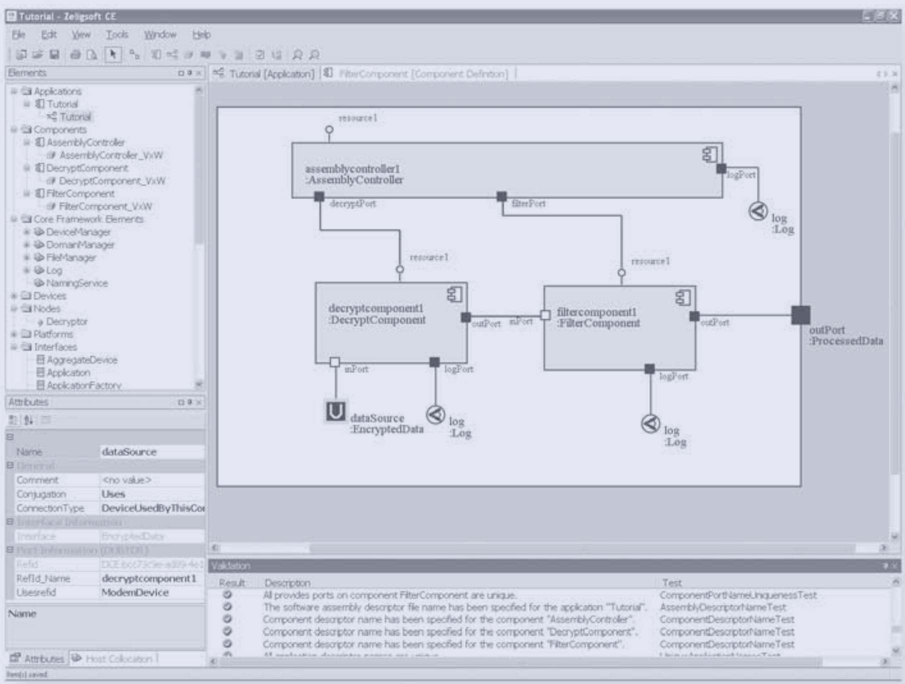
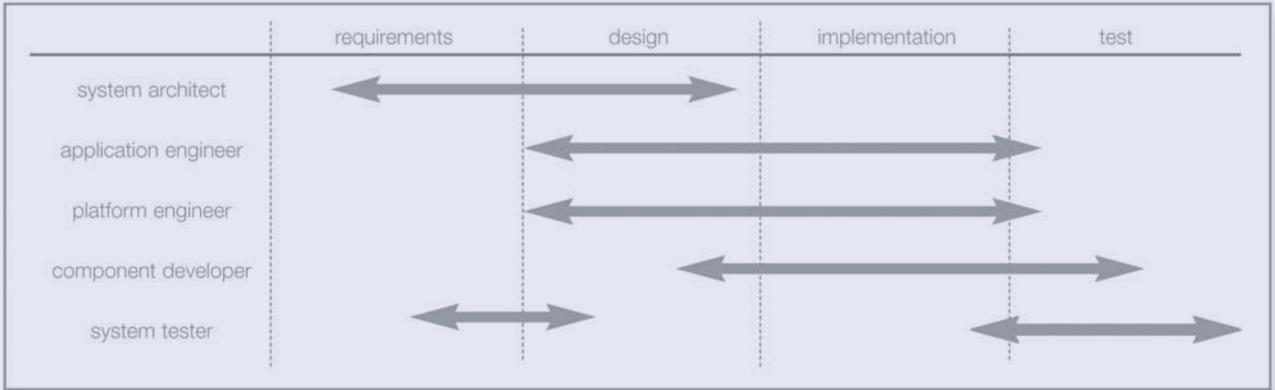
SCA Run-time Monitoring and Control:
Linking Design Artifacts to the Run-time Environment

Tim McGuire

zeligsoft


Tutorial - Zeligsoft CE
File Edit View Tools Window Help
Elements
Attributes
Validation
resource1
assemblycontroller1
:AssemblyController
decryptPort
filterPort
logPort
log
:Log
decryptcomponent1
:DecryptComponent
filtercomponent1
:FilterComponent
outPort
inPort
outPort
:ProcessedData
dataSource
:EncryptedData


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

SCA Run-time Monitoring and Control:
Linking Design Artifacts to the Run-time Environment

Tim McGuire

Software Communications Architecture (SCA) [1] systems are designed to be flexible in terms of run-time deployment. A single waveform can be deployed dynamically across a heterogeneous execution environment. The responsibility for this deployment lies with the SCA Core Framework (CF). Many projects building SCA compliant systems wrestle with the rules that govern this deployment. Even so, understanding deployments is a crucial aspect of testing and delivering a waveform. The danger in not understanding all possible deployments of an application is that untested deployments might occur in the field with disastrous results.

A previous paper, *SCA Deployment Management: Bridging the Gap in SCA Development* [2], covered modeling and validation of deployments from a static point of view. This paper will look at monitoring and analyzing deployments in a running system

1 INTRODUCTION

The Software Communication Architecture (SCA) [1] specifies the rules that govern deployment. The SCA Core Framework is responsible for deploying an application according to these rules. The SCA requires that software applications be specified independently from platforms (hardware abstraction layers). The platform provides services to the waveform and has capacities for things like memory and processing speed. The application contains dependencies to these services and capacities. This information is then used by the SCA Core Framework to deploy an application to a platform dynamically during system startup. One application can be deployed in multiple different ways depending on the platform and on its remaining resources at deployment time.

Software development for a waveform typically starts with a design model. Once the design is validated for SCA compliance, XML and source code artifacts are created. It is important to keep these artifacts in sync with the model; this can be a tedious exercise if done manually. The process is also error prone due to the amount of coordination required between the artifacts. A build environment is needed so that binaries can be built in a consistent manner. Once the binaries are built and the XML is authored, the artifacts are usually copied to a location known by the target operating environment. The application is installed and an instance is created on the target platform using some form of monitoring interface for the operating environment. Once the application is running, the deployment that occurred is analyzed and it is determined if it is one of the desired deployments.

Software development is usually done in cycles as indicated in Figure 1. One iteration requires the use of several unconnected tools for design, code authoring, compilation, transfer of files to the platform, and instantiation of the application on the platform. Each of these steps requires significant understanding of the underlying technology by the user. This makes each iteration more troublesome than it needs to be.

