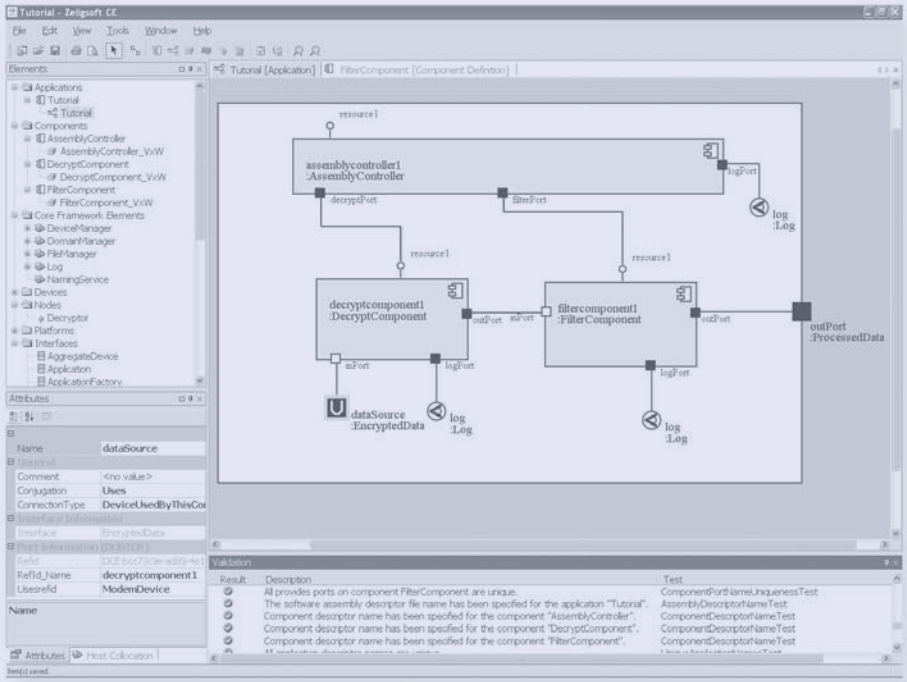
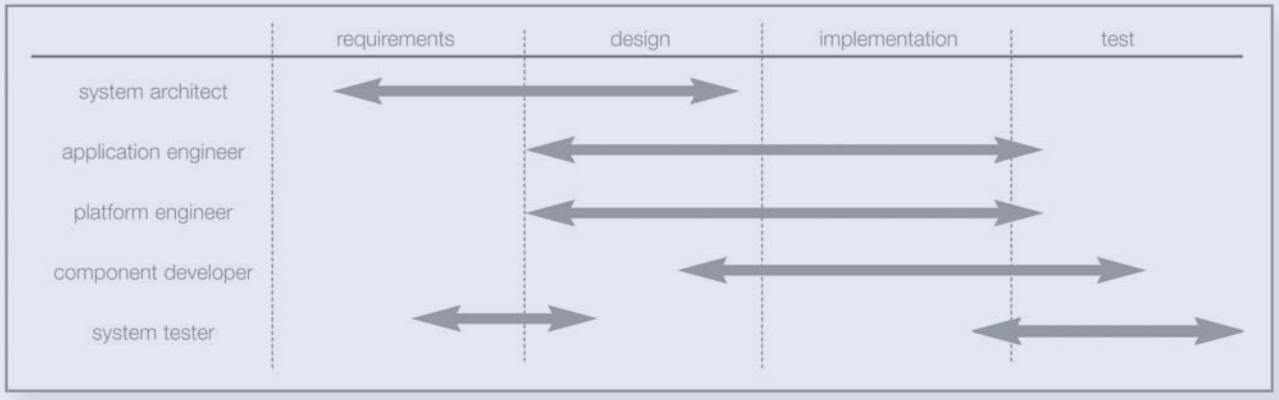
SCA Deployment Management: Bridging the Gap in SCA Development

John Hogg, Francis Bordeleau

zeligsoft


Tutorial - Zeligsoft CE
File Edit View Tools Window Help
Elements
Applications
Tutorial
Components
AssemblyController
DecryptComponent
FilterComponent
Core Framework Elements
DeviceManager
DomainManager
FileManager
Log
NamingService
Devices
Nodes
Decryptor
Platforms
Interfaces
AggregateDevice
Application
ApplicationFactory
Tutorial [Application]
FilterComponent [Component Definition]
resource1
assemblycontroller1
:AssemblyController
logPort
decryptPort
filterPort
log
:Log
decryptcomponent1
:DecryptComponent
outPort
inPort
filtercomponent1
:FilterComponent
outPort
:ProcessedData
dataSource
:EncryptedData
Attributes
Name
dataSource
Comment
Conjugation
Uses
Validation
Result
Description
Test
All provides ports on component FilterComponent are unique.
ComponentPortNameUniquenessTest
The software assembly descriptor file name has been specified for the application "Tutorial".
AssemblyDescriptorNameTest
Component descriptor name has been specified for the component "AssemblyController".
ComponentDescriptorNameTest
Component descriptor name has been specified for the component "DecryptComponent".
ComponentDescriptorNameTest
Component descriptor name has been specified for the component "FilterComponent".
ComponentDescriptorNameTest


requirements
design
implementation
test
system architect
application engineer
platform engineer
component developer
system tester

