SCA 4.1: A PROMISING FUTURE FOR THE SDR ECOSYSTEM

Antoine Schindler (antoine.schindler[at]thalesgroup[dot]com); Adrien Duprez
(adrien.duprez[at]thalesgroup[dot]com); Eric Saliba (eric.saliba[at]thalesgroup[dot]com);
Eric Nicollet (eric.nicollet[at]thalesgroup[dot]com).
Thales Communications & Security, Gennevilliers, France

ABSTRACT

The paper discusses the benefits predicted to be fulfilled by the coming SCA 4.1 standard, exploring areas of deployment performance, portability, applications costs optimization, security and scalability. For each of those areas, the identified benefits are presented and discussed. Faster boot times, more portable SDR applications, more secure architectures, optimized development costs and more scalable solutions are foreseen, yielding to positive conclusions regarding the potential value of SCA 4.1 for the SDR ecosystem.

1. INTRODUCTION

As a result of the efforts done by the JTNC with active support of WInnF, the SCA 4.1 standard, released by JTNC in a Draft version [1] beginning of February 2015 seems mature enough to apply for becoming the reference version for new SDR products, as suggested by many of the testimonials previously presented during the SCA 4.1 Preview Workshop of October 2014 [2].

Almost 10 years after SCA 2.2.2 [3] was published (May 2006) SCA 4.1 takes advantage of years of international development experience [4] to offer a leaner standard SDR architecture for Core Framework, Platform Devices and Servicesand SCA applications (typically waveforms).

This paper evaluates the benefits of SCA 4.1 in terms of deployment performance (§ 2), waveform portability (§ 3), optimization of development costs (§ 4), security (§ 5) and scalability (§ 6) . Those benefits are evaluated on the basis of information gathered during the development phase and taking into consideration the just-released Draft [1] standard.

The content of this paper is based on a number of Thales SDR assets: earlier prototyping studies done towards optimization of SCA 2.2.2 for SWaP (Size, Weight and Power) constrained radios, general expertise regarding development of secure SDR solutions, long and strong involvement in support and development of a Standards-based SDR ecosystem for military radios, leadership position in definition of two WInnF standards [5][6] that brought reference inputs for SCA 4.1 elaboration, themselves strongly influenced by results of the ESSOR program [7][8][9].

2. DEPLOYMENT PERFORMANCE

For the execution of a SDR component, its connections with the other components of a SDR application and its configuration and management, SCA 4.1 provides the same capabilities as SCA 2.2.2, while saving boot time, reducing memory footprint and CPU usage.

For each step of a component deployment, this section discusses in more details the optimizations and improvements brought by SCA 4.1.

2.1. Component loading

The most important part of a SCA platform boot time and a application instantiation is spent during the file system access and more specifically during the binary and XML files loading. One way of dealing with this issue is to reduce the size of the loaded binaries.

As required by the SCA 2.2.2 specification [3], a component implementing the *Resource* interface has to handle all the operations of this interface (e.g. *PropertySet::configure*). To be fully compliant with this specification, the component is not only required to provide the interface, but also to implement the required behavior (e.g. raising a *CF::InvalidConfiguration* exception containing the list of the provided properties), increasing the size of the component's binary.

Given this observation, the SCA 4.1 offers the ability to reduce the number of interfaces implemented by a component if it is not required to provide the associated services. This mechanism, called “optional inheritance”, is applied to all the components defined in the specification.

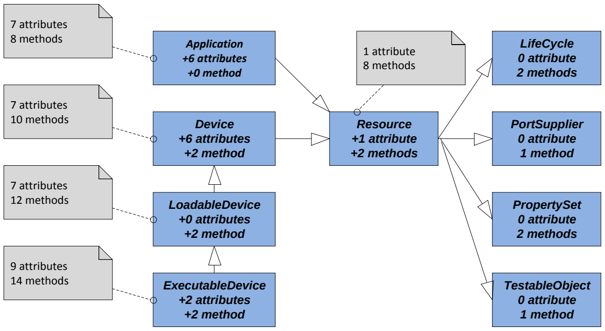

Figure 1: SCA 2.2.2 interfaces relationship

As depicted in Figure 1, an implementation of a component can require, as defined in the SCA 2.2.2 specification, to handle up to 9 attributes and implement 14 operations. However, depending of the implementation, all of these requirements are not necessary for the component to provide the services it is supposed to offer.