If users can automate much of the manual effort required for each iteration, then they can iterate through the development cycle more often. This has the benefit of allowing them to experiment and verify different designs at run-time. The ability to build and test the design more often reduces the overall risk.

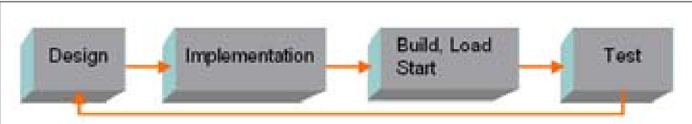


Figure 1: SCA Development Cycle

A convenient, consistent workflow will reduce the development cycle time when working with SCA systems. This workflow must contain intuitive steps for the following activities:

- **Creation** of the design model and validation for SCA compliance. The design model should contain the SCA platform. This allows for validation of the application against the platform so that dependencies and properties are checked and corrected before run-time integration is attempted.
- **Generation** of the Domain Profile (XML descriptor files), source files and a build environment from the design model. This ensures that all artifacts are always in sync.
- **Compilation** of binaries
- **Installation** of the application into the Operating Environment. This includes transfer of the Domain Profile and the binaries to the target.
- **Creation** of the application
- **Monitoring** and **controlling** of the run-time to allow the user to easily compare the run-time system to the design model.

The design model can be made to contain all the information needed for the steps above. This makes it possible to enable users to perform all these steps from a single tool. This is significant as it lessens the complexity, effort and probability of error which encourages the user to try different designs and tests. Convenient workflows promote an increase in development speed and repeatability which in turn encourages designers, developers and testers to investigate different designs and verify these decisions at run-time.

It makes the user more comfortable with the SCA and results in higher quality software, delivered in shorter time with reduced risk.

Modeling an application, a platform and deployments in a design model is important, but it does not answer questions like:

- If the waveform does not deploy, why did this happen? Which components did deploy and which ones did not?
- If a waveform deployed, is it one of the deployments that was modeled, or is it an unexpected deployment?
- Based on the set of known possible deployments, are there some that are considered more optimal than others? Are there deployments that do not meet functional requirements at run-time? Can the set be reduced to only those deployments that are desired using platform independent constructs such as host collocation or dependencies?
- Are there other deployments that might occur when devices are not available? This could be due to other applications running on the platform or devices that suddenly have no allocation resources? Unless every possible deployment is tested, how can the designers be sure that a tested deployment is chosen when the waveform is used in the field?

The concept that is missing here is the link from the design environment to the run-time environment. An SCA developer needs tools that allow for the design, as well as analysis and monitoring of SCA deployments.

The remainder of this paper discusses the importance of reducing the cycle time from design through to run-time. Section 2 reviews some of the concepts in the design model with respect to Applications, Nodes, Platforms and Deployments. Section 3 explains the advantages of monitoring and controlling an application on a platform from a run-time environment that is

similar to the design environment. It describes a typical application on a platform and the different run-time scenarios a user may encounter and workflows associated with these scenarios. Section 4 discusses testing and offers a workflow for using design artifacts during the testing stage. Section 5 reveals how Zeligsoft Component Enabler™ (CE) — the SCA backbone development tool — reduces the turnaround time required for a development cycle which allows more time for testing and reduces the risks that impact cost, time-to-market and software quality. Finally, section 6 summarizes this paper and explains how this technology can be used today to facilitate run-time control, monitoring and testing.

2 SCA DESIGN

This paper assumes the user has an application, a platform and deployments in a design model. Earlier papers describe modeling, validation and generation from a single design model. *Component Enabler Best Practices: SCA* [2] focuses on the description of the different roles involved in a process and on how CE can be used to support these roles. *Developing SCA Compliant Systems* [4] shows how MDA and tool automation can meet the challenges associated with SCA based development.

2.1 Applications

An *application* is a platform independent representation of a software design using components. The application in Figure 2 contains a number of component instances and the connections between them, demonstrating who talks to whom. The application has connections to a Log service on the platform using FindBy ports, and connections to ports on devices on the platform using DeviceThatLoadedThisComponentRef (DTLTCR) ports. For more information on these ports, see *Communicating SCA Architectures by Visualizing SCA Connections* [5]. It describes a simple, natural, graphical notation for depicting all SCA connections and architectures.

Once the application is complete and validated, artifacts can be generated and used by the Core Framework at run-time. *Validating SCA Compliant Systems: SCA Descriptor Validation using Zeligsoft Component Enabler* [3] discusses the need to produce correct and valid descriptor files early in the development process.

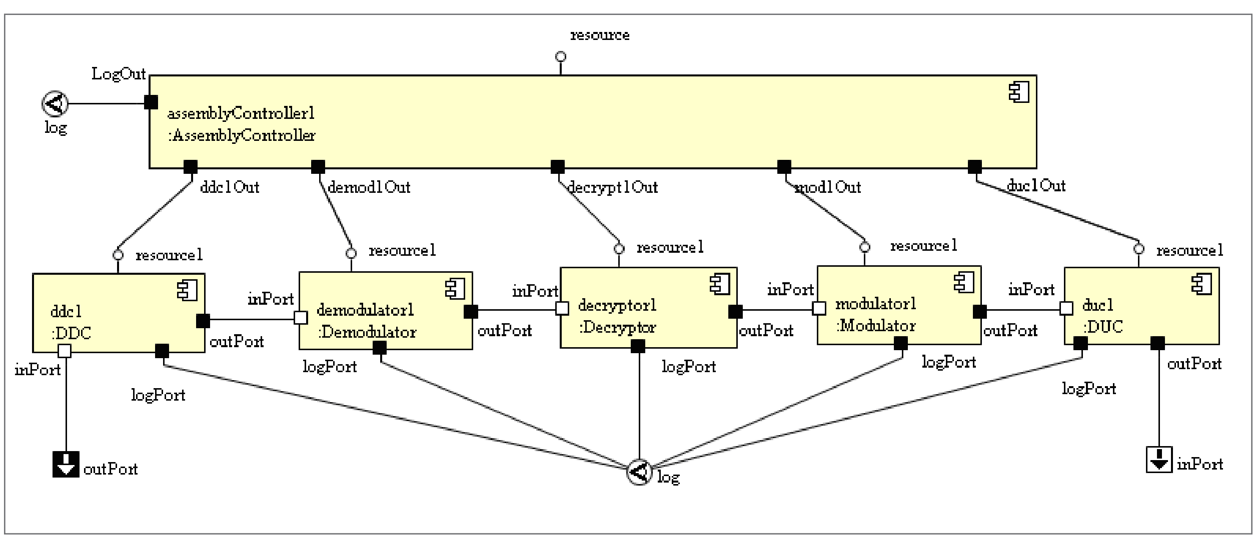


Figure 2: SCA Application

2.2 Platforms

A *platform* is the hardware environment on which applications execute and is comprised of one or more nodes. Connections between nodes indicate which nodes communicate with each other. Each node in Figure 3 represents a board with one or more devices and can be explored in more detail to reveal its devices, connections and so forth, as seen in Figure 4 and Figure 5.