SCA Deployment Management: Bridging the Gap in SCA Development

John Hogg, Francis Bordeleau

Abstract: Software Communications Architecture (SCA) systems are based on portable, component-based applications executing on flexible hardware platforms. The actual assignment or *deployment* of software components to hardware devices is done at system initialization time. How can the architect or system integrator ensure that this assignment will work properly and be fully tested?

This paper answers that question: through modeling, validation, analysis and enforcement of deployments using automated technology. Deployment management closes the gap between software and platform teams, increasing quality in fielded systems.

1 INTRODUCTION

The Software Communications Architecture (SCA) [1] is a powerful framework for realizing flexible, reusable component-based applications. Architects specify applications and hardware platforms as separate profiles, or sets of XML descriptor files. These profiles describe the requirements and capabilities of each component, application, device and platform. Together they allow sets of applications to be deployed on different hardware platforms with minimal porting effort.

Both application and platform profiles are well-defined by their descriptor files. Furthermore, sophisticated modeling tools are available to represent and validate software and hardware architectures and generate reliable descriptor file sets. The missing link is the connection between the two, or the *deployment* of application components to platform devices. Current tools have no representation of the actual assignment of components to devices and no simple means of validating the correctness of such an assignment. The SCA standard has a limited facility for enforcing a chosen deployment at run time, but it contains no descriptor file that can be used to enforce such a deployment. In short, deployment specification, validation and control has been poorly understood and poorly supported.

This paper presents a solution to the problem.

The remainder of this section provides background on the SCA context of deployment management and goes into more depth on the deployment problem. Section 2 explains how deployments can be specified or modeled. Unworkable deployments can be modeled as easily as valid ones, so section 3 describes deployment validation. Section 4 shows how the process of finding valid deployments can be automated. When the desired deployment or deployments have been selected and verified, section 5 explains how they can be enforced in delivered systems. Finally, section 6 summarizes this paper and explains how this technology can be used today.

1.1 SCA software and platform modeling

The Software Communications Architecture (SCA) enables the delivery of flexible, portable radios and other devices. The SCA framework is composed of component-based software *applications* (also referred to as *waveforms*) deployed on a flexible hardware platform. For an overview of how SCA architectures can be defined, validated and delivered, see [6] and [7].

