Journal of Computer Science 8 (6): 957-964, 2012
ISSN 1549-3636

Impact of the Use of Object Request Broker Middleware for Inter-Component Communications in C6416 Digital Signal Processor Based Software Communications Architecture Radio Systems

[1]Wael A. Murtada, [2]Mohamed M. Zahra,
[3]Magdi Fikri, [2]Mohamed I. Yousef and [4]Salwa El-Ramly
[1]Department of Satellite Communications and Ground Stations,
Space Sciences and Strategic Studies Division,
National Authority for Remote Sensing and Space Sciences, Cairo, Egypt
[2]Department Electronics and Communications Engineering,
Faculty of Engineering, Al-Azhar University, Cairo, Egypt
[3]Department Electronics and Communications Engineering,
Faculty of Engineering, Cairo University, Cairo, Egypt
[4]Department Electronics and Communications Engineering,
Faculty of Engineering, Ain Shams University, Cairo, Egypt

Abstract: Problem statement: This study presents an in-depth analysis of the performance of Software Communications Architecture (SCA) component-based waveform applications in terms of inter-component communications. The main limitation with SCA, in the context of embedded systems, is the additional cost introduced by the use of Object Request Broker (ORB) middleware. The ORB middleware handles the interaction between components and objects in SCA distributed environment. This interaction should be highly efficient, due to the real time nature of SCA systems and transparent to the application programmer. **Approach:** We can achieve high efficiency in SCA systems by enhancing the Inter-Process Communications (IPC) mechanisms in Operating systems (OS) micro kernels, while we achieve transparency through Interface Definition Language (IDL). Different encoding mechanisms like "External Data Representation (XDR), Network Data Representation (NDR) and Common Data Representation (CDR) facilitate inter-component communication transparently and efficiently". Marshalling procedures format data from the local machine representation to common network representations. A most common encoding mechanism for Common Object Request Broker Architecture (CORBA) systems is CDR representation. Measurements have been performed with ORBExpress DSP as a CORBA distribution and Open Source SCA Implementation Embedded (OSSIE) for SCA implementation. In order to perform these measurements we proposed two metrics for profiling the ORB that are invocation and marshalling. In addition, we propose three elements of data types to evaluate the performance of ORB middleware that are, Basic, Array and Sequence data types. **Results:** The CORBA bus is really the part, which brings an overhead to the SCA radio systems. This overhead is due to method invocations that have been carried out by ORB middleware. **Conclusion:** Performance benchmarks of ORBExpress DSP middleware show that, although using CORBA for inter-component communications introduces delays and overheads, the overall effect can be reduced by sending packets of data instead of basic data type elements.

Key words: Software Communications Architecture (SCA), ORB middleware, ORBexpress DSP, inter-component communications, stub code, encoding, marshalling

INTRODUCTION

Time delays are an inevitable aspect of any real world communication system. As radio applications and underlying hardware becomes increasingly complex, however, these latencies become more and more difficult to predict and understand (Tsou *et al*., 2007). Predictable latencies and deterministic behavior

Corresponding Author: Wael A. Murtada, Department of Satellite Communications and Ground Stations, Space Sciences and Strategic Studies Division, National Authority for Remote Sensing and Space Sciences, Cairo, Egypt

are necessary in order to meet the requirements of a wide range of today's communication needs. The advent of software defined radio and the use of the Digital Signal Processor (DSP) as a suitable device for radio communications further complicates these issues. Traditionally, timing information and latency characteristics could be determined by examining hardware designs and specifications. With current software radio designs, however, Operating System (OS) behavior, middleware and multi-threaded environments are some of the issues that factor into latency behavior (Balister *et al*., 2006). In this study, we focused on middleware latency that has a major impact of software latency behavior.

The Software Communications Architecture (SCA) is a component based software specification developed for the Joint Tactical Radio System (JTRS) that seeks to address many design issues in developing interoperable software radios. In order to achieve interoperability and portability of applications, the specification defines a number of operating environment requirements for compliant implementations such as POSIX OS standards and the use of Common Object Request Broker Architecture (CORBA) as middleware. CORBA is a standard released by the industry consortium Object Management Group (OMG) and defines the communication between the distributed components of a SCA radio waveform OMG. The SCA and its underlying standards are specifications only and require appropriate implementations for actual use.

