Construction and Execution of Adaptable Applications Using an Aspect-Oriented and Model Driven Approach

Sten A. Lundesgaard[1], Arnor Solberg[2,*], Jon Oldevik[2], Robert France[3], Jan Øyvind Aagedal[2], and Frank Eliassen[1]

[1] Simula Research Laboratory, Network and Distributed Systems,
P.O. Box 134, N-1325 Lysaker, Norway
{stena, frank}@simula.no
[2] SINTEF, ICT,
P.O. Box 124, N-0314 Oslo, Norway
{arnor.solberg, jon.oldevik, jan.aagedal}@sintef.no
[3] Colorado State University,
Fort Collins, CO-80532, USA
france@cs.colostate.edu

Abstract. Constructing and executing distributed applications that can adapt to their current operating context, in order to maintain or enhance Quality of Service (QoS) attribute levels, are complex tasks. Managing multiple, interacting QoS features is particularly difficult since these features tend to be distributed across the system and tangled with other features. The crosscutting nature of QoS features can make them difficult to evolve, and it can make it complicated to dynamically optimize with respect to provided QoS during execution. Furthermore, it complicates efficient construction of application variants that differ in their QoS characteristics to suit various execution contexts. This paper presents an aspect-oriented and model driven approach for constructing and a QoS-aware middleware for execution of QoS-sensitive applications. Aspect-oriented modeling techniques are used to separate QoS features from primary application logic, and for efficient specification of alternative application variants. Model driven engineering techniques are used to derive run-time representations of application variants from platform independent models. The developed middleware chooses the best variant according to the current operating context and the available resources.

1 Introduction

Distributed systems often execute in heterogeneous environments, in which the availability of resources such as bandwidth, memory, and computing power change over time. The increasing mobility and pervasiveness of computing systems require the consideration of the dynamic environment, when building suitable QoS features for maintaining desired QoS. Adaptive middleware addresses these challenges. It performs run-time configuration and adaptation by choosing between alternative application variants with similar functional properties but different QoS characteristics and

[*] Two first authors are in alphabetical order.

J. Indulska and K. Raymond (Eds.): DAIS 2007, LNCS 4531, pp. 76–89, 2007.

resource demands. Criteria for choosing an application variant are generally based on the context [1] or QoS characteristics [2][3].

Many concerns need to be considered when constructing alternative application variants, e.g., QoS preferences, context dependencies, and resource allocation. To manage this complexity, separation of concerns and support for defining and using suitable abstractions are needed. In Model Driven Engineering (MDE), abstractions and transformations between levels are used to manage complexity. For example, the Model Driven Architecture (MDA) [4] specifies three abstraction levels; a Computation Independent Model (CIM) describes the environment and specifies requirements; a Platform Independent Model (PIM) describes the parts that do not change from one platform to another; and a Platform Specific Model (PSM) includes descriptions of platform dependent parts. To further control the complexity of developing application variants that have similar functionality but differ in their QoS characteristics, mechanisms for separating crosscutting QoS features from the primary application logic are needed. Examples of QoS characteristics are security, integrity, robustness, and performance. Examples of corresponding QoS features are authentication, transaction control, error handling, and compression. Aspect-Oriented Software Development (AOSD) approaches [5]-[8] provide mechanisms for encapsulating crosscutting features. In the Aspect Oriented Modeling (AOM) approach presented in [8], crosscutting features are modeled as aspects and composed with the primary design model, to form integrated models.

This paper presents an approach for Construction and Execution of Adaptable applications (CEA-Frame). CEA-Frame integrates MDE and AOM techniques to model application variants in platform-independent terms and to automatically transform PIMs to PSMs. QoS features are separated from the primary functionality as aspect models and designed to fit particular operating contexts. For the execution we have developed a context- and QoS-aware dynamic middleware named QuAMobile, which identifies and chooses the application variant that is considered best for the current context and available resources. The alternative application variants are deployed using platform independent specifications, called *service plans* [11].

The separation of concern mechanisms in CEA-Frame improve the reusability of both design- and run-time artifacts through application-independent models of crosscutting QoS-features, and service plan specifications that separate meta-data from implementation code. Furthermore, modeling the QoS features separately in aspect models enables efficient representation of QoS variability from which a large number of application variants can be derived. The MDE based transformations make the transition from PIMs to PSMs faster, smoother and less error prone.

Sect. 2 presents the integrated construction and execution concepts, mechanisms and activities of CEA-Frame. In Sect. 3 the CEA-Frame is illustrated and validated using a live media streaming application example. Sect. 4 discusses related work. Sect. 5 draws some conclusions and outlines future work.