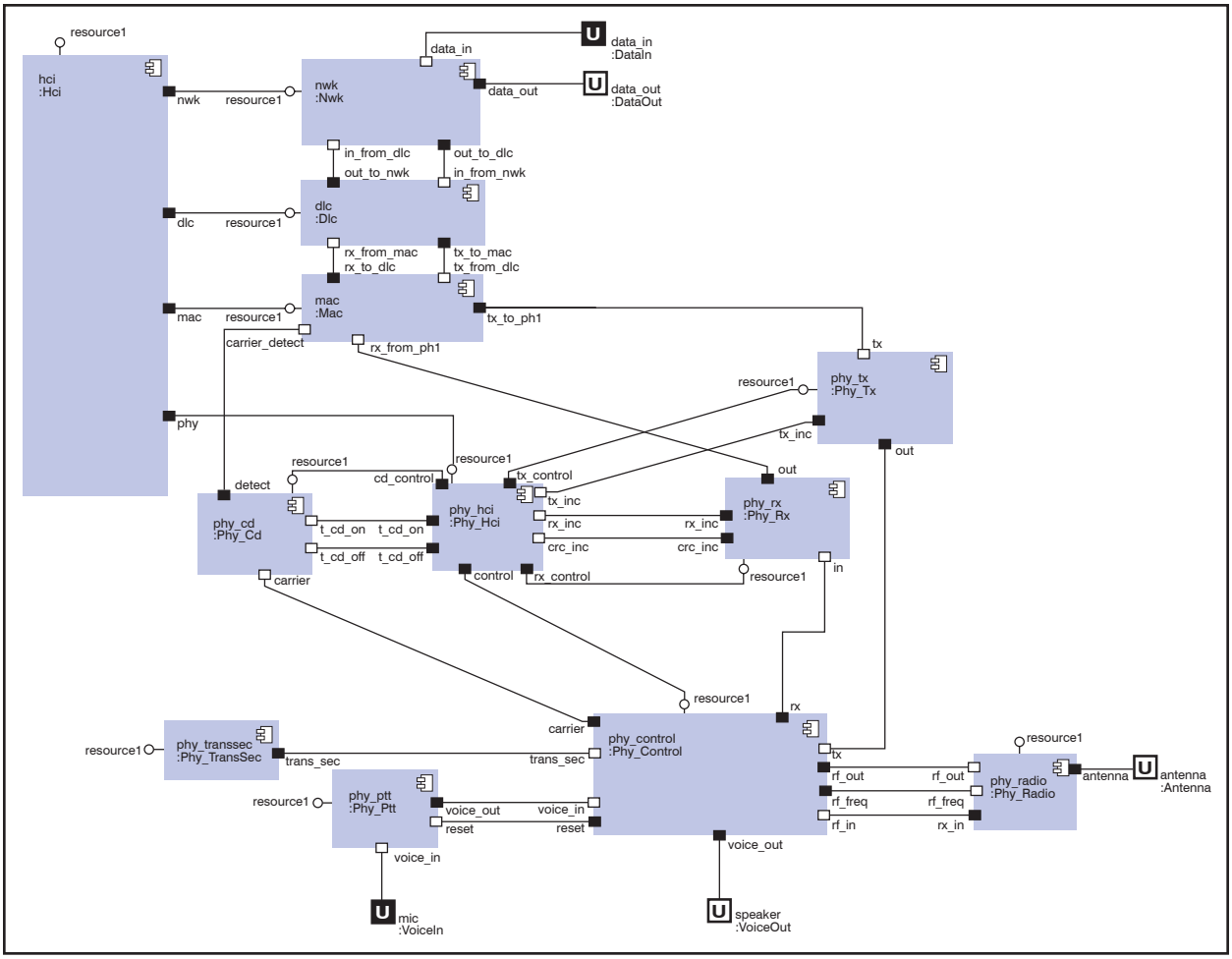
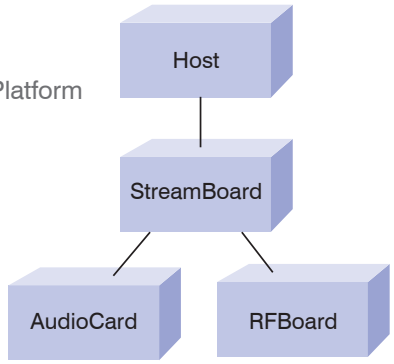


Figure 1: FM3TR Application[1]

Best-practice SCA development includes graphically modeling applications and platforms to understand and validate architectures and detailed designs. The software and platform *profiles* or sets of XML descriptor files are generated from these validated models. An example of an application (the Future Multiband Multiwaveform Modular Tactical Radio or FM3TR test waveform [2]) is shown in Figure 1. This diagram shows *components* (the rectangles) communicating through *ports* (the small black and white squares) associated with each other through *connections* (the lines).

The FM3TR application runs on a *platform* shown in Figure 2. A *platform* forms an abstraction of the processing capabilities available in a system. The platform is a logical abstraction of the physical processing elements. A platform can consist out of multiple nodes, each one abstracting the processing capacity of a specific

Figure 2:
FM3TR Platform



1. This is a sample decomposition of the FM3TR waveform, other decompositions could be considered as well.

hardware board in the system. The **FM3TR Platform** model of Figure 2 is composed of four nodes: **Host**, **StreamBoard**, **AudiCard**, and RFBoard.

Nodes are in turn composed of *devices* and *managers*. Devices provide resources to applications. Some examples of these resources might be operating system, memory, processing power (in MIPS) or throughput. *Managers* are responsible for controlling the devices. The **StreamBoard** node of Figure 2 is shown in Figure 3. It consists of a four devices (**gpp**,**dsp1**, **fpga1** and **fpga2**) and one manager (**devmanstreamboard1**).

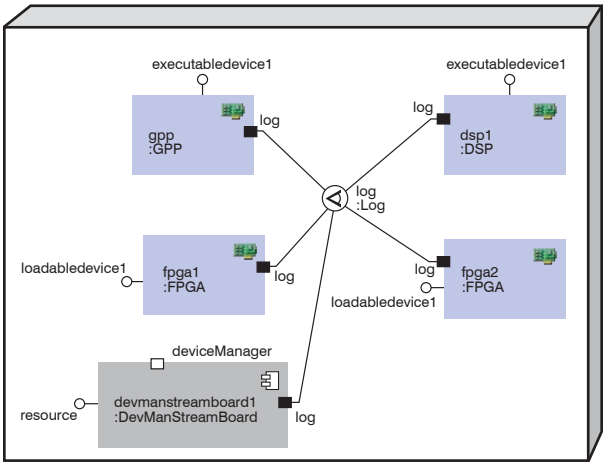


Figure 3: FM3TR Stream Board Node

There are two aspects to a device. First, it is a physical piece of hardware such as a general-purpose processor (GPP), field-programmable gate array (FPGA) or digital signal processor (DSP). A simple *device* such as a power supply just runs. The behavior of a *loadable device* such as an FPGA may be controlled through software loaded onto it and an *executable device* is a GPP capable of executing code.