Platforms can be validated and XML descriptor files can be generated for the complete platform. *Validating SCA Compliant Systems: SCA Descriptor Validation using Zeligsoft Component Enabler* [3] discusses the importance of generating correct and valid descriptor files early in the development process.

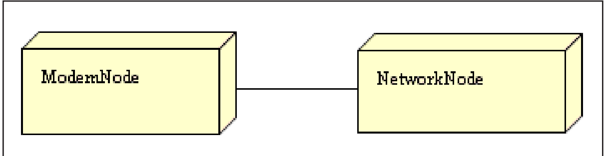


Figure 3: SCA Platform

2.3 Nodes

A *node* maps to a Device Configuration Descriptor (DCD) file in the SCA [1]. The nodes in Figure 4 and Figure 5 have device instances which in conjunction with their device definitions and implementations, represent the actual hardware devices. Connections to the Log and Event services are modeled using FindBy ports and are realized at run-time. Each node contains a device manager which manages its devices.

The Modem Node in Figure 4 contains device instances fpga1 and fpga2 which have ports inPort and outPort that are used to make connections with the application

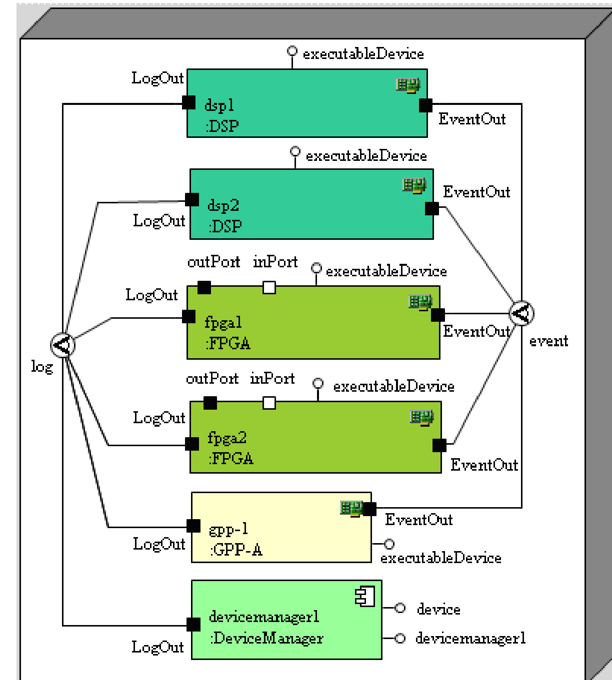


Figure 4: SCA Modem Node

The Network node shown in Figure 5 contains the Log service which components and devices will connect to at run-time. It also contains the domain manager which manages the entire system. It is responsible for the set-up and shut-down of Applications, for allocating Resources, Devices, and non-CORBA components to hardware devices. See the *Software Communication Architecture Specification* [1].

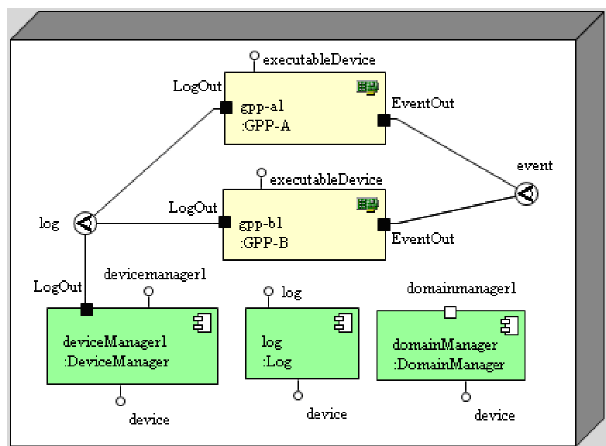


Figure 5: SCA Network Node

2.4 Deployments

A deployment specifies an assignment of the component instances in one or more applications to the device instances in a platform. Figure 6 is a graphical representation of a fully specified deployment model where each component instance has been mapped to a device instance. The icon next to a component instance indicates it has been assigned to a device instance. Another indicator is that the component instance appears under a device instance on the platform. A partially specified deployment is one where not all component instances are mapped to device instances. Below are some definitions of deployments.

- Valid — This is a deployment where all the constraints expressed using SCA constructs are met and therefore it validates in Component Enabler. This type of deployment can occur at run-time.
- Legal – This is a valid deployment that does not violate any of the functional and non-functional requirements of the application. This type of deployment requires manual inspection as well as functional testing to verify that the assignment of component instances to device instances makes sense, and that the waveform behaves as expected.
- Desired — This is a legal deployment that is defined by the user as a desired deployment in the model.

Before analysis, the set of valid deployments will be greater than or equivalent to the set of legal deployments which will be greater than or equivalent to the set of desired deployments. After analysis all three sets should be equivalent.