2 Construction and Execution of Adaptable Applications

CEA-Frame (Fig. 1) provides: i) methods for specification of application variants combining model driven and aspect-oriented modeling techniques, ii) mappings

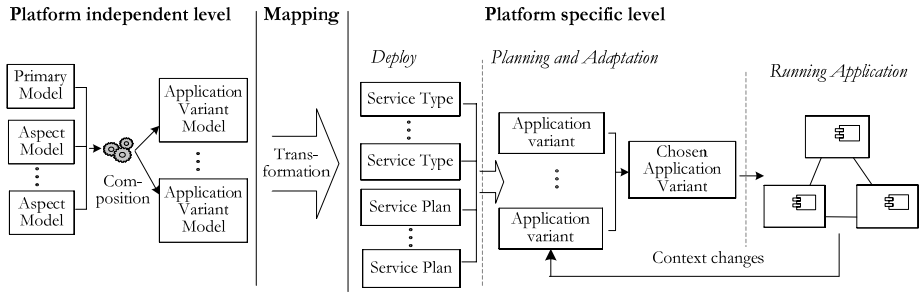

Fig. 1. Overview CEA-Frame

generating platform level constructs from platform independent specifications, and iii) a QoS-aware planning and adaptation supplied by the QuAMobile middleware.

At the platform independent level, a primary model and a set of aspect models are developed. Alternative application variants are obtained based on the following two mechanisms: i) compositions are used to derive application variants by composing the primary model with different subsets of the aspect models, and ii) variants of aspect models and primary models are described by means of model-based variability mechanisms such as specialization and parameterization. From the PIMs, service types in the form of Web-Service Description Language (WSDL) files and XML-based service plans are generated by our transformation engine. These mappings are implemented using the MOFScript Eclipse plug-in [10]. At the platform specific level, the QoS-aware planning process (in QuAMobile) uses the deployed service types and service plans to select the application variant that is considered best for the current context in order to meet the user's QoS preferences. This also includes checking context dependencies (e.g., run-time environment, communication technology, and storage facility dependencies), and predicting the end-to-end QoS according to the available resources (e.g., processing load, data rate and memory usage).

2.1 The Conceptual Service Model

The CEA-Frame defines *service, service type,* and *service plan* as central architectural concepts (see Fig. 2). A service type can be composed from a set of service types. An *application type* is a *service type*. Services realize service types and their meta-information is specified in *service plans*. Consequently, there may be different service plans for a service type. Services can be atomic or composite. Accordingly, there are atomic and composite plans. An atomic plan describes an atomic service, while a composite plan recursively describes a composite service by specifying the involved service types and the connections between them. In addition, both types of service plans contains: i) information about dependencies to context elements, ii) specification of the parameter configurations and iii) specification of the QoS characteristics. These are vital information for the QoS-aware planning and adaptation. It is tedious to

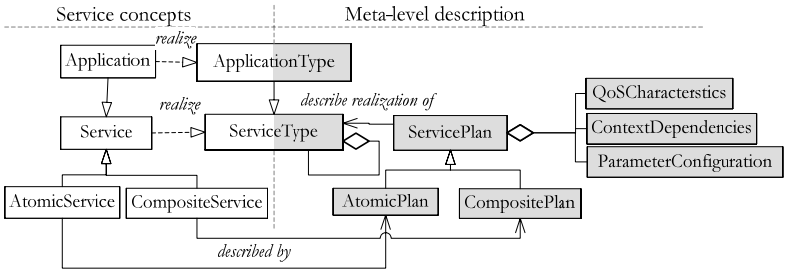

Fig. 2. The service and service plan concepts of CEA-Frame

develop the service plans manually, because many alternative application variants are required to support the different operating contexts of the application. In CEA-Frame the service plans are automatically generated from the more abstract PIMs. Service plans are further elaborated in [11].

2.2 Constructing Application Variants

The basis for the modeling in CEA-Frame is our Aspect-Oriented Model Driven Framework (AOMDF) [12], which combines aspect-oriented and model driven techniques to separate both vertical concerns such as technical platform, and user defined crosscutting concerns such as QoS. CEA-Frame extends AOMDF to support construction and execution of QoS-aware adaptable applications.

A design is expressed in terms of the following artifacts [7]: i) the *primary model (PM)* describes the application logic; ii) the *aspect models (AM)* describe crosscutting QoS features; iii) the *bindings* define where in the primary model the aspect models should be composed; and iv) the *composition directives* govern how aspect models are composed with primary models.

Before an aspect model can be composed with a primary model in an application domain, the aspect model must be instantiated in the context of the same application domain. An instantiation is obtained by binding elements in the aspect model to elements in the application domain. The result is called a context-specific aspect model. Context-specific aspect models and the primary model are composed to obtain an integrated design view [8]. Fig. 3 shows the modeling and mapping activities when constructing alternative application variants using CEA-Frame.