Using interface segregation (SCA 4.1 specifies 27 interfaces where SCA 2.2.2 specification has 18) and this "optional inheritance" mechanism, the SCA 4.1 provides a fairly fine granularity level to allow the component developer to have the right size for its implementation.

For example, a DeviceComponent (equivalent to a SCA 2.2.2 *Device* component) can implement only 1 operation instead of the 7 attributes and the 10 operations required by the SCA 2.2.2 specification.

This is an example among many others to use this SCA 4.1 useful feature that will help the SDR developers to noticeably reduce the size of the binaries and, as a result, the time needed to load them (in particular if they are retrieved from an encryptedfile system).

2.2. Component execution and registration

The essential improvement related to component execution and registration is that SCA 4.1 does not use the CORBA Naming Service.

As depicted in Figure 2, a SCA 2.2.2 *Resource* component deployed by an application factory requires the presence of a CORBA Naming Service. Indeed, the SCA 2.2.2 specification indicates that a waveform component shall bind itself in a Naming Context previously created by an application factory. This application factory can correlate the component's reference appearance in this context with the end of the component instantiation.

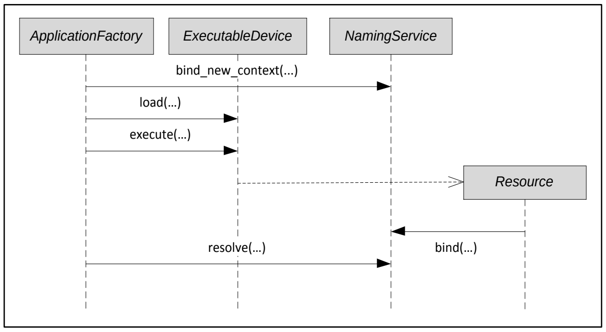

Figure 2: SCA 2.2.2 execution and registration

Following a streamlined approach, SCA 4.1 removes the use of the CORBA Naming Service as the way to exchange application component references, since introduction of the "push model" approach makes this registration mechanism unnecessary.

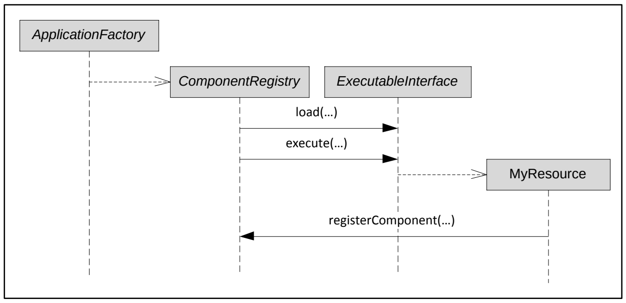

Figure 3: SCA 4.1 Resource execution and registration

The "push model" promotes the registration of a component with its manager by pushing its reference and the information carried by this component rather than pushing a reference that the manager will use later to pull the information it may need to manage this component.

This concept, illustrated in Figure 3, is applicable to all SDR components (Platform or Application), by passing a reference and all the necessary information to the ComponentRegistry interface, through which the component registers itself with its manager.

The time spent by the application factory to detect that the waveform component has bound in the Naming Context is saved, but it can be meaningless compared to the time saved by the removal of the Naming Service binary loading, execution and initialization.

2.3. Component deployment

Willing to avoid the unnecessary access to the file system, the SCA 4.1 "push model" also promotes the idea that a same XML profile should not need to be parsed twice. According to this idea, when a DeviceManagerComponent will parse the XML profile of its managed BasePlatformComponent, it will store all the information associated with this component (e.g. allocation properties) and provide it to the DomainManagerComponent when it will register itself into the domain using a *ComponentRegistry* interface. This optimization can save the substantial time spend doing unnecessary XML parsing during the DeviceManagerComponent registration into the domain.

