An Introduction to a UML Platform Independent Model of a Software Radio

Michel Barbeau and Francis Bordeleau
School of Computer Science
Carleton University
1125 Colonel By Drive,
Ottawa (Ontario), Canada K1S 5B6

Jeff Smith
Mercury Computer Systems Inc.
199 Riverneck Road
Chelmsford, MA US 01824

Abstract

This paper is a tutorial about a Platform Independent Model (PIM) of a Software Radio (SWR), which is being developed by the Object Management Group (OMG) SWR Domain Specific Interest Group (DSIG). The tutorial gives a short introduction to the concept of SWR and Model Driven Architecture (MDA). The tutorial then focuses on the two most important packages of the PIM, namely, the CF Control and Base Application packages.

1. Introduction

The objectives of this tutorial are to introduce the concept of Software Radio (SWR), to give an overview of the Unified Modeling Language (UML) [OMG 01] model of a SWR that is being standardized by the OMG [OMG 02a] and to review some of the related issues. It is important to stress that this is in-progress work and the tutorial provides an early exposure to it.

The concept of SWR has been defined by Mitola [Mito 00]. It is a radio consisting of a programmable hardware platform, which functions can be implemented in software. Standardization efforts are invested in this area for the sake of enabling the integration and interoperability of SWR components from different sources. Standardization efforts are being conducted by the Joint Tactical Radio Systems (JTRS) Joint Program Office (JPO) [Join 01], Software Defined Radio (SDR) Forum [SDR 02] and Object Management Group (OMG) Software Radio (SWR) Domain Special Interest Group (DSIG) [OMG 02a]. The JTRS JPO has standardized a model of a SWR called the Software Communications Architecture (SCA). UML diagrams are used solely for the specification of interfaces to the elements of a SWR. The SDR Forum conducts activities related to the SCA. It sponsors the development of a reference implementation of the SCA at the Communications Research Centre (CRC) [CRC 02].

The OMG SWR DSIG develops using UML a standard model of a SWR. The OMG SWR DSIG tries to maintain compatibility with the SCA (in fact their model is derived from the SCA), but in contrast to the SCA, the UML is used to model all aspects of a SWR (structural, behavioural, in addition to interfaces) and the Model Driven Architecture (MDA) approach is used [OMG 02b]. The MDA approach makes a separation between a Platform Independent Model (PIM) and a Platform Specific Model (PSM). A PIM makes abstraction of implementation platforms. For instance, in the SWR case the concept of distributed object is used. At the level of the PIM, this concept remains abstract and does not refer to one of its particular realization, such as CORBA and EJB. In contrast, a PSM would have to resolve this issue.

The use of the UML and MDA are well justified in this context. The architecture of a SWR is a complex problem to address. UML and MDA are tools for addressing complexity of software. The conceptual tools they provide for handling complexity are abstraction, encapsulation and hierarchical structure and behaviour. The UML and MDA also benefit from a wide acceptance. The UML and MDA artefacts are readable by a wide audience and supported by a variety of tools.

In this tutorial, we introduce the concept of SWR (Section 2) and UML and the MDA approach (Section 3). We make a presentation of the UML model of a SWR as being defined by the OMG (Section 4). Section 5 concludes the tutorial.

2. Software Radio

Digital Signal Processing (DSP) is a technology enabling the concept of SWR. In a nutshell, DSP consists of measuring analog signals, representing the measurements numerically (i.e. Analog to Digital Conversion (ADC)), doing some processing with the numbers and converting the results of this processing back to an analog signal (i.e. Digital to Analog Conversion (DAC)). The processing corresponds to a software representation of transformations such modulation, filtering and coding.

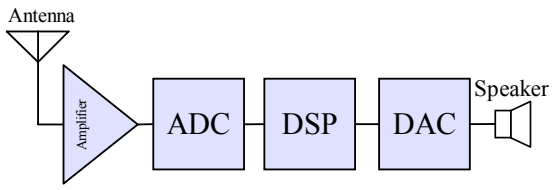

Figure 1. Architecture of a SWR receiver.