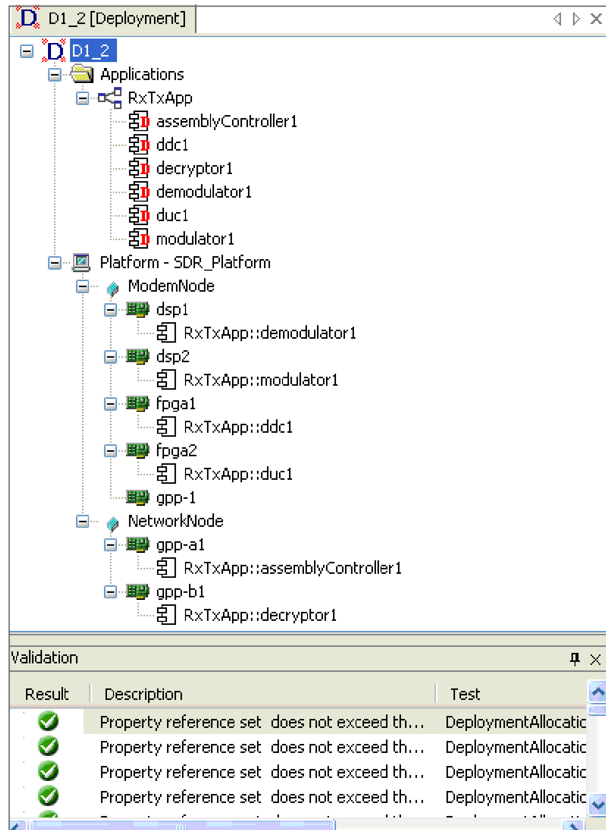


Figure 6: Valid Deployment model

2.4.1 Deployment Modeling and Validation

Modeling a deployment is a very powerful design capability. Validation of the deployment will verify that the assignment of the applications to the platform is SCA compliant. It will verify that all dependencies, relationships, connections and host collocations are satisfied so that integration issues are eliminated at design time rather than uncovered at run-time.

Before attempting run-time deployment of the deployment model specified in Figure 6 it is important to verify that all the constraints have been met. If the validation pane shows a green check mark for every test, then the deployment is valid. For more information on deployments see *SCA Deployment Management: Bridging the Gap in SCA Development* [6].

2.4.2 Deployment Calculation and Analysis

The ability to calculate the set of valid deployments reveals a lot of information at design time. If the resultant set of valid deployments is too large, it may mean the user has under-specified properties and relationships and there may be many that are valid but not legal with respect to requirements. If the set of valid deployments is too small, it could reveal the user has over-specified the constraints and therefore the application may not be very portable.

Calculating deployments is an exponential problem and cannot be done manually unless the input is very simple. For example, if every component instance in Figure 6 has one implementation then potentially each of the six component instances could be deployed to one of the seven device instances. This implies there are 7^6 possible run-time deployments. The problem becomes even more unmanageable if there is more than one implementation per component.

3 RUNTIME CONTROL AND MONITORING

So far we have looked at deployment modeling at design time. We will now investigate the different activities related to using these design time deployments on actual platforms at run-time.

The user creates a design model that describes the application and the platform. This design can be used to model, validate, and generate the implementations for the application. The design can also be used to monitor and control the implementations on an SCA compliant platform. *Controlling* an SCA application means installing, instantiating, starting, stopping, shutting down and uninstalling an application as well as modifying any properties of the application at run-time. *Monitoring* an SCA application means observing how the application is actually deployed and how it relates to the desired deployments modeled at design time.

An *actual deployment* is the assignment of an application's components that are currently running on devices on the platform. A *desired deployment* is a deployment the user expects to occur at run-time. This may not be the same as the current actual deployment. A typical model has multiple valid deployments. An application is modeled in a platform independent manner so that a component instance may potentially be assigned to any device instance on the platform. Valid assignments are constrained by implementation dependencies, connections and host collocations.

3.1 Control

Users need the ability to install an application in a seamless manner.

- In order to do this, the necessary artifacts must be copied to the desired location.
- Once the artifacts are in place, the application can then be created.
- When the application is running, it can be monitored and controlled through some form of a user interface.
- The application can also be started and stopped and uninstalled so that a new application can be installed or the user can return to an initial installed state for the existing application.

These workflows require multiple different steps and the user has to be intimately familiar with the working of the platform. Controlling of the platform and the application can be done through graphical tools. These tools can enable the user to perform all the steps in the workflow through intuitive context-sensitive menus; this will shorten the development iteration time. Ease of use facilitates experimentation with different architectures and properties to determine optimal designs.

3.1.1 Installing Applications

Installing an application verifies that the application's Software Assembly Descriptor file exists in the DomainManager's FileManager and that the files the application is dependent on also exists. This includes XML descriptor files and binaries referenced from the application. Automating these actions simplifies the development cycle workflow and reduces the probability of human error.

Figure 7 shows a monitor M1 without any installed applications in the Actual Deployment. The desired deployment D1_2 has been added to the monitor. The platform is running as indicated by the icon beside "Platform – Custom". The icon adjacent to the RxTxApp indicates the application is not installed. The icons indicate the allocation of component instances to device instances does not match the Actual Deployment (since the application has not been created yet). The desired deployment can be used to install the application using the DomainManager's installApplication() method which is found in the Software Communication Architecture Specification [1].

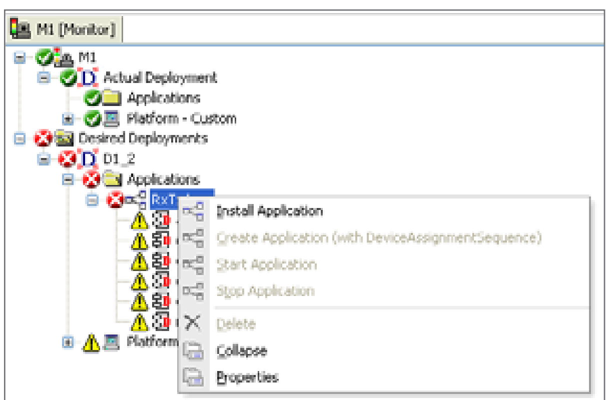


Figure 7: An Application not yet installed

Once installed, the application also appears in the Actual Deployment as shown in Figure 8. The icons still exist since the application is not created yet. The application can now be uninstalled using context menus. This intuitive workflow for installing and uninstalling an application, and the automation of copying artifacts to the required location, reduces the time necessary to set up for a test. The workflow thus reduces the overall time necessary for the software development cycle shown in Figure 1.

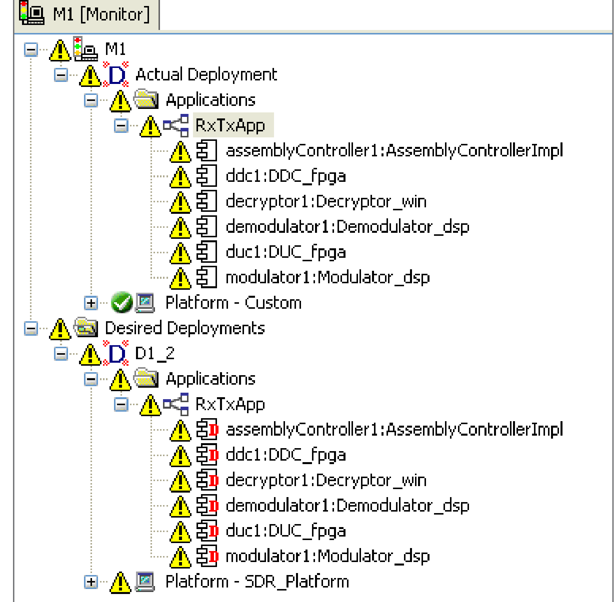


Figure 8: Installed Application

3.1.2 Creating Applications

Application deployment is controlled through the SCA Core Framework. XML descriptor files are parsed and based on the constraints specified, the binaries referenced in the descriptor files are loaded on the appropriate devices. The application is created using the ApplicationFactory's create() method specified in the Software Communication Architecture Specification [1].

