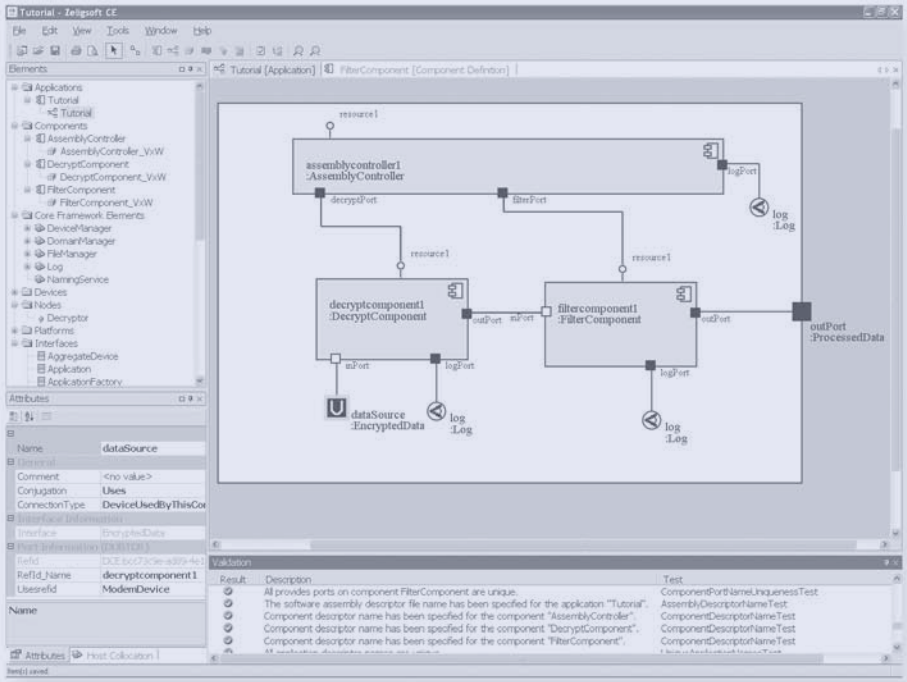
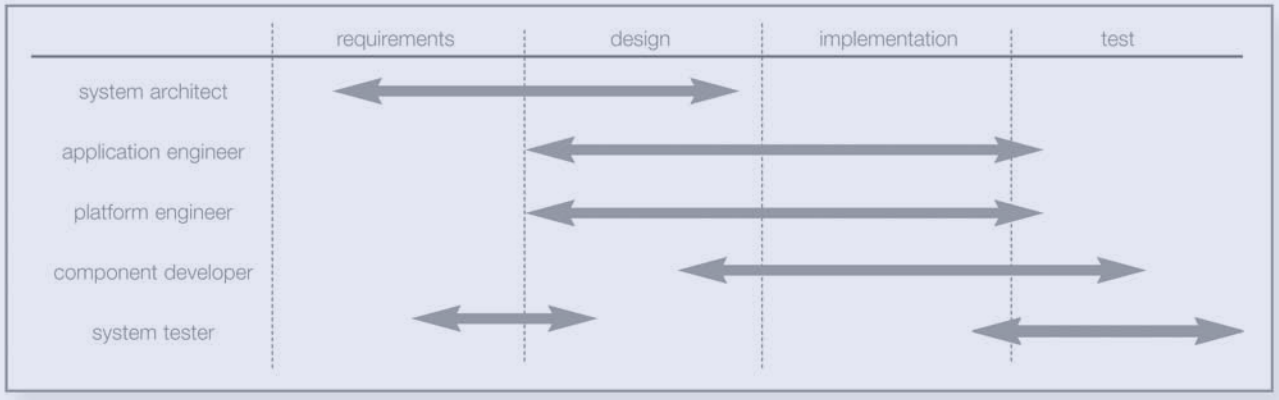
Developing SCA Compliant Systems

Mark Hermeling, 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

Developing SCA Compliant Systems

Mark Hermeling, John Hogg, Francis Bordeleau

Component-based software development, deployment and configuration for Software Defined Radio using the Software Communications Architecture (SCA) has seen an increased interest in the past few years. Companies are actively starting to build more and more radios that adhere to the SCA standard. Platform vendors are also incorporating the standard into their portfolio.

