The Enduring Myths of the Software Communications Architecture

Steve Bernier, NordiaSoft
Mathieu Michaud-Rancourt, NordiaSoft
François Levesque, NordiaSoft
Juan Pablo Zamora Zapata, NordiaSoft

Abstract - As with any new technology that makes bold promises, the adoption of Software Defined Radio (SDR) and the Software Communication Architecture (SCA) went through multiple phases that are best described by the Gartner Hype Cycle chart. The hype behind SDR technology and the early success stories of the SCA triggered significant publicity. The SCA was propelled by hype to a tipping point that Gartner refers to as the "Peaks of Inflated Expectations". The interest in the SCA started to wane as experiments and implementations failed to meet all the expectations. The bad press that followed dragged the SCA in a downward spiral of negative hype that is recognized as the "Trough of Disillusionment" phase. The SCA only started to pull out from the spiral after 2010 as a new generation of SCA-based products was brought to market. With more products being produced and the early international adopters, the SCA reached the "Slope of Enlightenment" phase. Nevertheless, the bad press that came with the disillusionment gave birth to many myths regarding the SCA. This paper identifies the enduring myths of the SCA and explains why they can be laid to rest.

Keywords - Framework; Software Components; Middleware; Software Architecture; SCA; CORBA; Performance

Introduction

One of the first major research projects to explore Software Defined Radios (SDR) was SpeakEasy in 1991 [1]. The project pioneered large-scale use of processors to perform most of its signal processing. It clearly established software could be used to process radio signals in real-time. At the same time, the project demonstrated that writing software for a specific hardware platform was an issue. It took 3 years to handcraft the software for the SpeakEasy platform that Moore's Law made obsolete in eighteen months. One of the conclusions [2] of the SpeakEasy project was that software needed to be made as platform-independent as possible.

Following the SpeakEasy project, the US DoD launched the Joint Tactical Radio System (JTRS) program to develop and acquire a new generation of radios to support tactical communications for all services [3]. The JTRS radios would become the first major wave of military software defined radios to be manufactured. Taking into account the lessons learned from the SpeakEasy project, the JTRS program imposed the use of an open architecture that would make software more independent from the hardware platform. The architecture is called the Software Communications Architecture (SCA).

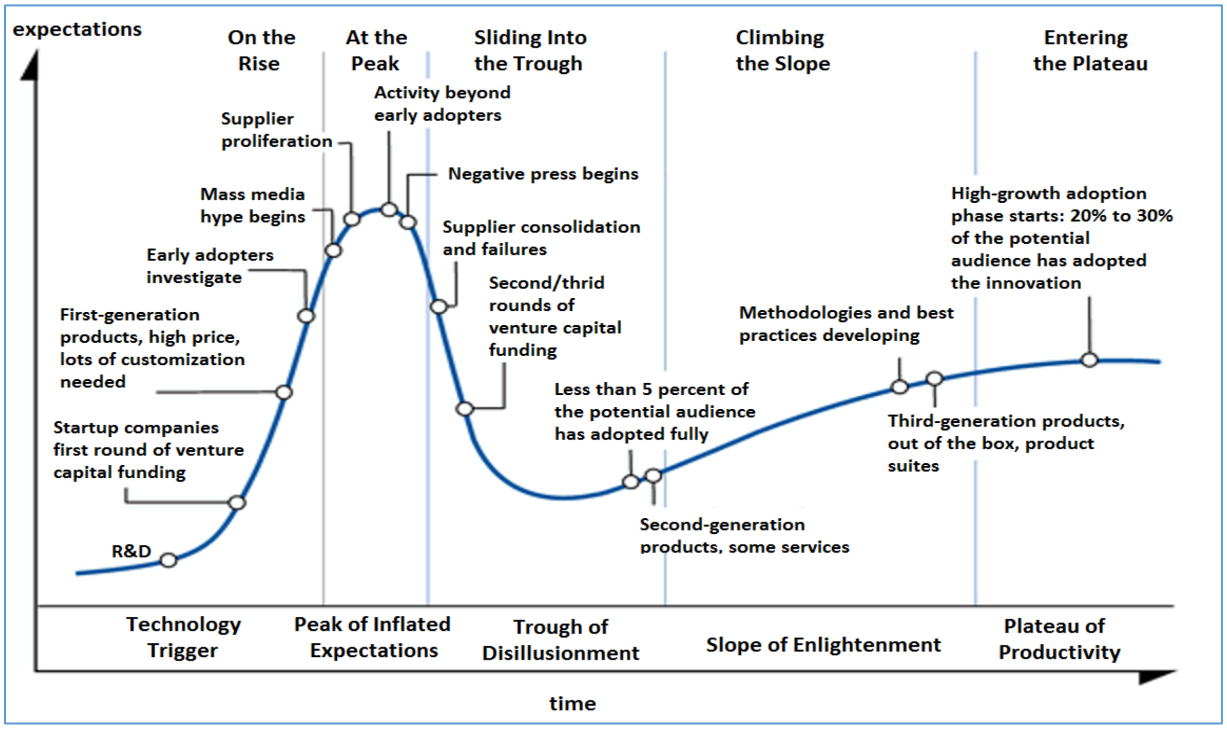

Figure 1 - Gartner Hype Cycle Chart

Even though the concept of SDR had been around for some years, it became significantly more popular when the multi-billion JTRS program was launched. The hype behind the concept of SDR fueled the popularity of the SCA which, like any new technology, went through multiple phases of adoption. The Gartner Hype Cycle chart (Figure 1) describes the different phases of adoption starting from the moment of inception to the mainstream usage. The JTRS program was presented as delivering PC-style radios that would easily support multiple communication applications. The program funded several manufacturers to investigate and develop the technology. The concept captured people's imagination and continued to gain in popularity until it became apparent that it was a possibility that not all the expectations would be met. The first radio prototypes took several minutes to boot and nearly as much time to switch from one communication protocol (i.e. application) to another. Eventually, the JTRS technology reached the phase called the "Peaks of Inflated Expectations" and the community suddenly became more critical of the technology.

The JTRS program was very disruptive because it was not only changing how radios were built, it was completely transforming the process by which radios were acquired. For the first time, radio platforms were being acquired separately from the communications protocol software (called waveform applications). Such an approach required significant adjustments in the acquisition process, which disturbed the ecosystem that was in place. But until the first prototypes of SCA radios came out, the detractors of the SCA had been fairly quiet. The interest in the SCA continued to wane as more experiments and implementations failed to meet all the expectations. The detractors of the SCA became more vocal about the performance issues of the early prototypes. Eventually, the fact that the early radios failed to meet all the requirements and the delays in deliveries of radios forced the US DoD to react by changing the scope of the program. The requirements became a moving target for an industry that was struggling with the new technology [4]. The situation led to a storm that pulled the JTRS in a downward spiral of negative hype that is recognized as the "Trough of Disillusionment" phase.