Creating an application from the monitor in Figure 9 involves right clicking on an application in the Actual Deployment and selecting Create Application.

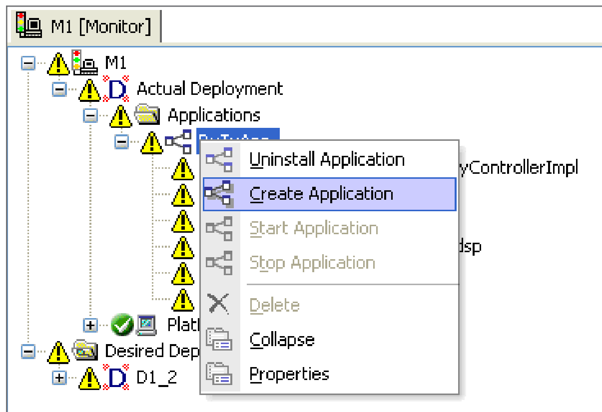


Figure 9: Creating an application

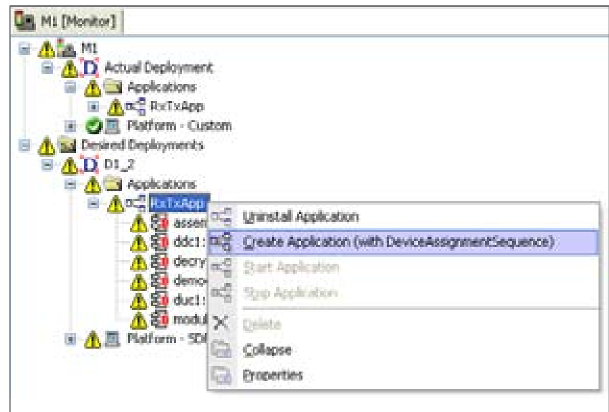


Figure 10: Desired deployment used to create an application

3.1.3 Creating Applications by enforcing Desired Deployments

Enforcing desired deployments is a powerful feature when users want a particular desired deployment to be instantiated on the running system. Enforcing a deployment involves creating a DeviceAssignmentSequence which specifies an assignment of component instances to device instances using their UUID's. The sequence is then passed as a parameter to the ApplicationFactory's create() method as specified in the Software Communication Architecture Specification [1]. This instructs the Core Framework to deploy the application exactly as described in the desired deployment model.

Creating DeviceAssignmentSequences manually for each desired deployment is time consuming and potentially error prone. If deployment models exist in the tool, then the creation of these sequences can be automated which further reduces the development cycle time specified in Figure 1.

Figure 10 is a run-time monitor. The application is installed and the actual deployment can be forced to match a desired deployment by selecting the desired deployment's applications and choosing Create Application (with DeviceAssignmentSequence).

3.1.4 Controlling Deployed Applications

Once the application is created, the user needs the ability to control it. Control includes operations such as starting and stopping the application. When an application has been tested or debugged, the inverses of the create and install operations will be required and are straightforward. Releasing the application removes its connections, terminates the application, returns all the allocated resources, and de-allocates the component's capabilities in use by the devices. Uninstalling the application first releases it, and then uninstalls it.

Figure 11 displays a created application where the icons indicate the Actual Deployment matches the Desired Deployment D1_2. Users can control the application by selecting the application and using its context sensitive menu.

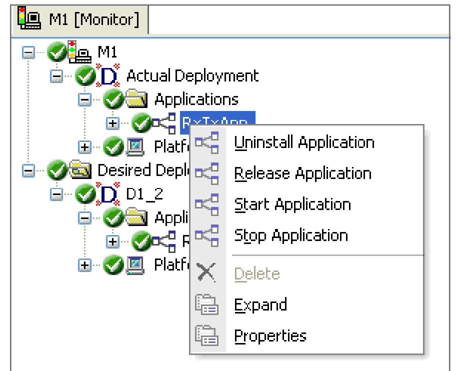


Figure 11: Controlling an application

3.1.5 Querying and Changing Properties

Users need the ability to query and change run-time properties on applications, nodes, components and devices. If this process is easy and intuitive, it will encourage them to change run-time property values and determine their effects on resulting behavior. Figure 12 displays a dialog displaying run-time properties for a device instance which may be changed during run-time monitoring and analysis.

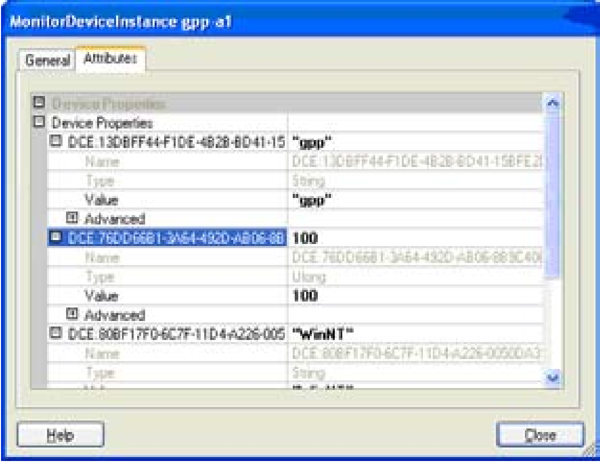


Figure 12: Run-time device properties

3.2 Monitoring and Analysis

Once the actual deployment is running, the next problem is observing what has happened. Did the deployment happen in the way the user expected? Automatic comparison and icons provide immediate correctness feedback to the user. A graphical representation of the running system allows the user to monitor which component implementations are running on which devices. However visual inspection is time consuming, non-scaleable and error-prone. If monitoring and analysis is done from the same tool as the design, then the tool can automate the comparison of the actual deployment (run-time) to the desired deployments (design).

