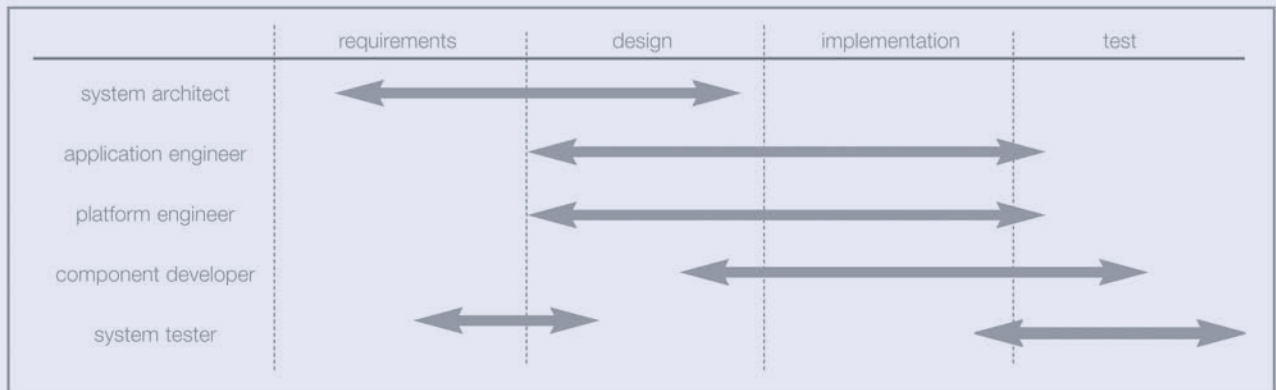
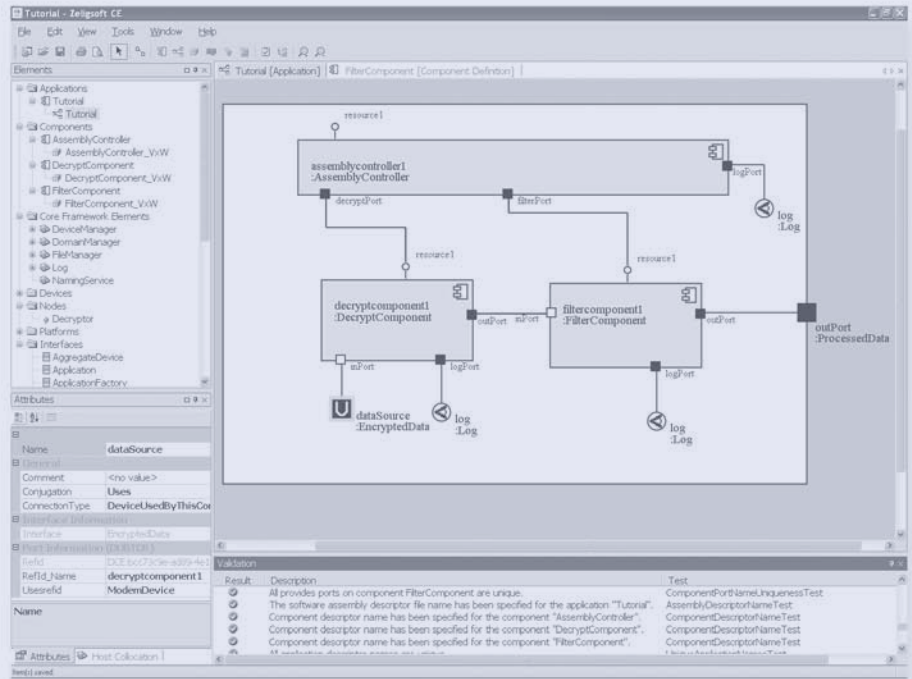




Component Based Software Design using CORBA

Victor Giddings, Objective Interface Systems
Mark Hermeling, Zeligsoft



Component Based Software Design using CORBA

Victor Giddings (OIS), Mark Hermeling (Zeligsoft)

Component based software design depends heavily on communication between components. In the past engineers have tried to minimize communication delays by combining components into monolithic systems or by using custom coded transportation mechanisms. This has resulted into in-flexible, fragile systems that are hard to design and test. This paper demonstrates through performance measurements that a high-performance ORB like OIS's ORBexpress makes flexible, re-usable component based software designs a reality.

