Addressing the Middleware Configuration Challenges using Model-based Techniques

Emre Turkay
Institute for Software Integrated Systems
Vanderbilt University
Box 1829, Station B
Nashville, TN 37235
emre.turkay@vanderbilt.edu

Aniruddha Gokhale
Institute for Software Integrated Systems
Vanderbilt University
Box 1829, Station B
Nashville, TN 37235
a.gokhale@vanderbilt.edu

ABSTRACT
Component middleware, such as J2EE, .Net and CORBA Component Model (CCM) have been increasingly used to develop and deploy large-scale distributed systems in different domains, including defense, enterprise, avionics and industrial process control. The applications in each of these domains require different levels and types of quality of service (QoS) guarantees from their underlying component middleware. In an effort to support a large number of applications, therefore, component middleware developers provide enormous flexibility in the way the middleware can be configured and fine-tuned for the target application. Ad hoc techniques used to configure the middleware are tedious and error-prone.

This paper describes a novel scheme we are using based on model-based systems engineering to address the concerns of complex middleware configuration. We present a modeling paradigm called Options Configuration Modeling Language we have used in the context of configuring a QoS-enabled CORBA component middleware.

Keywords: Component middleware, Model-driven Software Synthesis, Middleware Configuration.

1. INTRODUCTION
The emergence of next-generation large-scale, distributed and quality-of-service sensitive (LDQoSS) systems, such as internet-wide immersive tools for telemedicine and scientific applications, distributed mission training for defense, or time-sensitive, large-volume online stock trading in financial markets, stems from rapid technological advances in networking, hardware, storage, and component middleware technologies.

For example, networking technologies such as DiffServ [1] and Multi-Protocol Label Switching [2] are enabling network service providers to provision and deliver network-level quality-of-service (QoS) to LDQoSS systems. Complementing the networking advances are the substantial amount of R&D efforts that have focused on developing standards-based off-the-shelf component middleware, such as CORBA Component Model (CCM) [3], J2EE [4] and .Net [5] to support these LDQoSS systems.

Component middleware encapsulates specific services or sets of services to provide reusable building blocks that can be composed to develop LDQoSS systems rapidly and robustly than those built entirely from scratch. In particular, component middleware offers the following reusable capabilities:

- **Horizontal infrastructure services**, such as request brokers
- **Vertical models of domain concepts**, such as common semantics for higher-level reusable component services, and
- **Connector mechanisms between components**, such as remote method invocations or message passing.

Context: Customizing the flexible component middleware for LDQoSS systems
To be able to support the large variety of component middleware-based LDQoSS systems, therefore, developers of QoS-enabled component middleware provide maximum flexibility to enable LDQoSS system developers to configure and fine-tune the performance of component middleware appropriately at multiple levels, including the request broker, reusable services and message passing mechanisms.

Challenges: Choosing the right set of middleware configuration parameters for LDQoSS systems
The flexibility provided by component middleware developers is manifested in a large number of configuration parameters at the multiple levels described earlier. Examples of these configuration parameters include options to decide the internal request buffering strategies, the request demultiplexing and dispatching strategies, the data marshaling strategies, the appropriate concurrency models to be used, the end-to-end network connection management strategies, and the end-to-end priority propagation strategies, among others. This large number of configuration options incurs a high degree of complexity for LDQoSS system developers.
in making the right choices of the configuration options for their systems. This problem is further exacerbated by the fact that not all combinations of configuration parameters and their values form a semantically compatible set that can be supported by the component middleware. It therefore requires a high level of expertise and understanding of the component middleware to determine the optimum set of semantically compatible configurations to be used. Ad hoc techniques based on manually choosing these configuration parameters is tedious and error-prone, and has no scientific basis for analytically proving the correctness of the configured end system.

**Solution Approach: Use model-driven techniques to resolve middleware configuration challenges**

To address the challenges described above requires principled, analytically and empirically proven methods to configure and validate component middleware for LDQoSS systems. These methods must enforce the physical constraints of LDQoSS systems, as well as satisfy the system’s stringent QoS requirements. Model-based techniques hold promise in resolving these challenges since these technique are amenable to model checking, validation and verification. Section 2 describes how we are using model-based techniques to resolve the component middleware configuration challenges to support LDQoSS systems.

**2. RESOLVING THE MIDDLEWARE CONFIGURATION CHALLENGES VIA OCML**

Section 1 describes the challenges incurred in configuring component middleware appropriately to support LDQoSS systems. This section presents our R&D based on model-driven tools to address these challenges. We describe the Options Configuration Modeling Language (OCML) tool that is part of our Model-Driven Middleware (MDM) [6] toolchain called CoSMIC [7]. Model-Driven Middleware combines the strengths of modeling and QoS-enabled component middleware to support LDQoSS systems.

The Options Configuration Modeling Language (OCML) is a modeling language we have developed using the Generic Modeling Environment (GME) [8] to address the middleware configuration challenges. OCML is used to define the constraints and dependencies on the options used to customize the component middleware. Figure 1 depicts the OCML workflow diagram showcasing the dual use of the OCML tool that can be used both by component middleware developers and LDQoSS system developers. Section 2.1 describes the elements of OCML. Section 2.2 explains the dual use of the OCML tool.

**2.1 Language Definition**

This section describes the artifacts of the OCML language including the syntax, semantics, constraints and generative tools.

**2.1.1 Syntactic definition**

The OCML meta-model defines two sets of syntactic elements: (1) The hierarchical organization of the options a LDQoSS system will require and (2) the option dependency rules, which constrains the available combination of these options. In the following we describe each syntactic element of OCML.
The rule definition elements are simply the logical values associated with each other to generate visual logical expressions. The logical operators and rules also transform any kind of option based logical associations into the bool associations. For example, an and operation having a value of equality association with an integer option can further be connected to other logical operators or rules with a bool association. Rules also serve as a container for other rules. If a rule model contains only other rules and no options at all, it is implemented as a pure container class and without a requirement of an and operator it assumes all the options contained are connected with an and operation.