Open-Source Implementation: Embedded (OSSIE) PrismTech is an implementation of the SCA created at Virginia Tech for educational use as well as for research applications with software defined radio. OSSIE relies on a number of other open-source projects in order to address the standard. The original implementation runs on Linux and utilizes omniORB (Grisby *et al*., 2009) as the CORBA middleware implementation. We ported OSSIE on TI C6416 DSP and relied on ORBExpress DSP as an embedded CORBA middleware implementation for our measurements (Murtada *et al*., 2011). Additionally, TinyXML PrismTech is used for parsing the XML used in SCA profiles. In addition, OSSIE applications rely on a reusable interface library known as Standard Interfaces that simplifies the interaction of signal processing code portions with the implementation details of CORBA IDL.

This study introduces the middleware latency that contributes to inter-component latency in an OSSIE waveform. It is assumed that components reside on the same processor. Distributed radio applications that span multiple nodes present additional factors and are not examined in this study. While a typical TI C6416 DSP is used as a test case in this study, an effort is made such that the general concepts presented are applicable to other platforms. Timing measurements were performed on a test system contains TI C6416 as a Digital Signal Processor, Code Composer Studio IDE as a test environment, DSP/BIOS as a non POSIX Real Time Operating System (RTOS) and ORBExpress DSP as an embedded middleware CORBA distribution.

Software description: The efficiency of marshalling or stub code plays a vital role in SCA distributed environment. Improvement in application performance relies necessarily on efficient stubs (Puder *et al*., 2006). The performance of local client/server invocations and hence inter-component communications in SCA systems depend on efficient stub code when high-speed IPC mechanisms are used. Some of the encoding schemes like CDR, XDR and NDR facilitate the efficiency and transparency of inter-component communications (Tari and Bukhres, 2004).

We generate the stub code using Interface definition language (IDL) to C++ compiler from Interface definitions in IDL file. The client marshals parameters using stub code, uses OS kernel primitives to access the server skeleton, un-marshals these parameters at the server side and calls the associated server procedure (Henning and Vinoski, 1999). The server procedure marshals the result back to the client. Figure 1 illustrates the marshaling process between distributed client/server CORBA objects. The main advantage is that the programmer can determine and use the IDL remote interfaces like local interfaces. Stub code has two important features that are Portability and adaptability.

IDL compiler converts interface specifications, from the user code, into stub code. Generally, the stub code marshaling data into CDR, which is a CORBA standard mechanism for data encoding. CDR provides a "receiver-makes-it-right" approach to byte ordering (Ruh *et al*., 2000). Its functionality is to marshal the data between various computer architectures like Embedded C6416 DSPs, IBM personal computers, Sun workstations, Cray machines and VAX. It satisfies the ISO presentation layer specifications (Puder *et al*., 2006). CDR assumes that the basic and portable unit of encoding is the byte or octet. It encodes the data bytes in such away that any other decoders can decode the marshaled octets without loss of meaning (Tari and Bukhres, 2004). Generally, CDR encodes the stream of octets in both "little-endian" or "big-endian" representations. The marshaled octet stream in CDR encoding should be in multiples of four octets.