Starting at the *platform independent level*, the primary model is specified. Variability is specified using variability mechanisms provided in UML such as specialization, templates and multiplicity (e.g., "0..1" for optional elements). Then, QoS features are specified in aspect models. In our approach aspect models are reusable patterns that describe application specific QoS features when instantiated. In the composition, the aspects models are instantiated and composed with the primary model. An aspect model is instantiated by binding template parameters to actual values.

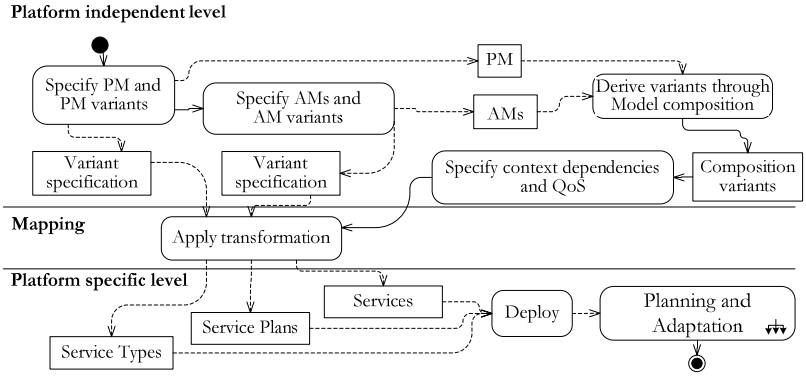

Fig. 3. CEA-Frame modeling and mapping activities

2.3 Execution of Adaptable Applications

In our implementation of CEA-Frame the distributed dynamic middleware QuAMobile and the Java virtual machine constitute the execution environment. QuAMobile implements a plug-in architecture for inserting domain specific managers: *service planner*, *context manager*, *resource manager*, *configuration manager*, and *adaptation manager* as depicted in Fig. 4. Service types and plans are interpreted during deployment using the Java Document Object Model (JDOM) open source parser. Service implementations reside in the *repository*, while service types and plans are published to the *broker*. During executing, service types and plans represent the meta-level model of the running application. This model is causally connected to the application, that is, any changes made to the meta-level causes corresponding changes in the application.

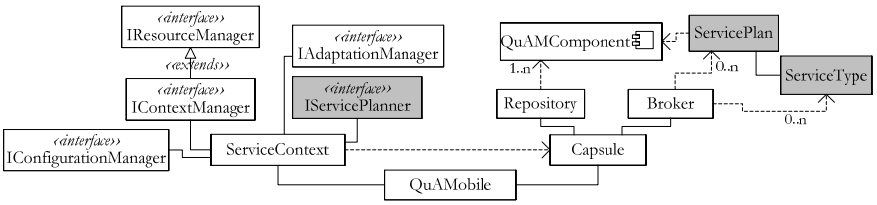

Fig. 4. QuAMobile core architecture

In dynamic heterogonous environments QoS guarantees can not be made. Instead QuAMobile re-plans and adapts the applications to meet the changes in context and resource availability. In the middleware the two plug-ins service planner and adaptation manager performs QoS-aware planning and adaptation. Service planning is a process that identifies suitable application variants for the context in which the application shall execute and choose the one that is considered most optimal with respect to the user's QoS preferences. The planning commences when the user (i.e., client software) submits a service request with user QoS preferences in the form of utility functions to the platform. In CEA-Frame, utility is a measure of usefulness and is

expressed by a real number in the range [0, 1], where 0 represents *useless* and 1 represents *as good as perfect*. Service planning is a four step process starting with i) identifying all the alternative application variants, ii) context dependency filtering, iii) QoS prediction, and iv) choosing the best suited variant according to the specified utility functions.

The adaptation mechanisms operate on a meta-level, where the service types and service plans are used for reasoning and altering the running application. When changes in the context are detected, i.e., there is updated context and resource information available, the service planner performs a re-planning of the running application. If another application variant matches the user's QoS preferences better, the middleware adapts the application. First, existing plans that constitute the meta-model of the running application are made available (reification). Then components involved in the adaptation are pushed to a safe-state (if this state is reachable), and changes are made to the meta-model. Lastly, the changes are absorbed by the application. Fig. 5 shows the activities involved in the execution of an adaptable application, and is a detailing of the planning and adaptation activity of Fig. 3.

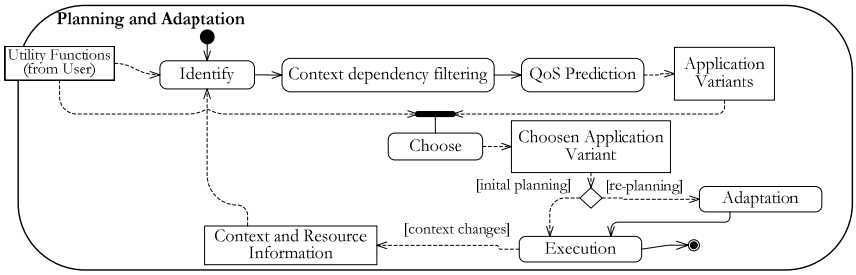

Fig. 5. Planning and adaptation activities

3 Illustrative Example

This section illustrates the CEA-Frame by describing the construction and execution of a live media streaming system. The system captures events (e.g., news and sports), encode onsite, and forward the media stream to streaming servers that the users access over the Internet (see Fig. 6). Users are mobile, and switch from Local Area Network (LAN) to a Wireless LAN (WLAN), and between WLAN subnets.