After the time wasted during the unnecessary file system access, SCA 4.1 tried to reduce the number of interactions between components. Even if the amount of time needed to realize an interaction between two SCA components can be negligible, the multiplication of those interactions can become a substantial waste of time. Moreover, the CPU usage dedicated to those interactions can also be considered as a waste if they are not necessary.

There again, the "push model" approach allows reducing the number of interaction needed for a component deployment.

For example, the registration of a DeviceManagerComponent in the domain requires, in the best case, only one method call using the *ComponentRegistry::registerComponent* operation.

The same approach is used to create a connection between two components. Where the SCA 2.2.2 specification requires 3 operation calls using the *PortSupplier* interface, the SCA 4.1 only needs 2 to realize all the required connections between two components using their *PortAccessor* interface.

The time saved by the reduction of those interactions can individually seem meaningless but, considering that this time is proportional to the number of components and the number of connections between them, it can quickly become substantial.

3. PORTABILITY

3.1. PIM IDL Profiles

3.1.1. Restructured Appendix E

One of the most promising innovations of the SCA 4.1 with regards to better supporting waveform portability is the restructuration of Appendix E, brought in by addition of new Appendix E-1, "*Application Interface Definition Language Platform Independent Model Profiles*", and captured by the new Appendix E name: "*Model Driven Support Technologies*"

Appendix E-1 fully endorses the WInnF Standard "*IDL Profiles for Platform-Independent Modeling of SDR Applications*" [6]. This brings an essential value in support of better portability of SDR applications, as explained in the coming chapters of this section.

The rest of Appendix E is composed of PSM-related appendices, for CORBA or native C++ namely.

3.1.2. Support for PIM to PSM migration

As illustrated by Figure 4, and detailed in the introduction of the referenced standards, PSM application specific interfaces separate the component's Business Logic from its Used / Provided Ports, which encapsulate any mappings or transformation to occur between the application specific interfaces and the Operating Environment (in particular towards the available Transport Mechanisms).

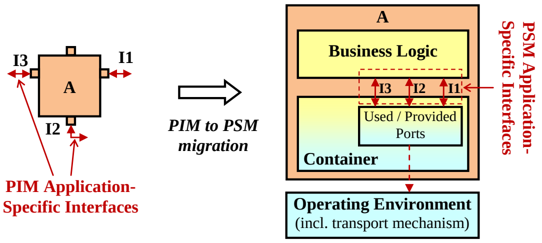

Figure 4: Notional PIM to PSM migration

Source : [1] Appendix E-1, p.8 & [6], p. 3

This approach is highly consistent with what the ESSOR program reported concerning its methodology for WF developments [9].

3.1.3. Definition of PIM Profiles

The defined PIM Profiles are specified using a thorough one-by-one specification approach parsing the underlying OMG standard [10], with detailed rationale provided.

The Full PIM IDL Profile is corresponding to the legacy SCA 2.2.2 component model, while the Lw and ULw Profiles are defined for resource-constrained environments such as DSP and FPGA OEs (see § 3.2 below).

3.1.4. Portability benefits

The previous descriptions corresponds to application of the Separation of Concerns design paradigm, that brings a clear separation between the Business Logic and the Container, which allows the Business Logic to be developed largely independently of platform assumptions, thus maximizing its portability. In particular it enables to select CORBA or alternate transport mechanisms, consistently with what the core specification enables for the Core Framework.

Realization of a PIM model with a consistent set of modeled components is one essential asset to ensure consistency of the design, enabling the SDR application porting activity to be essentially composed of direct porting of each of the SDR components, plus integration with the required ports, avoiding any modification of the functional behavior of the set of SDR components.

The choice of the programming language for the Business Logic is as well possibly done independently from the underlying PIM model, which is of decisive interest in PHY Layers where a given component may be needed, depending on porting assumptions, in FPGA or DSP language (see § 3.2 below).

3.2. Extension towards DSP and FPGA

3.2.1. Specification of GPP Component Models

Over its releases, SCA has provided a complete Component Model for development of GPP Components of a SDR application, with, essentially, prescriptions regarding:

- Reconfiguration Support: the *Resource* interface of SCA 2.2.2, with the equivalent set of optionally applicable elementary interfaces in SCA 4.1,
- Connectivity: Minimum CORBA mandated in SCA 2.2.2, a consistent PIM (with the Full IDL Profile) to PSM (multiple incl. CORBA) in Draft SCA 4.1,
- Operating System: the POSIX AEP until SCA 4.0, the Full POSIX AEP since then.