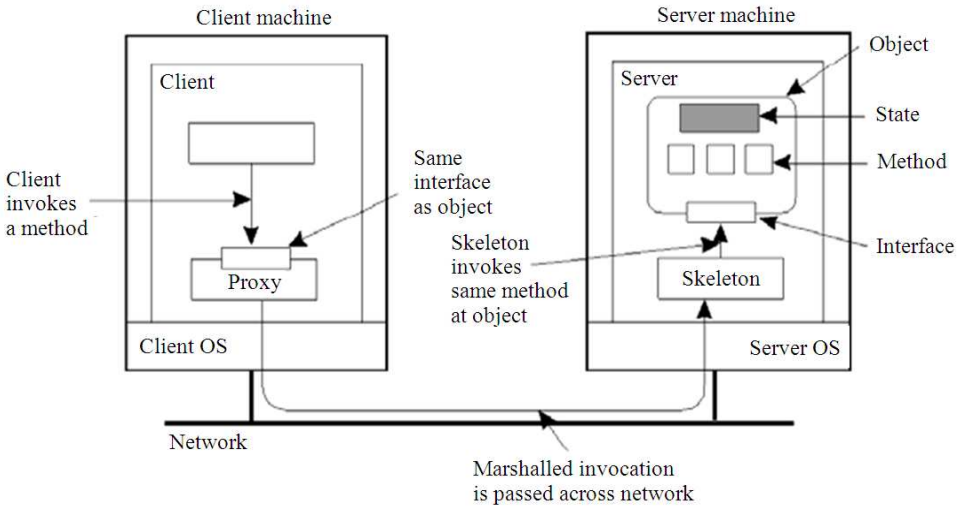

Fig. 1: Marshaling process between distributed client/server CORBA Objects

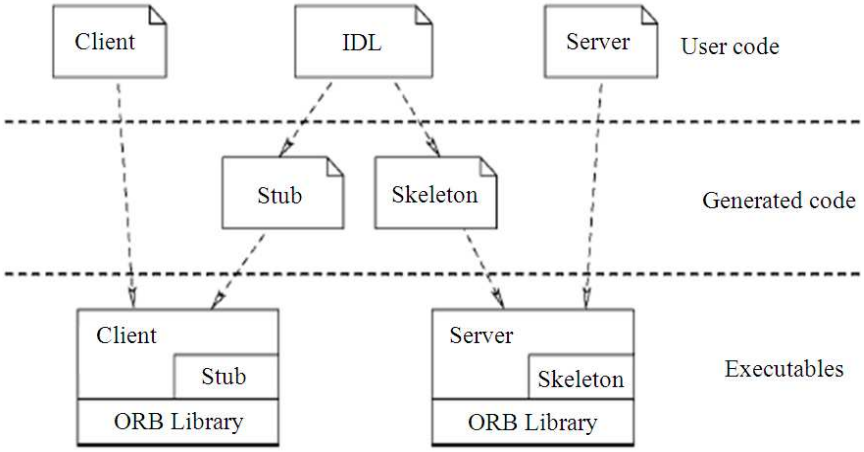

Fig. 2: The creation of CORBA application

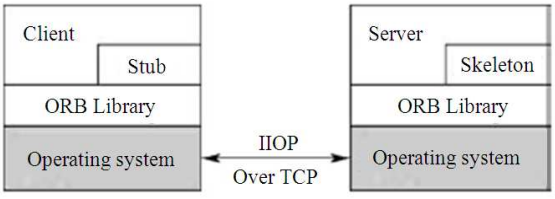

Fig. 3: Communication between client/server CORBA objects

When the marshaled data are not aligned as multiples of four octets, then it is padded with zeros. However, in ORBExpress middleware, the data buffer is in multiples of eight octets OIS. Similarly, when the marshaled data are not aligned as multiples of eight octets, then it is padded with zeros. Furthermore, ORBExpress DSP implemented a custom encoder, like CDR but it has some differences to improve the ORB performance, for use in marshaling process. Figure 2 illustrates the creation of CORBA based application using IDL compiler.

The CORBA specification defines the General Inter-ORB Protocol (GIOP) as its basic interoperability framework. GIOP is not a concrete protocol that is used directly to communicate between ORBs. Instead, it describes how specific protocols can be created to fit within the GIOP framework. Internet Inter-ORB Protocol (IIOP) is one concrete realization of GIOP as shown in Fig. 3. The GIOP specification fits into three ISO layers that are application, presentation and session. It consists of the following major elements: CDR, Message formats and IDL mapping.

