USING GENERIC COMPONENT ENVIRONMENTS IN JTRS RADIOS

Murat Bicer (Mercury Computer Systems Inc., Chelmsford, MA, mbicer@mc.com)
Jim Kulp (Mercury Computer Systems Inc., Chelmsford, MA, jkulp@mc.com)
Frank Pilhofer (Mercury Computer Systems Inc., Chelmsford, MA, fpilhofe@mc.com)

ABSTRACT

In the foreword of the Software Communications Architecture Specification, JTRS JPO states that *"the SCA has been structured to build on evolving commercial frameworks and architectures."*[1]. Building a specification on commercial standards enables the vendors to implement the required functionality using COTS products. Extended use of COTS in a system significantly reduces the acquisition, operation and supportability costs for the government. Following this principle, the SCA deliberately references commercial frameworks and standards such as minimumCORBA[2], Interoperable Naming Service[3], Event Service[4], Lightweight Log Service[5], POSIX[6], UML[7] and XML[8]. As the SCA specification evolves, more commercial standards can be adopted by the SCA. This paper describes commercial standards that can be candidates for adoption and investigates how existing implementations would be affected by them.

1. RELATED OMG STANDARDS

The Object Management Group (OMG) is an open membership, not-for-profit consortium that produces and maintains computer industry specifications. The OMG Software Based Communications Domain Task Force (SBC DTF) is currently working on the "PIM and PSM for SWRadio" specification in an effort to commercialize some additional aspects of the SCA. The OMG minimumCORBA, Lightweight Logging Service and Interoperable Naming Service specifications are already referenced by the SCA.

There is significant similarity between the SCA and the OMG CORBA Component Model (CCM). CCM was designed as a broadly applicable generic component middleware standard. Both the SCA and CCM are based on the idea of composing applications of interconnected components. This is not surprising, as the SCA specification was born out of the CCM specification before CCM was adopted and finalized by the OMG. The SCA has additional features and functionality to use resource management to decide which devices are capable of hosting a component implementation. The CCM has additional "enterprise computing" features, and a richer and more detailed model of component implementations. Unfortunately, due to some timing issues and the fact that CCM was originally completely inappropriate for embedded systems, the SCA could not be based on CCM, but only on CORBA.

The OMG "Deployment and Configuration of Component-Based Applications" specification (D+C) was created to enhance the deployment model of CCM, learn from the SCA, and provide a new commercial standard for component deployment. It embraces some of the SCA's deployment concepts, and introduces an updated deployment model into the CCM domain.

D+C defines 4 *managers* to control and manage the domain. These are the Repository Manager, the Target Manager, the Execution Manager and the Node Manager.

The Repository Manager is a service that allows installation, retrieval and deletion of component software packages. A D+C component package is analogous to an SCA Application. It consists of a set of component implementation artifacts and XML meta-data that represent a deployable application. The Repository Manager provides central access to package meta-data and artifacts. When a package is installed, the repository manager parses the meta-data for this package from the XML files and puts the information in IDL-defined data structures for retrieval. The Repository Manager essentially implements the install and uninstall functionality of the SCA Domain Manager.

The Target Manager is a run-time service that centrally maintains a domain's platform meta-data. It provides operations for retrieving the meta-data describing domain resources reflecting either their total or remaining capacity. Additional operations support the commitment and release of these resources as well as run-time updates such has hot-swapping.

D+C applications are executed by the Execution Manager based on a deployment plan. The Execution Manager is a singleton run-time service for the execution of deployment plans within a domain. It delegates operations to the Node Managers on each node. In order to do this, the Execution Manager decomposes the deployment plan into a set of partial component assemblies of component instances that execute entirely within a node. The Execution Manager essentially implements the plumbing part of the create functionality of the SCA Application Factory.

The Node Managers support the execution of a localized deployment plan. While the Execution Manager is a generic piece of the infrastructure, the Node Managers are specific to their node types.

A Node Manager service is not necessarily collocated with the managed node. It could run on a different node as long as it has the capability to launch component implementations on the managed node. This allows specialized hardware nodes such as FPGAs and DSPs to be a part of the domain even if they do not host an operating system or run concurrent processes.

Like the SCA, D+C allows implementations to define their requirements and targeted processor typesto define their capabilities, and defines a matching algorithm between the two. D+C has new concepts like hierarchical assemblies and an algorithm for an assembly to select between alternative component implementations. D+C also attempts to work in a platform independent manner, by delegating specifics about resources and capabilities to a platform-specific model that is separate from the D+C model. The resources and capabilities are essentially a contract between the component developer and the runtime environment(s) on the nodes. D+C is also designed for scalability, to support distributed systems with many computers.

