

# Patterns in Global Dynamic Middleware Scheduling and Binding

Victor Fay Wolfe & Lisa DiPippo  
University of Rhode Island  
{wolfe,dipippo}@cs.uri.edu

Peter Kortmann & Ben Watson  
Tri-Pacific Software  
{peter,watson}@tripac.com

Russ Johnston & Trudy Morgan  
U.S. Navy SPAWAR Systems Center SD  
{russ,trudy}@spawar.navy.mil

Louis DiPalma & Robert Kelly  
Raytheon Electronic Systems  
{Louis\_P\_DiPalma, Robert\_E\_Kelly—Jr}@raytheon.com

## 1 Introduction

This position paper describes patterns that we have discovered in the process of developing models and algorithms for global dynamic real-time QoS scheduling and binding in distributed object computing middleware. Some of these patterns are similar to previously defined patterns, while others are new due to the real-time nature of the applications in which they will be implemented. We first briefly describe the models and algorithms that we have developed. We then go on to discuss the patterns used by these algorithms. Finally, we describe some future work in the area of meta-configuration and speculate about what kinds of patterns might be useful in this area.

## 2 Global Dynamic Scheduling and Binding

This section describes a model and a set of algorithms that we have developed for global dynamic scheduling, and for binding in real-time middleware. In Section 3 we discuss patterns that allow these models and algorithms to work independently in a real-time middleware application.

### 2.1 Dynamic Real-Time Scheduling

Our global scheduling model assumes that there are clients which request service with some QoS requirements. It further assumes that there are servers that provide service at some level of QoS. There is a middleware scheduling service that implements a dynamic scheduling policy across the distributed system. Servers register with the scheduler to indicate the services that they provide, and the levels of QoS they provide for each service. The QoS parameters specified by a server can include execution time, accuracy, security, etc. The scheduler uses this information to:

- A. *Decide on an initial global priority assignment.* The scheduling service must assign a global priority for both the client execution and the servant's execution on behalf of the client. Priority assignment algorithms like *earliest-deadline-first* (EDF) [1], *weighted EDF* [2], and *least slack time* [1] might be used.
- B. *Adjust the global priority as the system dynamically changes.* For instance, in EDF scheduling the priority of a task may increase as time goes on. Also priority may be adjusted to handle overload.
- C. *Map global priorities to the underlying system priorities* (e.g. local priorities on a real-time operating system). For instance, in [3] we describe several algorithms for mapping global priorities to local priorities and accounting for the priority inversion that may result.
- D. *Perform resource management that dictates concurrent access to servers.* Basic priority inheritance [6] and serial locking approaches are among candidate resource management techniques. In [4,5] we describe how static priority

ceiling techniques can be adopted to manage distributed object servers. Similar strategies can be applied to object servers in a dynamic system using priority inheritance.

- E. *Handle overload conditions.* In a dynamic system overload may be handled via several techniques such as: *admission control*, which prohibits other tasks from entering; *load shedding*, which aborts tasks based on some heuristic measure of quality; and *load reduction* [7] which allows the QoS of certain tasks to be reduced.

## 2.2 Real-Time Dynamic Binding

The model used by our dynamic binding algorithms is similar to the scheduling model described in Section 2.1. It assumes clients and servers that specify QoS capabilities and requirements in the same way, and it assumes a middleware binding service. Each server registers with the binding service to specify the services that it can provide and the QoS capabilities for each service. The model assumes that a client may not know from which server to request a service. It also assumes that there may be more than one server which can provide the same service, possibly at different levels of QoS. When a client requires a service, it contacts the binding service and specifies its QoS requirements. The binding service implements a matching algorithm to determine which server will best be able to provide the requested service to the client. Once this match is made, the client is provided with the binding to the chosen server.