Building SDR products was a challenge that an open software architecture could not solve alone. Some will argue the JTRS program never really pulled out of the storm. Others think the program succeeded at fostering a new, and very disruptive technology [4]. As of the time of writing, the JTRS program led to the successful deployment of hundreds of thousands of military radios [5]. What cannot be denied is that SDR and SCA managed to weather the storm. The SCA was eventually adopted by major SDR acquisition programs worldwide. Both the German software defined future joint radio system (SVFuA) program [6] and the French COmmunications Numeriques TACtiques et de Theatre (CONTACT) program [7] have made the SCA a requirement. In fact, the SCA has become the worldwide de facto standard to build software-defined platforms for tactical military communications [5]. The adoption of the SCA as a technology is now moving into the "Slope of Enlightenment" phase. More companies are adopting the SCA and more products are being deployed. As SDR is gaining traction in other domains [8], SCA is starting to be more widely adopted and will transition to the next phase which is called the "Plateau of Productivity".

The SCA specification is open and free to download. The US DoD Joint Tactical Networking Center (JTNC) [9] is the custodian of the SCA specification and has recently published a major update (version 4.1 [10]). The specification evolves through change proposals, many of which come from Wireless Innovation Forum (WinnF) and its members [11]. The WInnF [12] is a non-profit mutual benefit corporation dedicated to advocating for the innovative use of spectrum and to advancing radio technologies that support essential or critical communications worldwide.

While the SCA is gaining in popularity, it still suffers from several myths that come from the "Trough of Disillusionment" phase. Some of those myths are very persistent and have been around for a decade. The following sections identify the most enduring myths of the SCA and explain why they can be laid to rest.

Myth #1: SCA is for military radios

The requirements imposed by the JTRS program led to the creation an open architecture to address obsolescence and enable technology insertion. The software architecture is called the Software Communications Architecture (SCA). As of 2017, over 500,000 SCA-based radio units have been deployed by different manufacturers located all over the world [5]. Because the SCA was initially developed for a tactical radio program, the SCA is often assumed to be specific to military radios. While that is a fair assumption to make, it could not be more wrong. The reality is that SCA is completely domain-agnostic. It has been designed to support any kind of embedded system that is made of various types of processors. Much like in the smartphone market, the SCA provides the ability for embedded platforms to host several applications. To do so, SCA divides the software in two categories of components: platform components and application components. Platform components provide access to the hardware embedded in the platform. SCA application components use the platform components to provide end-user services. Application components are very portable across different platforms and can be added post manufacturing. This characteristic is essential to ensure JTRS radios could run various applications (e.g. AM, FM, Link-16, UHF SATCOM DAMA, and more). Being able to efficiently port applications to many platforms reduces costs and facilitates technology insertion. It is important to underline that the SCA specification does not define any domain-specific APIs for platform components. The SCA is a domain-agnostic architecture that runs software components that can plug-and-play with each other. It only defines a few generic platform-related APIs for common hardware such as GPPs, DSPs, GPUs, FPGAs and a few services to support a file system, events, and logging. Adopters of the SCA must define their own set of domain-specific APIs.

Domain-specific APIs are defined in separate documents. They describe APIs that SCA Platform components must implement for SCA applications to use. For example, the automotive industry could define APIs that would let SCA applications get access to devices like a speedometer, steering wheel actuators, obstacle sensor, brake actuator, and more. Such APIs could be used to develop applications for parking assist, adaptive cruise control, lane departure control, and more. Using this approach, not only would cars be software-defined, but the SCA applications would be portable to any automobile platform that provides the proper set of domain-specific APIs, regardless of the processors, operating systems, or communication buses being used.

In the case of JTRS Military Tactical Radios, the domain-specific APIs are described in documents referred as the “JTNC APIs” [13]. The APIs define how to interact with a GPS, a timing source device, an audio device, a vocoder device, a transceiver device, a serial port, and an Ethernet port. Using the JTNC APIs, SCA applications can implement different communication protocols like AM, FM, Link-16, UHF SATCOM DAMA, and more. The JTRS SCA applications are kept in a repository managed by the US DoD and are portable to radios of various form factors built by different manufacturers.

The SCA defines how to implement reusable software components that can be connected to other components at run time. Components go through a standard life-cycle that covers, instantiation, initialization, configuration, connections, starting, stopping, disconnections, releasing, and interruption of execution. Interactions between components is not based on implementation knowledge. That means components can interact with each other even if they are implemented by different organizations which may even use different programming languages. Also, SCA components can transparently interact with components running anywhere: same or different program space, same or different processor/core, same or different boards, across a backplane or over the Internet. Interactions can be carried using different transport protocols to optimize performances.

Open software architectures like the SCA can be used to build any kind of system. So far, the SCA has been used mainly for military tactical radios. It has more recently started to be used for test instruments [14] [15] and electronic warfare [8]. It has also been used to build a few robots [16]. The key thing to remember is that, while the JTRS program relies on the SCA to build military software-defined radios, there is nothing military or radio-specific about the SCA. The SCA is a software component architecture to build reusable software for embedded systems.

Myth #2: SCA is too large
The high-tech world we live in is completely transforming the way business is conducted. The technical problems that need to be solved are constantly changing and that has an impact on the form factors and the delivery models for most products. Because of the fast-paced nature of the market, system integrators and manufacturers now have to deal with obsolescence before their product is even fielded. To better manage obsolescence, manufacturers are moving from hardware-centric solutions to software-centric solutions because software is more malleable and is easier to deploy than hardware [17] [18] [19] [20]. At the same time, more government programs now require the use of an open architecture to future-proof their investment by facilitating technology-insertion [6] [7] [21] [22] [23] [24] [25] [26].

Since the launch of the JTRS program in the late 90s, several other markets have been impacted by the concept of Software-Defined platforms. Being able to completely redefine a device's functionality through software is at the heart of Software Defined Networks (SDN) which started in the early 2000s [27]. More recently, the IT world was swept by the concept of Software Defined Storage (SDS) and Software Defined Data Centers (SDDC). The automotive industry could be the next to go through a profound transformation with the concept of Software Defined Cars (SDC) [20] [28].

As the industry is moving from hardware-centric solutions to software-centric solutions, it is safe to say that more software needs to run and more run-time memory is required. However, the perception that SCA is too large is mostly related to the fact that many early adopters (years 2000-2004) attempted to use the SCA on hardware platforms that had been designed over a decade earlier. Many of those platforms could only host small amounts of software. For instance, handheld radios deployed during the late 1990s had in the range of 8 to 16 MB of RAM, some had even less. Deploying the SCA infrastructure on such legacy platforms proved to be a real challenge and early adopters of the SCA suffered through the transition to software-centric platforms. However, most platforms introduced in the early 2000s came with more memory. For instance, the first Apple™ iPhone, introduced in 2007, came with 132 MB of RAM [29].

Technically speaking, a platform is not software-defined simply because it contains some software. Microcontrollers and reprogrammable chips started to be used long before the term software-defined radio (SDR) was coined [30]. The key characteristic of modern software-defined product is the possibility to completely redefine its functionality through software without sending the product back to the manufacturer [31]. Software-defined platforms generally contain at least one processor that is more generic-purpose (GPP) and used to federate all of the embedded hardware (other processors, sensors, and actuators). It also coordinates the boot sequence which consists in programming the hardware by loading, launching, and configuring software/firmware on all the different processors of the platform.