3.1 Modeling and Mapping

The illustrative example of the modeling and mapping process is structured according to the CEA-Frame activities depicted in Fig. 3.

Specify Primary Model and Primary Model Variants. The application level composite structure of the media streaming primary model is shown in Fig. 6.

SgnlCommunication initiates and controls the media stream on request from the *MediaPlayer*. *LiveMediaSrc* provides the video images, and *MediaStrmService* sends the stream to *MediaPlayer* through the *StrmCommunication* service. These services

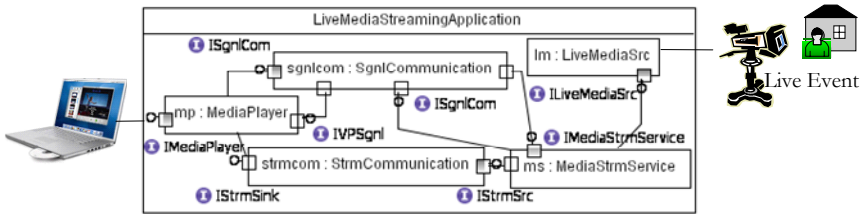

Fig. 6. Live media streaming system, application level composite structure

are all composite. In this example we will look into details of the *StrmCommunication* service and its variants (Fig. 7). This service has both alternative compositions and parameter configurations from which variants are derived, high-lightening variability mechanisms and variant derivation provided in CEA-Frame.

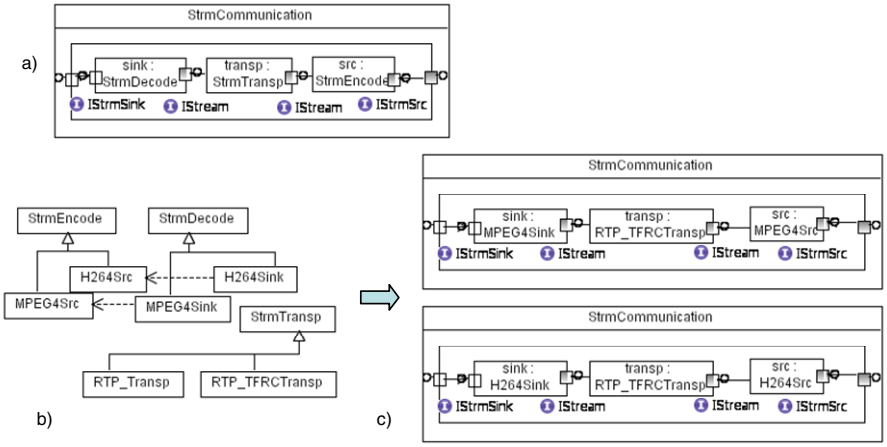

Fig. 7. a) primary model of *StrmCommunication* and b) variations of types

The types of the parts contained in the general *StrmCommunication* (Fig. 7a) are abstract and represent variation points. Possible variations of these types can be represented as a specialization hierarchy as shown in Fig. 7b. Here the allowed specializations for the encoder and decoder are MPEG-4 and H.262, and the allowed transport protocols are RTP and RTP_TFRC. Fig. 7c shows two of the four possible variants for this case. The dependency relationships in the specialization hierarchy ensure compliance for the source and sink of a particular variant.

Specify Aspect Models and Aspect Model variants. QoS features are specified in aspect models. For wireless communication bit errors represent an inherent problem. To ensure a satisfactory video quality, Forward Error Correction (FEC) algorithms can be used to minimize the effect of bit errors. Also, due to the handover and roaming between WLAN sub-nets, pre-fetching (using a buffer) can be used to reduce jitter. To improve smoothness and timeliness of the video when streaming over WLAN, the two aspect models depicted in Fig. 8a and Fig. 8b are specified.