As discussed in the following section, SCA 4.1 provides consistent solutions to address the matters related to Execution Support for DSP and FPGA OEs.

3.2.2. Flexible and consistent connectivity approach

For Connectivity, the DSP and FPGA OEs can fully benefit from usage made by SCA 4.1 of the PIM/PSM paradigm, which foundations are discussed in § 3.1 above.

At the PIM level, SCA 4.1 uses the ULw PIM IDL Profile of WInnF IDL Profiles Standard [6], therefore bringing an optimal solution for PIM specification of Application Specific Interfaces for DSP and FPGA components. This resulted from an international convergence effort realized within the WInnF work group that developed the said WInnF Standard, increasing the readiness level of the initial contribution from ESSOR Architecture submitted for SCA 4.0 development (see [8]).

At PSM level, SCA 4.1 clearly allows for usage of an unlimited set of possible of Connectivity mechanisms, as indicated in p.7 of SCA 4.1 Appendix E-1, with standard Connectivity mechanisms (CORBA [10], MHAL Connectivity [11] and MOCB [12]) or proprietary solutions being possibly used.

3.2.3. Mature POSIX-based AEPs

For Operating System, which only represents a matter for DSP OE since FPGA do not use Operating Systems, SCA 4.1 essentially endorsed two Profiles among those allowed by the WInnF AEPs for Resource Constrained Processors [5], one Lightweight (Lw) Profile, and one Ultra-lightweight (ULw) Profile.

The WInnF specification brought important improvements to the previous achievements in successfully conducting an international convergence effort that delivered the underlying WInnF specification.

3.2.4. The remaining gap: Reconfiguration Support

Reconfiguration Support for DSPs and FPGAs remain untreated, and should be a point to consider for future standardization efforts.

3.2.5. Portability benefits

The additions reported above enable DSP and FPGA environments to be consistently covered by SCA 4.1, therefore significantly expanding the boundaries of what SCA can bring for coming radio products while application of previous SCA 2.2.2 proved to be de facto limited to GPP environments.

This opens perspectives for significant increase of the proportion of SDR applications being designed with high degree of portability, while enabling conformant SDR platforms to more easily host SDR components as well developed in their DSP and/or FPGA processors.

4. OPTIMIZED APPLICATIONS COSTS

4.1. Application backwards compatibility

Application backwards compatibility is an optional Unit of Functionality for the Core Framework allowing the Operating Environment to benefit from SCA 4.1 features (e.g. the “optional inheritance” mechanism), while remaining capable to manage SCA 2.2.2 applications. This therefore optimizes costs thanks to preservation of past investments in SCA applications.

This helps very significantly for adoption of SCA 4.1, since SDR platforms will possibly migrate towards SCA 4.1 without adding the costs, risks and schedule barrier to simultaneously handle migration of existing SCA 2.2.2 SDR applications towards SCA 4.1.

Since the application backwards compatibility Unit of Functionality imposes some overheads compared to the situation where the Core Framework would only comply with SCA 4.1, and since this prevents to benefit from a number of SCA 4.1 improvements (e.g. Naming Service removal), final convergence towards SCA 4.1-only architectures are seen as the end-term perspective.

4.2. Leaner SDR Applications development cycles

4.2.1. Earlier defects detection

Adoption of a generalized PIM/PSM paradigm for the SDR Applications, as enabled by Appendix E [1]; is known for being a powerful preventive method for improvement of the designs, thanks to the underlying separation of concerns between the logical design (business logic) and the physical design (implementation specific) [13]. It is a way to detect design defects at an early stage of the development process.

4.2.2. Easier introduction of code generation

The adoption of PIM to PSM paradigms provides ways, thanks to automated generation of the Container code, to increase software quality, reducing the integration risks and the maintenance costs [13].