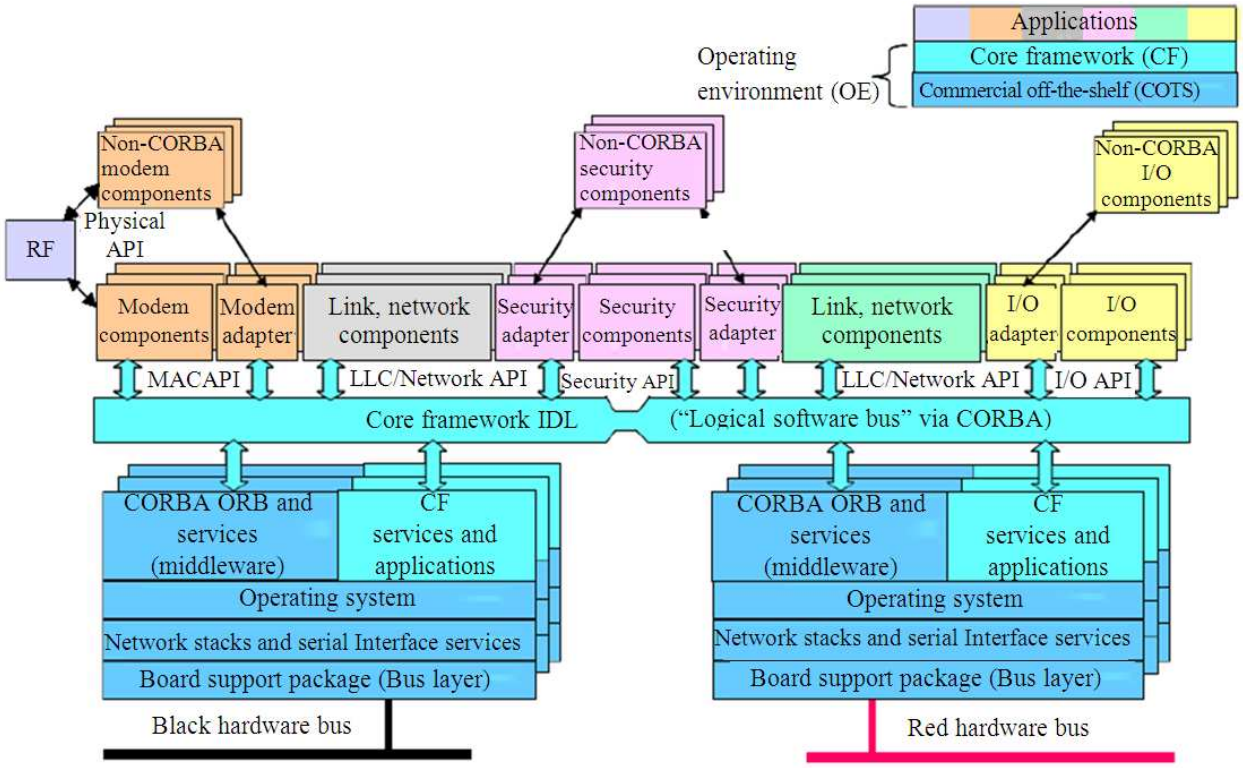

Fig. 4: SCA software structure

Commercial ORBs like ORB express DSP OIS, e*ORB PrismTech and omniORB (Grisby *et al*., 2009) use IIOP (Ruh *et al*., 2000) to communicate between distributed CORBA objects.

Here, we give an overview of a SCA system specifying the software architecture requirements and software components responsible for the deployment, configuration and control of the waveforms on the TI C6416 DSP hardware. The software infrastructure provided by SCA for the distributed application deployment is based on the CORBA software bus.

SCA: The Software Communications Architecture defined by the JTRS specifies an Operating Environment. It allows for the abstraction between software and hardware. In an ideal view, we can install any waveform application on any platform. The waveform development is a component based software design. For maximum reuse and reconfiguration, the SCA defines waveforms and platforms as a set of interconnected components (Bard and Kovarik, 2007). These components can be reused and are independent. They encapsulate their behavior and provide certain functionality, exposed through interfaces. We divide this system into three main parts as shown in Fig. 4: Core Framework (CF), CORBA ORB middleware and an Operating System.

The ORB and the Operating System depend on the underlying hardware system but all the Core Framework must have the same function.

It provides:

- Collection of services used by the waveforms and the other applications
- Software, which enables the installation, configuration, management and the control of waveforms
- File system to manage the waveforms
- Hardware interfaces to enable the abstraction of the platform

As shown in Fig. 5, we can divide the CF into components (Bard and Kovarik, 2007). It is also important to know that every entity in SCA has a Universally Unique Identifier (UUID) as defined by the DCE UUID standard adopted by CORBA OMG. This UUID allows for the identifying of every entity when the CF discovers the platform resources, hardware devices and software components, using the CORBA services.