This paper provides an introduction to the SCA and highlights the goals it is trying to achieve and the technology it uses. The technology puts many challenges in front of the developers of systems. This paper highlights these challenges and their inherent pitfalls.

Model Driven Architecture (MDA) is an initiative from the Object Management Group. MDA introduces models as the primary artifact of development and promotes modeling in favor of coding. This paper shows how MDA and tool automation can meet the challenges associated with SCA based development.

1 INTRODUCTION

The United States Department of Defense (DoD) requires that all radios delivered to any of the armed forces adhere to the Software Communications Architecture (SCA) **[SCA]** standard. The vision of the DoD is to develop a state-of-the-art, economical communications facility for the armed forces that will allow all branches to cooperate.

The main goal of the SCA is to reduce cost and increase interoperability. Software Defined Radios (SDRs) developed using the SCA can be upgraded and extended both from the hardware and the software side of the radio.

This is very beneficial for the DoD, the user of the radios. It also benefits the producers of the radio, who have a stable, proven standard to build their systems with and can readily depend on existing tools and platforms. However, some extra work is required to make the radios compliant with the standard.

This paper introduces the technology that the SCA uses at a high-level. It covers the responsibilities of the developers of the radio. It highlights what is needed to make the radios compliant and some of the bottlenecks and difficulties that can arise in the development process. It also introduces ways to overcome these difficulties by using Model Driven Architecture (MDA) techniques and tool-driven automation.

2 TECHNOLOGY

The SCA addresses component-based systems that require a high-level of platform independence. These systems need a way to query available hardware (devices) in order to deploy needed resources (software). This requires much coordination between the software and the hardware.

The SCA introduces

- A separation between application (software) and platform (hardware drivers) based on componentization of the entire system
- A Component Framework that addresses coordination
- An API that allows for standardized communication between the Core Framework, components, and devices
- A descriptive language that describes the application and the platform, and their relationships using XML
- A communication layer that allows for seamless communication
- An operating system standard that provides services at the lowest level

This paper looks at each of these individually and introduces their significance in the larger picture. The paper concludes with a high-level overview of how an SCA system is deployed during run-time.

2.1 Application and Platform

The SCA makes a distinction between application and platform, software, and hardware drivers respectively.

The application is defined by individual components. Each component provides a certain amount of functionality. Interconnecting the components into an application comprises the complete system functionality.

A component has a well-defined interface that specifies what services it provides and what services it requires from other components. Connections between components are considered to be part of the application and are created dynamically during deployment.

The platform defines the hardware that the components can use. The platform is built through drivers, which are software representations of the real hardware elements. The SCA refers to these drivers as devices.

Components may have requirements; for example, a certain component may require 10 Mb of RAM or 5 MIPS of processing power. Alternatively, a component can request a specific type of Field-Programmable Gate Array (FPGA) or general purpose processor.

The goal here is independence. It should be possible to run an application on multiple platforms, as long as the platforms fulfill the application requirements. Alternatively, it should be possible to run multiple applications on one platform.

The components and platform together form a high-level, implementation-independent description of the system. They describe what the individual pieces of the system are, rather than how the system works in detail. This system description is used by the Component Framework to load components onto devices and execute the system. See section 2.3 for more information.

2.2 XML Descriptors

Components, devices, applications, and platforms are described using a set of XML descriptor files that have a well-defined syntax. These files contain information about the external interfaces of the components and devices, their implementations, and how they are connected to form applications and platforms. In short, the descriptor files contain all the information that is required to deploy and configure an application on a platform.