1 INTRODUCTION

Software Defined Radio systems are complex embedded systems. The Software Communications Architecture (SCA) standard provides a component based way of designing, deploying and configuring these systems. Component Based Software Design (CBSD) is key in this endeavour; it allows the designer to build a waveform (an application) from independent components. Each of these components can be independently designed, built, tested, deployed and re-used.

A proper component based application utilizes a number of components; each component has a well defined interface and well defined functionality. All of the components are connected together: they communicate over a middleware layer. The functionality of all components combined delivers the functionality of the entire application. In a component based design each component performs a specific functionality and does it well.

However, many current component based designs are very monolithic, they combine a lot of functionality in one single component. The entire system is then composed of a few components (typically 3 or 4 or fewer). This approach is typically chosen for performance reasons, designers try to optimize the performance of the system by combining a lot of functionality in one single component. Designers try to minimize use of the middleware layer. This practice is no longer necessary given modern middleware layers

like CORBA. Embedded CORBA ORBs like OIS's ORBexpress are highly optimized. The level of performance they provide makes the need for monolithic designs a thing of the past.

This paper will explore proper component based design in the context of the SCA and will cover how ORBexpress provides the required communication performance. Even though this paper is mainly focussed on the SCA the arguments are equally applicable to other component based standards such as the OMG Deployment and Configuration standard.

2 BENEFITS OF COMPONENT BASED DESIGN

Embedded systems have been steadily increasing in complexity. Software development methodologies have also changed. Component based software design is the latest technology intended to make this increase in complexity manageable.

CBSD defines an application as a set of connected components. Each component has a definition of its interface and its attributes. The component definition defines what the component needs from its environment (uses interfaces) to perform its services and what services it offers to the environment (provides interfaces). Every component has one or more implementations. Each of these implementations implements the definition of the component. Every implementation has a number of pre-requisites that must be met for that implementation

to be usable. For example, a pre-requisite could be 'PPC processor', 'Integrity RTOS' or '4MB of available RAM'.

This approach provides a number of benefits to software projects:

Divide and conquer

Components can be developed, tested and deployed individually. This simplifies development and increases quality and efficiency.

Increased re-use

A component is truly reusable. It's dependencies on the environment are well specified. If the environment satisfies the dependencies then the component can be used. The component can simply be "wired" into the application.

Design reuse

The "plumbing" of a component based application is standardized by the component framework. Proper use of automated design tools can reduce the amount of code that needs to be hand-written. This increases efficiency and quality.

Multiple platform support

A component based system can be deployed to multiple platforms. Every component can have multiple implementations, one implementation per execution environment.

Platform consolidation

Using component based techniques enables platform consolidation. Multiple applications can be executed on the same platform, provided memory space and processing power is sufficient.

Seamless distribution

Component based systems can execute across a multitude of processing devices: General Purpose

Processors (GPPs), Digital Signal Processors (DSPs) and Field Programmable Gate Arrays (FPGAs). This distribution is made easier by a standardized communication and connection strategy.

Standardization

Standardization on a component based framework will lead to availability of development tools leading to more efficient development workflows. Additionally, standardization will also lead to COTS components, where a project can go "shopping" for a component or out-source a component to a partner.

These benefits are readily achievable through existing component based standards like the Software Communications Architecture of the JTRS program run by the US DoD. Other standards like the deployment and configuration standard of the OMG are still emerging.

3 THE NEED FOR MIDDLEWARE

Component based systems require middleware to operate. Each component is described through a set of files containing information about the component, its implementations, its dependencies and its interface. The application is described through a file that references descriptions of the components used in the application. The application descriptor file defines instances of the components and the connections. The meta-information in these descriptor files is typically given in a well structured XML syntax.