4.2.3. Simplification of the test phases

The number of requirements introduced by SCA 4.1 has been reduced, and the phrasing of requirements has been modified, when needed, to capture an identical intent while being more prone to automated requirement verification tanks to usage of static code analysis tools.

Even for the requirements that demands static code inspection, SCA 4.1 is more suited for using automatic inspection tools (e.g. AEP conformance). This is another way to test earlier in the development process and thus reducing the cost of the defect analysis and correction.

5. SECURITY

SCA 4.1 brings security benefits with respect to equipment integrity as well as software assurance.

SCA 4.1 defines evolutions in components registration, by introducing the “push model” and removing the Naming Service. These SCA 4.1 evolutions enhance equipment integrity for the following reasons.

The “push model” states that a component registration to the domain manager requires a single transaction, as described on Figure 5. In former SCA specification, a component used the CORBA Naming Service to declare itself to the domain manager, which enabled the component to sniff and use any object reference in the same Naming Context. Furthermore, a component registration to the domain manager expected a multi-transactional scheme to learn all the component information. This simpler scheme should avoid erroneous behaviors.

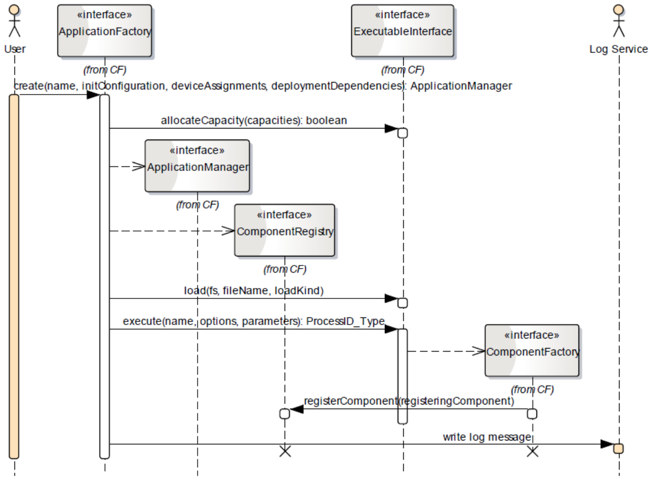
User
«interface»
ApplicationFactory
(from CF)
«interface»
ExecutableInterface
(from CF)
Log Service
create(name, initConfiguration, deviceAssignments, deploymentDependencies): ApplicationManager
allocateCapacity(capacities): boolean
«interface»
ApplicationManager
(from CF)
«interface»
ComponentRegistry
(from CF)
load(fs, fileName, loadKind)
execute(name, options, parameters): ProcessID_Type
«interface»
ComponentFactory
(from CF)
registerComponent(registeringComponent)
write log message

Figure 5: *ApplicationFactory* Application Creation Behavior

Source : [1], p.86

The “push model” also restricts the object access to information discovering. The manager does not publish equipment objects list anymore. The object is initialized only with a registration reference (an instance of the *ComponentRegistry* interface) provided by the DomainManagerComponent. Access to the full DomainManagerComponent services for the external components is no more available unless for the components which are required to.

The SCA 4.1 registration interface is a standalone service (see Figure 6), and therefore the objects cannot access to other system level interfaces.

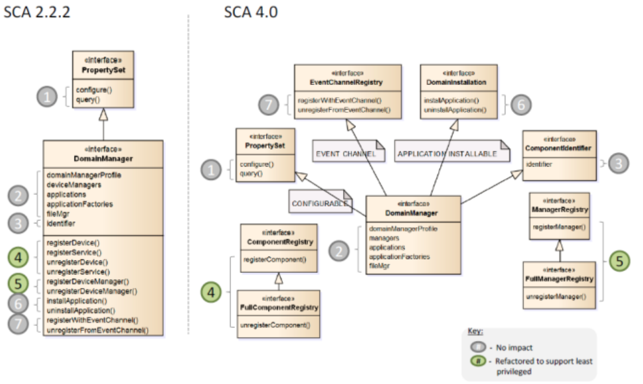

Figure 6:Domain Manager interface transformation

Source : [14], p.3

Further, the static port connection proposal with SCA 4.1 should enhance the platform stability and control. By statically defining the components connections, erroneous or illicit configuration actions leading to abnormal objects connection should be reduced, either wrong connections between legitimate components or connections with illegitimate components.