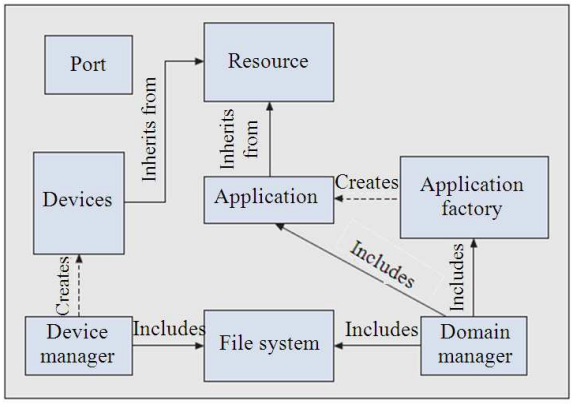

Fig. 5: SCA architecture

CORBA performance: As the real-time system behavior and memory footprint are important for real-time embedded developers, ORB performance is crucial for selecting a CORBA middleware. There is a widespread belief that inserting a CORBA middleware to a system imposes another "layer". CORBA has a small overhead to the embedded system; this overhead varies among CORBA implementations (Puder *et al*., 2006) and the CORBA "layer" replaces the overhead added by message processing delay which would have been imposed anyway OIS.

Benchmarking ORB performance: One of the important factors of ORBs benchmarking is the time it takes to complete message invocation. Message transfer time using TCP/IP socket program without ORB represents an important measurement. For a precise calculation of the CORBA overhead time, we can get the difference between TCP/IP socket time and ORB in the loop time (Ruh *et al*., 2000).

CORBA overhead types: Space and Time are the two major CORBA overheads in the embedded systems. We define the CORBA "Space" overhead as the number of bytes required for the data that constructing CORBA message at GIOP or IIOP in TCP/IP networks (Ruh *et al*., 2000). There are additional overheads associated with TCP/IP systems such that:

- Ethernet imposes 26 bytes per each frame
- IP inserts 12 bytes per each packet
- TCP inserts 24 bytes (plus options field) per each packet

Furthermore, the CORBA IIOP protocol imposes between 40-80 bytes per each CORBA message to this overhead. The major component of the system overhead comes from how the system developers define the interfaces (Henning and Vinoski, 1999). From performance perspective in CORBA IIOP protocol, it is important to note that if we send a fewer big messages, the system performance increases than the case of sending many small ones.

We define another CORBA overhead, which is "Time", as the time that ORB takes for processing each message (Ruh *et al*., 2000). This overhead is divided into two main components that are fixed and Variable components OIS. When benchmarking different CORBA implementations, we should study both overheads.

Fixed overhead: We define fixed overhead as the overhead due to ORB message invocation with null arguments. The major components of fixed overhead are ORB demultiplexing time, performing up-call, OS context switching, OS system calls and other similar tasks occurs every message transfer. Locating objects quickly, invoking operations on objects and avoiding unnecessary OS context switching and OS system calls are the main parameters for minimizing the fixed overhead (Tari and Bukhres, 2004).

Variable overhead: We define this overhead by marshaling and un-marshaling times, which varies in terms of number of parameters and parameters' data type of a given message. The ORB must quickly marshal and un-marshal data; efficiently use a transport and avoid building a multiple copies of data (Henning and Vinoski, 1999).

MATERIALS AND METHODS

Profiling ORBExpress DSP ORB middleware: The performance of the ORB has a great impact on the overall performance of the distributed SCA radio system. This is because all inter-component communications are established using CORBA messages OMG. We propose two specific test scenarios to generate profiling results:

Invocation: Measures the roundtrip cycle count for a simple method invocation with no arguments.

Marshaling: Measures the roundtrip cycle count for a simple method invocation with basic arguments.