The descriptor files are fed to a middleware layer. This layer consists of a CORBA communication layer and a management layer. The CORBA layer provides seamless communication across a distributed platform. The management layer is responsible for creating the application based on the descriptor files and establishing the CORBA links between application components, as well as for general management and maintenance tasks.

The management layer (referred to as “Core Framework” in the SCA) is responsible for exploring the application and the requirements of the component implementations in the application. Once it has determined that the execution environment can meet the requirements of the application it will instantiate the components across the processors in the execution environment. Lastly, it connects the interfaces of components together.

The management layer communicates with the components through CORBA.

4 SCA COMMUNICATION ANALYSIS

Communication in an SCA based system is based on CORBA. The SCA Core Framework uses CORBA to initialize the application and connect all the components.

Communication in an SCA compliant system can be divided in the following categories:

Waveform control and management

This is typically communication from the Core Framework and the AssemblyController (central control of the application). This could also be communication caused by interaction with the user.

This communication includes one and two-way function calls but there is little data included and the real-time requirements are low.

Signaling

Signaling is used to update the receiver of state changes in the sender. This communication has a low data content and is often one-way. It is important that the communication is completed with low latency. This type of communication is common when sender and receiver collaborate, but not very intensively.

Behavior

Behavioral communication is used when sender and receiver need to perform a lot of collaboration to complete a common task. This communication is frequent, contains data, and needs to have minimal latency.

Streaming

This is pure data communication from one component to another component. This is mostly one-way communication that has strict requirements on minimal latency and often high rates of throughput.

The SCA mandates the use of CORBA in communication, though it allows the uses of non-CORBA connections. This latter option is available if and only if the connection is initiated through CORBA.

A lot has been written and said about the perceived performance of CORBA ORBs. “They consume extra memory”, and “they add overhead to execution paths”. Therefore CORBA is frequently not considered for high throughput or minimal latency communications.

There are two ways to deal with these objections.

The traditional solution is to use engineering workarounds to improve performance. We examine the engineering workaround approach in section 5.

A second solution is to use a modern, high-performance embedded ORB, an ORB that provides communication flexibility without the run-time overhead. We will look at that approach in section 6.

5 SCA COMMUNICATION IMPLEMENTATION

The waveform control and management category is not problematic. Most of this type of communication is done at startup, some of it is during the life-span of the system, but the real-time demands on it are pretty low. The SCA mandates that this category is to communicate through CORBA.

The signaling category is also often not a problem. Real-time demands are typically not very high and data throughput is low.

The behavior and streaming categories deserve some extra attention. Performance will significantly degrade when sender and receiver are performing a lot of behavior or streaming communication, especially if this communication is done across processor or even board boundaries.

Current SCA projects have dealt with this degraded performance in two ways:

1. Combining functionality in the same process for behavioral optimization
2. Use alternate means of communication for streaming optimization

Option 1 results in large monolithic components that contain functionality for a number of unrelated tasks all linked into the same executable.

Option 2 utilizes non-standard communication protocols or high performance communication fabrics which results in less portable applications.

Communication is a major consideration when deciding on the software architecture to use for a specific application. Software teams have been erring on the side of caution when dealing with performance. They used both option 1 and 2 to optimize communication paths in their system.

This results in systems with one single component per processor. That component contains all the functionality assigned to that processor. This reduces the amount of CORBA communication needed since all the intra-processor communication can be done through native language functional calls (typically C++ or C).

This also results in systems that use custom coded high performance transportation links and hence are less portable.

These are valid ways to get an application to work, and technically it complies with the requirements of the SCA specification, but it runs counter to its intent. This approach negates the benefits that can be gained from component based systems.

Monolithic systems are very specific to the targeted execution environment. Replacing components can only happen at processor boundaries. Porting the components to new target environments means porting of the custom transportation links.