SCA was designed to support technology insertion. It enables a platform to host multiple applications that could be installed post-manufacturing. Thus, SCA platforms must provide some form of mass storage device that can host new applications as they are added. Older generation radios had very little mass-storage capacity and in most cases, applications could not easily be reloaded post-manufacturing. The fact is that JTRS SCA radios and legacy military radios are separated by over a decade of technology advancements. Using the SCA on legacy radios was certainly a challenge.

However, the fact that JTRS SCA radios require more runtime and mass storage memory than legacy radios cannot be attributed to the SCA but to the requirements for which the SCA has been introduced (i.e. portability, reuse, lower cost, technology insertion, etc.). That being said, some of NordiaSoft customers have implemented SCA radios that require less than 32 MB of runtime memory for the full software stack which includes a real time operating system kernel, a TCP/IP networking layer, a flash file system, a POSIX layer, the drivers, the SCA platform components for the speakers, the microphone, the transceiver, the GPP, the DSP, and the FPGA. That also includes the NordiaSoft SCA Core Framework, the ORBexpress RT CORBA stack from OIS, and an SCA application made of 3 components.

In summary, platforms designed for software-defined systems typically provide more than enough memory for the SCA to be used. The SCA has been deployed on platforms with various form factors ranging from small battery-operated devices to larger truck-mounted multi-channel radios. As of today, over 500 thousand SCA radios have been deployed [5].

Myth #3: SCA is too slow
This myth implies that using the SCA to implement a software-defined radio is a barrier to real time performances. Like most of the myths, this one is rooted in the early days of the JTRS program when radio manufacturers were attempting to use the SCA on platforms that had not been designed to run sophisticated software.

During the early days of the SCA, one of the first issues to be reported was the long time it took to boot a radio. The myth is that some JTRS Radios took over 10 minutes to boot and reach a state where they could transmit or receive. While some early JTRS radio prototypes were indeed slower to boot, measurements published by Harris describe a much less dire situation. It is important to point out that the JTRS program had experimentation phases (called Step 2A, 2B, and 2C) during which radio manufacturers were asked to investigate different aspects of the SCA and JTRS APIs (now called the JTNC APIs). The prototypes created during that phase were based on legacy hardware and were not designed for optimum boot performances.

	AN/PRC-117F (C) legacy radio	1st iteration JMTR radio (2001)	2nd iteration JMTR radio (2004)	3rd iteration JMTR radio (2005)
Power up to Application	1x	8x	1.6x	Meet field requirement
Application switch time	1x	16x	2.3x	Meet field requirement
Parameter change time	1x	4x	2x	Meet field requirement
Rx/Tx turnaround time	1x	3x	1.75x	Meet field requirement

Table 1 Harris Early SCA Radios

The first measurements published by Harris [32] describe the performances of a prototype called the JTRS Manpack Test-bed Radio (JMTR) which were not as good as the existing Harris AN/PRC-117F (C) non-SCA radio (see “1st iteration” in Table 1). However, with some optimizations, the JMTR was able to deliver much improved performances [33] (see “2nd iteration” in Table 1). Ultimately, Harris was able to build an SCA radio that meets the field requirements [34] (see “3rd iteration” in Table 1). However, the early measurements that described issues were published in papers and presentations still accessible from the Internet which are being used to perpetuate the myth about the SCA being too slow.

This myth is very persistent despite the fact that hundreds of thousands of SCA radios, of various form factors, including handhelds, have been manufactured and deployed worldwide with great success [5].

Myth #4: CORBA is too slow
This myth is almost invariably linked to TCP/IP, which is the default transport for all ORBs. There is no doubt the overhead of TCP/IP can become a significant performance barrier for communications between programs running on the same processor or across a high-performance switched fabric. The issue with TCP/IP is even more important for embedded real-time systems like software defined radios that must handle large quantities of data that travels between different processors. Nonetheless, CORBA has been used to build many types of embedded real-time systems. It has been used for aircraft and flight control systems, airborne early warning systems, missile defense systems, weapon systems, air traffic control systems, distributed interactive simulation, automated stock trading systems, process automation, vehicle control systems, and more [35] [36] [37] [38] [39] [40] [41] [42] [43].

Unfortunately, CORBA is often compared with TCP/IP sockets and various messaging utilities like POSIX message queues, and ZeroMQ. The problem with such comparisons is that it compares apples to oranges. CORBA handles endianness, provides a control protocol, provides independence from transports, and supports different program-space configurations. Neither sockets, POSIX message queues, nor ZeroMQ offer the same features. The concept of a control protocol like the CORBA General ORB Interoperability Protocol (GIOP) is essential to create reusable components. What makes a component reusable is the ability to alter its behavior. Users must be able to start/stop a component, choose which components need to be connected together, configure component properties (code rate, frequency, modulation type, etc.). Controlling a component is typically implemented via a control protocol, which is precisely what GIOP provides.

The next few paragraphs introduce the result of an experiment designed to compare the performances of a number of solutions that provide different levels of support for heterogeneous embedded distributed systems. Figure 2, compares the speed of transmitting clear data via TCP/IP with the time it takes to send data that is marshalled/unmarshalled using JSON, CDR, and XDR. The data was produced with the same experiment for all the approaches. It uses a generator component that transmits 10,000 packets that contain 1000 doubles each. The experiment measures how long it takes for the packets to loop back to the generator component. The packets being sent by the generator are received and retransmitted by two pass-through components in a row. The last pass-through component in the chain sends the packet back to the generator which records the time of arrival and computes the round-trip time. Each experiment measures the average round-trip time for each packet in microseconds. The experiments were conducted using FreeScale i.MX6 Quad SABRE Automotive board that comes with an ARM® CORTEX®-A9 quad core processor with auto scaling of the clock up to 1 GHz, running Linux kernel version 3.14 with 2 Gigs of RAM. The ORB used for the experiments involving the CDR or CORBA is ORBexpress RT 3.0.3 patch Q61 ExRiSl for GCC 4.8.2 from Objective Interface Systems [44]. The experiments that involved JSON were conducted using jsoncpp [45].

In the context of a heterogeneous system, TCP/IP cannot be used without handling endianness. Therefore, when calculating the overhead of the SCA, the time required to handle heterogeneity cannot be considered overhead. Figure 2 shows the relative cost of dealing with endianness using different well-known techniques. The data is labelled as follows:

- "JSON" data is marshalled/unmarshalled with the JavaScript Object Notation (JSON) [46].
- "TCP/IP XDR" data is marshalled/unmarshalled with the External Data Representation (XDR) [47] commonly used on unix-flavored systems.
- "TCP/IP CDR" data is marshalled/unmarshalled with the Common Data Representation (CDR) [48] used by CORBA.
- "CDR-NOMARSH" data shows the performances of CDR when it detects that the source and destination processors have the same endianness and marshalling can be avoided.

