

A Generative Programming Approach to Middleware development

Venkita Subramonian and Christopher Gill
{venkita,cdgill}@cse.wustl.edu
Department of Computer Science and Engineering
Washington University, St.Louis,MO

1 Introduction

The OMG's Model Driven Architecture [1] is emerging as a way to enable seamless interaction between applications developed using various middleware platforms like CORBA [2], Java/EJB [3], .NET [4] and XML/SOAP [5]. Component based development technologies like the CORBA Component Model [6] and Enterprise Java Beans aim to provide meta-level frameworks for simplifying development on any particular middleware platform. These component frameworks ease the development, configuration and deployment of *application components*. Each of these *configurable* application components can be developed using low level programming frameworks that provide finer grained programming primitives. While the application components can be assembled using component based development tools, the implementations of the components themselves and the infrastructure below the component framework are usually achieved via manual assembly. Several design patterns aid in making this manual assembly possible so that the resulting end product is extensible and reusable. The *Generative Programming* [7] paradigm offers techniques which could be used by the component implementor to (1) configure the component implementation based on the configuration settings of the component and (2) configure the infrastructure more flexibly. In this position paper, we investigate the applicability of C++ Template Meta Programming [7] - a specific Generative Programming technique - to the development of middleware infrastructure mechanisms.

2 Middleware Infrastructure Configuration

Modern software engineering methodologies emphasize the design of *configurable* and *customizable* components. Each instance of a component may have its own unique customizations. This is analogous to a car assembly plant, in which each car might have its own customized fittings [7]. Applying these ideas to middleware infrastruc-

ture development is an emerging area of research involving many techniques such as Aspect-Oriented Programming, Subject Oriented Programming, Intentional Programming, Generic Programming and Template Meta-Programming. These techniques complement each other in various ways. In this section, we discuss the applicability of Template Meta-Programming, in particular, to the configuration of dispatching infrastructure mechanisms.

Real-time Dispatching Middleware Infrastructure:

Real-time dispatching infrastructure mechanisms [8] assume different roles under different contexts, although the basic principle remains the same: to dispatch a particular *command* object based on its QoS attributes. For example, dispatching infrastructure can be used in the context of a real-time event channel [9] to push events to consumers at the appropriate priorities, or it could be used in the context of an ORB to dispatch requests to servant objects at the right priority. Furthermore, different configurations might be appropriate in different contexts. For example, the configuration in an ORB core might preserve priority lanes, but to avoid overhead, might not render requests dynamically.

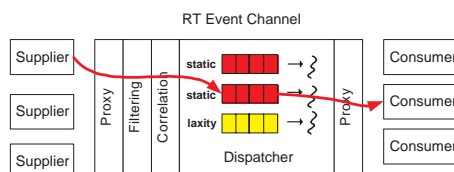


Figure 1: Example Dispatching Configuration [8]

Figure 1 shows an instance of a dispatching infrastructure that consists of real-time threads and queues based on a scheduling strategy chosen by the application. This strategy could be decided as part of the configuration of an application component.

Configuration of Dispatcher mechanisms: Configuring this infrastructure consists of setting up (1) the appropriate number of queues, (2) the thread priorities, and (3) the attributes corresponding to each individual queue based on the dispatching type used. For example, for a

deadline based strategy, the most eligible item is the one with the earliest deadline, whereas for RMS [10], the dispatch queue could just be a FIFO. This configuration information could be supplied by an off-line scheduler.

Factory based configuration: One approach to configuring infrastructure is to use the *Factory* [11] pattern to instantiate the appropriate queues and threads based on configuration parameters given as part of the *create* method of the factory. This approach configures the dispatching mechanisms at runtime even though, in some situations, the dispatching and scheduling strategies are known in advance at design time itself. Moreover, this approach suffers from accidental complexities like the fact that the QoS descriptor structure contains fields irrelevant to the type of scheduling discipline in use. For example, the QoS descriptor information might look like the following:

```
struct QoSDescriptor
{
    long period;
    long deadline;
    long worst_case_execution_time;
    int importance;
}
```

If RMS is chosen as the scheduling discipline, all fields other than *period* become unnecessary overhead.