These systems are also not truly component based, so testing and development is still very arduous. A lot of the flexibility that the SCA intended is lost through these monolithic designs.

Lastly, using non-CORBA solutions means more design and implementation effort is required from the engineers. Other means of transportation need to be devised and implemented.

The fear of low performance delivered by the CORBA layer is understandable, though not valid when using CORBA ORBs that have been optimized for embedded systems like *ORBexpress*. Section 6 will substantiate this claim using some example performance numbers gained from using *ORBexpress* on embedded systems.

6 CORBA PERFORMANCE

The performance of an application using CORBA for a request is determined by several factors. The amount of data to be transferred during the request is determined by the parameters of the request, which is defined by the needs of the application. Fewer, shorter parameters can be transferred in less time than those that are more in number or greater in size.

The execution environment performance makes a large difference. Some operating systems are faster than others, e.g., during a context switch. Processors vary widely in processing power as generally measured by clock speed. Finally, as we will see, the transport is very much a factor in both latency and throughput.

However, one of the most important determining factors is the ORB used. ORB performance differs by multiple orders of magnitude in performing the same operation. High performance ORBs avoid unnecessary context switches and minimize data copying to optimize both latency and throughput. ORBexpress has been shown to be the performance leader in ORBs available today.

Figure 1 shows ORBexpress performance for components distributed in three different ways: on different processors or boards, on different processes within the processor, and within the same process. It plots the latency time, in microseconds, for a request carrying an increasing number of bytes of data from one component to another. Each request is a CORBA two-way request, i.e., the specified amount of data is sent as payload in the request message from one component to the other and the receipt of the data is acknowledged by the ORB with a reply message.

The upper line, labelled “TCP between boards” shows the latency for a request sent from one board to another using TCP/IP over Ethernet. The “zero point” of this curve, about 370 microseconds, is determined largely by the processor speeds involved in sending and receiving the data. The server is running on a 400 MHz PowerPC process running LynxOS 4.0, while the client is on a 475 MHz AMD K6 processor running Linux.

The curve labelled “TCP loopback” is the latency between two components within the same board, but