3.2.1 Monitor Log Window

If the user's Core Framework contains a Log service and a node on the platform contains a reference to the Log service as in Figure 5, then SCA log messages will appear in the monitor's log window as long as component instances are connected to the service.

Log messages are useful additional information for debugging the run-time deployment and any problems that might have occurred. It contains information such as connection status between ports in the application and connections between the application and platform.

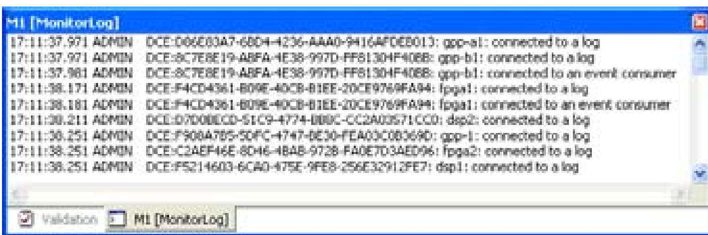


Figure 13: Monitor log window

3.2.2 Run-time Analysis

If an application is running on a platform within the same tool where the design model exists, then comparisons can be automated facilitating answers to important questions. Does a desired deployment match the actual run-time deployment? Is there an actual deployment that does not match any desired deployments?

The possible outcomes for an actual deployment are listed below. We will look at these scenarios in more detail.

- The application fails to deploy on the platform.
- The actual deployment matches a desired deployment
- The actual deployment matches a deployment other than the desired deployment
- The actual deployment does not match any of the modeled deployments and is therefore unexpected.
- The actual deployment matches several modeled deployments.

3.2.2.1 The application fails to deploy on the platform

When the Core Framework cannot allocate component instances to device instances such that all constraints are met, the application fails to deploy on the platform. The user must determine why the run-time deployment failed. This investigation can be a long and tedious process. Log messages are inspected for information related to the failure. The status of the platform has to be verified. The design model is scrutinized for component dependencies that exceed the allocation properties on devices. Relationships, connections to the platform and host collocation all have to be studied.

Deployment modeling and validation described in section 2.4.1 demonstrates how deployment validation can inform the user whether a deployment model is a valid one or not.

3.2.2.2 Actual deployment matches the desired deployment

Figure 14 shows a run-time monitor where the actual deployment matches a desired deployment as indicated by the icons. Each component instance has been allocated to a device instance at run-time matching the desired deployment in the Run-time monitor. This is the expected behavior when the user forces a particular valid deployment. This could also occur if the Core Framework chooses a run-time deployment that matches one of the valid deployments modeled in the tool. Since the run-time monitor contains the desired deployments and the actual deployment, the comparison between the two can be automated, increasing efficiency and removing human error. Once the user determines the actual deployment is also the desired deployment, run-time testing can begin.

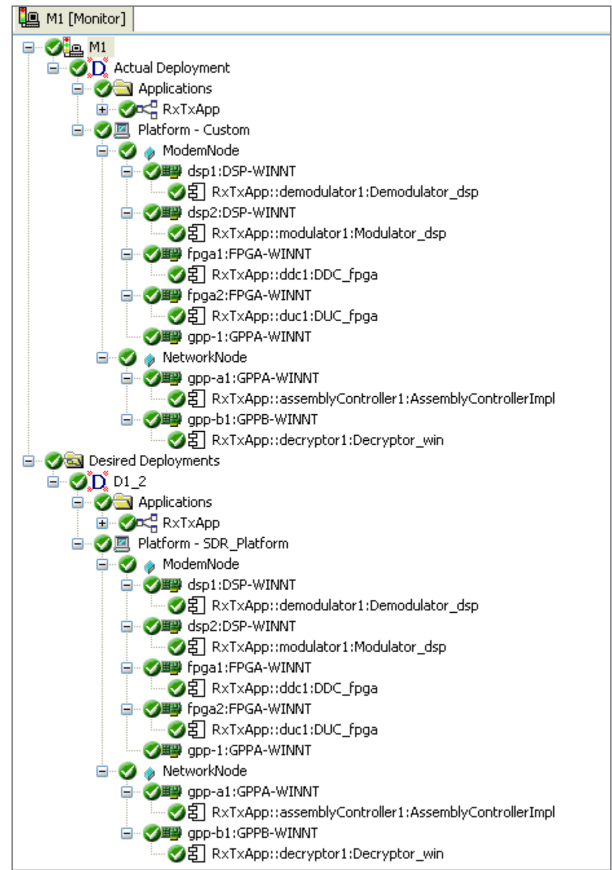


Figure 14: Actual deployment matches desired deployment

3.2.2.3 Actual deployment does not match the desired deployment

A more interesting scenario is when the actual run-time deployment does not match what the user expected. When a deployment has occurred outside the expected set of deployments, this is potentially a serious problem. It means the component allocation as a whole is unspecified. The developer must determine why this happened.

Because the design and run-time artifacts are in the same tool, the user can quickly switch to deployment modeling and recalculate all the possible deployments. If the actual deployment matches one of the newly calculated deployments, then the issue was that the valid deployment was not modeled. Since it is now modeled and considered known, the run-time analysis and testing can continue to determine if the new valid deployment is a legal and desired deployment. Unexpected run-time deployments need to be eliminated so that all possible deployments are modeled, documented and tested, so there are no surprises.

In Figure 15, automatic comparison of the actual deployment to desired deployment indicates that something is not as expected. Since the icons indicating a problem are propagated to the root

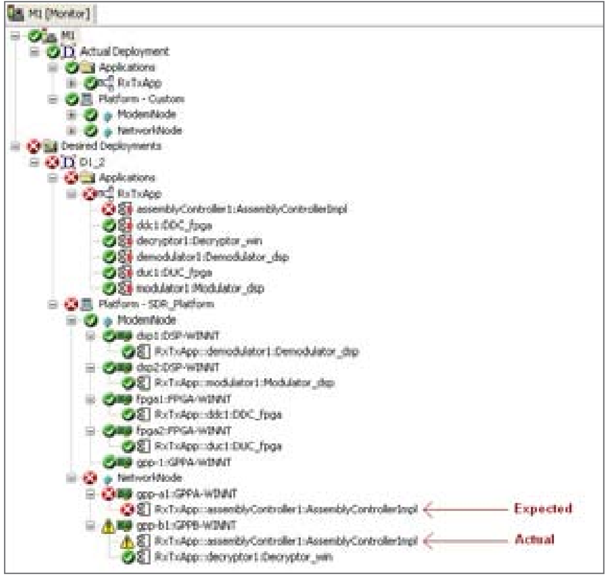


Figure 15: Actual deployment does not match the desired deployment

"Desired Deployments", it is obvious there is an issue even if the information is collapsed.