These distinctions are important because a physical device also has a logical device analog. The logical devices are themselves components and control their underlying physical devices. The physical devices on the **StreamBoard** node of Figure 3 include two FPGAs and a DSP. These are controlled through their logical devices that run on the GPP.

But how does the architect or developer know that the FPGA and DSP logical devices execute on the GPP? And as a more challenging question, how does the architect or developer know which of the components in Figure 1 run on which of the devices in the Stream Board node or in the other nodes of Figure 2? The SCA profile XML descriptor files provide full definitions of each individual application and of the platform, but these descriptions are all independent. No SCA descriptor file connects them, the connection happens during startup of the software on the hardware. Deployment specification, validation and enforcement have been the missing links in SCA best-practice development.

1.2 Deployment

The term “deployment” has multiple meanings. For example it can mean delivering working systems to the field. It can also mean physically loading components (including logical devices) on physical devices. In this context it specifically means assigning components to physical devices so they can be loaded and executed.

The deployment of a component to a device is constrained in many ways. The component has various needs; for example, it will require a certain type or family of processor, it may require a particular operating system, and it may need other capabilities such as floating-point hardware or certain FPGA characteristics.

The platform architect can make deployment decisions up-front because a platform’s logical devices will execute on the platform itself, and therefore the entire environment is visible. The SCA allows the user to specify the physical device where a logical device will be deployed using a `deployondevice` element. If no `deployondevice` is specified, the logical device is deployed to the same physical device as the devicemanager. The platform deployment is therefore well-understood from the beginning.

By contrast, application deployment to a platform is not defined either in the platform or the application. The SCA is designed to support application portability, so neither the platform nor the application can have this information. Furthermore, there is no standardized XML descriptor "deployment profile" to capture this mapping.

Is the deployment problem really significant? How many potential deployments can there be? Assignment of components to devices is exponential, so

- 6 components and 2 devices have 64 potential deployments
- 9 components and 3 devices have 19,683 potential deployments
- 20 components and 4 devices have 1,099,511,627,776 (over a trillion!) potential deployments

Since a single SDR application can easily have 20 components, a platform usually supports multiple applications and a platform can easily have half a dozen devices, these numbers just scratch the surface.

Of course, many component-to-processor assignments are clearly unworkable. An FPGA component will never be able to execute on a GPP and vice versa. Nonetheless, the problem size is well beyond the capability of manual analysis for significant systems.

A valid deployment satisfies SCA deployment constraints, as discussed later. A verified deployment has been thoroughly tested. To ensure proper radio behavior in the field, it is necessary to find all valid deployments of applications onto a platform, verify these thoroughly, and ensure that only these deployments can occur in fielded systems.

1.3 Deployment Management Contexts

How does the architect find, validate and enforce these verified deployments? Detailed workflows and use cases will be different for each software radio builder and vary in difference contexts, but some basic approaches are common.

Several situations are possible in the delivery of a software radio platform with a set of applications:

- The platform and a set of applications may be developed together in the delivery of an entirely new product (a "green field" situation)
- A new application may be added to an existing platform, which may or may not have other applications
- An existing non-SCA application may be ported to an SCA platform
- An existing SCA application may be ported to a new SCA platform

Each of these situations has unique workflow aspects and concerns. This paper does not dwell on their differences, but rather the common issues of deployment. Special concerns in specific contexts will occasionally be discussed.

2 DEPLOYMENT SPECIFICATION

A representation of deployments is a fundamental requirement for understanding, validating and enforcing them. Analysis is only possible when there is something to be analyzed. This “something” is a representation or model of a deployment, and is natural and straightforward.

At its simplest, a deployment model is a mapping from components to devices. Each SCA component must be deployed at run time on exactly one device. An SCA component can have multiple implementations for different platforms or optimized for different characteristics.

A deployment model can be presented in several ways. An intuitive browser-like presentation is shown in Figure 4. The example combines the application and platform of the previous diagrams. A deployment has one or more applications (here, the single application of Figure 1) containing icons of component instances that must be deployed. It also has a platform (here, the platform of Figure 2) containing nodes which in turn contain devices (some of which were shown in Figure 3). Finally, the devices contain the component instances deployed to them. The top part of the diagram shows the component instances that must be deployed; the bottom part shows the actual deployment.

The deployment illustrated in Figure 4 is partial. The “D”s on the component instances in the application show that they have been deployed onto a device. Some of the application component instances have not been deployed; they are lacking the “D” adornment. A complete deployment has been specified when no component instances in applications are unadorned.

The gesture for deploying (or redeploying) a component instance is a natural drag-and-drop. With this interface the user can quickly specify a full deployment of a complex platform and set of applications.