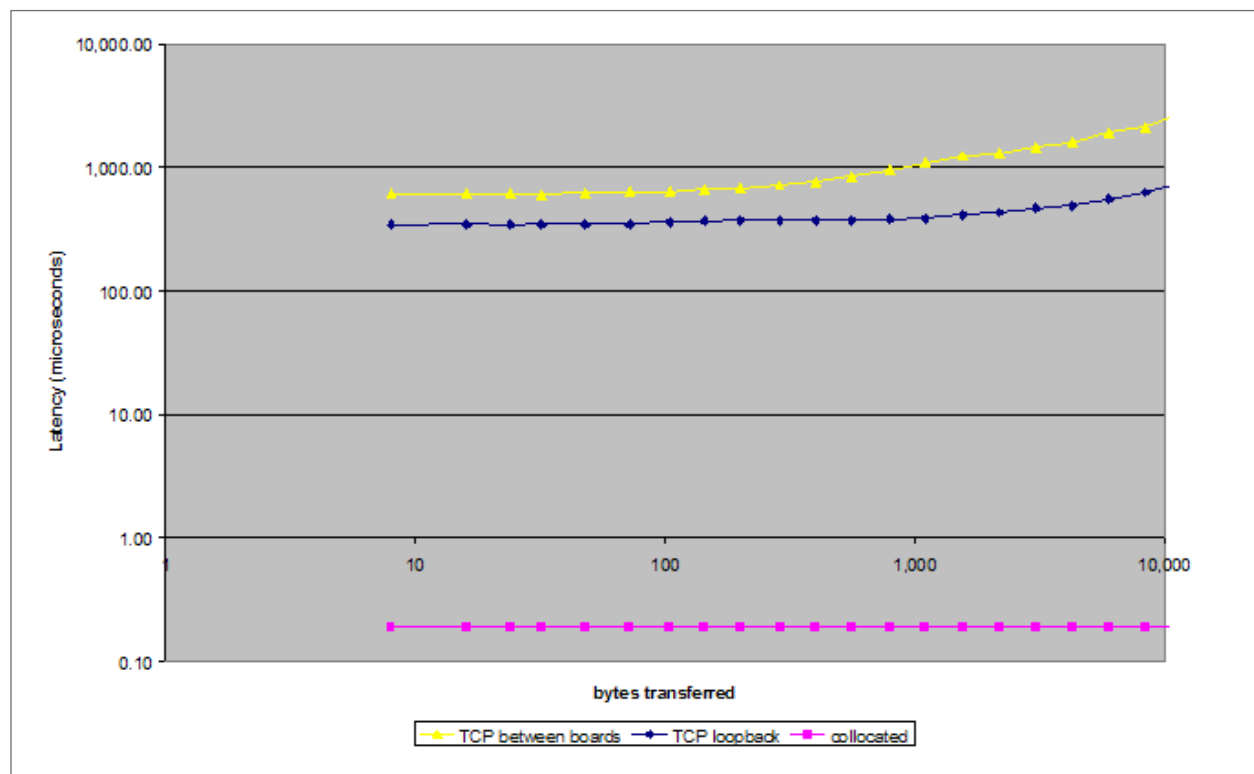


Figure 1: Overhead of Component Distribution

within different operating system processes, (both hosted on LynxOS 4.0 on a 400 MHz PowerPC). The transport used in the “loopback” version of TCP/IP is provided by the operating system. Since the CPU performance of all the hardware is roughly the same, the difference between the first two curves is attributable to the extra cost of using TCP over Ethernet between boards.

The lowest curve, labelled “collocated” shows the latency between two components that are in the same operating system process. ORBexpress optimizes such interactions by essentially stepping aside and using the native language facilities to perform the request. The latency shown, roughly 190 nanoseconds, is consistent with the cost of two C++ dispatching calls and one “if test” that are used to implement this interaction.

This lowest curve represents the cost using CBSD without distribution. Since it is so small, there is little justification for monolithic designs.

This does not completely answer the question of the performance cost of CBSD with distribution. One of the unexamined questions is how much of the latency seen above is due to the transport technology, e.g., TCP/IP over Ethernet, and how much is due to the mediation of the ORB in the interaction.

Figure 2 offers some insight into this issue. It shows a comparison between the use of TCP/IP loopback and “shared memory”. Most operating systems allow blocks of memory to be “shared” between processes and this has been used as the basis of an ORB transport. The results in this graph show a speed-up by a factor of five simply due to the use of an alternative transport.

The overwhelming majority of the cost of a CORBA request-reply interaction is due to the inter-process communication mechanism; selection of a higher-performance mechanism resulted in vastly improved performance. It also offers the promise that a higher-