In addition, we propose three different argument types to evaluate ORBExpress DSP middleware performance:

- Basic Data Type
- Array
- Sequence

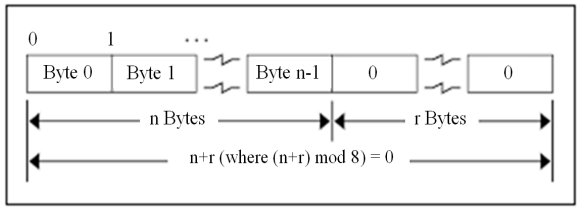

Fig. 6: The encoded array

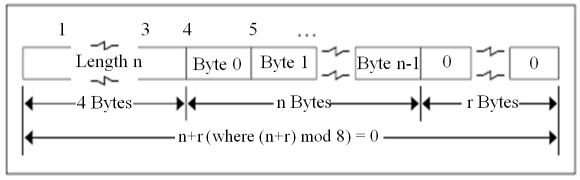

Fig. 7: Encoded unbounded sequence element with four bytes length identifier

Table 1: CPU clock cycles needed to characterize ORBExpress DSP marshaling profile

	Octet	Short	Float	Double
Basic marsh.	8427	8253	8075	7798
Array marsh.	24609	37566	67461	125708
Sequence Marsh.	24940	40826	73456	121013

Assuming that, both array and sequence data types are with a length of 1024 elements and all method implementations were empty. In the evaluation process, we should pass a CORBA Environment variable as an argument on each request due to the lack of exception support in embedded C++ language under Code Composer Studio (CCS) IDE. We should make this even for interfaces with no arguments defined in their IDL definitions. Client and Server where launched as different tasks with priorities two and one, respectively (Murtada *et al*., 2011). We summarize the ORBExpress DSP profiling results for different data types in Table 1.

ORBExpress DSP middleware has a custom encoding mechanism that differs from the CDR encoding. It assumes eight bytes buffer for data marshaling process OIS.

Fixed-length arrays of elements starting from 0 through (n-1) are encoded by checking the number of elements (n) if it is a multiple of eight. In case of it is not a multiple of eight, the array is padded with zero bytes.

Counted arrays provide the ability to encode variable-length arrays; it is called sequences, of homogeneous elements. The sequence is encoded as the element count n, an unsigned Long integer identifier, followed by the encoding of each of the sequence elements and starting with element 0 and then progressing through element (n-1). Sequences are variable-length vectors, or open arrays. Sequences can contain any element type and can be bounded or unbounded. An unbounded sequence can hold any number of elements up to the limits of unsigned Long integer. In addition, bounded sequence can hold any number of elements up to the maximum bound. In case of bounded sequence, there is an additional four bytes to represent the maximum bound of the bounded sequence. To describe the encoding process in Arrays and Sequences respectively, see Fig. 6 and 7.

For an ORB, the very first time a client makes a request to a server; the execution is longer than subsequent requests PrismTech. This extra delay is due to the binding of new connections, which is a one-off overhead. However, subsequent calls do not need this overhead. In ORBExpress DSP ORB middleware, when we establish the mirror transport; which we used for transport between client and server in our study; all subsequent requests between client and server have the same time OIS. In addition, what time an ORB establishes a connection and how many connections have been used between processes is different with different ORBs (Puder *et al*., 2006). ORBExpress DSP multiplexes invocations over a single connection between client and server. This is true even if multiple threads within the same client are all talking to the same server process. Other ORBs may use additional connections, which will cause usage of additional resources and thus limiting the scalability of certain ORBs (Tari and Bukhres, 2004).

RESULTS

There are a couple of interesting points in ORBExpress DSP middleware profiling results as shown in Table 1. First, for basic data types, if the length (in bytes) of the basic data type increases, then the marshaling speed becomes faster. This is due to the time needed for data padding process, for basic data types of the size less than eight bytes, when encoding the basic data types. For instance, the middleware encoder is obliged to pad a data buffer that contains an Octet data type with seven bytes to be marshaled correctly; while six bytes padding is needed to marshal a Short data type. When we target a high performance in CORBA messaging, it is logical for ORBExpress DSP middleware to enlarge the encoder buffer size to eight bytes rather than the four bytes buffer used in CDR encoding.