An SCA personality module built on the D+C model can support compliance with the current SCA specification.

With this layered enhancement, a D+C enabled CCM implementation that is compliant with the Lightweight CCM profile becomes an attractive foundation for an SCA Core Framework, greatly enlarging the specification's COTS content beyond CORBA and POSIX, and leveraging adopted OMG standards to work in favor of the SCA and waveform applications.

2. CURRENT STATUS

With the SCA specification's current standing, it is possible to use D+C internal to a Core Framework implementation. In such an implementation, SCA interfaces and the domain profile would be unchanged. The Core Framework would internally translate SCA Software Packages to D+C packages, and then delegate installation and deployment to the D+C infrastructure. Using D+C in a CF implementation would increase the COTS content of the end-product significantly and focus the SDR aspects of the SCA model as a personality of a COTS component deployment system.

Such an implementation of the SCA 2.2 Core Framework would make use of the Deployment and Configuration facilities to deploy and configure waveform applications, while not taking advantage of advanced features like hierarchical assemblies. Such an implementation also wouldn't use the CORBA Component Model, as the change of the CF::Resource interface to a CORBA component would imply backwards-incompatibility (e.g. by using component primitives to navigate ports rather than the PortSupplier interface).

A D+C-based SCA would expose the same CF interfaces, and use the same XML descriptor files, as defined by the SCA 2.2 specification. Usage of the D+C functionality would be completely encapsulated inside the Core Framework. The Core Framework would then delegate many operations to the COTS D+C implementation and thus be a smaller software module than a typical standalone Core Framework.

The subsections below enumerate the potential changes in the components of a CF constructed this way.

2.1 CF::DomainManager

Application installation and device management would be impacted by the presence of a D+C platform. The installApplication operation would delegate to the Repository Manager, and the registration of devices would cause the Domain Manager to send domain updates to the D+C Target Manager.

When a CF client requests the installation of an application, the Domain Manager would create a package in the D+C package repository. To accomplish this, the Domain Manager has to read the package's descriptors, and to translate the SCA Software Assembly Descriptor into a D+C Component Package Description. The details of this transformation can be seen in [9]. The Domain Manager then creates the package in the D+C repository and instantiates an Application Factory object encapsulating the resulting package configuration. Note that, while the SCA descriptors must be parsed from XML, the D+C repository has an IDL-based binary interface so no D+C XML is ever produced.

When a Device Manager registers a new device with the Domain Manager, the Domain Manager would send a domainUpdate to the D+C Target Manager reflecting the change in configuration. For Executable Devices, the Domain Manager also instantiates a wrapper that acts as a Node Manager in order to take launch requests from a D+C Execution Manager, translating these requests to operations on the underlying Executable Device and Resource interfaces.

2.2 CF::ApplicationFactory

The Application Factory is a CF component that represents an installed application and that supports creating and configuring an instance of that application. Clients of the Application Factory can provide a subset of device assignments.

This represents the planning and launching phases according to the D+C specification. In a D+C based CF, the Application Factory would include a planner, or re-use and extend a planner provided by the platform. A tradeoff could

be made regarding advance planning (i.e. at application installation time) or late planning (at application instantiation time).

The Application Factory then passes the resulting DeploymentPlan to the Execution Manager, launching the application (using ExecutionManager::prepare, ApplicationManager::startLaunch and Application::finishLaunch).

Because D+C is concerned with the deployment and initial configuration of an application only, its resulting Application object does not support run-time configuration or port navigation. The Application Factory provides an object that implements the CF::Application interface, delegating configuration to the assembly controller and port navigation to the respective resource.

2.3 CF::ExecutableDevice

The D+C Execution Manager uses the Node Manager interface to launch components on nodes. As mentioned above, the Core Framework needs to provide a Node Manager for each Executable Device during startup. This wrapper can be generic, i.e. it is not device specific. The wrapper delegates preparePlan requests to appropriate load and execute primitives on Executable Devices. Processing nodes that are already part of the D+C platform need no such wrapper. Only SCA-enabled executable devices dynamically registered with the domain manager need such wrappers. Note that the wrappers can be objects local to the domain manager.

3. POSSIBLE EVOLUTION PATHS FOR SCA

Currently, there is an overlap between the SCA specification and the D+C (with Lightweight CCM) specifications. The previous chapter explained how this relationship can be exploited when implementing a CF built on a COTS D+C system.

