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

Balachandran Natarajan
Institute for Software Integrated Systems
Vanderbilt University
Box 1829, Station B
Nashville, TN 37235
b.natarajan@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 different 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. Application developers, however, resort to ad hoc techniques to configure the middleware, which 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 (OCML) we have used in the context of configuring a QoS-enabled CORBA component middleware.

Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development:Modeling Paradigm; C.3 [**Special-purpose and Application-based Systems**]: Real-time Systems:Event-based Systems; D.2.2 [**Software Engineering**]: Design Tools and Techniques:Model driven

Keywords

Component middleware, Model-driven Software Synthesis, Middleware Configuration

*This work was sponsored in part by AFRL Contract#F33615-03-C-4112 for DARPA PCES Program and grant from Siemens

ACM Southeast Conference '04, April 2-3, 2004, Huntsville, Alabama, USA.

1. INTRODUCTION

1.1 Emerging Trends

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.

1.2 Challenges Provisioning LDQoSS Systems

This section outlines the middleware configuration challenges faced by domain-specific LDQoSS system developers and provides our solution that address these challenges.

Context: Customizing the flexible component middleware for LDQoSS systems

To be able to support the large variety of component middleware-based LDQoSS systems 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 optimal 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 is an emerging paradigm that 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.

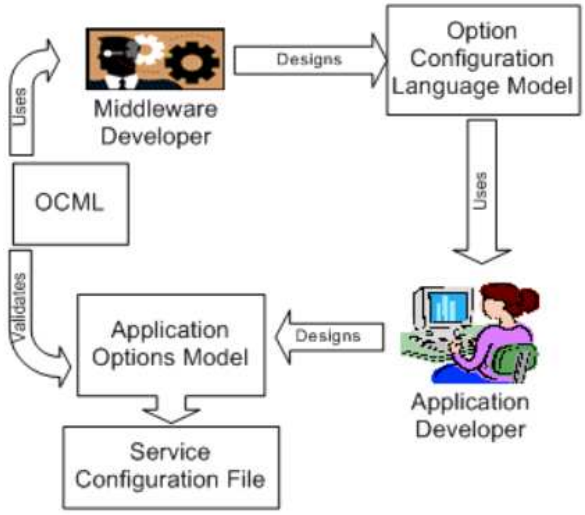

Figure 1: OCML Workflow

2.1 Language Definition

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

2.1.1 Syntactic definition

The OCML language defines two sets of syntactic elements: (1) The hierarchical organization of the middleware configuration 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.

Option Category Elements

The modeling elements used in the hierarchical organization of the options are listed and described below together with their corresponding icons we used in the GME modeling environment:

Option category Middleware configuration options are arranged hierarchically within the different option categories. An option category may include other option categories recursively. All the options have a default value. Moreover, the option category and all the options can have a description attribute, which is a textual description of that category or the option.

Numeric option Options having an integer numeric value are named as integer options. An integer option can be limited in range when it will have maximum and minimum value attributes.

String option Options having an alphanumeric value are named as string options .

Enum option Options that can be selected from a set of values are named as enum options.

Enum option value The enumerated values that can be contained in enum options are represented with enum option value elements. One of the enum option values in the same enum option group can have a default attribute set to *true*, which makes it the default value of the associated enum option.

Flag option The flag option can be set to a combination of the values from the given value set.

Flag option value Each flag element for a flag option is defined as a flag option value.

Rule Definition Elements

A rule definition element is a combination of logical operations. A rule can contain other rules or references to a set of options associated via some combinations of logical operators that are explained below.

And operator Logical "and" operator.

Or operator Logical "or" operator.

Not operator Logical "not" operator. Not operator can have only one input argument.

Implies operator The outputs should be satisfied if the inputs are satisfied.

If and only if operator If the input side is satisfied, then the output must be satisfied too (and vice versa).

Connector A connector is used to associate two or more integer or string options with the equality association.

Associations

The logical operators and options can be bound to each other via *associations* that are used to build legal and semantically meaningful rules for options composition. There are five kinds of associations in OCML as described below. These associations are contained by the rule elements.

Bool association All the logical operators except connector and rules can be associated with the bool association. Connections can only be sources of the bool associations.

Selection association All the options can be sources of selection associations. They are associated with the logical operators and rules except connectors.

Value equality associations (string and integer) String and integer options can be associated with rules and logic operators, except connector, with the value equality association. Value equality associations have value attributes, which have the same type with the source option's type (string, numerical and boolean).

Option equality associations (string and integer) A set of string, or integer options can be associated with a connector via an option equality association. A connector can only be associated with the same type of options.

Range association Numeric options can be associated with rules and logic operators, except connector, with the range association. Range association has two attributes – max and min.

2.1.2 Semantics

The set of options is a collection of configuration parameters for a specific application domain. They are provided in a structured form so that they can be used for two purposes: (1) the description attributes of each option is used in the document generation process and (2) they are referenced in the rule definition section and their references are connected with the logical operators to form meaningful logical expressions (*i.e.*, rules).

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. For example, if a rule model contains only other rules and no options at all, then it is implemented as a pure container class. Moreover, 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 *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.