In the experiments that involve JSON, XDR and CDR, even the two pass-through components unmarshal/marshal before sending the data to the next component.

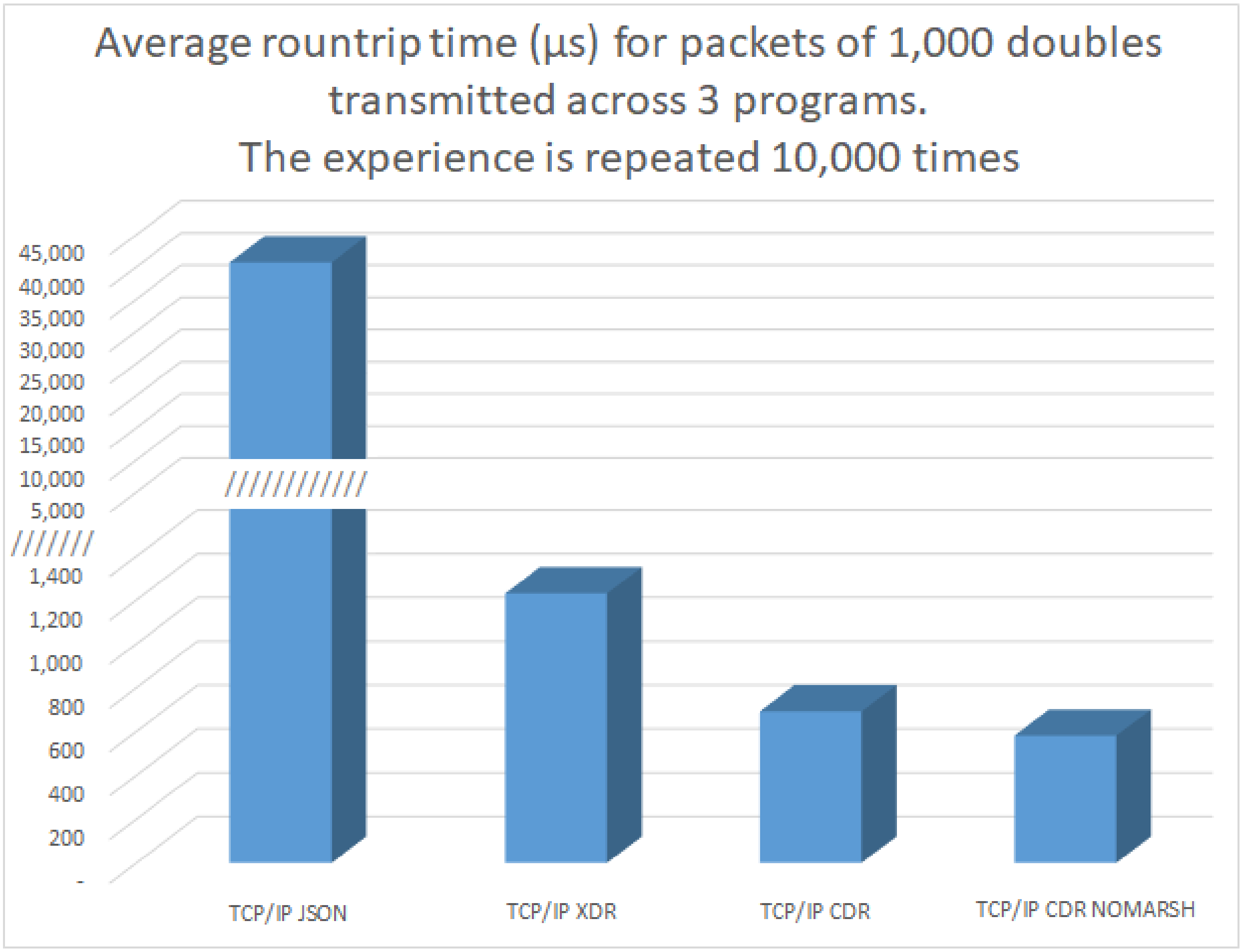

Figure 2 - Performances of different approaches for marshalling/unmarshalling data

JSON uses a textual representation of the binary data. It can be used with several programming languages including C/C++. However, while JSON can be faster than XML-RPC [49] [50], it has not been designed for real-time performances [51]. In our experiments, JSON was 61 times slower than CDR. Part of the reason why JSON is slower is the time it takes to convert from binary to text (and vice versa) which is longer than simply reordering bytes as XDR and CDR do. However, JSON performances also suffer from the fact that it inflates the number of bytes to be transferred. For instance, to marshal 1000 doubles of 8 bytes each, JSON produces 13,876 characters (i.e. bytes).

Both XDR [47] and CDR use a binary representation that is more suitable for high-performance systems. XDR imposes a network-endianness (often called a network-byte-order) to represent the marshalled data. While using a binary representation is faster than a textual representation, relying on a network-endianness is not optimal. In the case network-endianness is Big Endian (the case with XDR) and both the source and destination processors use Little Endian, data still gets marshalled/unmarshalled unnecessarily.

CDR does not use a network-endianness. Instead, it sends the raw data along with an indicator that identifies the source endianness. It is up to the destination program to marshal the data when it uses a different endianness than the source processor. In short, using CDR, the data is never marshalled when both the source and the destination processors use the same endianness. Furthermore, as it can be seen in Figure 2, CDR is clearly faster than XDR even when marshalling/unmarshalling is required.

All of the measurements presented in Figure 2 rely on TCP/IP as the transport layer. However, as mentioned previously, TCP/IP imposes a level of overhead that may be an issue for embedded real time systems. An important feature of CORBA that many developers don't seem to know about is the possibility to use different transports. Over the years, CORBA has been used with a number of different high-performance transports [52] [53] [54] [55] [56] [57] [58]. Newcomers to CORBA would benefit from reviewing the existing literature regarding alternate transports and CORBA performances [59] [60] [61]. Since not all CORBA products are created equal, it is also worth reviewing the performance comparison between different CORBA products that Lockheed Martin has made public [62] [63].

Another key feature of CORBA is that it can support in-process invocations. This performance improvement is possible when two interacting CORBA objects run in a same program space [64] [65] [66] [67] [68]. In such a case, it is possible for the ORB to carry out invocations by calling a local function, bypassing any marshalling/unmarshalling and request multiplexing/demultiplexing. This optimization yields optimum real-time performances on small single processor systems. But perhaps the most important aspect of the alternate transport feature is that it allows CORBA objects to switch from one transport to another without changing a single line of source code in the business logic. Further, a single CORBA object can use different transports to carry out interactions with different objects in parallel. This ability provides tremendous benefits in terms of reducing the cost of porting and adapting applications from one system to another.

Figure 3 introduces four more experiments. All the labels that contain the keyword “SCA” identify measurements that were produced using an SCA version 4.1 application. The SCA application, which relies on CORBA, reproduces the same experiment which consists of sending packets of 1000 doubles that flow through two pass-through components and loop back. The application is made of three SCA components: a packet generator component and two pass-through components. The Core Framework being used to run the SCA application is the “eCo Core Framework” from NordiaSoft which implements SCA version 4.1. All the experiments are executed on the same ARM processor using the same ORB and same version of Linux as with the previous experiments.