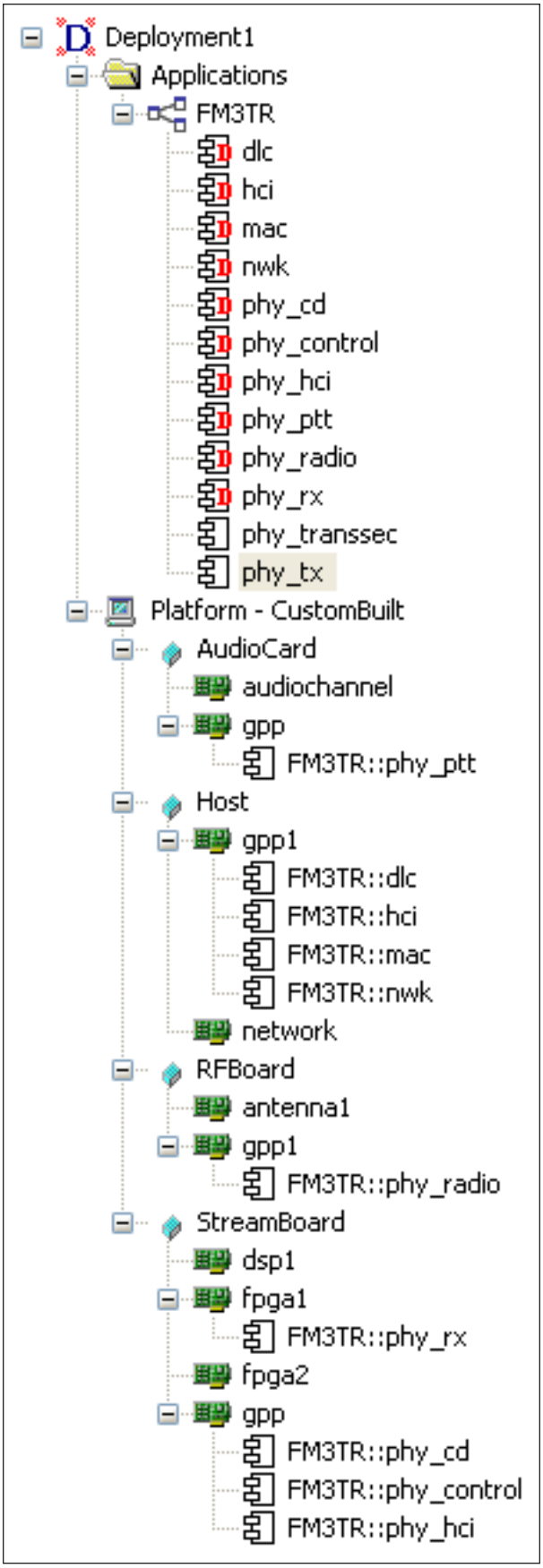


Figure 4: Partial Deployment

3 DEPLOYMENT VALIDATION

Of course, the mere fact that a deployment can be specified does not mean that it makes sense. Specifying a deployment is easy. Specifying the *right* deployment can be much harder.

The number of potential deployments (i.e. assignments of components to devices regardless of validity) grows exponentially with the number of platforms and components. The vast majority of these assignments are invalid, because a valid deployment must satisfy a variety of constraints. Some of these involve a single component and the physical device on which it's deployed; some involve multiple elements, possibly from multiple applications.

A full description and discussion of deployment constraints is beyond the scope of this paper. Examples of general categories of deployment constraints are:

- Environment dependencies of OS and processor: can the logical device execute the component's code?
- Allocation dependencies: is there enough of a quantified resource to be shared between deployed components? Are non-quantified resources available?
- Host collocation constraints: are components required to be host collocated deployed on the same device?
- Uses relationships (such as <devicethatloadedthiscomponentref>, <deviceusedbythiscomponentref>, <usesdevice>): do the required ports and interfaces exist on the referenced logical device?

For a concrete example, consider one of the cases in the last category above. An SCA connection end may specify a port and a <devicethatloadedthiscomponentref>. Validating this connection in a deployment involves several steps:

1. Find the component instance specified by the <refid> attribute of the <devicethatloadedthiscomponentref>. (It may be the component at the other end of the connection, but may also be any other component that is part of the application.)
2. Find the physical device on which the component instance of Step 1 is deployed from the deployment model.
3. Find the logical device on the platform representing the physical device of Step 2.
4. Find a port on the logical device of Step 3 matching the name specified in the connection.
5. Find the port or interface at the other end of the connection. It may be another <devicethatloadedthiscomponentref>, in which case reiterate the steps above. It can also be any of the other connection ends supported by the SCA.
6. Validate the ports for consistency. Are they of suitable type (Uses to Provides or Interface)? Are the IDL interfaces consistent (identical or compatible through inheritance?)