The second point of interest is that, there is a little difference between marshalling of arrays and sequences. Even though they are both of the same length, but the array marshalling is faster than the

sequence marshalling when the data type size is less than eight bytes. Here, we can summarize all the cases for marshaling the arrays and sequences as follows:

- If the sequence element length is less than eight bytes, then it needs to be padded with zeros
- For array marshaling, the ORBExpress DSP encoder checks for the whole array size if it is a multiple of eight. In case of not a multiple of eight, then the array is padded with zeros
- If the sequence element size is a multiple of eight, then there is no padding needed. Thus, the sequence marshaling is faster in this case. This is due to the encoder rule that is the ORB is obliged to marshal the whole array, but it is not mandatory to marshal all elements in the sequence. ORB can marshal selected elements in the sequence rather than all elements in the case of array. For instance, if we have consecutive zeros in an array, the ORB is obliged to marshal all elements of the array including consecutive zeros. Nevertheless, it is not mandatory to marshal these consecutive zeros in the sequence

There is a need to characterize ORBExpress DSP middleware for non-argument method invocation (Balister *et al*., 2006). We find that, when a client invokes the server with a simple method with no arguments, then it takes about 10.4 microseconds at 720 MHz processor speed.

It was not that difficult to get actually the number of cycles involved in establishing a mirror transport connection, so here are the cycle counts for both ORBExpress DSP configurations as shown in Table 2. Note that, small configuration forces the compiler to optimize for the code size using a specific compiler switches. In addition, fast configuration uses the compiler switches that optimize code for speed and making the code potentially faster on some platforms OIS.

From Table 1, ORBExpress DSP takes 8075 clock cycles to make a roundtrip marshalling call with a single float, while it takes 67461 clock cycles to send an array with 1024 floats. Actually, it only takes 65.9 clock cycles per float element in the array. Thus, we need the block processing in SCA radio systems.

Table 2: Clock cycles needed for mirror transport establishment in both ORBExpress DSP configurations

Configuration type	Number of clock cycles	Time (μ sec.)at 720 MHz
Small DSP configuration	9427	13.4
Fast DSP configuration	8625	11.2

DISCUSSION

Impact on data rate performance: The framework overhead incurred during instantiation and waveform deployment can be arranged to happen off-line, therefore not affecting the system throughput PrismTech. The only aspect of the SCA that impacts the system throughput is the dependency on CORBA for inter-component communications. The maximum system data rate depends on many factors: algorithm processing delays, framework delays, analog to digital conversion rate (Balister *et al*., 2006). In order to isolate the impact of the framework, we use the results shown in Table 1 to estimate an upper bound for the system data rate. This table shows the number of clock cycles that takes to send a CORBA message with different parameters to an empty interface in a collocated server. That is, there are no processing or transport delays.

The maximum achievable data rate is given by $1/(T_{fr} + T_m + T_{tr})$, where T_{fr} is the delay due to interface adapters, T_m is the delay due to middleware processing and T_{tr} is the delay due to transport mechanisms. In our system, we are only considering T_m because no interface adapters are required and no transport mechanisms have been developed at this time. T_m is given by:

$$T_m = \frac{D_t S_s}{N_p n} \quad (1)$$

From Eq. 1, D_t is the measured transfer delay as shown in Table 1, S_s is the number of samples per symbol, N_p is the packet size and n is the number of bits per symbol. To estimate the maximum data rate allowed by the framework, we assume $S_s = 8$ and $n = 1$. The clock speed in our system is 720 MHz. Substituting these values into Equation 1 for a single float type transfer, that according to Table 1 takes 8,075 cycles, the maximum data rate achievable is 11,145 bits per second. However, if we consider sending an array of 1024 floats, the transfer takes 67,461 clock cycles allowing a maximum data rate of 1,366,122 bits per second. This result highlights the need of block processing when dealing with an SCA radio system where SDR developers have to tradeoff latency and performance.

CONCLUSION

This study shows the behavior of inter-component communications in a SCA OE and demonstrates the necessity of having good knowledge about it to design efficient waveforms.

