Applying Model Driven Architecture Techniques to Achieve Reconfigurable Software Defined Radios

Francis Bordeleau, Mark Hermeling, John Hogg

Affiliation: Zeligsoft Inc.
Emails: {francis, mark, hogg}@zeligsoft.com

1. INTRODUCTION

Complex software systems, like Software Defined Radios (SDR), regularly need to be reconfigured during their lifetime due for example to the need to replace a faulty component, to optimize resource usage, or to adapt to a changing environment. *Dynamic reconfiguration* is the ability to modify and extend a system while it is running. Because of the mission critical (and often life-critical) role played by SDR technology, dynamic reconfiguration constitutes a key development aspect. In such systems, it is essential to test and validate all possible deployments and configurations of the system before deploying it.

While the problem of dynamic reconfiguration of software systems is not new, the emergence of component-based software and Model Driven Development (MDD) technologies in the last decade has opened the door for new solutions. In [1], the authors provide a comprehensive conceptual framework within which to address problems and solutions related to dynamically reconfigurable systems in a systematic and consistent manner. The model identifies and categorizes the various types of change that may be required, the relationship between those types, and the key factors that need to be considered and actions to be performed when such changes take place.

This paper focuses more particularly on the deployment aspect of reconfigurable systems. It defines an MDD-based approach to address the problem of dynamic reconfiguration of SDR. The MDD approach is used to specify 1) the application and the services it requires; 2) the platform and the services it provides; and 3) the deployment of the application to the platform. The MDD-based models allow for early specification of the issues at hand and provide a single source of information that can be used from architecture through to deployment, configuration and reconfiguration. The models can be used to analyze and validate possible deployment and reconfiguration scenarios.

The first part of the paper introduces the MDD based approach used to specify the application, the platform and the deployment relations between them. The second part highlights what information can be learned from the models through analysis of different deployment scenarios.

2. BACKGROUND

1.1 Component-Based Software

The idea of *component-based software development* is to build an application by assembling a set of reusable components that can be independently deployed, configured, and connected together. The key particularity of a component, compared to traditional software artifacts like procedures, functions or objects, is that a component is a run-time reusable entity as opposed to a design-time reusable entity.

Components enable the building of families of products based on standard building blocks. It also enables portability and reconfigurability; a component is not linked to a particular hardware platform. It can be configured (and reconfigured) at run-time to fit on different platforms or different platform configurations. This makes applications portable and reconfigurable.

1.2 Model-Driven Development

The idea of Model-Driven Development (MDD) is to use models as the primary artifact for software development as is the case in traditional engineering disciplines like mechanical, civil, and electrical engineering. A key benefit of models, besides being easier to understand than traditional source code based designs, is that they can be used for analysis and validation of different system properties, and for the generation of other types of artifacts like other models, code, and documentation.

In embedded systems like SDR, MDD can be used to create distinct component models for applications and platforms. The application and platform models are then linked through dependency relations and deployment constraints. The models can be used in the development process to analyze potential deployments of an application on a given platform or to validate the feasibility of a specific deployment. The information contained in these models can also be used at run-time to deploy, configure, and reconfigure the applications on the platform.

3. SYSTEM MODELING

To achieve maximum portability and reusability we define both the application and the platform as an assembly of independent components.

1.3 Components

The Unified Modeling Language (UML) [2] defines a *component* as representing a modular part of a system that encapsulates its content and whose manifestation is replaceable within its environment. A component provides functionality to other components through *provides ports* and uses functionality from other components through *uses ports*. Components can be assembled into larger entities by connecting their interfaces together through connectors. Components can be individually configured according to the role they play in an application.

Components are defined independently of the device they execute on. A component is associated with a set of implementations that can be deployed on different hardware devices. Each implementation is defined in terms of its specific deployment requirements (processor type, operating system, memory, dynamic link libraries, and so forth). The different component implementations can be written using different programming languages, like C or C++ for a general purpose processor or VHDL for an FPGA..

1.4 Platform

A *platform* forms an abstraction of the processing capacities available in a system. The platform is a logical abstraction of the physical processing elements. The processing elements (devices) provide resources to applications. Some examples of these resources might be memory, processing power (in MIPS) or high-speed communication links. The components that make up a platform are referred to as ‘devices’ and ‘managers’. *Devices* can provide specialized resources like ‘operating system’ and ‘processor’. Other resources indicate capacities like ‘memory’, ‘processing power’ or ‘throughput’. Managers are responsible for controlling the devices.

A platform can consist out of multiple nodes, each one abstracting the processing capacity of a specific hardware board in the system. Figure A (left side) provides an example of a platform (ThePlatform) model consisting of a single node (SigProcNode) with two devices (fpga and gpp) and one manager (devicemanager).

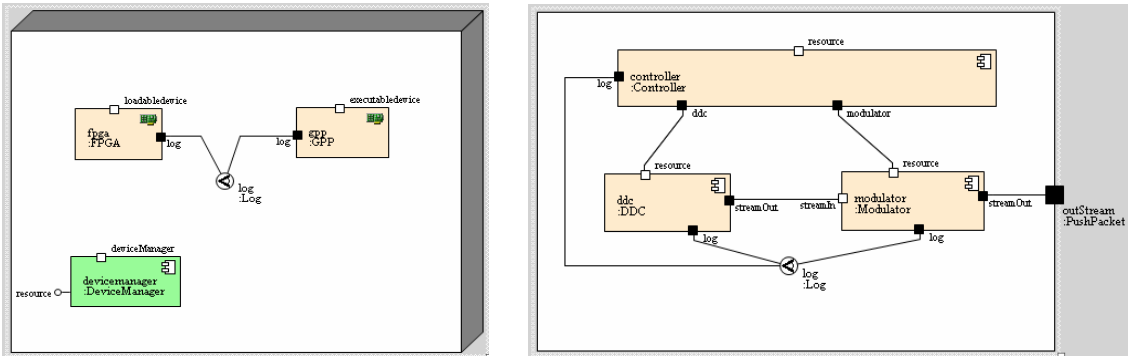

Figure A -- SigProcNode and ModulatorApplication

1.5 Application

An *application* provides the functional behavior of a system and makes use of the resources of the platform. An application is made up of a number of components. Each of these components provides certain functionality of the system. Components in the application are executed on devices in the platform. Components can depend on resources provided by the devices; this effectively limits the set of devices that a component can be executed on.