<Devicethatloadedthiscomponentref> deployment validation involves only one application. Other deployment constraints must consider interactions between multiple applications. Manually finding, validating and maintaining a validated and verified deployment or set of deployments becomes increasingly complex as the number of applications, component instances, devices and connections grows.

Fortunately, it is not necessary to manually validate a deployment. Deployment constraints may be large in number and complex to analyze, but they are well-defined. Deployment validation has been automated. Using automated validation, an architect or developer can easily determine whether a deployment will satisfy SCA constraints.

Of course, a single valid/invalid bit of information would be of limited help to the user. The user must know exactly what makes a deployment invalid, where the error is and perhaps how to fix it. Automated deployment validation includes navigation to errors and suggested solutions.

Iterative development is a proven best practice for all but the smallest of problems. “Big-bang” solutions result in large, costly errors discovered late. This is equally true of deployment definition and analysis. There is no need to specify a full deployment before validating; it is both possible and highly desirable to validate a partial deployment. Of course, a partial deployment will never validate cleanly. The undeployed component instances will be treated as errors. When no other errors remain, though, the complete deployment risk will be significantly reduced.

There are several different approaches to building up partial deployments, and strategies will vary depending on whether the applications are under construction or are being ported from a previous platform. The least sophisticated approach in a green-fields situation is to deploy as much as possible as soon as possible: when a component instance and its target device are created in the model, add the deployment immediately. This may be crude, but it will help flag errors and incorrect assumptions early.

In a porting situation a complete set of applications and the platform may be available from the beginning, and a more organized approach will lead the user to an optimal solution faster. The two main strategies in this case are application-first and device-first. In the application-first strategy an entire application is deployed and validated, possibly in several iterations. When a good deployment has been selected the next application is deployed and validated by itself. The two application deployments are then combined and validated. This process is repeated until a complete validation has been constructed.

The disadvantage of an application-first approach is that critical resource contention may not be discovered until late in the process. For this reason a device-first approach may be used separately or in parallel with the application-first workflow. In device-first deployment, a device is selected and a set of components from all applications are deployed to it. If the device has a scarce resource (such as memory or bandwidth) this can be detected early and alternatives (such as redeploying components or even entire applications elsewhere) can be investigated.

The workflow described above is semi-automated: the representation and validation of deployment is tool-based, but the deployment decisions are entirely manual. It is in many respects a trial-and-error approach. Further automation can be applied, and it will be discussed next.

4 AUTOMATED DEPLOYMENT CREATION

The basic goal of the deployment task is to find a deployment or set of deployments that will be verified and ensured in fielded systems. The task of constructing the set of candidate valid deployments can be entirely manual. It can also be partially or fully automated.

In principle, finding all valid deployments is simple: simply construct all potential deployments and select any that are valid. As we've seen, there are combinatorial reasons why a brute force solution won't work. However, clever pruning strategies can reduce the time required, although analysis will always be computationally intensive.

When a set of valid deployments has been found, the user inspects them, selects a set of interesting ones and saves them as part of the software, platform and deployment model. They may be the basis of further evolution as the platform and applications evolve.