The complete set of descriptor files is referred to as the Domain Profile, which consists of the set that describes the application (the Software Profile), and the set that describes the platform (the Device Profile). The Domain Profile is a key factor in the deployment and configuration of a component-based system.

The Domain Profile contains a large amount of information. Large systems can have tens of thousands of lines of XML code spread over hundreds of files.

2.3 Component Framework

The Component Framework is responsible for deploying and configuring applications, and for managing the application and the platform. It reads the information in the Domain Profile at startup and tries to deploy the application to the platform, while adhering to the relations in the Domain Profile.

In short, the Core Framework knows how to download a component to a device, connect components together so that they can communicate, start and stop the component, and deal with error conditions. It takes care of management tasks so that the components themselves do not have to deal with them.

The Component Framework is referred to by the SCA as the Core Framework (CF) and is the core part that guides the run-time deployment of a system.

2.4 API

The CF needs to communicate with all the elements in a system – both application and platform. To do this, certain basic interfaces have been prescribed. The SCA specifies the interfaces that a component and a device are required to implement.

The interfaces cover aspects of dynamic deployment, such as where the Core Framework queries the platform to see which components can be loaded on which device. The interfaces are also used to connect component instances together during runtime, to start and stop them, and to do some event handling and logging.

One of the goals of the SCA is also to increase inter-operability between components. To achieve this, certain domain-specific APIs have been designed. The domain- specificinterfaces deal with concepts such as antenna configuration, sending and receiving data streams, and so forth.

2.5 Communication Layer

The communication layer provides abstraction from the physical act of communication to the Core Framework, the devices, and the components. Software Radios frequently consist of multiple physical processing devices that provide processing functionality. Communication needs to be possible between components and devices that reside on the same or separate processors.

The SCA has chosen the industry proven CORBA **[CORBA]** communication layer to provide the messaging. Every SCA compliant radio has a CORBA compliant ORB incorporated into it.

2.6 Operating System

The operating system is the lowest layer of software abstraction on most embedded systems. It harnesses the power of the processor and provides services to the processes running on top of it. The operating system provides the lowest level of hardware abstraction and provides services for task switching, semaphores, memory management, and so forth.

The SCA states that the operating system needs to be POSIX compliant. This means that software that was written for operating system A can be recompiled and run on operating system B.

3 CHALLENGES

The SCA standard provides a number of benefits to both the user and the developer of radio systems. However, complying with the standard is not a trivial affair. This section introduces a number of challenges that need to be met when building SCA compliant systems.

3.1 Architecture

The system architecture details who talks to whom and what they say to one another. It describes how the system is divided into an application and a platform. It describes how the components in the application are connected to each other and to the devices in the platform. It also describes the platform with the connections between the devices.

The architecture is defined during high-level system design. The architect team divides the system into components, which are individually handed off to separate teams for construction.

An SCA based system is like any system; however, there are a number of areas related to the system architecture where some difficulties arise – most notably in the areas of visualization, platform independence, and correctness.

Visualization

The system architecture needs to be communicated within the development team. This is best done through diagrams. Diagrams can be made on white boards, or with Microsoft Visio, PowerPoint, or modeling tools.

The SCA domain differs in that none of the tools mentioned above allow users to add SCA specific detail to the architecture. Elements such as component instances, assembly controllers, ports and interfaces, connection attributes, and many other properties cannot be set or directly indicated.

An architecture without these SCA specific properties is incomplete and could potentially lead to misunderstandings amongst development teams.

Platform Independence

The SCA goes to great lengths to provide platform-independent modeling. This means that it is possible to define an application and a platform independent of each other. A component in the application can then have relationships with certain aspects it requires from a platform. Multiple platforms might satisfy these relationships.

Expressing and visualizing platform-independent relationships correctly requires detailed knowledge of the SCA.

Correctness and Compliance

Once the architecture has been specified, it needs to be analyzed for correctness and compliance. For example, take a connection between components in the application. The ports on the components on both ends of the connection need to have compatible interfaces. Another example is ensuring that all the SCA required data has been provided, including properties such as the CORBA version and the assembly controller setting for the application.