At the heart of a SWR is a processor that does DSP. For the sake of flexibility, radio-

frequency components of the hardware architecture of a SWR are made as general as possible, see Figure 1. Antennas are either wideband or multi-band. On the receiver side, the amplifier and ADC are also made wideband.

3. UML and the MDA Approach

The Model Driven Architecture (MDA) is a three-tier approach: the Platform Independent Model (PIM), Platform Specific Model (PSM) and Enterprise Deployment Model (EDM), see Figure 2. A PIM is expressed using UML and describes how an application or a system is structured, while making abstraction of implementation details. For instance, the PIM of a file transfer application would define a get operation in a generic way as an action that transfers the blocks of a file from a source file system to a target file system. The PSM, also expressed with UML, adds constraints and implementations details. The PSM of the file transfer application would specify that FTP over TCP/IP is used to implement the operations. The EDM is the final product and corresponds to code written in a specific programming language for a specific OS. The EDM for the file transfer application could be source code in C for Linux. There is a one-to-many relationship from a PIM to PSMs and another one-to-many relationships from a PSM to EDMs.

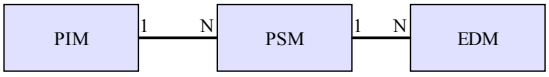

Figure 2. The MDA approach.

The MDA approach gives rise to model driven development, see Figure 3. The PIM and PSM are developed according to the rules defined within a metamodel, a model of models. The metamodel itself can be described with UML. From the PIM to the PSM there is mapping relationship which adds infrastructure (e.g. a concrete distributed object framework such as CORBA). From the PSM to the PIM there is a refactoring relationship that abstracts infrastructure.

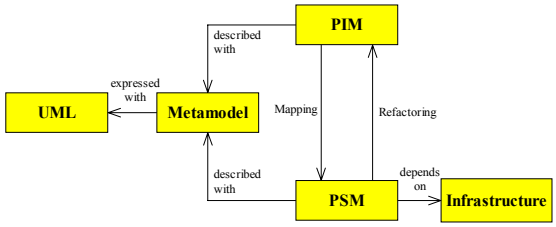

Figure 3. Model driven development.

4. UML Model of the SWR

The OMG SWR DSIG community uses the following terminology: waveform, application and domain. A SWR can be loaded with waveforms. A *waveform* consists of transformations. In a SWR transmitter, transformations encode data (e.g. voice) and produce a signal that can be transmitted over the air. In a SWR receiver, transformations decode a signal received over the air and produce data (e.g. packets). In other words, a waveform is a software representation of a radio communications standard or mode.

An *application* is a program which execution realizes a given waveform. An application provides an interface through which it can be configured, controlled and monitored.

A *domain* is a set of hardware devices and applications. A given domain is under the control of a domain manager.

Packages

The top level of the OMG UML model of a SWR is organized into packages, see Figure 4. A UML *package* is a folder metaphor. It is used to group together a collection of connected elements of a model, such as classes, associations and even other sub packages. The graphical representation of a package evokes a folder and is a rectangle with a small tab on the top left corner. The name of the package is written inside the rectangle.

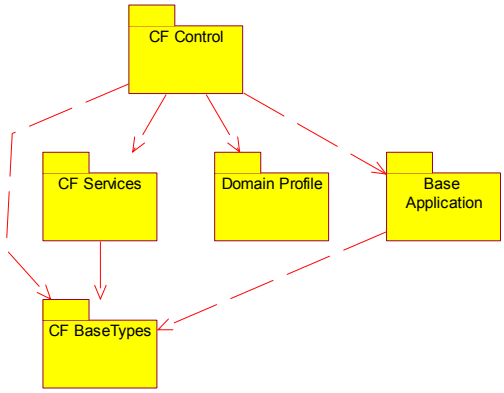

Figure 4. Package view of a SWR.