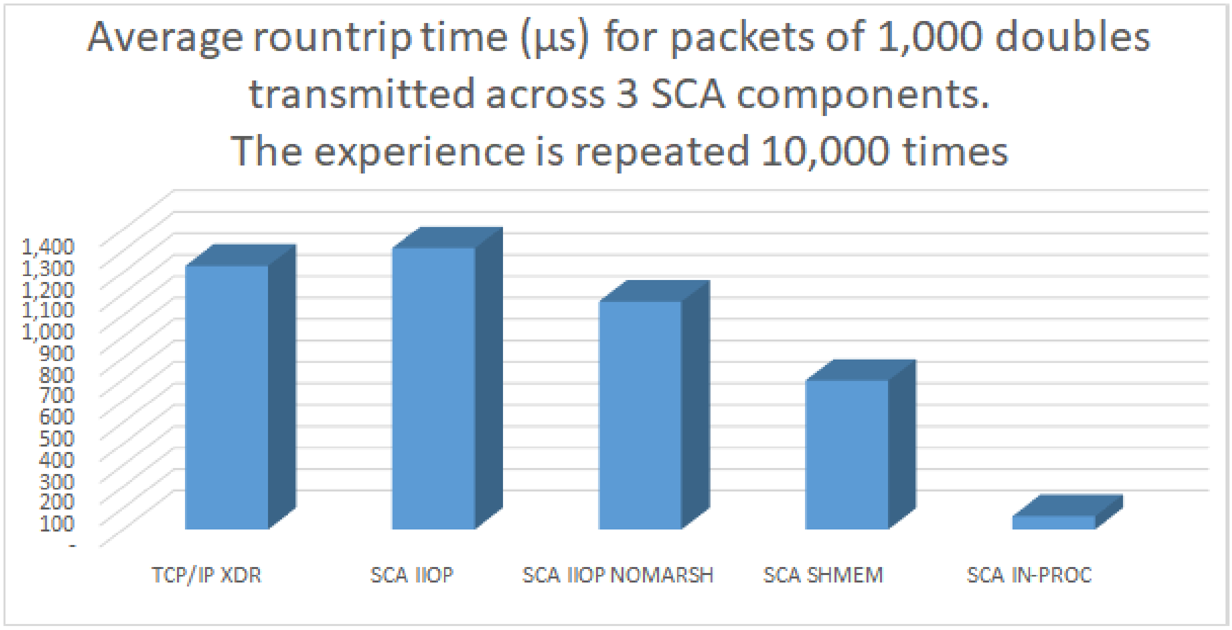

Figure 3 - Performances of SCA Components using different transports

Figure 3 shows data is labelled as follows:

- "TCP/IP XDR" is the same non-CORBA experiment as in Figure 3. It serves as a reference to compare performances with the four different experiments that use an SCA application.
- "SCA IIOP" describes the performance of the SCA application with the ORBexpress TCP/IP transport (known as IIOP).
- "SCA SHMEM" describes the performance of the SCA application with the ORBexpress shared memory transport.
- "SCA IN-PROC" describes the performance of the SCA application with all three components running in a single program space.

As illutrated by Figure 3, using SCA offers performances that match the use of programs that use TCP/IP and XDR instead of CORBA. In fact, when no marshalling/unmarshalling is required (see "SCA IIOP NOMARSH"), SCA outperforms the program that uses TCP/IP and XDR. Of course, the advantage of using SCA is that it provides independence from the program space mapping, programming language, transport protocol, and endianness. And it does this at virtually no cost from a performance stand-point. Thanks to CORBA, SCA also provides a sophisticated control protocol that allows SCA components to be controlled by any third-party component.

It is important to note that the SCA application used in all four experiments is exactly the same. There are no changes to the source code between the three experiments. Without the SCA, development would typically be done by implementing four different programs to handle TCP/IP, TCP/IP with XDR, shared memory, and in-process invocations (i.e. no TCP/IP).

As indicated in a survey conducted by the Embedded Systems Design Journal [69], over 50% of multi-processor systems are heterogeneous. A more recent survey [70], indicates close to 40% of the systems being upgraded included a different processor and 24% included new connectivity capabilities. Furthermore, 14% of the upgrades involved new or different peripherals and 10% included a new operating system. In terms of performances, the survey indicated that 59% of the systems require real-time capabilities, 56% require digital signal processing, and 54% require networking capabilities.

In short, as systems evolve, they become more complex, more heterogeneous, and still require real-time performances. To satisfy the requirements of the market, sophisticated technologies like SCA and CORBA must be used to ensure best performances across heterogeneous platforms and to maximize reuse of the existing source code which represents over 60% of the financial resources in most projects. Using a proven technology in terms of time-to-market is even more important at a time where, according to the same survey, close to 62% of the projects finished late or were canceled.

Myth #5: SCA is too expensive

One of the most prevalent SCA myths states that the technology is too expensive to implement.
As outlined by US Navy Captain K. Peterson [71], sceptics often cite the doubling of cost estimates of the Joint Program Executive Office Clusters 1-5 throughout the 2005 - 2007 period, or even the cancellation of Ground Mobile Radio (GMR) program in 2011. But those claims are frequently offered as sound bites without full explanations of the reasons for failure, which include among other things, program scope and management, constant feature increase, and even the laws of physics. Also having a significant impact in the overall results was the lack of understanding and embracement of the technology. Some organizations reluctantly used the SCA because it was an imposition (i.e., procurement requirement) and often struggled and failed to achieve minimum levels of SCA success. Their approach was to keep building proprietary SDRs as before, and only address the SCA requirements as an afterthought. On the opposite side, there are organizations that have found tremendous success using the SCA as they have been able to substantially reduce porting costs, shorten time to market [5] and have achieved better communication between developers [72].

For the first group of companies (reluctant embracers), they still had to pay the costs of the SCA technology and learning curve, but without realizing the benefits. The approach was akin to using a C++ compiler and not benefit from object-oriented features. The problem lies with the lack of understanding that the SCA is not only a specification, but also a design philosophy. SDR development often requires developers to have expertise in inter process communications (IPCs), real-time operating systems (including considerable knowledge on the usage of threads, tasks, address spaces, timers, etc.), and operating environments (build files, compilers, munchers, linkers, shared libraries, dynamic loaders, file systems, etc.). Under such approach, the signal processing expertise of some team members was often sidelined in favor of expertise on IPCs, OS, OEs and so on. With the SCA, developers also had to deal with software engineering concepts such as components and reuse, platform abstractions, and API development.