3 A new approach - Configuration Generators

C++ Template Meta Programming aids in the development of configuration generators which generate configurations at compile-time using the power of C++ templates. For example, the above QoS descriptor structure could be written using template metaprogramming constructs, (IF, SWITCH, CASE, etc.) described in [7]. These constructs are used to build a configuration generator which actually generates the customized component. For example, the QoSDescriptor above could be generated by a QoSDescriptorGenerator which is a class template parameterized with the scheduling strategy. In contrast with the factory approach, this solution *composes* only the *necessary* fields in the QoSDescriptor. Moreover, with the factory approach there could be some invalid configurations or combination thereof, that may not

be detected until runtime. With the new approach, it is possible to make the generator flag invalid configurations as being erroneous at compile-time itself. Thus, the new approach facilitates building *type-systems* which enforces domain level semantics, all within the confines of the C++ language.

Even though the example above is trivial, the Template Meta Programming mechanism provides a very powerful tool to *generate* component configurations and customizations. Its power is realized when the configuration parameters from CORBA CCM are passed on to such generators which generates the appropriate customized versions for the underlying infrastructure.

4 Related Work

AOP approach to OS: This is an ongoing research area [12] to find cross-cutting concerns within OS code. Our approach differs from this work with respect to the technique used (AOP vs Template Meta Programming) as well as the domain under consideration (OS vs Middleware).

QoS-CCM: [13] describes ongoing research on real-time extensions to CCM. This research area aims to identify the configurable real-time and other QoS properties which could be integrated with CCM, which currently does not have QoS specifications. Our approach intends to complement this effort with techniques to enforce the higher level component configuration (CCM) with lower level infrastructure configuration.

Aspect CCM: [14] describes an extension to the CCM using Aspect Oriented Programming techniques. This approach addresses the cross-cutting concerns encountered in various phases of the component development life cycle. This approach complements the template programming approach since the latter is used to generate various instances of *a* particular component, whereas the former provides mechanisms to deal with requirements cutting across *several* components.

In conclusion, we note that infrastructure configuration can be thought of along two dominant dimensions: (1) self-contained environments and (2) aspects that cross those environments. Our work complements each of these by offering a configurable substrate.

References

- [1] Object Management Group, *Model Driven Architecture (MDA)*, OMG Document ormsc/2001-07-01 ed., July 2001.
- [2] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 3.0 ed., June 2002.
- [3] Sun Microsystems, "Enterprise JavaBeans Specification." java.sun.com/products/ejb/docs.html, Aug. 2001.
- [4] Microsoft Corporation, "Microsoft .NET Development." msdn.microsoft.com/net/, 2002.
- [5] J. Snell and K. MacLeod, *Programming Web Applications with SOAP*. O'Reilly, 2001.
- [6] Object Management Group, *CORBA Components*, OMG Document formal/2001-11-03 ed., Nov. 2001.
- [7] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Boston: Addison-Wesley, 2000.
- [8] D. L. Levine, C. D. Gill, and D. C. Schmidt, "Dynamic Scheduling Strategies for Avionics Mission Computing," in *Proceedings of the 17th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, Nov. 1998.
- [9] C. O'Ryan, D. C. Schmidt, and J. R. Noseworthy, "Patterns and Performance of a CORBA Event Service for Large-scale Distributed Interactive Simulations," *International Journal of Computer Systems Science and Engineering*, vol. 17, Mar. 2002.
- [10] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *JACM*, vol. 20, pp. 46–61, Jan. 1973.
- [11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.
- [12] Y. Coady, G. Kiczales, M. Feeley, N. Hutchinson, and J. S. Ong, "Structuring operating system aspects: using aop to improve os structure modularity," *Communications of the ACM*, vol. 44, no. 10, pp. 79–82, 2001.
- [13] N. Wang, K. Balasubramanian, and C. Gill, "Towards a real-time corba component model," in *OMG Workshop On Embedded & Real-Time Distributed Object Systems*, (Washington, D.C.), Object Management Group, July 2002.
- [14] P. Clemente, J. Hernandez, J. Murillo, M. Perez, and F. Sanchez, "Aspectccm: An aspect-oriented extension of the corba component model," in *Proceedings of the 28th Euromicro Conference (EUROMICRO'02)*, IEEE, 2002.