A package can be connected to other packages. Such a connection is represented by a dashed arrow with an open arrowhead from a referee package to a referenced package. Elements defined in the referenced package become visible in the referee package. The representation of a collection of connected packages is called a *package view*.

The UML PIM of a SWR consists of five packages, namely, Core Framework (CF) Base Types, CF Services, Domain Profile, Base Application and CF Control. The CF Base Types is a utility package that defines data types used by other elements of the UML PIM of a SWR. Data types are defined as classes. For example the Property type used in the Base Application package is defined here. The CF Service package comprises

four support services, namely, the event service, file service, logging service and naming service. The Domain Profile package defines XML files that describe a domain. An important type of file is the one describing how an application should be assembled. Not all packages are currently at the same stage of progress within the OMG SWR DSIG. Considerable effort has been invested so far for the Base Application and CF Control packages. These two packages are reviewed in more detail in the sequel.

4.1 The Base Application Package

The Base Application package defines the elements that are at the core of applications. These elements are defined as interfaces and classes, see Figure 6. Every class is pictured as a rectangle with three compartments. The top compartment contains the name of the class. The middle compartment contains a list of typed data members. The bottom compartment contains signatures of operations.

An interface defines signatures of operations that are realized by classes. The interface itself does not address implementation and has no data members and internal structure. An interface is represented by an <<*interface*>> class stereotype. An interface can be connected to other interfaces through generalization relationships. This kind of relationships is represented by a solid arrow with a closed arrowhead directed towards the parent interface. A realization relationship is represented as a solid arrow with a closed arrowhead from a class to a supported interface.

Central to the Base Application package are the Resource and Port abstractions. A *Resource* is the abstraction of a software element of a SWR application. The Resource class realizes a number of interfaces, namely, the ResourceInterface, LifeCycle, PropertySet, PortSupplier and TestableObject interfaces. Together they define an Application Programming Interface (API) which is used by other elements of the SWR to control a resource.

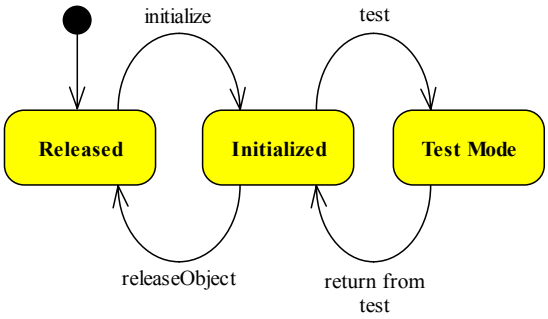

Figure 5. Statechart diagram for Resource instance.

In UML, statechart diagrams are used to model the behaviour of instances of classes. A statechart diagram consists of states and transitions. A state is represented by a rectangle with rounded corners. The initial state is illustrated by a filled circle with a single outgoing transition. A transition is pictured by an arc from a source state to a target state. The arc has an associated action label. A statechart diagram of Resource instance is pictured in Figure 5.

The LifeCycle interface declares operations which invocations initialise or release internal elements of a resource (e.g. set values of data members or configure parameters). The TestableObject interface declares an operation with which can be launched test routines built-in a resource. It is a black box form of testing. A resource may have several properties. A property is an identity-value pair. The value of an individual property can be configured or inspected using operations declared on the PropertySet interface.

A Resource has zero-to-many ports, which can be retrieved individually by name using the operations declared on the PortSupplier interface. Ports can be connected to each other using operations declared on the Port class. Each connection is half-duplex and a port can be connected to several other ports.

The ResourceInterface amalgamates all the aforementioned interfaces and declares additional operations in order to start or stop the execution of a resource and to extract its identity.

A ResourceFactory is an optional path that can be taken to create resources by identity and specified properties or to release resources by identity. It is inspired from the Factory design pattern. In addition, a Resource instance may be shared. For each resource that it produces, the ResourceFactory maintains a count of references. The resource is effectively released only when the reference count reaches the value zero. A shutdown operation releases the ResourceFactory itself.