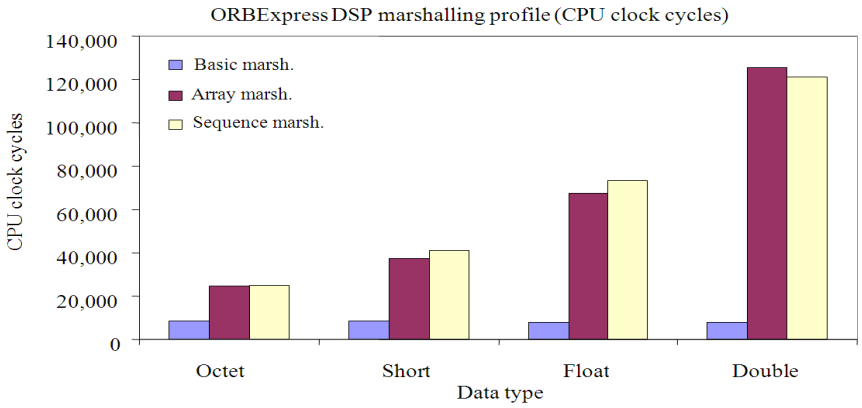

Fig. 8: ORBExpress DSP marshaling profile in terms of CPU clock cycles

In ORB marshalling profile, the latency time for basic data types needs too many clocks compared with block marshalling, which is a significant overhead. Thus, we need block processing to enhance the system performance in terms of inter-component messaging. This leads us to consider the impact of the choice regarding the size of data transmitted. Good Knowledge of the communications in SCA architecture requires a good understanding of CORBA. The challenge is to find the good trade-off between component reusability and system latency due to ORB communication messages. Performance benchmarks show that, although using CORBA for inter-component communications introduces delays and overheads, we can reduce the overall effect by sending packets of data instead of single elements. On the other hand, ORBExpress DSP introduces an eight bytes encoder buffer that targets a large chunk of marshaled data to increase ORB performance. From Fig. 8, using the arrays to marshal elements, of the size less than eight bytes, is better than using sequences. However using the sequences to marshal elements, of the size equal to a multiple of eight bytes, is better than using arrays.

REFERENCES

Balister, P.J., M. Robert and J.H. Reed, 2006. Impact of the use of CORBA for inter-component communication in SCA based radio. Proceedings of the SDR Forum Technical Conference, (SDRFTC' 06), Durham Hall, Blacksburg, VA., pp: 1-4.

Bard, J. and V.J. Kovarik, 2007. Software Defined Radio: The Software Communications Architecture. 1st Edn., John Wiley and Sons Inc., ISBN: 0470865180, pp: 462.

Grisby, D., S.L. Lo and D. Riddoch, 2009. The omniORB version 4.1 user's guide. AT and T Laboratories Cambridge, Cambridge.

Henning, M. and S. Vinoski, 1999. Advanced CORBA Programming with C++. 1st Edn., Addison Wesley, Reading, Mass., ISBN: 0201379279, pp: 1083.

Murtada, W.A., M.M. Zahra, M. Fikri, M.I. Yousef and S. El-Ramly, 2011. Design and implementation of an efficient software communications architecture core framework for a digital signal processors platform. Am. J. Eng. Applied Sci., 4: 429-434. DOI: 10.3844/ajeassp.2011.429.434

Puder, A., K. Romer and F. Pilhofer, 2006. Distributed Systems Architecture: A Middleware Approach. 1st Edn., Elsevier, Amsterdam, Boston, ISBN: 1558606483, pp: 323.

Ruh, W.A., T. Herron and P. Klinker, 2000. IIOP Complete: Understanding CORBA and Middleware Interoperability. 1st Edn., Addison-Wesley, Harlow, ISBN: 0201379252, pp: 262.

Tari, Z. and O. Bukhres, 2004. Fundamentals of Distributed Object Systems: The CORBA Perspective. 1st Edn., John Wiley and Sons Inc., Hoboken, NJ., ISBN: 0471464112, pp: 424.

Tsou, T., P. Balister and J. Reed, 2007. Latency profiling for SCA software radio. Proceedings of the SDR Forum Technical Conference and Product Exposition, (FTCPE' 07), Virginia Tech, Blacksburg, VA, USA., pp: 1-6.