Efficient and Extensible Multithreaded Remote Servers*

Ricardo Jiménez-Peris¹, M. Patiño-Martínez¹, F. J. Ballesteros², and S. Arévalo¹

¹ Technical University of Madrid
² Madrid Carlos III University

Abstract. In many cases, servers must impose a protocol of calls to their clients, and at the same time handle multiple client requests. The MT-Rendezvous design pattern greatly simplifies both tasks: separate server threads handle separate clients or sessions, and each different call protocol is handled by means of rendezvous.

One of the most significant performance problems in this kind of system is the latency introduced by network messages exchanged between clients and servers. Another design pattern, CompositeCalls, has been used to achieve dramatic performance improvements. With CompositeCalls clients send entire programs to the server so that the number of messages exchanged can be greatly reduced. Moreover, servers can be dynamically extended by using CompositeCalls.

Therefore, an expressive and efficient server model can be obtained by mixing both patterns within the same framework. However, as both patterns overlap, its integration is not a trivial task.

In this paper we describe how can both patterns be combined, including a brief description for its instantiation in Ada 95. Besides, we show concrete applications where the compound pattern, CompositeRendezvousCalls, can be employed, including a transactional framework for distributed Ada applications, TransLib.

1 Introduction

One of the mechanisms proposed for client-server interaction is rendezvous. It has some advantages over the RPC model. In particular, servers can easily impose a protocol of calls to their clients. Consider a server design that imposes a protocol for client calls, which is the common case; e.g. open must be called first, then a sequence of read and/or write, and finally close. Rendezvous is probably the best way to go, because enforcing call protocols with RPC is somewhat awkward and error prone. With RPC, it is necessary to record results from

---

* This work has been partially funded by the Spanish Research Council (CICYT), contract number TIC98-1032-C03-01 and the Madrid Regional Council (CAM), contract number CAM-07T/0012/1998.

¹ A more involved protocol results when considering different (specialized) open primitives: openRead, openAppend, openModify.
previous calls, test those results in following calls, and maintain shared data structures to associate session state with client identifiers (or addresses).

However, using rendezvous also poses some drawbacks. In particular, it is hard to implement multithreaded servers, using different call protocols for different clients. The \textit{MT-Rendezvous} design pattern \cite{2} simplifies such task providing support for multithreaded rendezvous. The server code just deals with a single client. That is, when the first call from a client is received a server thread is created. That call and future calls from the same client will be managed by the same server thread. Thus, server development is greatly simplified. Some scenarios where multithreaded rendezvous-based servers are convenient are transactional environments \cite{1}; like \textit{Transactional Drago} \cite{3,2}, and DVM \cite{10}.

In a distributed environment, typical client/server interactions require many network messages. For instance, to copy a file within the same server the whole file will cross the network twice. Reducing the number of messages required to perform a given task can dramatically improve latency. The \textit{CompositeCalls} design pattern \cite{5} allows a client to send multiple calls, bundled together, to a given server (they will be referred as composite calls or programs). Simple control structures, such like “\texttt{while I can read, write what I have read}” can be submitted to the server. Every call made to a server entry point, within a composite call, is actually a local procedure call. Finally, results are returned back to the caller node. Therefore, by using \textit{CompositeCalls}, a lot of network traffic can be avoided, improving the latency for the service. An additional benefit of \textit{CompositeCalls} is that it allows dynamic extension of the server (e.g. a \texttt{copy} service could be added by building upon the primitive \texttt{read}, and \texttt{write} services).

Improvements in server latency, expressiveness and simplicity can be achieved by careful integration of the \textit{CompositeCalls} and \textit{MT-Rendezvous} patterns. Such integration presents several difficulties because both patterns intersect, addressing common issues in rather different ways (e.g. they model server calls in different ways). Despite that, combining both patterns brings clear benefits in distributed applications:

1. Servers need to include just basic (primary) services. Thus, its design is simplified. Later, they can be extended dynamically by means of composite calls.
2. Client/server interaction can be more cleanly structured, by using the expressive power of rendezvous.
3. Client/server interaction can be more efficient. All server calls made within a composite call do not cross the network.
4. Server code is more easily written, as it has to deal with a single client; while, at the same time, more throughput can be obtained on multiprocessors by servicing in parallel different clients (i.e. due to multithreading).
5. A single call, expressing an entire rendezvous-based session, can be built; i.e. separate steps in a call protocol enforced by rendezvous can be combined in a single entity.

In what follows, we will show the patterns involved in sections\cite{2} and \cite{3} Section\cite{4} shows how to the \textit{CompositeCalls} pattern can be mixed with \textit{MT-Rendez-