The biggest mistake that some organizations have made is to ask the same developers to take care of all of the above instead of making them work in the area where they have expertise. When an organization can focus some team members to exclusively work on preparing an SCA platform, others to exclusively work on signal processing logic, and the remainder of the team on building SCA Application components from the business logic, the productivity increases and the organization becomes more effective. Successful organizations have done so and have substantially reduced costs by incentivizing developers and designers to create reusable software. In those organizations, infrastructure has been separated from the application's business logic enabling high levels of reuse, having an impact on agility and costs, since development time decreases, robustness increases, and integration becomes easier. Time-to-market is also impacted due to shorter development cycles and faster integration of 3rd party business logic [73].

The final item that portraits SCA as too expensive relies on anecdotal records from the early 2000s when no COTS SCA development tools were available. At that time, companies had to develop the entire SCA infrastructure in-house (SCA Core Framework for a specific Operating Environments as well as SCA compliant Applications) with little or no software tooling. This is akin to implementing a compiler and linker before starting to implement applications. Such an approach was very time-consuming and proved to be expensive. Today, there is an entire SCA ecosystem of COTS products and services. This frees developers to focus on their core expertise without having to become SCA experts. The result is more efficient time usage across the industry.

Myth #6: SCA version 4.1 is CORBA agnostic

The SCA 4.1 specification follows the Object Management Group (OMG) Model Driven Architecture (MDA) paradigm, where software is initially defined as a Platform Independent Model (PIM) and later transformed or augmented into Platform Specific Models (PSM). The reasoning for such separation lies in the fact that the business logic of a software application can be separated from the details that are specific to some software technologies or hardware platforms. When building a parking assist application for an automobile, a PIM could identify how signals coming to/from the steering wheel, brake actuators and obstacle sensors are to be processed and relayed to the other hardware/software pieces of the application. A PSM can further refine such a PIM adding details regarding implementation technology like data formats, processor architectures, operating systems, programming languages, inter process communication mechanisms, and more.

In the case of version 4.1 of the SCA, the main specification document was modified to create a separation between the higher level concepts (PIM concepts) and the lower-level implementation technologies (PSM concepts). The key difference between SCAv2.2.2 and SCAv4.1, is that the latter is structured to allow the definition of additional PSMs that could be based on middleware technology different than CORBA. While the documents of version 4.1 of the SCA specification have been revamped to eliminate any implicit reference to CORBA as the communications transport, no other PSM alternative has been standardized or even proposed. In other words, at the moment, the SCA defines a single PSM which is CORBA-specific. Thus, any compliant implementation of SCA v4.1 can only be implemented using the SCA CORBA PSM. The following paragraphs provide reasons that might explain why this is the case.

First, the current SCA specification is not completely platform-independent (PIM). It is very likely that a non-CORBA PSM would have an impact on the behavior and the interfaces of some components as defined in the current PIM. For example, for systems with a single address space, what would be the benefit of having meta-data (i.e. the XML domain profile) to describe multiple implementation for each component. Having a single implementation would also have an impact on the rules to resolve the final value of a property. Furthermore, it would not be useful for an ExecutableDevice to advertise the characteristics of an operating environment (e.g. processor). In fact, one could even question the need for an ExecutableDevice since its purpose is to provide the Core Framework with access to processors in a way that is independent from a specific operating system.

The SCA has been developed to be used with a range of operating environments. It can be used with systems that only have one processor as well as with systems made of several processors of the same type or different types. It supports systems with that support a file system as well as those that do not. It supports systems with a single address space as well as systems with multiple address spaces. Those core concepts are part of the SCA PIM. This implies that whatever middleware is chosen for a PSM, it must support the core SCA concepts (i.e. PIM).

A new PSM proposal would also need to address post manufacturing technology insertion. Currently, CORBA provides the ability for a binary to interact with other binaries that were unknown at build time. This means binaries can interact without having to first share any source code. Using a PSM that would change this would bring a number of issues. It would likely eliminate the ability to allow post-manufacturing technology insertion. For instance, how would it be possible to install a new application into an existing platform? How would the existing software stack learn about the new application if no interactions are possible? How would the DomainManager be informed that a new application has been installed? Addressing those issues would certainly lead to changes to the DomainManager API which would require changes to the PIM.

If a new SCA PSM relies on a more basic form of IPC (e.g. Shared Memory, POSIX Message Queues, TCP/IP, etc.), the PSM would have to define message formats for each possible interaction. Source code would need to be written to create and parse each of the message formats, which would become transport-specific. Such a PSM would substantially impact portability of SCA components and therefore increase costs as well as time to market. To implement the PIM concepts of the SCA specification, a PSM would ideally need to offer support for 1) single and multiple processors, 2) single and multiple address spaces, 3) homogeneous and heterogeneous systems, 4) independence of transport, 5) location transparency, and 6) independence from programming language.

In summary, the SCA defines a single PSM which is CORBA-specific. Although it is possible to create a non-CORBA SCA PSM, there are a multitude of technical reasons why the SCA community has not created additional PSMs.

Conclusion

Over two decades ago, the military became the first customer to drive a transformation from hardware platforms to software defined platforms. It started with tactical radios but was soon followed by the commercial world with software-defined IT infrastructure. Time has shown that the US DoD JTRS program was indeed revolutionary. The PC-style radio transformed the industry of tactical radios worldwide. As for any new technology, the SCA went through several phases of adoption and emerged as a sound technology that it is now starting to make its way into different markets.

This paper addressed a number of myths regarding the SCA. It starts by highlighting that the SCA is in fact a software component architecture targeted at embedded systems. While it has been used mainly for military software-defined radios, the SCA has been designed for a very wide range of products. The paper also explains that the SCA requires very little memory in addition to the software-defined functionality applications already require. In most cases, the additional memory requirements come from the support for post-manufacturing upgrades and technology insertion that many of the early software defined products don't support. The myth about the SCA being too large is rooted in the early days of the JTRS program were radio platforms used very little software and thus, provided tiny amounts of runtime memory. The reality is that many battery-operated devices were built using the SCA with great success.

The myth regarding the real-time capabilities of the SCA is also addressed along with the importance of software reuse across different platforms to reduce costs and time-to-market. In the context of embedded systems, where most systems are multiprocessor and heterogeneous, it is imperative to be able to quickly port the software which often holds the bulk of the intellectual property that is so important for product differentiation. This paper compares the performances of different options to deal with platform independence. It explains that Plain Jane TCP/IP cannot be used as a benchmark reference. It also shows that CORBA comes with the required performance enhancing features to build a wide range of real-time embedded systems.

This paper supports that the myth of the SCA being too expensive comes from the early days of the JTRS program when radio manufacturers had to develop everything from scratch. In addition to everything else, the radio manufacturers initially had to become experts at developing middleware and frameworks. This is akin to asking a car manufacturer to develop its own compiler/linker tools before implementing infotainment applications. Today, there are COTS solutions that, not only provide a SCA-compliant framework, but offer sophisticated modeling and code generation tools. COTS solutions greatly enhance the speed of development and thus reduce costs and minimize time-to-market. COTS solutions have now been used on several generations of products and widely deployed. COTS solutions allow more manufacturers to hit the ground running building SCA-based products.