Options, rules and logical operators should be associated with the associations listed above to define complete logical expressions. The logical meaning of each association is described below;

- **Bool associations** are used to associate a logical operator with another logical operator. They are used to define complex logical expressions. For example, an and operator can be inverted with a not operator by adding an association where the source of the association is the and operator and the destination is the not operator.
- To build a rule on the condition of an option is presented at the application configuration selection association is used. Association of the selection association with an option and a logical expression means if that option is selected then the specific port of the logical operation should be set to true otherwise false.
- **Value equality associations** are used to check if the value of a string or an integer option is equal to a given constant value.
- The purpose of the existence of option equality associations is to provide a dynamic equality check. To provide a complete logical expression with the option equality association a connector element should be used. Connector element simply has the logical value true if all the associated options have the same value.
- **Range association** is specific for the numerical options and checks if the value of the given option is in the range of [min, max] where min and max are the attributes of the connection.

2.1.4 Constraints
Complementary to the syntactic rules defined in the OCML metamodel are some constraints. The modeling paradigm’s constraint checking ensures the models meet the specified constraints thereby validating the models for correctness. The models are also required to conform to the following constraints:

- A connection can be associated with only one type of option. For example, Integer options and String options cannot be associated with the same connection.
- Connections are exceptional logical expressions *i.e.*, they cannot be destinations of bool associations. Their usage is restricted to option equality checking.
- Two different enum value options contained by the same enum option must not be provided directly or
indirectly to an and operator. Although the other constraint checks are handled with the OCL statements in the meta-model definition, this check is handled at the model interpretation phase.

2.1.5 OCML Generative Tools

OCML generative tool made up of a model interpreter generates two different outputs. These include documentation of all supported options in HTML format and the Configuration File Generator source code, which are described below.

Documentation generation

The generated HTML documentation includes information about every option and cross references for the dependencies. The OCML meta-model contains an option paradigm where options are hierarchically categorized into option categories. Every option category and the options themselves contain description attribute, which can include hyper-text information. The HTML documentation contains the collection of these descriptions in a human readable format when rendered within a HTML browser. The rules paradigm of an OCML model includes all the dependencies of the options. The documentation also displays the cross-references for the dependent objects and together with the textual representation of the rules.

Configuration file generator

The second output generated by the OCML interpreter is the source code of the Configuration File Generator (CFG) application. CFG is a graphical interface for describing the configuration of the application domain of the OCML model. The CFG application provides an easy to use features as listed below:

- Easy navigation through all the options the application provides and selected with a single-click.
- Prevents the generation of invalid/unoptimized configuration by providing an on-line automatic constraint manager.
- The UI environment displays the generated documentation and automatically navigates through this document so that the user can easily see the information about the option which is currently in focus.
- The user interface is platform independent.

2.2 Dual Use of OCML tool

OCML has two phases, which are used by both the application developer and the middleware developer. The middleware developer uses OCML to model the options in categorical order and also defines the dependencies of these options as rules. For this stage OCML tool is used as a modeling language to create the model generated by the middleware developer. OCML also provides an interpreter for the designed model, which when executed, generates the CFG and the documentation. Both the CFG and the documentation is generated according to the model defined by the middleware developer. The application developer uses the CFG and the generated documentation to set up the configuration of a specific application, which is validated against the rules modeled by the middleware developer. This dual use of OCML tool is represented in Figure 1.

2.3 OCML Use Case Scenario

OCML is initially designed to be a modeling tool for the configuration of the CIAO [9] component middleware configuration options, however it is generic enough to be used as a modeling tool for different tools and libraries. However CIAO configuration options provides a good example for demonstrating OCML.

The CIAO options are categorically divided into four sections as simple and advanced resource factories, server strategy factory, and client strategy factory. All these categories are defined as different Option categories and the options modeled within them hierarchically.

A sample rules governing options and expressed textually is shown below.

- If the CIAO's ORBAllowReactivationOfSystemIds is set to value 0 then ORBActiveHintsInIds cannot be declared and therefore cannot have a specific value.
- If the ORBConnectionPurgingStrategy is set to value NULL, then neither ORBConnectionCacheMax nor ORBConnectionCachePurgePercentage can be declared and can have a specific value.

After the CIAO developers design the models for CIAO options and the rules explained above, the model is interpreted and the interpretation process generates the CIAO-specific CFG and documentation.

Further, the application developer uses the CFG generated in the previous process to define the configuration set for her specific application. The CFG provides online help when the application configuration is done and it also restricts the user from specifying configurations which conflict with the rules defined by the middleware developer.

3. CONCLUSIONS

Component based QoS enabled middleware provides solutions for the various aspects of the application development and deployment process and also provides policies and mechanisms for provisioning and enforcing large-scale DRE application QoS requirements.

CoSMIC is developed to provide solution for the complexity of choosing syntactically and semantically compatible set of configurations for a specific application which uses QoS enabled middleware.

OCML is developed as a part of the CoSMIC MDM tool suite. While OCML defines a language for options configuration modeling, it brings out a solution to resolve the complexity of middleware configuration.

OCML project is a work in progress and latest information and source code can be obtained from www.dre.vanderbilt.edu/~turkaye/ocml. The CoSMIC MDM tool suite available for download at www.dre.vanderbilt.edu/cosmic. CoSMIC tool suite is developed in association with the CIAO component middleware, which is available for download at www.dre.vanderbilt.edu/CIAO.
4. REFERENCES