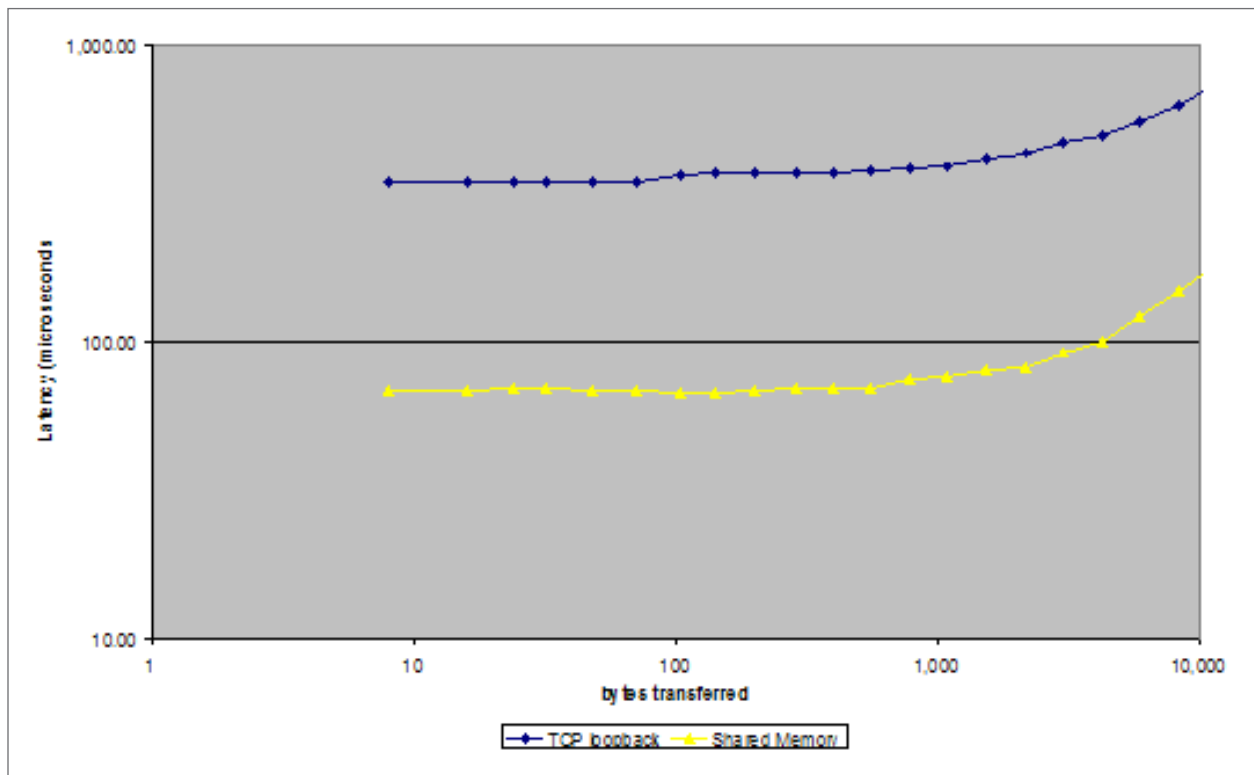


Figure 2: Effects of Different CORBA Transports

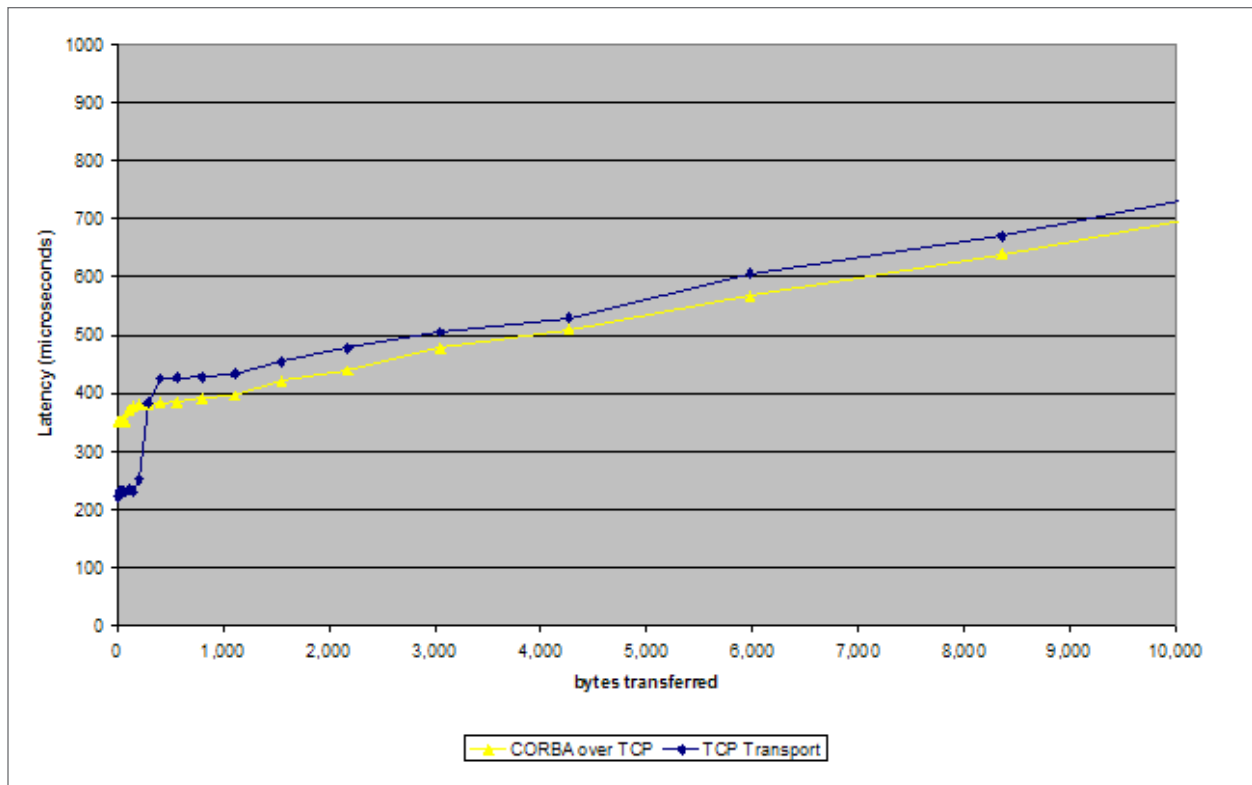


Figure 3: Overhead of CORBA in Loopback

performance hardware design will result in a higher-performance application, irrespective of the use of CBSD.

To round out this view of performance, we will now complete the separation of the performance of the underlying transport and operating system from the overhead introduced by CORBA.

Figure 3 compares the performance of two programs using TCP loopback. The CORBA over TCP line is the performance of the CORBA benchmark program, while the TCP Transport line is a simple “C” program that transfers data between two processes. (Both of these benchmark programs are shipped as demos with the ORBexpress products and usable with any transport. This allows customers to make their own measurements on their own selected infrastructure.)

The overhead of CORBA is represented by the difference between the two curves. The overhead is attributable to three areas:

Message Packaging and Formatting: The information about the request to be performed, including the identity and target of the request, as well as the parameter information, must be copied into request and reply messages that are sent to and from the server.

Multiplexing and De-Multiplexing: The request is multiplexed with other requests bound for the same server and multiplexed over the transport connections that have been established. When the request arrives at the server, it must be demultiplexed and routed to the correct object implementation.

Request execution: The server provides the execution resources, including a thread to execute the request operation, when the request is received. For real-time requests, this is complicated by the requirement to resolve resource contentions in a manner consistent with real-time priorities and policies.

Note that the performance overhead is very small or even negative for part of the graph. The small initial

overhead is due to the high-performance implementations of the operations described in the previous paragraph. The negative overhead in later parts of the graph illustrates another benefit of the use of commercial middleware: a commercial ORB will be optimized more than “roll-your-own” middleware because of the incentives involved.