Finally, the latest version of the SCA specification documents have been structured to separate the high-level concepts (PIM concepts) and the lower-level implementation details (PSM). This has been done in response to a perceived pressure coming from the industry to allow alternatives to CORBA. However, even after many years, the only PSM available relies on CORBA. The paper explains that no other PSM has been proposed because no better alternatives to CORBA have been found. The reality is that in order to meet the requirements of the PIM, CORBA is still the most appropriate solution even from a performance stand-point. Another reason for not having a new PSM is that news developers eventually realize that CORBA has performance enhancing features they did not use to build their initial prototypes.

In summary, this paper provides evidences that can be used to put the enduring myths of the SCA to rest. It provides a wealth of reference material that will help readers distinguish myths from the reality. The hope is that this paper will elevate the discussion above the old myths.

References

[1] "SpeakEasy," [Online]. Available: https://en.wikipedia.org/wiki/SpeakEasy.

[2] R. Lackey and D. Upmal, "SPEAKeasy, the Military Software Radio," *IEEE Communications Magazine,* vol. 33, no. 5, 1995.

[3] S. I. Erwin, "Pentagon to Invest in PC-Style Radios," *National Defense, NDIA's Business & Technology Magazine,* 2002.

[4] B. Rosenberg, "From radios to waveforms: How JTRS is remaking itself as JTNC," *Defense Systems,* 2012.

[5] Wireless Innovation Forum, "SCA Standards for Defense Communications," 2015. [Online]. Available: http://www.wirelessinnovation.org/assets/Collateral_and_Supporting_Docs/sca%20sell%20sheet%20march%202015.pdf. [Accessed 24 10 2017].

[6] "Digital Tactical Communications for German Army Command Vehicles," *Defense Procurement International,* 2017.

[7] Defense Industry Daily staff, "Making CONTACT: France's Billion-Euro Radio Program," *Defense Industry Daily,* 2012.

[8] J. M. G. Bickle, "SCA 4.1 Perspective and Product Transition," in *2015 SCA Workshop - Wireless Innovation Forum*, Melbourne, Florida, 2015.

[9] "JTNC Home," [Online]. Available: www.public.navy.mil/jtnc. [Accessed 24 10 2017].

[10] "SCA 4.1 Specification," 08 2015. [Online]. Available: http://www.public.navy.mil/jtnc/SCA/SCAv4_1_Final/SCA_4.1_ScaSpecification.pdf. [Accessed 24 10 2017].

[11] Lee Pucker, "Wireless Innovation Forum Contributions to the SCA 4.1," *Military Embedded Systems,* 2014.

[12] "Wireless Innovation Forum," [Online]. Available: http://www.wirelessinnovation.org/.

[13] "JTNC Resource Catalog," Joint Tactical Networking Center, 20 8 2015. [Online]. Available: http://www.public.navy.mil/jtnc/Pages/resources.aspx?filter=cat-api. [Accessed 24 10 2017].

[14] L. Desjardins, "Cobham intros AXIe, but the software is the real surprise," EDN Network, 2016.

[15] EEWEB, "Building a Platform to Launch the Future," *EEWeb | Modern Test & Measure,* no. August 2016, 2016.

[16] S. Hong, J. Lee, H. Eom and G. Jeon, "The robot software communications architecture (RSCA): embedded middleware for networked service robots," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International* , Rhodes Island, Greece, 2006.

[17] PWC Global, "Software disruption accelerates".

[18] Dali Wireless, "Pipeline Magazine: The Evolution of Wireless Networks: Shift from Hardware-centric to Software-centric," October 2016. [Online]. Available: https://www.daliwireless.com/evolution-wireless-networks-shift-hardware-centric-software-centric/. [Accessed 07 2018].

[19] TU - Automotive, "SDX: the software-defined car," 2016. [Online]. Available: http://analysis.tu-auto.com/telematics/sdx-software-defined-car. [Accessed 07 2018].

[20] "The Software Defined Car™ is Here: Are you ready for it?," July 2015. [Online]. Available: https://movimentogroup.com/blog/the-software-defined-car-is-here-are-you-ready-for-it/. [Accessed 24 10 2017].

[21] "ESSOR: European Secure SOftware defined Radio," [Online]. Available: http://www.occar.int/36. [Accessed 24 10 2017].

[22] "European Component Oriented Architecture (ECOA)," [Online]. Available: http://www.ecoa.technology/. [Accessed 24 10 2017].

[23] Wikipedia, "Generic Vehicle Architecture," [Online]. Available: https://en.wikipedia.org/wiki/Generic_Vehicle_Architecture. [Accessed 24 10 2017].

[24] "What is NGVA?," [Online]. Available: http://hamersham.com/what-is-ngva/. [Accessed 24 10 2017].

[25] "Land Vehicle with Open System (LAVOSAR I)," [Online]. Available: https://www.eda.europa.eu/what-we-do/activities/activities-search/land-vehicle-with-open-system-(lavosar-i). [Accessed 24 10 2017].

[26] "VICTORY - The Vehicular Integration for C4ISR/EW Interoperability," [Online]. Available: https://victory-standards.org/. [Accessed 24 10 2017].

[27] Wikipedia, "Software-defined networking," [Online]. Available: https://en.wikipedia.org/wiki/Software-defined_networking. [Accessed 24 10 2017].

[28] "Software-defined car takes shape," June 2017. [Online]. Available: http://www.eenewsautomotive.com/news/software-defined-car-takes-shape. [Accessed 24 10 2017].

[29] "Apple iPhone (Original/1st Gen/EDGE) 4, 8, 16 GB Specs," [Online]. Available: http://www.everymac.com/systems/apple/iphone/specs/apple-iphone-specs.html. [Accessed 24 10 2017].

[30] Wikipedia, "Software Defined Radio," [Online]. Available: https://en.wikipedia.org/wiki/Software-defined_radio. [Accessed 24 10 2017].

[31] WInnF Members, "What is Software Defined Radio?," Wireless Innovation Forum, [Online]. Available: http://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf. [Accessed 24 10 2017].

[32] M. Turner, "Software Defined Solutions - New Technology and JTRS Push the Envelope," in *SDR'02 Technical Conference and Product Exposition*, San Diego, 2002.

[33] M. Turner, "Software Defined Radio Solutions - Getting to JTRS compliant military SDRs and Beyond," in *SDR'04 Technical Conference and Product Exposition*, Scottsdale, 2004.

[34] M. Turner, "Software Defined Solutions - Experience making JTRS work, from the SCA, to Waveforms, to Secure Radios," in *SDR'05 Technical Conference and Product Exposition*, Anaheim CA, 2005.

[35] "Examples of ORBexpress in action," [Online]. Available: http://www.ois.com/Markets/military-and-aerospace.html. [Accessed 24 10 2017].

[36] "Meeting the Challenges of Ultra Large Large Scale Systems via Model Scale Systems via Model Driven Driven Engineering Engineering," 2007. [Online]. Available: https://resources.sei.cmu.edu/asset_files/Presentation/2007_017_001_23038.pdf. [Accessed 24 10 2017].