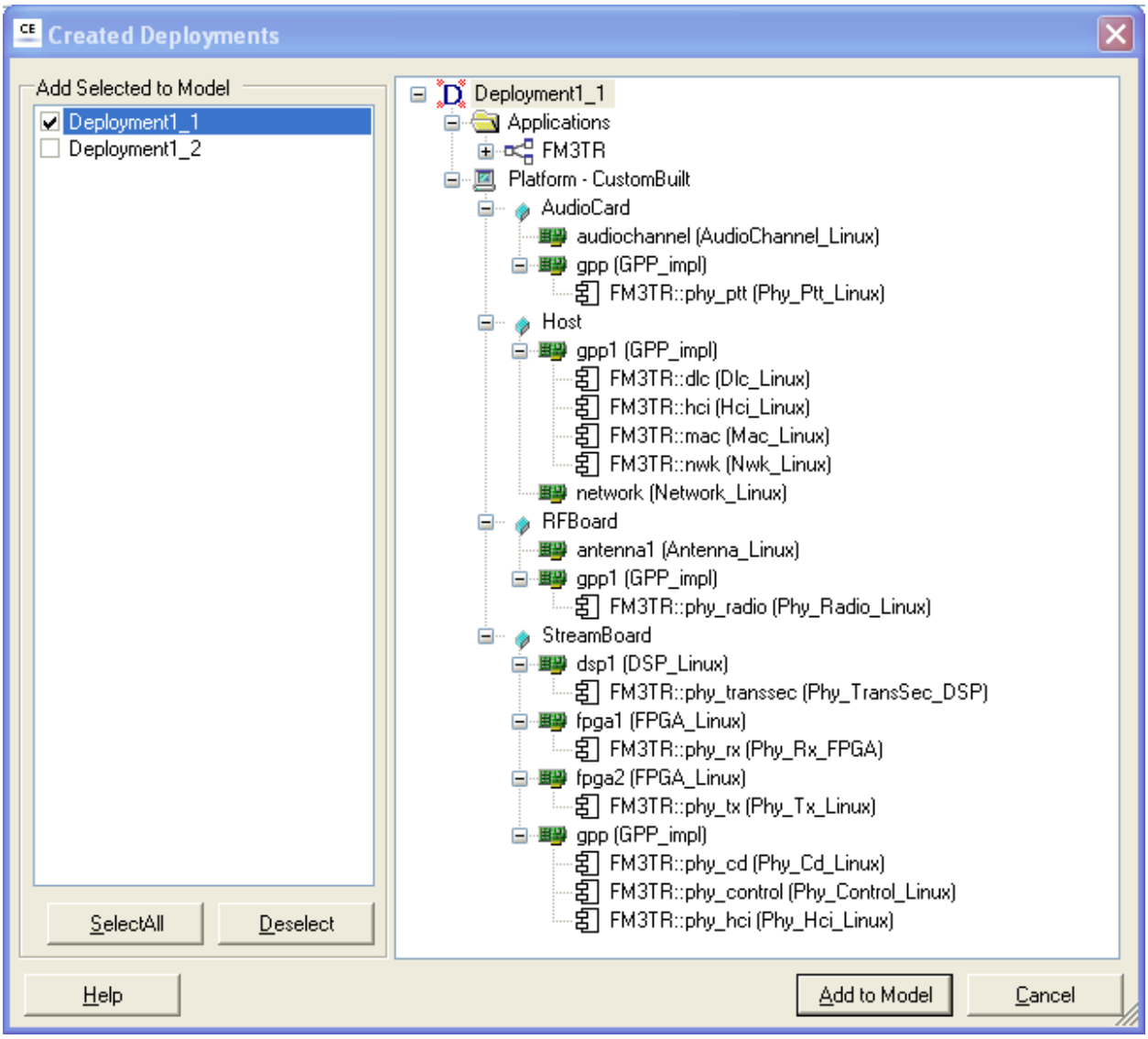


Figure 5: Possible Deployments

The simplest and most simplistic use of deployment automation is to create all possible deployments from a completely undeployed initial state. The resulting set of valid deployments may be large, however, and it is advisable to specify a maximum bound on solutions to report. Furthermore, if the set of valid deployments is large, then manual effort will probably be needed after the fact to evaluate their relative merits on other grounds.

In practice, the creation of valid deployments will not be entirely automated. The user will start by specifying at least a few "anchor points" of critical components to processor devices. These will be used as the basis of deployment automation: only the deployments that include the given partial deployment will be presented.

Figure 5 shows the set of possible deployments given the partial deployment of Figure 4. As each deployment is selected in the left pane it is displayed in the right pane. Checking the box beside a set of deployments and clicking "Add to model" brings them into the model.

The sheer number of valid deployments may also be valuable information to the user. It is a clear signal that deployments must be constrained in some way. Good practice demands that every fielded deployment must be tested in the lab.

Automated deployment generation can answer several questions for the user. The goal is indeed to find a desirable deployment. However, deployment analysis may also find undesirable deployments: deployments that are valid but have bad properties (for instance, bandwidth issues across the platform). These undesirable deployments can then be prevented by profile constraints or by runtime control in Section 5.

4.1 Deployment Analysis as a Porting Tool

The discussion so far has suggested the main problem is to prune down the discovered valid deployments to a manageable number. But what happens if no valid deployments are found?

This is a very real possibility, especially when porting a set of applications to a new platform. The applications may have been tailored to specific capabilities of their original environment. The porting task is to adapt them to the new platform and its capabilities. This is much simpler than starting from scratch, but there is no silver bullet; expertise and judgment are essential.

When no deployment is found, the architect faces a detective task: what stops the parts from fitting? Fortunately, a lot of evidence is available. The first key exhibit is the deployment on the original platform. Platform architectures will have similarities in their designs and the architect's first task will be to construct a similar deployment in the new environment. Since no valid deployment was found, this deployment will not be valid, but it can be validated. The errors reported in this validation can be critical in understanding the effort and cost of the entire porting effort. They identify the gaps in the resources required by the applications and provided by the platform.