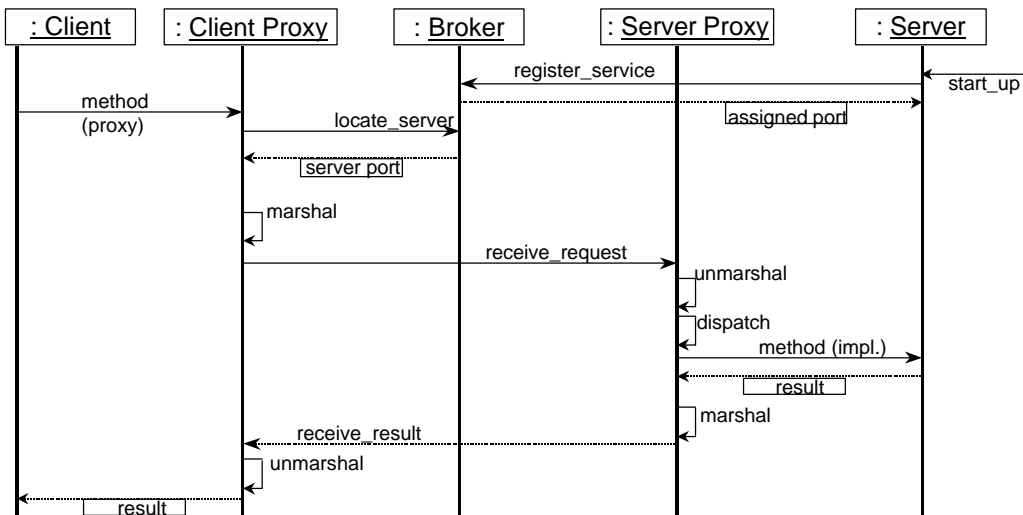
As in the scheduling model, our binding model can implement different matching algorithms depending upon the application requirements. In order to implement such algorithms, the binding service may be required to maintain information about the tasks executing on each node, along with their QoS specifications. Some possible binding algorithms include simple fitting algorithm, such as *first fit*, *best fit* or *worst fit*, as well as *probabilistic algorithms* that we have developed to take into account possible future requests when making a binding decision. In our probabilistic algorithms the binding service maintains not only information about tasks execution on each node, but also probability distributions representing the likelihood of a particular task being requested on a server. This information allows the binding service to choose a server that will be likely to fit the current request along with the next “mythical” request on that server.

## 3 Patterns in Dynamic Scheduling and Binding

While developing the scheduling and binding models and algorithms described in Section 2, we have recognized certain patterns emerging. By identifying and elaborating upon these patterns, we can make our models stronger and can provide a basis for other work to use similar global scheduling and binding policies. In this section we describe the specific patterns that we have identified. These patterns are extensions of previously published-patterns [9].

### 3.1 Binding

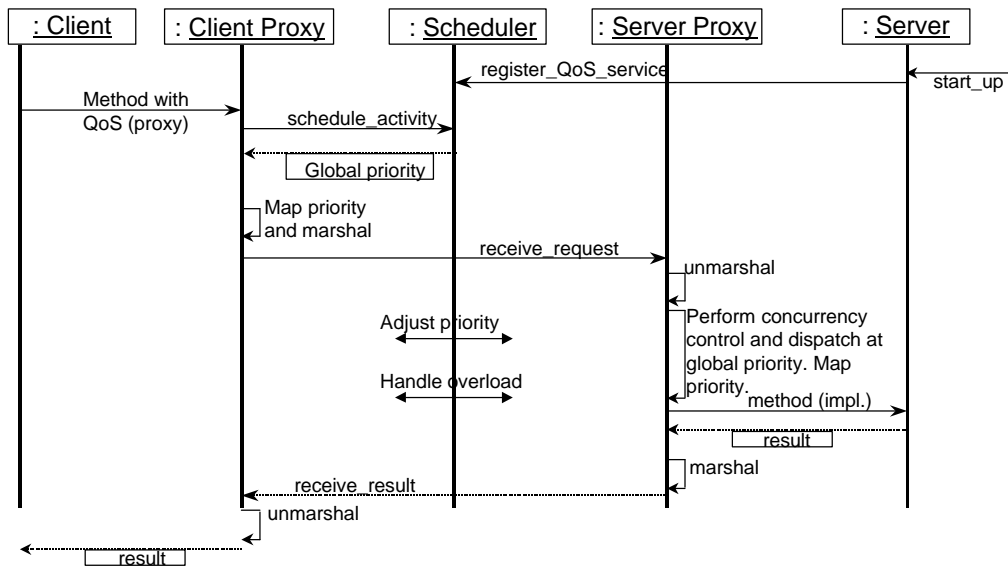
The dynamic binding model described in Section 2.2 is very similar to the *Broker Pattern* in [9], shown in Figure 1. In the Broker Pattern, a client wishes to locate a server. In general, the server can be specified by name, as is done in a Naming Service, or by service, as is done in a Trader Service. In our model, the location of the server is based upon the chosen matching algorithm. Then the Broker can specify the server port to the client and the client can make the request to the server. Thus, our binding model can utilize the Broker Pattern directly.