[37] S. Aslam-Mir, J. L. Paunicka and E. J. Martens, "CORBA in Control Systems," in *OMG Workshop On Distributed Object Computing For Real-Time and Embedded Systems*, Washington, DC, 2003.

[38] L. DiPalma and R. Kelly, "Applying CORBA in a Contemporary Embedded," in *OMG's 2nd Workshop in Real-Time and Embedded Distributed Object Computing*, 2001.

[39] J. F. Masiyowski, "CORBA in SIGINT Systems - A Case Study," in *OMG's 2nd Workshop in Real-Time and Embedded Distributed Object Computing*, 2001.

[40] S. Aslam-Mir, "Experiences with Experiences with real-time embedded CORBA in Telecom time embedded CORBA in Telecom," in *OMG Real-Time and Embedded Distributed Object* , 2000.

[41] 3GPP Support Office, "LTE 3GPP Technical Specification - TS 32.357 V2.0.0 (2010-03)," 2010.

[42] C. Lee, J. Kim, S. Hyeon and S. Choi, "FGPA Design to Support a CORBA Component," in *SDR 08 Technical Conference and Product Exposition.*, Washington, DC, 2008.

[43] C. M. Estes, "Distributed Control System for the National Ignition Facility," in *OMG Embedded Object-based Systems Workshop*, Santa Clara, CA, 2001.

[44] "OIS - High Performance Secure Communications Middleware," [Online]. Available: www.ois.com. [Accessed 24 10 2017].

[45] "GitHub jsoncpp," [Online]. Available: https://github.com/open-source-parsers/jsoncpp.

[46] Wikipedia, "JavaScript Object Notation," [Online]. Available: https://en.wikipedia.org/wiki/JSON.

[47] Wikipedia, "External Data Representation," [Online]. Available: https://en.wikipedia.org/wiki/External_Data_Representation.

[48] Wikipedia, "Common Data Representation," [Online]. Available: https://en.wikipedia.org/wiki/Common_Data_Representation.

[49] N. C. Zakas, "Is JSON better than XML?," 2008. [Online]. Available: https://www.nczonline.net/blog/2008/01/09/is-json-better-than-xml/.

[50] J. Wyse, "Why JSON Is Better Than XML," 2014. [Online]. Available: https://blog.cloud-elements.com/json-better-xml.

[51] P. Hintjens, "Chapter 7 - Advanced Architecture using ZeroMQ," [Online]. Available: http://zguide.zeromq.org/py:chapter7#toc0.

[52] V. Fotopoulos and C. Heaberlin, "Reliable UDP (RDP) Transport for CORBA," in *OMG Embedded and Real-Time 2002 Workshop*, 2002.

[53] F. Kuhns, D. C. Schmidt, C. O'Ryan, O. Othman and B. Trask, "Implementing Pluggable Protocols for TAO," [Online]. Available: http://www.dre.vanderbilt.edu/~schmidt/DOC_ROOT/TAO/docs/pluggable_protocols/. [Accessed 24 10 2017].

[54] C. R. A. Gonzalez, F. M. Portelinha and J. H. Reed, "Design and implementation of an SCA core framework for a DSP platform," *Military Embedded Systems,* 2007.

[55] G. Middioni, "CORBA over VMEbus Transport for Software Defined Radios".

[56] F. Casalino, G. Middioni and D. Paniscotti, "Experience Report on the Use of CORBA as the Sole Middleware Solution in SCA-Based SDR Environments," in *SDR'08 Technical Conference*, Washington, DC, 2008.

[57] B. Trask, "Using CORBA Asynchronous Messaging, Pluggable Protocols and the Real-Time Event Service in a Real-Time Embedded System," in *Object Managements Group's First Workshop on Real- Time and Embedded Distributed Object Computing*, Falls Church, VA, 2000.

[58] N. Scandella, "A Plug-in Transport with Dissimilar ORBs and a Connectionless Network," in *OMG Embedded Object-Based Workshop*, Santa Clara, CA, 2001.

[59] B. Balfour, "ORB Performance: Gross vs. Net," in *OMG Real-time & Embedded Distributed Object Computing Workshop*, Falls Church, VA, 2000.

[60] D. C. Schmidt, A. Gokhale, T. H. Harrison and G. Parulkar, "A High-performance Endsystem Architecture for Real-time CORBA," *IEEE Communications - Distributed Object Computing.*

[61] S. Bernier, H. Latour and J. Zamora, "How different messaging semantics can affect SCA applications performances: a benchmark comparison," *Analog Integrated Circuits and Signal Processing,* vol. 69, pp. 227-243, 2011.

[62] G. H. Thaker, P. J. Lardieri, D. Krecker, K. O’Hara, M. Price and C. Winters, "Systems Integration Systems Integration Achieving Bounded End-to-End Latencies Achieving Bounded End-to-End Latencies with Real-time Linux and Realtime CORBA with Real-time Linux and Realtime CORBA," Lockheed Martin - Systems Integration, [Online]. Available: http://www.atl.external.lmco.com/papers/1120.pdf. [Accessed 24 10 2017].

[63] Lockheed Martin - Advanced Technology Laboratories, "ORB Tests," [Online]. Available: http://www.atl.external.lmco.com/projects/QoS/orb/index.html. [Accessed 24 10 2017].

[64] "High Performance CORBA for Real-time and Embedded Applications - Optimized Collocation," [Online]. Available: http://www.ois.com/Products/orbexpress-rt-for-c.html. [Accessed 24 19 2017].

[65] D. Schmidt, N. Wang and S. Vinowski, "Object Interconnections Collocation Optimizations for CORBA," [Online]. Available: www.cs.wustl.edu/~schmidt/PDF/C++-report-col18.pdf. [Accessed 24 10 2017].

[66] "The omniORB version 4.0 User's Guide - Chapter 2 The Basics - Colocated Client and Implementation," [Online]. Available: http://omniorb.sourceforge.net/omni40/omniORB/omniORB002.html. [Accessed 24 10 2017].

[67] S. Bernier, "SCA Myths and Realities," in *SMi Annual Software Radio Conference*, London, UK, 2006.

[68] S. Bernier, C. Auger, J. Zamora, H. Latour and M. Michaud-Rancourt, "SCA Advanced Features – Optimizing Boot Time, Memory Usage, And Middleware Communications," in *the 2009 Software Defined Radio Technical Conference*, Washington, DC, 2009.

[69] J. Turley, "Survey says: software tools more important than chips," *Embedded Systems Programming,* 2005.

[70] Aspencore, "2017 Embedded Market Study," *EETimes Embedded,* 2017.

[71] U. N. C. K. R. Peterson, "Joint Tactical Networks," in *WInnComm 2015*, San Diego, CA, 2015.

[72] K. Dingman and A. DiBernardo, "Porting...It's more than just Software," in *WInnComm 2014*, Schaumburg, Illinois, 2014.

[73] D. C. Schmidt, "http://www1.cse.wustl.edu/~schmidt/reuse-lessons.html," [Online]. Available: http://www1.cse.wustl.edu/~schmidt/reuse-lessons.html. [Accessed 24 10 2017].