3.2 Descriptors

An SCA compliant system is described by a set of XML descriptor files. These files provide detailed information about each of the components and devices, and how they are connected to form the application and platform. These descriptor files contain many references between components and devices and are rather complex to write.

As an example, the application starts with the Software Assembly Descriptor (SAD), which specifies component instances that are each described in their own component description files. The connections in the SAD refer to properties on components that refer to properties on devices, which are contained in the device descriptors. Collectively, the set of descriptor files can contain over a hundred files and reach sizes into the tens of thousands of lines.

The challenge with the descriptor files lies in the complex syntax and the cross-references between files. Authoring these files is a very time-consuming and error-prone activity.

3.3 Integration

One of the goals of the SCA is to enable component sharing between projects, including COTS (Commercial Off-the-Shelf) applications, components, devices, and platforms. A COTS component is delivered as a number of executables, together with the XML files that describe the executables.

Integration of COTS components into an application requires users to understand the interface of the component and the different uses that might be made of the component. This requires a thorough reading and understanding of the XML files. As indicated in section 3.2, this is not a trivial exercise.

3.4 CONCLUSION

Most of the challenges that the SCA poses can be compared to the challenges faced with writing source code. We need to model the source code to understand how the different source modules are related. We need compilers to verify integrity and to ensure one source module makes correct use of another.

These challenges increase the chance of costly errors that can linger in the software development cycle and only become apparent when the system is tested. Some examples are integration problems due to incorrect interfaces on components or incorrect descriptor files. Some of these problems can be costly to find — debugging embedded systems is an expensive and resource- consuming activity.

4 SOLUTION

The challenges of building SCA compliant systems can be met through visual modeling and automation. Visual modeling has been gaining popularity in recent years. Combined with automation, visual modeling can provide significant benefits, including support for engineers building SCA compliant systems.

Visual modeling displays information about the components and devices that users are working with in a visual paradigm. Automation means that tools perform the work. Users perform less manual work in areas such as content validation of the model or generation of the descriptors.

4.1Model

Visual depictions of components, devices, and the connections between them increase the engineer's understanding of the system. They prevent misunderstanding and improve communication.

The diagrams can also be used as a basis for system documentation.

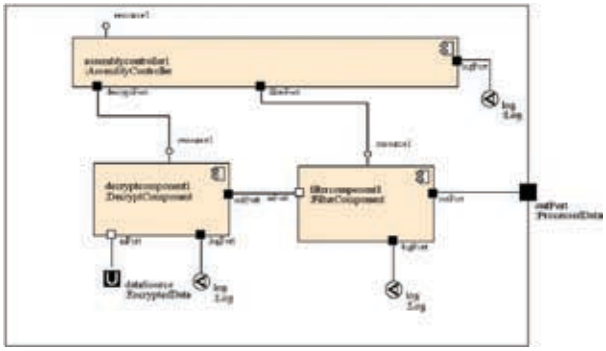


Figure 1: Visual model of an application

Figure 1 shows a model of an application. The diagram combines information from about ten different XML files. The diagram allows an engineer to understand the content of the application with minimal effort and minimal time.

However, modeling is more then just diagrams. A good visual modeling tool contains a related hyperlinked set of the elements that together contain all the information needed for the entire system. This hyperlinked set of elements is usually referred to as the model. The fact that the model is hyperlinked allows for easy navigation through the model space. Users can navigate through diagrams and relationships. Easy workflows let users locate all the information that comprises the model space.

4.2 Validate

Once the model is created, it needs to be validated. As previously stated, the SCA standard defines many rules and regulations that the system needs to meet. Examples of these regulations include attributes (which have to be set to one of several allowable values), and application-to-platform relationship evaluation.

Validation is an important aspect of modeling. It is comparable to syntax checking source code. A model with errors is like a piece of C code that does not correctly compile. It means that the system is not correctly described. Further work with the model can lead to undefined behavior, for example, during download and execution. These undefined behaviors can be very difficult to find during testing.