Figure A (right side) provides an example of a simple application (ModulatorApplication) model that consisting of three components (ddc, modulator, and controller).

1.6 Component Framework

A *Component Framework (CF)* provides a set of services and facilities to application and platform developers. The CF analyzes the application and its dependencies as well as the resources of the platform. Based on this information it will deploy and configure the application to the platform. During deployment, it will perform the actions necessary to download component implementations to the devices that are assigned to. The CF is also responsible for establishing the connections between the application components and between the components and the devices on the platform, as well as configuring all components.

In the context of SDR, the Software Communication Architecture (SCA) [3], which has been defined by the US Department of Defense (DoD) Joint Tactical Radio Systems (JTRS), constitutes the leading standard. The SCA Core Framework (CF) provides the basic functionality to deploy, configure, and manage SDR systems. While not directly addressing dynamic reconfiguration, the SCA CF provides basic functionality that can be extended to develop reconfigurable SDR systems.

4. EVALUATING DEPLOYMENTS

In order to ensure safe reconfiguration of an application on a specific platform, it is essential to analyze all possible deployments of the application on that platform. Any illegal (or undesired) deployments must be explicitly excluded from the set of possible deployments.

A *Deployment* is defined as a specific allocation of application components to platform devices. Figure B specifies a deployment, called Deployment1, of ModulatorApplication on ThePlatform in which the ddc component is allocated to the fpga device, and the controller and modulator components are allocated to the gpp device.

Whether a particular deployment is valid depends on the implementations associated with the components, the requirements of these implementations and the resources provided by the devices. The ddc requires an implementation that matches the fpga to be able to execute on that device. If both the controller and the modulator execute on the gpp then that device needs to have sufficient resources in terms of memory and processing power.

A deployment is *valid* if all of the deployment constraints of each component can be satisfied by the device allocation. More than one valid deployment for a given application on a platform may exist.

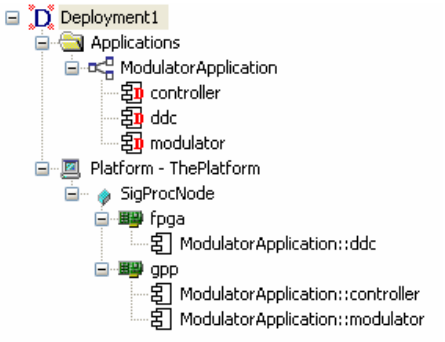

Figure B -- Deployment of the application on the platform

The following three questions arise when dealing with application reconfigurability:

- **Does a valid deployment exist?**
  The most basic use case for assessing the deployability of a component-based application on a platform consists in determining if a valid deployment of the application on the platform exists
- **What is the set of all possible deployments?**
  Some component frameworks, like the SCA standard, allow for dynamic planning of deployment at run-time. In such systems, the deployment of an application can result in many different concrete deployments. Being able to determine the set of all possible deployments is of critical importance. Developers must ensure that no forbidden deployments are possible and that all desired deployments are possible.
- **Is a specific deployment valid?**
  Another critical issue of application deployment consists in being able to determine if a specific deployment of an application on a platform is possible.

Answering these questions through manual evaluation of the application and the platform is virtually impossible. An automated system is required for this.

5. MODEL DRIVEN AUTOMATION

Applications and platform models, as defined in the previous sections, can be used to calculate and analyze all possible deployments of an application on a platform at design time. The process of determining whether a single application or a set of applications can be deployed on a platform becomes one of finding an implementation to component assignment and a component to device assignment that satisfies all the predetermined deployment constraints. Since the set of all possible assignments is finite, a simple rule-based system based on backtracking techniques may be used to iterate through the search space and evaluate the possible deployments. For every implementation and component assignment put to the test, validation is done to evaluate whether each of the constraint is met. If a constraint is not fulfilled, then the validation fails, while providing a reason for the failure.

6. CONCLUSION

Model driven techniques can be used to express embedded systems like SDR in terms of applications and platforms. Applications can be given *requires* relationships to resources provided by the platform. Deployments can be expressed by allocating components to devices on the platform. The information in the model can be used to perform different types of deployability analysis to the application. The model driven nature of the approach presented is vitally important. It allows for automation to assist the designer in analyzing and evaluating the different possible deployments of a set of applications on a platform, a tedious task and error prone when done by hand. The analysis and evaluation of the different possible deployments is of crucial importance in mission-critical systems like SDR systems.

7. References

[1] D. Walsh, F. Bordeleau, B. Selic. “A Domain Model for Dynamic System Reconfiguration”, In *Proceedings of ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML 2005)*. Montego Bay, Jamaica, October 2-7, 2005.

[2] Object Management Group, Unified Modeling Language (UML) 2.0. http://www.uml.org/

[3] Joint Tactical Radio System (JTRS) Joint Program Office (JPO). Software Communications Architecture (SCA). http://jtrs.army.mil/sections/technicalinformation/fset_technical_sca.html