**Figure 1 - Broker Pattern**

### 3.2 Scheduling

The scheduling model described in Section 2.1 suggests a pattern for global real-time scheduling. In this pattern the global scheduler is accessed for each client request to a server. The execution model for this pattern is depicted in Figure 2.



**Figure 2 - Global Scheduling Pattern**

Notice that this pattern is very similar to the Broker Pattern shown in Figure 1. The main difference is in the functionality of the Broker/Scheduler entity. The function of the Broker is to provide a server to the client based on some specification. In the Global Scheduling Pattern, the client knows the server on which it wishes to execute an activity, but it needs the Scheduler to assign a global priority to the its request. Furthermore, the proxies perform scheduling functions including mapping of the global priorities to local system priorities, and resource management such as concurrency control. The two

double headed lines emanating from the scheduler indicate that the scheduler may have asynchronous interactions with either the client proxy, the server proxy, or both in order to adjust global priorities or to handle overload conditions (e.g. .to indicate an abort if load shedding is being used).

## 4 Current Work

This paper has briefly described two patterns that we are using in our development of dynamic real-time QoS middleware services – the basic patterns for global scheduling and for binding. There are several other patterns on which we are working:

- *Combined Scheduling Binding Pattern.* In many applications, global scheduling and binding may be required together. That is, when a client makes a request, it may not know which server will provide the best service, and it also may require a global priority assignment on the chosen server. In this case, a new pattern arises in which the client makes a single request to a broker-like entity, and this entity performs the matching and the priority assignment. The implementation of this pattern may actually have a binding service and a scheduler working collaboratively to perform this broker-like task.
- *Service Configurator Pattern.* This pattern, presented in [8], allows for a meta-service that dynamically configures other services. In our current work, we are looking to apply this pattern to dynamic real-time scheduling and binding. The QoS middleware services that we have developed have the ability to implement multiple policies for priority assignment, resource management, overload, and binding. Applying the Service Configurator Pattern will allow for dynamic configuration of the scheduling and binding services so that different policies can be chosen depending upon environmental and system conditions.
- *Application Specific Conversation Patterns.* Each of the patterns that we have discussed in this paper require a per-request model in which clients must access the service (scheduling or binding) for each request. We are examining particular applications to discover “conversation” patterns where clients and servers communicate in a predictable manner. Using such patterns, we will modify the scheduling model so that each conversation is scheduled as a unit, rather than each request. Similarly, we could bind clients to servers for the duration of a conversation, rather than for each request.

## References

- [1] Jane W. S. Liu. *Real-Time Systems*. Prentice Hall, Inc., 2000.
- [2] V. Fay-Wolfe, L. DiPippo, R. Ginis, M. Squadrito, S. Wohlever, I. Zyk. Expressing and Enforcing Timing Constraints in a Dynamic Real-Time CORBA System. *The International Journal of Time-Critical Computing Systems*, 16, 253-280 (1999).
- [3] L. DiPippo, V. F. Wolfe, L. Esibov, G. Cooper, R. Johnston, B. Thuraisingham, J. Mauer. Scheduling and Priority Mapping for Static Real-Time Middleware, *Real-Time Systems Journal*, 20, 155-182, 2001.
- [4] M. Squadrito, L. Esibov, L. DiPippo, V. F. Wolfe, G. Cooper, B. Thuraisingham, P. Krupp, M. Milligan, R. Johnston, R. Bethmangalkar. The Affected Set Priority Ceiling Protocols for Real-Time Object-Oriented Systems. *International Journal of Computer Systems Science and Engineering - Special Issue on Object-Oriented Real-Time Distributed Systems*. Mar. 1999.
- [5] L. Cingiser DiPippo, V. Fay Wolfe. Object-Based Semantic Real-Time Concurrency Control with Bounded Imprecision. *IEEE Transactions on Knowledge and Data Engineering*, vol. 9 no. 1. Feb. 1997.
- [6] R. Rajkumar, Synchronization in Real-Time Systems: A Priority Inheritance Approach. Kluwer Academic Publishers, Boston, MA, 1991.
- [7] Ethan Hodys. *A Scheduling Algorithm for a Real-Time Multi-Agent System*, University of Rhode Island Technical Report – TR00-275.
- [8] Douglas Schmidt, Prashant Jain. Service Configurator -- A Pattern for Dynamic Configuration of Services. (*updated April 28th, 1997*) C++ Report, SIGS, Vol. 9, No. 6, June, 1997.
- [9] Douglas C. Schmidt, Michael Stal, Hans Rohnert and Frank Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, Wiley & Sons, 2000.