4.2 The CF Control Package

The CF Control package addresses the control and management of resources, ports (defined in the Base Application package) and devices within a domain.

Application Class

An instance of the *Application* class is the abstraction of an application, which is a member of a domain (see Figure 7). The Application class declares operations for the control, configuration and monitoring of an application. An application consists of Resources, is a kind of Resource and is created using the services of an ApplicationFactory. Operations declared in the Resource class are overloaded in the Application class. For instance, the invocation of the releaseObject operation on an instance of the Application class releases all resources that an application has and returns the

capacity of hardware devices granted to the application.

There is an important aspect of applications that deserves further explanations. Every application has a profile. Its *profile* is a specification of how the application is assembled in terms of resources and needs of capacity on hardware devices. This specification is expressed using the XML language and is stored either in a file or in memory. Hence, the application has an attribute named profile which is either a reference to a file (which can be resolved using the file service (of the CF Services package)) or an internal representation of an XML specification.

ApplicationFactory Class

The creation of an application is done by name using the create operation declared on the ApplicationFactory class. An instance of the ApplicationFactory class is associated with one named type of applications and one XML specification for the assembly of this type of applications. The create operation takes three parameters: the name of the instance the application being created, a mapping of resources to hardware devices and a configuration consisting of a set of properties (i.e. identity-value pairs).

Device Class

An instance of the Device class (see Figures 7 and 8) is a software abstraction of a hardware device. It is analogous to the notion of device presents within Unix. It is hence another kind of resource. The Device class defines attributes that describe the capabilities of a device in terms of ports and properties.

There are three sub classes of the Device class, see Figure 8. An instance of the AggregateDevice class is a device composed of other devices. An instance of the LoadableDevice class can be loaded with software (e.g. a driver). This software determines its behaviour. An instance of the ExecutableDevice class declares operations for the control of the execution and termination of a software loaded on a device.

DeviceManager Class

Instances of the Device class must be registered with an instance of the DeviceManager class. Its keeps a list of registered devices and a map of instances of the Device class to hardware devices, which are designated with labels (e.g. on Unix, name of entries if the /dev directory). The device manager is responsible for the creation of the file system (defined in the CF Services package). The device manager should register itself with the domain manager, to be discussed next.

DomainManager Class

An instance of the DomainManager class is the entry point to a domain, which is a set of hardware devices and applications. It provides operations for the control and configuration of a domain. A key operation addresses the installation of applications within a domain. Indeed, before an application can be created, using the create operation of the appropriate application factory, it must be installed within a domain. The installApplication operation takes the name of a file that defines the profile of a type of applications. It verifies that all files required for creating an application of this type are present in the file system. If yes, is reports the success of the installation by generating an event and creating a log entry using the event service and logging service defined in the CF Services package.

After the installation, the application can proceed with the creation and assembly of all resources that are part of an application. Note that a DomainManager also knows the application factories and provides operations that return handles to them. An instance of the DomainManager class can be registered with a naming service (defined in the package CF Services), which can be lookup by distributed client programs. At this point the paper has given almost the big picture of the SWR PIM of the OMG SWR DSIG.

The DomainManager class realizes four interfaces: the DomainRegistration, ApplicationInstallation, DomainHCI and DomainEventChannels interfaces, see Figure 9.

DomainRegistration Interface

A domain contains hardware devices, services (see the CF Services package) and applications. When an instance of the Device class is inserted in the domain, the device and its domain manager must be registered. This is normally done when a SWR is initialized, following a start-up or reboot. To this end, the DomainManager realizes the DomainRegistration interface. A device manager is registered with the registerDeviceManager operation while a device and its association with a device manager are registered with the registerDevice operation.

The following services may be inserted within a domain: event service, file service, log service and naming service. A new service is inserted within a domain with the registerService operation.