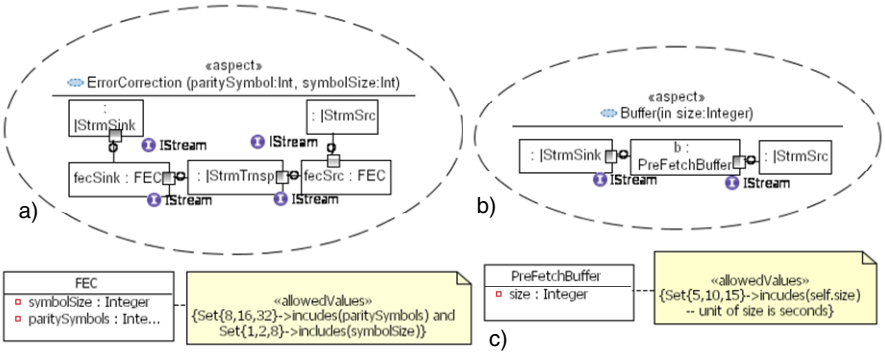

Fig. 8. Aspect Models a) error correction, b) pre-fetching, and c) allowed values

The aspect models are parameterized. The *allowedValues* stereotype is used to specify the values of these parameters for the particular application (Fig. 8c). For the *ErrorCorrection* and *Buffer* aspects, parameters that must be specified are buffer sizes, symbol sizes, and parity symbols. The set of combinations of these parameter configurations signify a corresponding set of aspect model variants (i.e., nine *Error-Correction* variants and three *Buffer* variants) with different QoS characteristics and resource demands. For example, increasing the values for the parity symbols and the symbol size increase the protection level of the error correction, but at the cost of CPU usage and start-up time.

Derive Variants Through Model Composition. The aspect models consist of template forms of composite structure diagrams, expressed using a template variant of the Role Based Meta-Modeling Language (RBML) [18]. RBML is a pattern description language which characterizes a family of UML models. The aspect templates are instantiated by binding template parameters to values. The parameters are marked using the symbol "|" (see the aspects models in Fig. 8). When the role binding is specified the primary model is composed with the aspect models according to specified composition rules.

We obtain four alternative compositions of the *StrmCommunication* service, two of which are shown in Fig. 9 (pre-fetching without FEC and usage of the primary model without including any aspects is not shown).

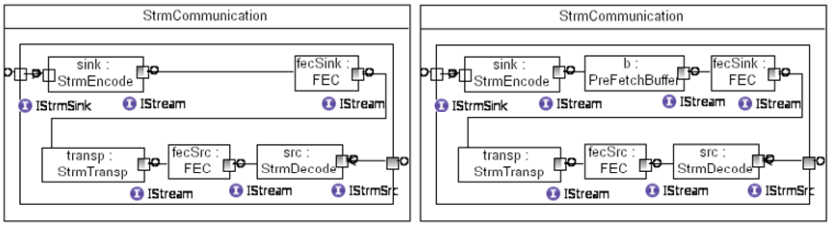

Fig. 9. Composition variants of *StrmCommunication*

Specify Context Dependencies and QoS. Applying CEA-Frame, application specific QoS characteristics, resources, and context elements need to be defined. The QoS characteristics and resources definitions in our example are based on the ISO/IEC 9126 QoS characteristics catalogue [14] and the General Resource Model (GRM) [15]. The specifications are modeled according to the guidelines of the UML profile for QoS standard [13]. A subset of QoS characteristics resource and context types used for the live media streaming application is shown in Fig. 10a.

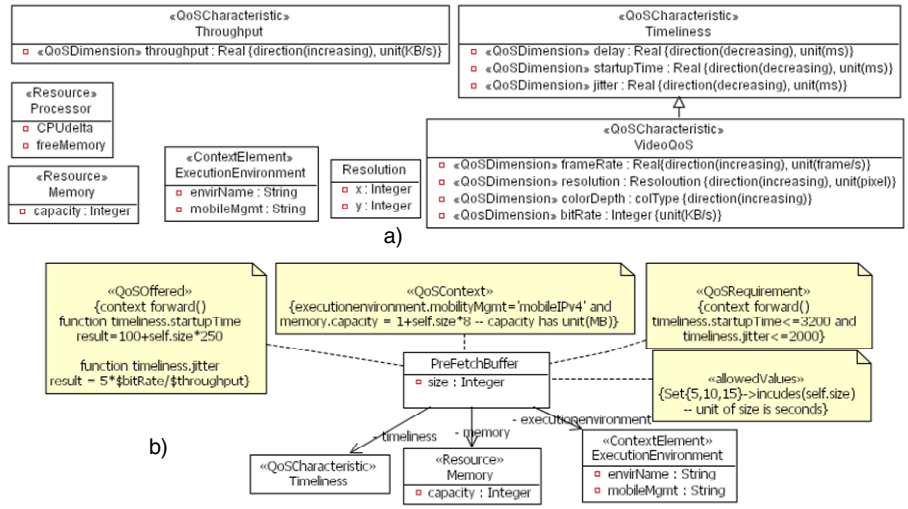

Fig. 10. a) QoS characteristics, resource, and context types, and b) context dependencies and QoS specification

The specified services are associated with context dependencies, QoS requirements, and QoS prediction functions. For context dependency specifications we use the stereotype *QoSContext.* The *QoSOffered* stereotype us used to specify predicted QoS. Both stereotypes are provided by the UML profile for QoS standard [13]. In addition we have defined the *QoSRequirement* stereotype, which is used to specify the QoS levels a service needs to fulfill, e.g., min and max values. A *QoSRequirement* specification is identified based on expected usage of the service.

