for transferring data objects can be adapted as well. The parameter \( P_{4.1} \) is mapped to a lower level of hierarchy to a parameter of an Adaptive TCP connector which allows the selection of various available TCP protocols. Therefore, five Sensors are defined, representing information about the access network (\( CS_{1} \) to \( CS_{5} \)) and the location of the user (\( CS_{5} \)). The definition of the Sensors and the Mapping Function \( MF_{1} \) is shown in figure 3. The Adaptive TCP connector contains pathes for sending via TCP, I-TCP, or TCP-HC. I-TCP is based on the split connection approach using two separate TCP connections and an additional component on the base station to copy packets between the two connections. For the wireless link an optimized protocol (SNR, Selective Repeat Protocol) is used to support wireless links with high error rates. TCP-HC, i.e.

\(^{1}\text{We have adopted the data types defined by OCL for the metamodel. The data type Set is a template type, denoted by Set(type). The notion Set(string) defines a data type representing a set of strings.}\)
TCP with header compression can be used to support asymmetric metrics. According to the Mapping Function $M_{F_1}$, I-TCP is selected if the last link is a wireless link with a packet error rate $> 5\%$ and the network is the home network, for which we assume that I-TCP is supported by the base stations. TCP-HP is selected if the data rate is higher than the data rate for the downlink, i.e. an asymmetric connection is used. Otherwise, standard TCP is selected.

4 Related Work

The pipeline design pattern [11] introduces the idea of loosely coupling components via pipes. The pattern was adopted as the foundation of the coupling of components via connectors in a loosely coupled way in our meta-model.

Architecture Description Languages (ADL, [9]) allow the design of software systems at architectural level. They introduce basic elements for architecture description, e.g. components, connectors, ports and roles, and allow an hierarchical composition (see e.g. [4]). We adopted some of these elements for our meta-model. However, ADLs focus on distributed systems and configurations are usually static. Exceptions are Darwin [3] and Rapide [7], both languages allow the definition of constraint changes of configurations. C2 [10] and Wright [1] support configuration changes at runtime. In all languages specific elements for modelling context, meta-information and parametrization of components and connectors for adaptation control are missing.

Several solutions allow the creation of adaption paths or graphs with focus on paths for transforming media objects [8, 2] or the dynamic instantiation of paths or graphs at runtime [5, 6]. Transformation operations are described by the input and output data types only to characterize their functionality. Adaptation graphs can also include components with unbound object types (e.g. for caching or queuing) and allow the explicit description of adaptation control based on context-to-parameter mappings.

5 Conclusion and future Work

The main advantages of the described meta-model approach are: the integration of operations for the adaptation of application data, communication and the application structure (integration), the modeling of adaptation operations as replaceable and reusable components and connectors (flexibility), the clear definition of adaptation operations, used contextual information and its mapping to adaptation control (transparency) and the independence of the meta-model from implementation platforms (platform independence). We have already applied the modeling approach to various application scenarios (e.g., an adaptive email for mobile devices with disconnection support and an adaptation framework for device-independent web-applications) to illustrate the feasibility of our approach. Furthermore, we have built a modeling tool based on Eclipse. Current work investigates the transformation of the meta-model to various platform specific models (e.g. EJB and OSGi) as well as model-to-code transformations. The transformations are specified based on Java code and QVT. Further work will be done to extend the modeling approach with additional elements, e.g. for describing comprehensive context models.

References