Moving forward, the JTRS JPO can choose between two options to determine the path for the evolution of the SCA specification. Figure 1 shows the relationship between the SCA and the OMG specifications, as well as the options for SCA evolution.

The first option would be to continue making minor modifications to the SCA and the relevant OMG specifications through change proposals. This would bring these specifications closer, thus increasing the COTS content of the SCA CF implementations. The burden of maintenance would stay the same as this approach does not necessarily reduce the size of the SCA specification.

More beneficial results could be achieved by the second option: explicitly referencing the aforementioned OMG specifications in the SCA (like the current SCA references the LightWeight Logging service). This would recognize that these additional aspects of the SCA do not need to be specialized for SDR, just like CORBA does not need to be specialized for SDR.

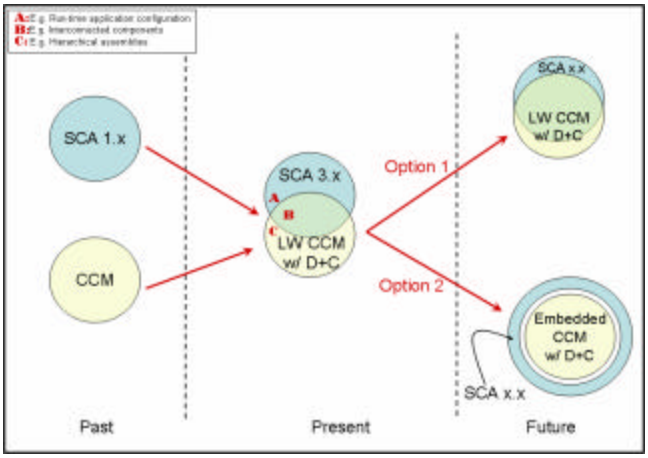

Figure 1 – SCA Evolution

Even though such an adoption would have some impact on the specification, large parts of the SCA will remain unaffected. This includes most of the Base Application Interfaces, the Framework Control Interfaces, the Framework Services Interfaces, and usage of the logging and event services. Also, the Application Environment Profile remains unchanged.

The adoption of D+C/LWCCM will affect the metadata used by waveform applications and devices. The Software Package, Software Component and Software Assembly descriptors will be transformed into the Component Package and Component Implementation descriptors in binary form from the D+C specification. This allows waveform applications to benefit from D+C features like hierarchical assemblies. The SCA can extend the Target Manager to accept domain updates from the Domain Manager during device registration.

The adoption of LW CCM will require a slight modification to the CF::Resource interface. CORBA Components already include some of the functionality provided by the CF::Resource interface, such as ports, navigation and configuration. In a CCM-based SCA, Resources would be implemented as CCM *components*. This could be achieved by changing the definition of CF::Resource into a component, such as:

```
interface Startable {
    void start ();
    void stop ();
};
```

```
component Resource
  supports Startable, LifeCycle,
TestableObject {
    string identifier;
};
```

The Resource Factory interface would be replaced by CCM entry points.

The CF would then use D+C's primitives to configure and interconnect components. At startup, CF would still provide an Application object that would provide port navigation and component configuration at runtime.

An Application Factory interface would still be provided as a part of the CF to handle dynamic deployment planning. The Application Factory will act as a planner and prepare D+C Deployment Plans at runtime that will then be fed to the Execution Manager. Support for some additional incremental and dynamic planning patterns could be implemented in this SCA/SDR Application Factory, beyond those typically used with D+C.

3.1 Migration of Existing Waveforms

If the SCA specification is modified as described above, the existing waveforms would need to be migrated to this new platform.

An SCA waveform (i.e. Application) consists of two parts: component implementations and metadata. The metadata contains information to configure and interconnect the application as well as component requirements that are matched against device capabilities during deployment.

The transformation of metadata, from SCA Software Assembly Descriptors to D+C Component Package Descriptors, is explained in detail in [9].

For the migration of component implementations, there are three options:

The first option is modifying the component implementations to implement the "new" CCM-ized Resource component interface, and rebuilding the components. This approach has the disadvantage of changing the source code. This would be impossible if source code of the waveform is not available.

The second option is using a legacy wrapper that provides the CCM-ized Resource component interface on the outside, and delegates to the implementation via the old-style Resource interface on the inside.