QoSOffered specifies QoS prediction functions that the middleware uses to calculate the QoS for a given set of context and resource QoS values. For example, *StrmCommunication* is associated with functions that predicts a long start-up time when the *PreFetchBuffer* is part of the composition. However, when connected to WLAN these functions predict increase in the frame drop rate and jitter. When streaming live media (e.g., news and sport events), the user defined trade-off may be to have low start-up time as long as the frame drop rate is below a certain limit. Fig. 10b shows examples of application specific context and QoS specifications using the *QoSContext, QoSOffered,* and *QoSRequirements* stereotypes (the Object Constraint Language (OCL) is used for specification).

For composite services, *QoSOffered* is dependent on the QoS offered by its parts. Thus, QoS prediction in these cases need to take into account that the composite do not know what parts it consist of, since new parts can be added when composing aspect models. For example the offered *startupTime* for the *StrmCommunication* is a summation of the *startupTime* of its parts; consequently, the different composition variants will have different start-up times as expressed with the following OCL-based predictor function:

(*self.timeliness.startupTime = self.parts.collect(part:Property | part.type.feature->select (f:Feature | f.name = 'timeliness'))->collect (f:Feature | f.type.attribute->select (a:Property | a.name = 'startUpTime')) -> sum ()*).

Apply Transformation. To bridge the model and platform levels of the adaptable application, automated transformations are used for mapping UML models to application variants and service type specifications. The PIM transformation source consists of the four composed models derived from the compositions of aspect and primary models, two of these are shown in Fig. 9. We refer to these as the base compositions. Additional input to the transformation, for our example, is the specialization hierarchy specifying primary model variants, the allowed values associated with aspect models parameters determining aspect model variants, and context and QoS specifications. From these a total number of 432 alternative variants of the *StrmCommunication* service can be derived (4 base compositions*4 primary model variants*9 ErrorCorrection aspect model variants*3 Buffer aspect model variants). Thus, this specific case illustrates the general challenge that the set of variants can be very large. To avoid a large number of variants, one can identify combinations of the parameter values that imply significant variation in the end-to-end QoS characteristics. Only these are deployed as possible run-time variants. In the example this led to a reduction of combinations of the three different sets of parity symbol lengths and symbol sizes for the FEC service from nine combination to the following three value pairs: {8, 1}, {16, 2}, or {32, 8}). The number of derived PSM variants then becomes 144.

The transformations have been implemented using the MOFScript Eclipse plug-in [10]. MOFScript was one of the proposed languages in the standardization process of MOF Model to Text Transformations, which has been adopted and is now in its finalizing stage [9]. In general, the implemented transformations map CEA-Frame PIM concepts such as *QoScharacteristics*, *QoScontext*, and service specifications in primary and aspect models, to CEA-Frame PSM concepts such as service types, service plans, and service realizations.

3.2 QoS-Aware Planning and Adaptation

In our example QuAMobile is installed on a laptop and a streaming server. The installation creates a common service context that provides protocols for service discovery and context information sharing between the domain specific management plug-ins. The service planner residing on the streaming server is configured as master, i.e., centralized planning and local adaptation. To illustrate the QoS-aware planning and adaptation (tasks shown in Fig. 5) it is assumed that the user has the laptop connected to the LAN. After some time the user disconnects and moves over to WLAN.

Deploy. Generated service types (WSDL), service plans, and components are deployed and published on the machine on which the service is to execute.

Identify. When the user requests access to the live streaming service, alternative application variants are synthesized from the published services and discovered service plans. QuAMobile identifies all of the 720 application variants (144 variants of the *StrmCommunication* and additional 5 variants of the *LiveMediaSrc* services, resulting in 720 variants of the *LiveMediaStreaming* application (see Fig. 6).

Context Dependency Filtering. Application variants that can not execute in the current operating context are filtered, by comparing gathered context information against the specified context dependencies (*QoSContext* in the composite models). In QuAMobile, it is the context manager that gathers and processes data about the context and makes information available to the service planner plug-in. For the identified application variants, it is the specified dependencies to the screen resolution that are caught by the context dependency filter, since three of the *LiveMediaSrcs* services require a screen with a higher resolution than what the laptop has. After context dependency filtering 288 variants remains.

QoS Prediction. End-to-end QoS characteristics are predicted using the specified functions (*QoSOffered* stereotype) in a bottom-up style, i.e., start by calculating the QoS of each atomic service and finishing of with the composite service. The QoS prediction functions are specified and deployed as text strings; hence, the expressions are calculated for each planning and adaptation process. Predicted QoS are checked against QoS requirements specified by the application developer (*QoSRequirement* stereotype).