This reinforces the value of deployment management throughout the development lifecycle, not just at then end. It's always cheaper and safer to find and understand issues early.

```
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE deployment_enforcement>
<!--Generated by Zeligsoft Component Enabler
    http://www.zeligsoft.com-->
<deploymentenforcement>
    <application id="DCE:ec08a5aa-fdad-4339-94ab-6f1afd2f729d" name="FM3TR" />
    <deviceassignmentsequence>
        <!--Component instance hci is deployed on device instance gpp1-->
        <deviceassignmenttype>
            <componentid>DCE:757d7094-0718-458e-95c5-3f34ff071cea</componentid>
            <assigndeviceid>DCE:bb99f471-b19a-4108-87a6-4ed55bad96ab</assigndeviceid>
        </deviceassignmenttype>
        <!--Component instance phy_ptt is deployed on device instance gpp-->
        <deviceassignmenttype>
            <componentid>DCE:a74d83fa-6af9-4036-9206-97e11005a9df</componentid>
            <assigndeviceid>DCE:8d11e069-918f-43ab-aefe-d4fcc96ffe21</assigndeviceid>
        </deviceassignmenttype>
        ...
        <!--Component instance mac is deployed on device instance gpp1-->
        <deviceassignmenttype>
            <componentid>DCE:2ce4a918-f266-4ded-8e6b-cdeba3128abd</componentid>
            <assigndeviceid>DCE:bb99f471-b19a-4108-87a6-4ed55bad96ab</assigndeviceid>
        </deviceassignmenttype>
    </deviceassignmentsequence>
</deploymentenforcement>
```

Figure 6: FM3TR Deployment Descriptor

5 DEPLOYMENT ENFORCEMENT

Once the architect or system integrator has completed a full analysis of deployments for the given applications and platform, one assignment of component instances to devices can be selected as the golden deployment, which will then be fully tested.

Unfortunately, the automated analysis described above might indicate that other deployments are also valid. Without further control, any of these could occur at system initialization time. What should the system integrator do to ensure the desired configuration every time?

The ideal answer would be, "Create a standard SCA descriptor file that will be used to enforce the given deployment." Unfortunately, there is no such descriptor file in the SCA standard. Nonetheless, there is something close. The `ApplicationFactory::create()` operation includes an optional `deviceAssignments` parameter. This specifies a mapping of component instances to devices.

From there it is a simple matter to define a descriptor file for each application and assign it a .dep.xml extension. The deployment of Figure 5 is specified by the descriptor file of Figure 6 (note that most of the file has been deleted to conserve space). Just such a file is generated as part of each application in a full deployment descriptor generation.

This file is not directly usable by a SCA Core Framework but can be easily integrated into any SCA environment. The XML format of the file maps directly to the IDL format of the `deviceAssignments` parameter and this gives the user almost[2] full control over the deployment.

In some cases the user may wish to only partially control deployment. This gives greater flexibility in deployments involving additional applications. This is done simply by generating from a partial deployment.

2. The control is "almost full" because the create() parameter does not specify the component implementation to be used in the deployment. If two valid deployments are available either could be chosen. This will not happen in common SCA development practice.

6 SUMMARY

Deployment modeling, validation and enforcement has truly been a missing link in SCA development; there have been no tools or even approaches and vocabulary to answer the questions that architects, developers and system integrators face. This paper described how deployments can be specified, how they can be automatically validated, how valid deployments can be generated, and how desired deployments can be enforced in fielded systems.

Automated support for all these aspects is available today in Zeligsoft Component Enabler™ (CE) 2.0. With CE 2.0, the SCA team can close the gap between platform and software developers to deliver a reliable, trusted system. For more information on how you can apply this technology to your project, go to www.zeligsoft.com or contact sales@zeligsoft.com.

6 REFERENCES

1. **JTRS**
   Joint Tactical Radio System (JTRS) Joint Program Office, *Software Communications Architecture Specification V3.0*, August 27, 2004. http://jtrs.army.mil/sections/technicalinformation/fset_technical_sca.html

2. **DERA**
   FM3TR Decomposition
   http://www.computing.surrey.ac.uk/personal/pg/E.Willink/wdl/Fm3tr.html

3. **OMG**
   Interoperable Naming Service Specification, November, 2000. ftp://ftp.omg.org/pub/docs/formal/00-11-01.pdf

4. **OMG**
   UML 2.0 Superstructure Specification (convenience document), October 8, 2004. http://www.omg.org/cgi-bin/doc?ptc/2004-10-02

5. **Raytheon**
   Joint Tactical Radio System (JTRS) SCA Developer's Guide, June 18, 2002. http://jtrs.army.mil/sections/technicalinformation/fset_technical_sca.html

6. **Zeligsoft**
   Mark Hermeling, John Hogg and Francis Bordeleau, *Developing SCA Compliant Systems*, 2005
   http://www.zeligsoft.com/Technology/Resources.asp

7. **Zeligsoft**
   Mark Hermeling, John Hogg and Francis Bordeleau, *Component Enabler Best Practices: SCA*, 2005
   http://www.zeligsoft.com/Technology/Resources.asp

Contact Information

Website: www.zeligsoft.com

Email: info@zeligsoft.com

Toll-free (North America): 1-800-ZELIGSW (1-800-935-4479)

Direct dial: +1 819-684-9639

Version 1.1
Printed in Canada