2.1.3 Constraints

Complementing the syntactic rules defined in the OCML language 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 language definition, this check is handled at the model interpretation phase.

2.1.4 OCML Generative Tools

The 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 tool contains an option paradigm where options are hierarchically categorized into option categories. Every option category and the options themselves contain a description attribute, which can include hypertext 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 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. The CFG is used to generate the XML formatted configuration file of a specific application which uses the tools modeled with OCML. CFG allows users to select the possible options conforming to the defined rules in the OCML model rule paradigm. CFG contains two dynamically created parts; (1) options hierarchy data structure and (2) rule validation functions. The CFG application provides an easy to use feature 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 LDQoSS system developer and the middleware developer. First the middleware developer uses OCML to model the options, categorize them in a hierarchical order and define the rules governing their dependencies. For this stage the 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 LDQoSS system developer uses the CFG and the generated documentation to set up the configuration of the 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

In its current form OCML has been 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 middleware platforms.

This section illustrates through use case scenarios how OCML is used to model CIAO configuration options. 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 are modeled within them hierarchically.

A sample rules governing option dependencies and expressed textually is shown below:

- Figure 2 depicts a rule for CIAO's `ORBAllowReactivationOfSystemIds`. If this option is is set to value 0 then the `ORBActiveHintsInIds` option cannot be declared and therefore cannot have a specific value.
- Figure 3 show how when the `ORBConnectionPurgingStrategy` is set to value `NULL`, then neither `ORBConnectionCacheMax` nor `ORBConnectionCachePurgePercentage` can be declared and can have a specific value.

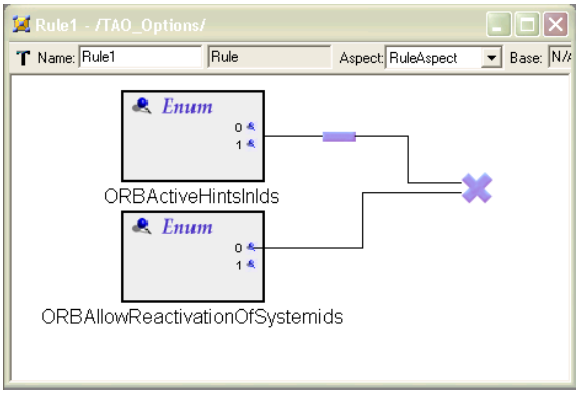

Figure 2: OCML Rule Example 1

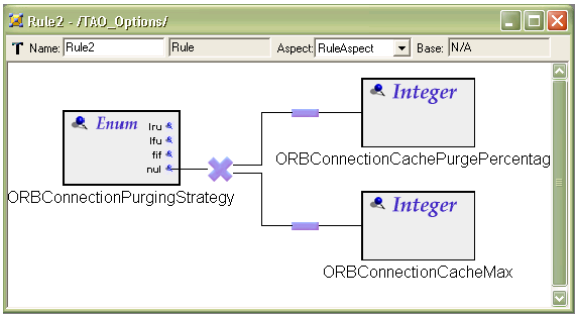

Figure 3: OCML Rule Example 2

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.

Thereafter, for a CIAO-based LDQoSS system, the developer uses the CFG generated in the previous process to define the configuration set for his/her specific application. The CFG provides online help when the application configuration is done and it also restricts the user from specifying configurations that 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 a tool-chain 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 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/cosmic`, which is also the web site for the CoSMIC tool suite. 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

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," *Internet Society, Network Working Group RFC 2475*, pp. 1–36, Dec. 1998.

[2] E. Rosen, A. Vishwanathan, and Callon R, "Multiprotocol Label Switching Architecture," *Internet Society, Network Working Group, Standards Track RFC 3031*, pp. 1–61, Jan. 2001.

[3] Object Management Group, *CORBA Components*, OMG Document formal/2002-06-65 edition, June 2002.

[4] Sun Microsystems, "Java^{TM} 2 Platform Enterprise Edition," http://java.sun.com/j2ee/index.html, 2001.

[5] Microsoft Corporation, "Microsoft .NET Development," msdn.microsoft.com/net/, 2002.

[6] Aniruddha Gokhale, Douglas C. Schmidt, Balachandran Natarajan, Jeff Gray, and Nanbor Wang, "Model Driven Middleware," in *Middleware for Communications*, Qusay Mahmoud, Ed. Wiley and Sons, New York, 2003.

[7] Center for Distributed Object Computing, "Component Synthesis using Model Integrated Computing (CoSMIC)," www.dre.vanderbilt.edu/cosmic, Vanderbilt University.

[8] Akos Ledeczi, Arpad Bakay, Miklos Maroti, Peter Volgysei, Greg Nordstrom, Jonathan Sprinkle, and Gabor Karsai, "Composing Domain-Specific Design Environments," *IEEE Computer*, Nov. 2001.

[9] Nanbor Wang, Douglas C. Schmidt, Aniruddha Gokhale, Craig Rodrigues, Balachandran Natarajan, Joseph P. Loyall, Richard E. Schantz, and Christopher D. Gill, "QoS-enabled Middleware," in *Middleware for Communications*, Qusay Mahmoud, Ed. Wiley and Sons, New York, 2003.