7 CONCLUSIONS

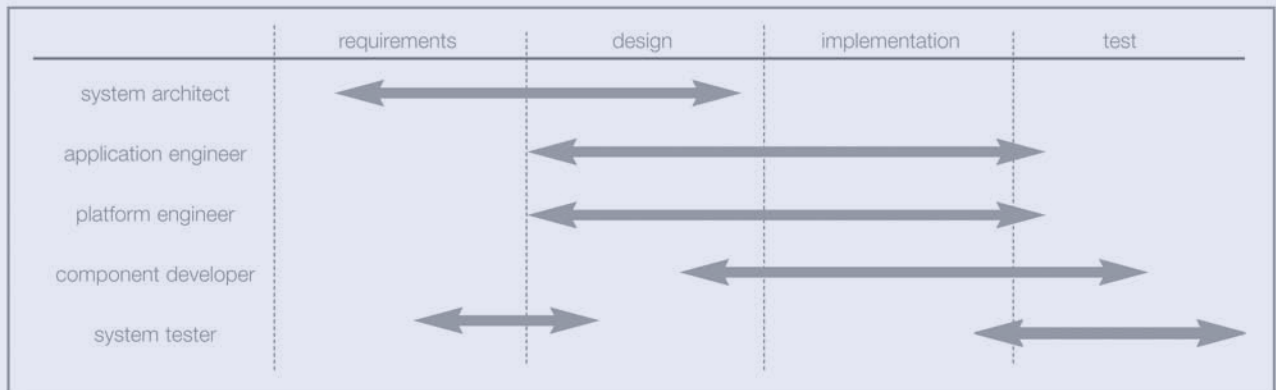
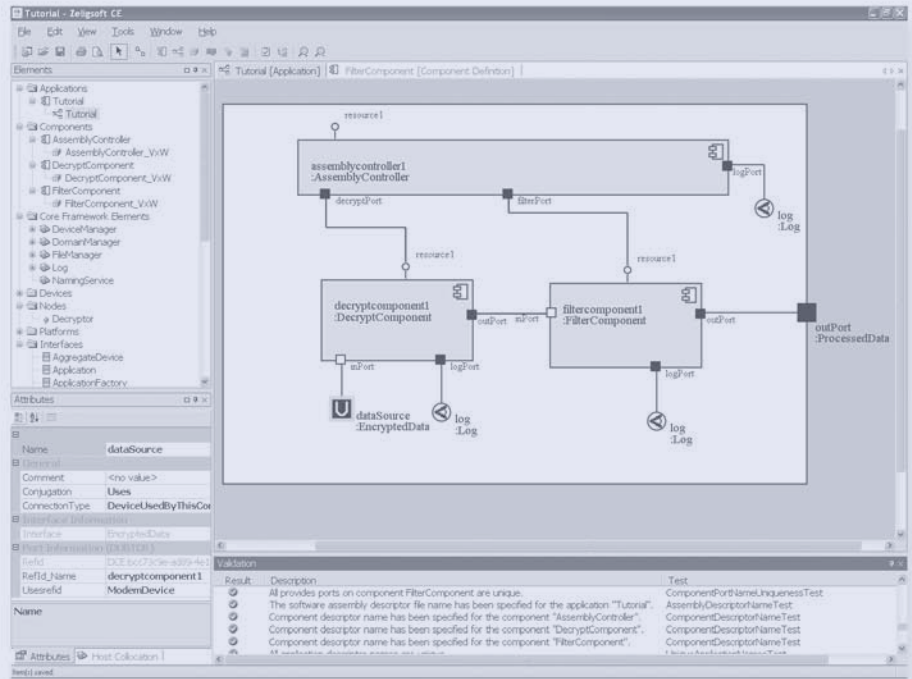
The first generation implementations of SDR have lost the benefits of using finer-grained CBSB in the mistaken belief that CORBA overhead would negatively impact performance. We have illustrated that there is almost no overhead with the use of CORBA-based components if the components are collocated in the same process. The overheads disappear because the ORB “steps out of the way” while the portability and re-use benefits are retained.

Even when components are distributed in different processes or on different processors, the overhead is negligible. In fact, with a well-crafted ORB, the overhead can be less than using a communication mechanism other than CORBA.

The combination of Zeligsoft Component Enabler and OIS ORB*express* enables users to design component based systems according to component based principles. This combination of tools makes it easier to deliver flexible, re-usable SCA compliant systems on-time and within budget.

8 REFERENCES

1. **Zeligsoft Component Enabler**
www.zeligsoft.com
2. **Objective Interface Systems Orb*express***
www.ois.com
3. **CORBA**
www.corba.org





Contact Information

Website: www.zeligsoft.com

Email: info@zeligsoft.com

Toll-free (North America): 1-800-ZELIGSW (1-800-935-4479)

Direct dial: +1 819-684-9639