3.2.2.4 Comparing several desired deployments

Several desired deployments can be compared to the actual deployment in the same run-time monitor. Figure 16 shows that desired deployment D1_2 matches the actual deployment since it displays the icon. The other desired deployments do not match the actual deployment as indicated by the icon. If all desired deployments are fully specified, then at most one can match the actual deployment.

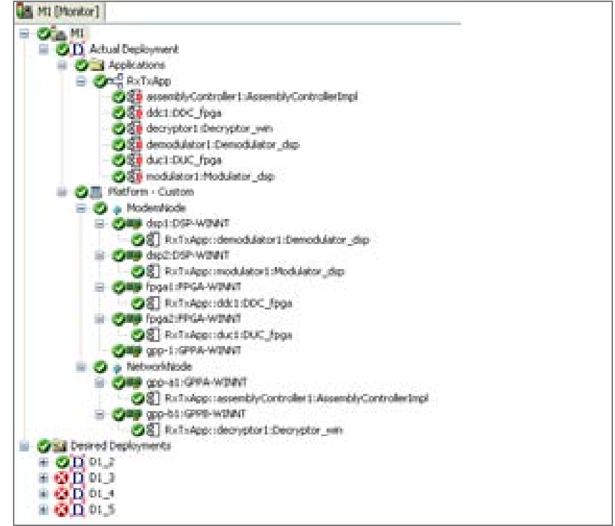


Figure 16: Actual deployment matches one desired deployment

The actual deployment can match several desired deployments if one or more of these deployments are partially specified. In this case the user would see two or more desired deployments where everything matches.

The automation of these comparisons increases efficiency, removes human error and allows the user to focus on the results of the comparisons rather on performing the comparisons manually. It also simplifies deployment visualization for users who are not experts in the SCA.

4 TESTING

Testing is important and can be a labour intensive stage in the development cycle illustrated in Figure 1. If testers can reduce the time and effort necessary to set up, perform and analyze a particular test, then the time for an iteration can be reduced. Reduced iteration time allows for more iterations and therefore more complete testing can be performed, reducing risk.

4.1 Testing Individual Desired Deployments

Testing begins with the set of all valid deployments that are legal with respect to non-functional requirements. That is, they have been validated and then inspected by a domain expert to verify the allocation of component instances to devices instances and do not violate any non-functional requirements, such as performance or security. All valid deployments that are not considered legal at this point should be removed from the set of valid deployments, using SCA and user defined constraints.

One by one, the tester must enforce each desired deployment on the platform at run-time and then monitor and evaluate it to verify that it meets functional requirements. If a deployment satisfies functional requirements it can be declared a legal deployment. If it does not, it should be made an invalid deployment by adding constraints. Optimal legal deployments should be considered desired deployments. Sub-optimal legal deployments should be made invalid and therefore illegal using constraints.

The goal of development is to determine a set of desired deployments and make that set equivalent to the set of legal deployments which should be equivalent to the set of valid deployments.

4.2 Situational Testing

Once the testing is considered complete by enforcing deployments, the next step is to let the Core Framework choose deployments under different conditions. Questions that should be considered are:

- If the application is the first one deployed on a platform, is the actual deployment one from the set of desired deployments? If not, the valid deployment set is different than the desired deployment set. The new deployment should be tested and either removed from the valid set using constraints or made legal and added to the desired deployment set.
- What is the actual deployment when an application deployment is attempted after one or more applications are deployed on the platform? The result should either be a failure to deploy, or the actual deployment should again be a desired deployment.
- What is the actual deployment when one or more devices are effectively disabled by reducing its allocation properties to zero or setting its state to busy? If the set of desired deployments are known in advance and are equivalent to the set of valid deployments, then the actual deployment should be one from this set.

5 DEVELOPMENT CYCLE REDUCTION

Zeligsoft CE significantly reduces the time and effort required for each iteration of the software development cycle in Figure 1. It is the backbone of the SCA development process and it guides the user from the initial design through to the testing phase. Zeligsoft CE provides for a systematic approach to SCA software development, thereby reducing risk.

When designing with Zeligsoft CE, models of the complete domain can be validated at any time. Constraints between applications and a platform can be verified at design rather than at integration time, reducing the

risk and cost of integration. Valid deployments can be inspected at design time to verify whether they meet the non-function requirements or not.

During the implementation stage, the complete domain profile (XML Descriptor files) can be generated from Zeligsoft CE at the push of a button. SCA wrapper code for components and a complete build environment can be generated using flexible and customizable templates. Since the model is the source, the SCA, source code and build artifacts are always synchronized with each other. Smart generation guarantees user code will not be overwritten.

Compiling and linking binaries for an application can be initiated from Zeligsoft CE with build output generated into its output window. Build avoidance ensures only the required binaries are rebuilt. The advanced scripting feature of Zeligsoft CE can automate the transfer of the Domain Profile and the binaries to the target. Once they are in place, context menus are used to install and uninstall the application.

Setting up test environments is simple using Zeligsoft CE. Creation of an application where the desired deployment matches the actual deployment is as straightforward as using the context menu from the application in the run-time monitor's desired deployment. Once the application is created, it can be started and stopped or uninstalled from Zeligsoft CE. Properties can be viewed or changed, and if the Log service is available, the monitor's log window can be viewed.

Comparison of the actual deployment to desired deployments is automated, increasing efficiency and removing human error. Icons quickly indicate where the differences are.

6 SUMMARY

In SCA development to date the turnaround time to complete an iteration from design through to testing has been too long and difficult. There has been a disconnect between design artifacts and run-time analysis. Run-time integration is expensive and time consuming. Design tools are put aside and run-time tools are used during testing. Comparing the run-time environment to the intended design required manual inspection of run-time logs and comparison of UUIDs, which is painstaking, error prone and non-scaleable.

Modern software engineering practices can change that. Model driven workflows allow the user to express the system from early design all the way through to implementation, unit test and system test. Models are useful since they allow the entire team to understand the solution space. Furthermore, models can be used to generate artifacts, and monitor and control execution.