ApplicationInstallation Interface

The ApplicationInstallation interface declares operations with which applications are introduced within a domain. A new application is introduced

with the installApplication operation. The argument of the operation is the name of a file containing the profile of the application.

DomainHCI Interface

The DomainManagerHCI interface declares a set of operations that a HCI client would use to retrieve domain information. For instance, the getApplicationFactories operation returns the list of factories which are available to the client.

DomainEventChannels Interface

The DomainEventChannels interface declares a set of operations that a client can use for registering or unregistering with a domain event channel. A registered client is notified when applications are either installed or uninstalled.

5. Conclusion

This tutorial has introduced the PIM of a SWR being developed by the OMG SWR DSIG. This is ongoing work. The reader may follow the progress made by the DSIG at the following URL: swradio.omg.org. UML models and related documents are available on-line. Future work is the development of PSMs and EDMs. Another important issue that needs to be addressed is the validation of the mappings from the PIM to the PSMs and from a PSM to the EDMs.

Acknowledgements

The authors would like to acknowledge financial support from Communications and Information Technology Ontario (CITO) and Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[Arch 01] Architecture Board ORMSC, "Model Driven Architecture," Document number ormsc/2001-07-01, 2001.

[CRC 02] Communications Research Centre (CRC) Canada, "SCA Reference Implementation," www.crc.ca/en/html/scari/home/home, 2002.

[Join 01] Joint Tactical Radio Systems (JTRS) Joint Program Office (JPO), "Software Communications Architecture Specification," V2.2, www.jtrs.saalt.army.mil/SCA/SCA.html, 2001

[Mito 00] J. Mitola III, "Software Radio Architecture – Object-Oriented Approaches to Wireless Systems Engineering," John Wiley & Sons, Inc., 2000.

[OMG 01] Object Management Group (OMG), "OMG Unified Modeling Language Specification," Version 1.4, September 2001. (available at: www.omg.org)

[OMG 02a] Object Management Group (OMG), Software Radio DSIG, swradio.omg.org, 2002.

[OMG 02b] Object Management Group (OMG), "Model Driven Architecture (MDA)," www.omg.org/mda, 2002.

[SDR 02] Software Defined Radio (SDR) Forum, www.sdrforum.org, 2002.

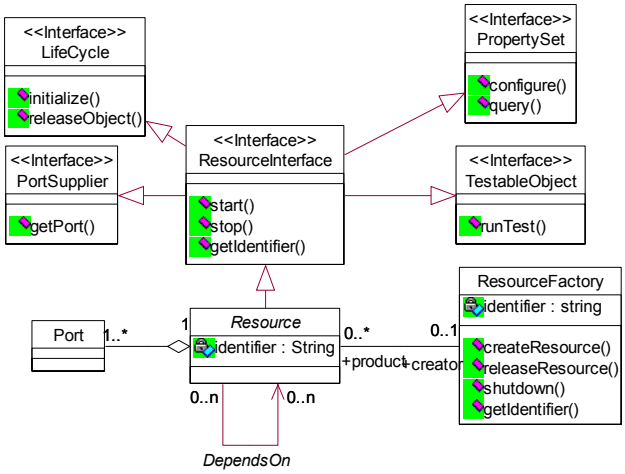

Figure 6. Class diagram of Base Application.