Choose. Utility functions are used to specify the user's QoS preferences and tradeoff between user QoS dimensions, e.g., *start-up time* ≥ 0.6, *detail-level* ≥ 0.6, and *smoothness* ≥ 0.6. By using the provided utility functions (see Fig. 11Fig. 11) and the predicted QoS QuAMobile calculates the utility of the application variants and chooses the one, which i) meets the specified minimum utility values and ii) has the highest utility-to-user QoS ratio. When the laptop is connected to the LAN it is the application variant with the *StrmCommunication* composition without the *FEC* and *PreFetchBuffer* services that is chosen, i.e., the primary model as depicted in Fig. 7. This variant is selected since the increase in utility for the *detail level* and *smoothness* dimensions are small compared to the increase in *start-up time*.

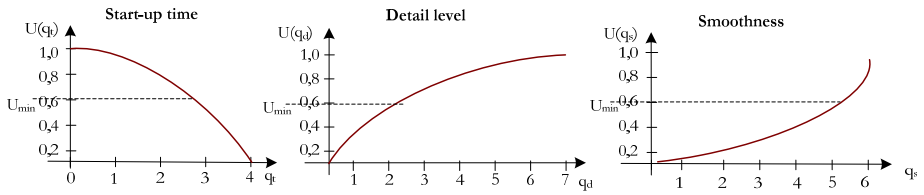

Fig. 11. Utility Functions

Execution. The application variant chosen is forwarded to the configuration managers on the laptop and streaming server. They create the components, configure, and bind them together. Execution of the initial application variant is like any other non- adaptable applications.

Adaptation. In our example the user disconnects the laptop from the LAN during the streaming of a particular news event, i.e., the streaming connection is moved over to WLAN by the *RTP_TFRCTransport* service. The context change makes the current application variant unsuitable, as the bit error rate associated with WLAN causes video frame to be dropped, i.e., too low utility for the *smoothness* dimension. QuAMobile therefore initiates re-planning and chooses the application variant which includes both the *FEC* and *PreFetchBuffer* services. This variant has a better balance between the *start-up time* and *smoothness* QoS dimensions. During adaptation service plans are used as a meta-model of the running application, enabling QuAMobile to make changes to the running application.

4 Related Work

Atkinson et al. [19] combine model driven and aspect-oriented development. Aspect-oriented techniques are used for refining specific aspects of the model (vertical separation of concern) by architecture stratification. This approach differs from the aspect approach employed in CEA-Frame, in that the aspects are not composed but represent refinements of a particular part of the model at higher level stratums. Thus, each stratum represents the whole system. Furthermore, Atkinson et al. define possible refinements as pattern-based aspects applied through framework instantiations. In our approach we use standard AOM and MDE mechanisms such as compositions and transformations.

MDE is used by Kulkarni et al. [16] for providing separation of concern between system concerns at both model and code level using templates and code weaving. This is similar to the AOM approach we employ, except that we use parameterized UML to specify aspects and perform model level composition avoiding the need for code level weaving. Clarke et al. [17] and Ray et al. [7] also apply aspects for separation of concern. The aspects models are weaved with application models, by adding and replacing both classes and operations. Kiczales et al. [5] employ aspect models for multiple concerns; functional behavior and crosscutting concerns. Hyper/J multiple models are integrated, making it possible to model alternative static application variants. CEA-Frame integrates aspect models with the application logic in a similar manner, but has additional support for parameter configuration, context, and QoS requirements. In addition, MDE principles are used to generate platform specific artifacts.

There are examples of adaptive middleware platforms that are combined with software engineering tools; $2K^{Q+}$ [2], QuO [3], and CoSMIC [20]. $2K^{Q+}$ provides an environment for specifying alternative service compositions, their QoS characteristics, and adaptation steps. A platform dependent compiler produces executable code for adapting the application. QuO introduces description languages for specifying QoS, which is compiled to executable code for monitoring QoS and controlling the interaction between distributed objects. CoSMIC is a MDE toolkit, which model compositions and QoS requirements at the platform level (a component based QoS-aware CORBA middleware). CEA-Frame addresses the same problems as $2K^{Q+}$, QuO, and CoSMIC, but at a platform independent level. This avoids specification of all possible context

and resource allocations, and enable integration of the framework with any development environment and middleware platform. Furthermore, CEA-frame pushes the task of identifying and choosing a variant to run-time, giving a larger solution space and higher probability of finding the best application configuration.

5 Conclusion and Future Work

The task of developing and operating distributed applications for heterogeneous dynamic environments is particularly difficult in the presence of multiple crosscutting QoS features. Our approach to tackle this problem is to separate the QoS features from the application logic, and place the responsibility of choosing and configuring the application at the middleware level.