Software assurance should be more efficient through introduction of the “conditional inheritance” mechanism for platform and application components: by default, a component shall no more inherit from all the interfaces (see Figure 7). Further, SCA 4.1 allows an equipment to limit the complexity of its Core Framework: SCA conformance can be claimed on one of the three following Profiles: Lightweight, Medium or Full. These features induce an opportunity to gain assurance on the software development (fewer services, fewer interfaces, fewer tests) reducing any useless source code and increasing reliability.

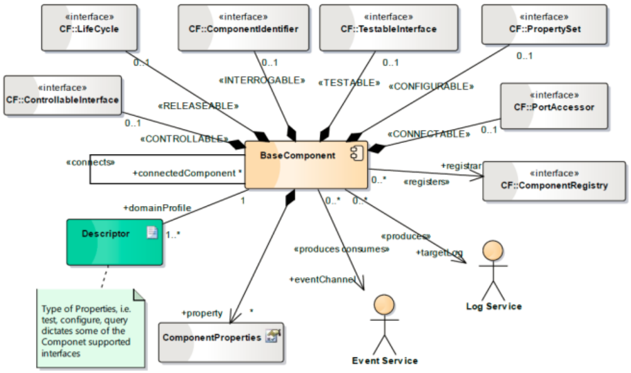
«interface» CF::LifeCycle
«interface» CF::ComponentIdentifier
«interface» CF::TestableInterface
«interface» CF::PropertySet
«interface» CF::ControllableInterface
«interface» CF::PortAccessor
«interface» CF::ComponentRegistry
BaseComponent
«INTERROGABLE»
«TESTABLE»
«CONFIGURABLE»
«RELEASEABLE»
«CONTROLLABLE»
«CONNECTABLE»
«connects»
+connectedComponent *
+registrar
«registers»
+domainProfile
Descriptor
+property
ComponentProperties
«produces»
«consumes»
+eventChannel
«produces»
+targetLog
Log Service
Event Service
Type of Properties, i.e. test, configure, query dictates some of the Componet supported interfaces
0..1
0..*
1
1..*

Figure 7: Base Component UML

Source : [1], p.34

These new SCA 4.1 features and improvements reduce implementation complexity and induce build-in SCA equipment security efficiency. This would be helpful for further security evaluation and accreditation.

6. SCALABILITY

SCA 2.2.2 was designed to be a one-size-fits-all solution for SDR products (SDR applications and platforms).

This could not take into account the specificity of platforms implementation constraints (SWAP requirements, type of processor, etc…), and did not bring enough flexibility to suit the needs of SDR applications.

This section discussed the scalability improvements brought by SCA 4.1.

6.1. Core Framework Profiles

Based on the new Profiles definition (Lightweight, medium and Full), the SCA 4.1 introduces a more scalable architecture. The customization process is applicable to the base components (Base Device Components and Base Application Components) but moreover to the Framework Control Components and the services provided by the Operating Environment. This means that the SCA 4.1 architecture is able to adapt itself to a wide range of more or less constrained underlying platform architectures.

6.2. Connectivity/CORBA-neutrality

One important barrier for adoption of SCA was the mandate for CORBA usage. The removal of this obligation, often dubbed as “CORBA-neutrality”, thanks to the PIM description of the interfaces between the various components of the SDR system; is a very significant progress to facilitate adoption of SCA technology on a broader variety of reconfigurable platforms, while preserving the investments and performance achieved by those having adopted a CORBA-based architecture.

6.3. Optional inheritance

The “optional inheritance” mechanism, as exposed beforehand, is providing means for SDR application components and SDR platform components to be tailored to their strict needs, avoiding a number of implementation overheads.

This will as well facilitate SCA 4.1 adoption since eliminating the previous need to get familiar with interfaces and associated concepts that are finally not used, wasting design efforts.

SCA 4.1 is therefore much more suited to the strict needs of the developed capabilities than SCA 2.2.2.