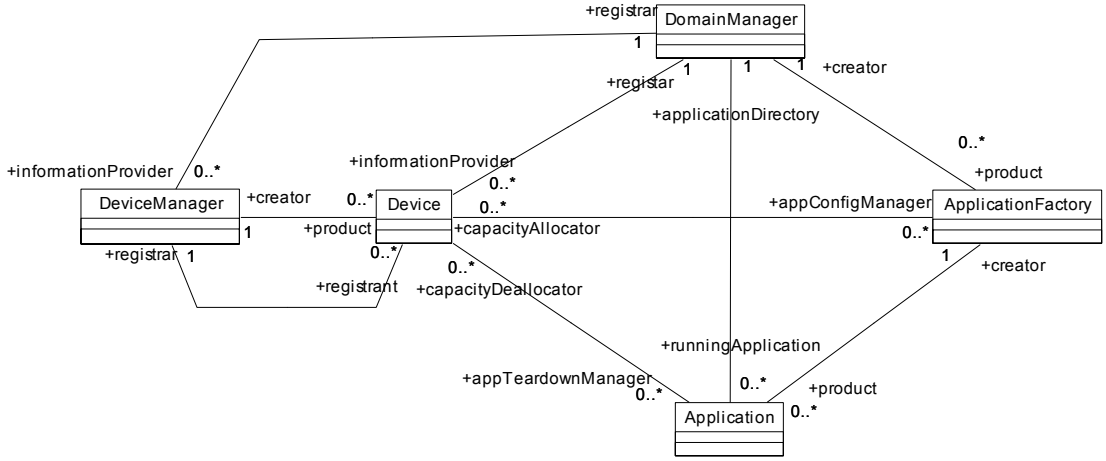

Figure 7. Class diagram for Application and Device.

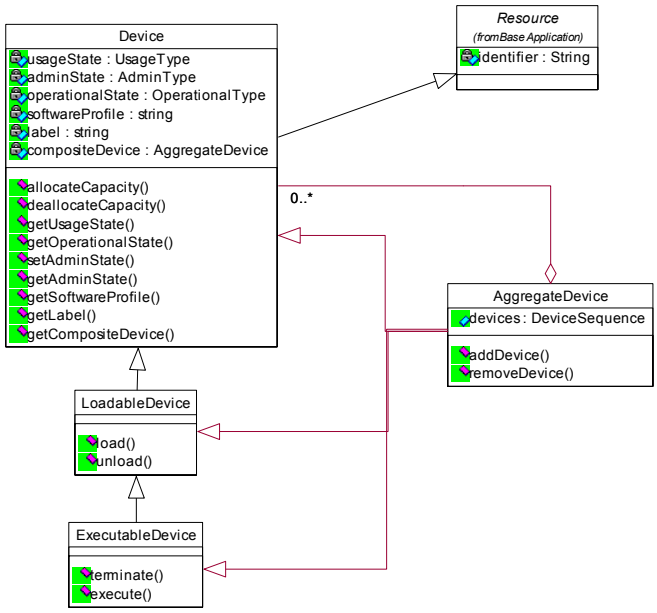

Figure 8. Class diagram for devices.

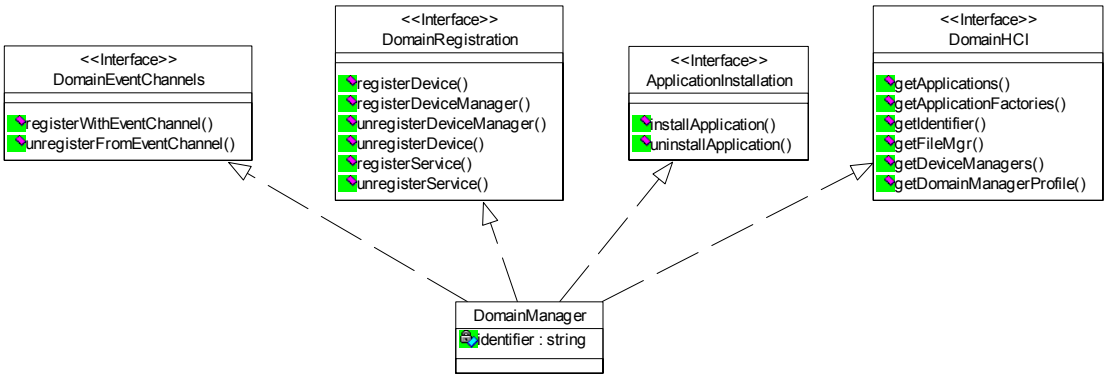

Figure 9. The DomainManager class.