Zeligsoft CE closes the development loop by guiding the user from the design through to run-time testing. Elements created in the CE model are used at run-time to validate the design. The run-time monitor resembles the deployment model and therefore it is easy for the user to view the run-time results. Since the design and run-time artifacts are both in CE, the actual deployment and the desired deployment can be automatically compared. Deployment models can be added to the run-time monitor for automatic differences analysis. Run-time icons quickly alert the user of any disagreements between what is running and what the user expects. Unexpected run-time deployments can be quickly identified and resolved. The run-time environment can be analyzed and controlled from CE. Applications can be loaded, created and unloaded on the platform. Properties can be queried and configured and log messages can be inspected in the run-time environment.

Because CE is used throughout the SCA process it is easy to switch back to the design environment, and make and validate changes to the model. Artifacts can be recreated to reflect the changes and loaded on the platform for run-time testing. This link between the design and run-time environments allows the user to model a little, build a little and test a little. It creates a repeatable iterative process that encourages experimentation, reduces risk and manages design and operating environment complexity.

For more information on how CE can benefit your SCA project go to www.zeligsoft.com or contact sales@zeligsoft.com.

7 REFERENCES

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

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

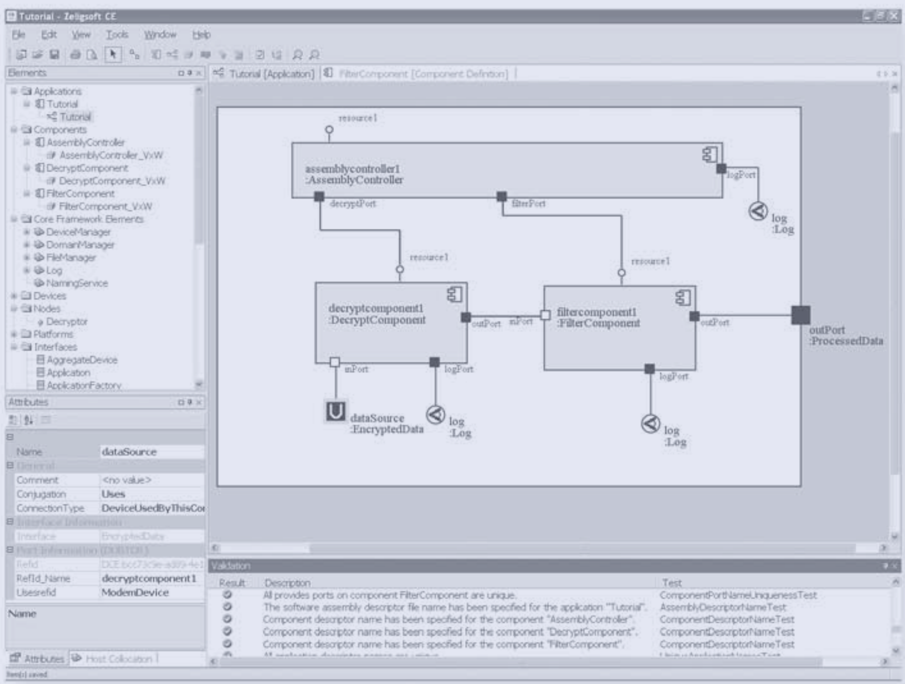
3. **Zeligsoft**
   Mark Hermeling and Francis Bordeleau, *Validating SCA Compliant Systems*
   http://www.zeligsoft.com/Technology/Resources.asp

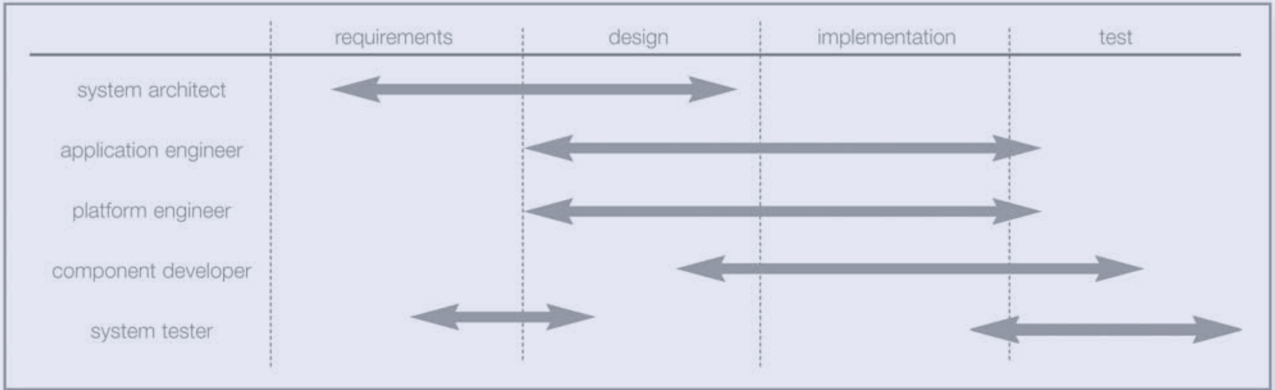
4. **Zeligsoft**
   Mark Hermeling, John Hogg and Francis Bordeleau, *Developing SCA Compliant Systems*
   http://www.zeligsoft.com/Technology/Resources.asp

5. **Zeligsoft**
   John Hogg, *Communicating SCA Architectures by Visualizing SCA Connections*
   http://www.zeligsoft.com/Technology/Resources.asp

6. **Zeligsoft**
   John Hogg and Francis Bordeleau, *SCA Deployment Management: Bridging the Gap in SCA Development, 2005*.
   http://www.zeligsoft.com/Technology/Resources.asp

zeligsoft


Tutorial - Zeligsoft CE
File Edit View Tools Window Help
Elements
Tutorial [Application]
FilterComponent [Component Definition]
resource1
assemblycontroller1
:AssemblyController
logPort
decryptPort
filterPort
log
:Log
resource1
resource1
decryptcomponent1
:DecryptComponent
outPort
inPort
filtercomponent1
:FilterComponent
outPort
outPort
:ProcessedData
inPort
logPort
logPort
dataSource
:EncryptedData
log
:Log
log
:Log
Attributes
Name
dataSource
Comment
<no value>
Conjugation
Uses
ConnectionType
Refid_Name
decryptcomponent1
Usesrefid
ModemDevice
Validation
Result
Description
Test


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

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

Zeligsoft, Zeligsoft CE and Waveform Builder are trademarks of Zeligsoft Inc.
All other trademarks are the property of their respective companies.
Version 1.1
Printed in Canada © Copyright 2005.