A legacy wrapper could be easily generated from the component metadata, as it would need knowledge of the component ports and properties. From that, appropriate wrapper IDL and implementation code could be generated automatically. A Resource's configurable properties would be exposed as the component's attributes, and local CCM primitives for port interconnection (i.e. the connect operation in the Receptacle interface) would be delegated to the Resource's getPort and the Port's connectPort operation. Note that there is no overhead involved in the actual connection. The overhead for configuration and other operations on the component interface (e.g. the TestableObject::runTest operation) would be a single local indirection, as the legacy wrapper would just forward the invocation to the respective operation on the Resource interface. As for footprint, the legacy wrapper does not contain any complex operations but is only a straightforward component implementation. An automated tool could thus provide the full migration, first transforming the metadata as detailed in [9], and then adding a legacy wrapper to each implementation.

Finally, the third option is running legacy component implementations in a SCA-legacy-enabled Node Manager. This approach is the least invasive and allows the execution of SCA 2.2 components without change. Its disadvantage is the need to write a specialized Node Manager. This also reduces the COTS content of the system.

3.2 Impact on Existing SCA Devices

3.2.1 Devices Components are "Deployed On"

D+C uses the term *"resource"* to describe the available features and assets in a target environment. During the creation of the deployment plan, the implementation requirements of the components and connections are matched against the resources of the nodes and interconnects to identify suitable nodes and interconnects for deployment. This concept is similar to SCA's capacity concept for *allocation* type properties, where the Application Factory determines the candidate devices.

Executable Devices can be modeled as Node Managers in the new D+C compliant environment. D+C allows centralized management of node capacities through the Target Manager, so that a Node Manager need not be concerned with capacity allocations.

Existing Executable Devices can be migrated to this new environment by using wrappers. A generic wrapper could easily act as a Node Manager on the outside, and delegate calls to a legacy loadable or executable device. Such a Node Manager could only be used to load and/or execute legacy SCA components. In order to support new CCM compliant SCA Resources, additional Node Manager behavior would need to be implemented.

3.2.2 Devices "Used By" Components

D+C node managers are infrastructure elements that are not application software components. In other words, D+C nodes are potential targets only for execution, comparable to

SCA Executable Devices. Other types of devices that are *used by* waveform components (such as an Audio Device) can be modeled as node *resources* in D+C. An SCA Resource that connects to a Device through a *usesdevice* relationship can be modeled in the D+C environment by defining an *ImplementationRequirement* on the component implementation. The *ImplementationRequirement* would have a *ResourceUsageKind* of *InstanceUsesResource*, which specifies that the relationship is a "uses" relationship between a component instance and a D+C resource. This D+C feature allows such a resource to be represented by any type of "handle". When mapping the SCA *uses device* relationship, this handle is a CORBA object reference. Just like the SCA, the software component uses the required device by using the CORBA object reference.

In other words, D+C unifies the SCA capacity allocation and *used by devices* concepts under the *Implementation Requirement* concept.

4. CONCLUSION

An SCA CF built on a COTS D+C capable Lightweight CCM platform would be fully-compliant with the SCA specification while increasing the "COTS" content of the implementation. By increasing COTS content, we mean using commercially available implementations of generic component-based software standards to implement the functionality required by the SCA. Such an approach would reduce the operation and supportability costs. The existing SCA compliant waveforms would run on this platform without any change.

In the future versions of the SCA specification, D+C and Lightweight CCM can be explicitly referenced, to take full advantage of these OMG specifications' advanced features such as hierarchical assemblies of components. The SCA already references other commercial standards. This approach would take this adoption one step further and leverage more available commercial technologies that are suitable for, but not specific to, SDR. As the size of the SCA specification decreases by utilizing commercially available standards, the cost of maintaining and supporting the specification drops significantly.

5. REFERENCES

[1] Modular Software-programmable Radio Consortium, "Software Communications Architecture Specification", *MSRC-5000SCA*, V2.2, November 17, 2001.

[2] Object Management Group, "minimumCORBA", *OMG document* orbos/98-05-13, May 19, 1998.

[3] Object Management Group, "Interoperable Naming Service Specification", *OMG document* formal/00-11-01.

[4] Object Management Group, "Event Service Specification", *OMG document* formal/01-03-01: Event Service, v1.1.

[5] Object Management Group, "Lightweight Log Service Specification", *OMG document* formal/03-11-03: v1.0.

[6] International Organization for Standardization, "POSIX.1. Application Program Interface" *ISO/IEC* 9945:1996.

[7] Object Management Group, "Unified Modeling Language Specification", Version 1.3, March 2000.

[8] World Wide Web Consortium, "Extensible Markup Language 1.0.", *W3C Recommendation*, Feb 1998.

[9] Mercury Computer Systems, Inc., "SCA to D+C XML Transformation", *White Paper*, March 2005.