CEA-Frame combines AOM and MDE techniques for efficient construction of a potentially large number of alternatives application variants needed due to the dynamics and heterogeneity of the execution environment. A context and QoS-aware middleware is developed to handle adaptation. The framework provides: i) methods and activity descriptions for constructing adaptable applications, ii) variability mechanisms using aspects and model composition as well as parameterized primary and aspect models, iii) separation of crosscutting QoS features iv) automatic model transformation and code generation, and v) a QoS-aware planning and adaptation process that configures and adapts the application to suit the operating context and resources available. The implementation of the framework has been validated by using it to construct and execute a live video streaming application.

The construction of application variants is accomplished by separating QoS variability specifications from variability of the primary model and the composition of the primary model with different subsets of the aspect models. The automatic transformations support efficient derivation of a large number of alternative application variants and eliminate tedious error-prone manual implementations. At the platform specific level separating specifications of the alternative application variants and their QoS characteristics (using the service plan concept) improves reusability of the services. All information needed for the middleware to filter, order, and choose a suitable application variant, is generated from platform independent models. CEA-Frame is based on standards such as the UML profile for QoS [13], GRM [15], ISO/IEC 9126 [14], and MOF Model to Text [9].

To develop the CEA-Frame, we will work further on the model composition techniques and related tool support. We are also working on OCL-based templates that are easier to work with and more readable.

References

1. Capra, L., Emmerich, W., Mascolo, C.: CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications. IEEE Trans. on Software Engineering 29(10), 929–945 (2003)
2. Nahrstedt, K., Xu, D., Wichadakul, D., Baochun, L.: QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments. IEEE Communications Magazine 39(11), 140–148 (2001)

3. Loyall, J., Bakken, D., Schantz, R., Zinky, J., Karr, D., Vanegas, R., Anderson, K.: QoS Aspect Languages and Their Runtime Integration. In: O'Hallaron, D.R. (ed.) LCR 1998. LNCS, vol. 1511, pp. 303–318. Springer, Heidelberg (1998)
4. OMG, MDA TM Guide v1.0.1, http://www.omg.org/docs/omg/03-06-01pdf
5. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingier, J., Irwin, J.: Aspect-Oriented Programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–241. Springer, Heidelberg (1997)
6. Ossher, H., Tarr, P.: Using Multidimensional Separation of Concerns to (Re)shape evolving Software. Communications of the ACM 44(10), 43–50 (2001)
7. Ray, I., France, R., Li, N., Georg, G.: An Aspect-Based Approach to Modeling Access Control Concerns. Journal of Info. and Software Tech. 46(9), 575–587 (2004)
8. France, R., Ray, I., Georg, G., Ghosh, S.: An aspect-oriented approach to design modeling. IEE Proc. Software, vol. 151(4) (2004)
9. OMG: MOF Models to Text Transformation Language Final Adopted Specification. Technical report, OMG document ptc/06-11-01 (2006)
10. MOFScript Eclipse plug-in, http://www.modelbased.net/mofscript
11. Lundesgaard, S., Lund, K., Eliassen, F.: Utilising Alternative Application Configurations in Context- and QoS-aware Mobile Middleware. In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 228–241. Springer, Heidelberg (2006)
12. Simmonds, D., Solberg, A., Reddy, R., France, R., Ghosh, S.: An Aspect Oriented Model Driven Framework. In: Proc. the Enterprise Distributed Object Computing Conference, pp. 119–130 (2005)
13. UML profile for modeling QoS and Fault Tolerance characteristics and Mechanisms. Adopted standard, OMG May 2005, Document ptc/05-05-02 (2005)
14. ISO/IEC JTC1/SC7, 1999a, Information Technology -Software product quality -Part 1: Quality model, ISO/IEC, Report: 9126-1
15. Object Management Group, UML Profile for Schedulability, Performance, and Time Specification, ad/2000-08-04 (2002)
16. Kulkarni, V., Reddy, S.: Separation of Concerns in Model-driven Development. IEEE Software 20(5), 64–69 (2003)
17. Clarke, S., Harrison, W., Ossher, H., Tarr, P.: Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code. In: Proc. of 14th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Application, pp. 325–339 (1999)
18. France, R.B., Kim, D., Ghosh, S., Song, E.: A UML-Based Pattern Specification Technique. IEEE Trans. on Software Eng. 30(3), 193–206 (2004)
19. Atkinson, C., Kühne, T.: Aspect-Oriented Development with Stratified Frameworks. IEEE Software 20(1), 81–89 (2003)
20. Gokhale, A., Balasubramanian, K., Krishna, A., Balasubramanian, J., Edwards, G., Deng, G., Turkay, E., Parsons, J., Schimdt, D.: Model Driven Middleware: A New Paradigm for Developing Distributed Real-time Embedded Systems. Science of Computer programming (2005)