6.4. Nested Application

The support of “Nested Applications” enable to handle applications composed of other applications, bringing promising capabilities for elaborated configurations on complex systems.

This will enable for eliminate the combinatory explosion of systems with multiple options for different segments, where an overarching application description was required for each of the possible combinations.

This will as well facilitate integration of new capabilities within systems.

SCA 4.1 is therefore more suited to handling of complex systems than what SCA 2.2.2 was capable of.

7. CONCLUSIONS

This paper explored various areas where SCA 4.1 is expected to bring key value to SDR solution developers.

Faster boot times will be experienced, thanks to simplification of components registration mechanisms; more portable SDR Applications will be developed, thanks to support of PIM/PSM design paradigms coupled to high flexibility in choice of implementation options; more secure architectures will be available, thanks to suppression of most vulnerable parts of previous architecture and avoidance of non-required interfaces; optimized development costs will be achieved thanks to leaner overall architecture and simplification and automation of the testing phases; last, more scalable solutions will be available, enabling to better adapt the designs to expectations and platform constraints.

This implies that SCA 4.1 has the potential to bring decisive value to bring forward the SDR ecosystem, making it more efficient for existing areas where SCA is used, and making SCA 4.1 much more attractive for new adopters.

Reports from implementation by stakeholders will enable, in near future, to better evaluate things, moving the knowledge basis on SCA 4.1 from previsions, such as those reported in this report, to implementation results.

REFERENCES

[1] JTNC, “Software Communications Architecture Specification” version 4.1<DRAFT>, http://jtnc.mil/sca/Pages/sca1.aspx, 31 December 2014.

[2] WInnF, JTNC, “SCA 4.1 Preview Workshop”, Aberdeen, Maryland, 9-10 October 2014.

[3] JPEO JTRS, “Software Communications Architecture Specification, Version 2.2.2”, http://jtnc.mil/sca/Pages/sca1.aspx, 15 May 2006.

[4] WInnF, “SCA Standards for Defense Communications: Global Adoption, Proven Performance”, http://groups.winnforum.org/SCA_Committee.

[5] WInnF, “Lw and ULw POSIX AEPs for Resource Constrained Processors”, Version V1.0.0, WINNF-14-S-0009, http://groups.winnforum.org/Specifications, 25 July 2014.

[6] WInnF, “IDL Profiles for Platform-Independent Modeling of SDR Applications”, Version V1.0.1, WINNF-14-S-0016, http://groups.winnforum.org/Specifications, 25 August 2014.

[7] WInnF, “ESSOR/SCA Next LwAEP harmonization”, Version V1.0.0, WINNF-11-I-0007-V1.0.0, http://groups.winnforum.org/d/do/4690, 24 May 2011.

[8] WInnF, “ESSOR/SCA Next CORBA Profiles harmonization”, Version V1.0.0, WINNF-11-I-0008-V1.0.0, http://groups.winnforum.org/d/do/4691, 24 May 2011.

[9] ESSOR HDR Base WF – Methodology and Results for Developing a Portable Coalition Waveform Software – WinnComm’14, 10-13 March 2014.

[10] OMG, “Common Object Request Broker Architecture (CORBA) Specification Part 1: CORBA Interfaces”, Version 3.2, formal/2011-11-01, http://www.omg.org/spec/CORBA/3.2/Interfaces/PDF, November 2011.

[11] JTNC Standards, Public Application Program Interfaces (APIs), “Modem Hardware Abstraction Layer”, Version 3.0, http://jtnc.mil/sca/Pages/api1.aspx, 02 October 2013.

[12] JTNC Standards, Public Application Program Interfaces (APIs), “MHAL on Chip Bus”, Version 1.1.5, http://jtnc.mil/sca/Pages/api1.aspx, 26 June 2013.

[13] OMG, “Model Driven Architecture (MDA) Guide”, Version 2.0, ormsc/2014-06-01, http://www.omg.org/mda/presentations.htm, June 2014.

[14] Chalena M. Jimenez, Kevin W. Richardson, and Donald R. Stephens, JTRS, “SCA4 – An Evolved Framework”, http://jtnc.mil/sca/Pages/references1.aspx, 24 July 2012.