Furthermore, a model that does not correctly validate does not adhere to the standard. This means that the system might not pass certification test by JTEL, the JTRS certification authority [JTRS].

4.3 Generation

Generation is a direct extension of validation. A model should contain all the information that is available about the SCA aspects of the system. With this information present, validation provides a useful statement on the compliancy of the system. The information can also be used to automatically generate correct-by-construction XML descriptors for the system.

Generation removes the tedious and error-prone step of manually writing the XML descriptors.

5 ZELIGSOFT CE

Zeligsoft Component Enabler [ZSFT] is a visual modeling tool that provides the facilities for users to conveniently model, validate, and generate SCA compliant systems. It provides a user-friendly, model-based workflow to define components and applications, devices and platforms.

The next section walks through some of the features and benefits of CE.

5.1 User-Friendly Interface

Users of a model-based development tool like CE spend substantial time navigating through dialogs, windows, diagrams, and menus. The CE User Interface allows for simple navigation. It puts all the functionality and power of tool automation at the user's finger tips.

5.2 UML-Based Approach

The modeling paradigm behind CE is based on version 2.0 of the industry standard Unified Modeling Language [UML]. UML 2.0 offers platform-independent modeling and structure diagrams that show run-time communication links between instances.

Many software developers are familiar with the UML and hence are able to use CE with a minimal learning curve.

5.3 Provides SCA guidance

The SCA is a standard with many rules that are not necessarily intuitive, especially to new users. A thorough understanding of the standard is needed to design SCA-compliant applications and platforms, and to be able to write the correct set of XML descriptors that specify them. CE removes the need to be an XML expert through its validation and generation feature. The tool guides users through the pitfalls of the standard and highlights elements that are missing or in violation of the standard.

Once the model is defined, the generation feature generates the set of required XML descriptor files. The generated descriptor files are correct-by-construction, which removes costly debugging sessions resulting from typing errors or misunderstandings of the standard.

5.4 XML Import

Integration of COTS or legacy components is an important part of the construction of an SCA compliantsystem. CE can import the XML descriptor files from COTS or existing SCA components. CE offers drag-and-drop functionality to bring the component into the system. An added benefit of the XML Import feature is that the XML is validated. Any potential errors in syntax or semantics of the XML are highlighted.

5.5 Documentation Generation

SCA based systems require thorough documentation of the application, platform, components, and devices. The model already contains all the information about the system. CE allows users to generate documentation straight from this model.

Using templates, the documentation generation feature lets users specify the layout, content, and format of the documentation.

5.6 Increased Speed and Quality

A modeling tool like CE provides a more efficient workflow for users. Users are guided through SCA requirements, so they make fewer mistakes. Generating XML descriptors through a model-based flow is approximately 10 times faster than manual descriptor authoring. Development times are shortened and system quality is increased.

Software developers take pride in their work and enjoy working with state-of-the-art modeling tools like CE. It allows them to deliver better work faster and removes the frustrating task of manually authoring XML code. Developers with high-quality tools deliver better results more quickly. Job satisfaction is higher, so employee turnover is lower. Developers that are familiar with the SCA are difficult to find and train. CE eliminates these concerns.

6 CONCLUSION

Developing SCA based systems requires extra effort and coordination from the development team. However, this extra effort is well worth it. The SCA promises easy maintenance, portability, interoperability, and use of COTS systems.

Visual modeling tools and automation help developers stay on track, shorten development time, and increase software quality. Tools provide the flexibility to work within the SCA standard.

Zeligsoft CE provides all the functionality needed from a good visual modeling tool – an intuitive User Interface, SCA guidance, importation, validation and generation of documentation and descriptors are all aspects that are important to users.

7 REFERENCES

- **SCA**
  Software Communications Architecture
  http://jtrs.army.mil/sections/technicalinformation/fset_technical_sca.html

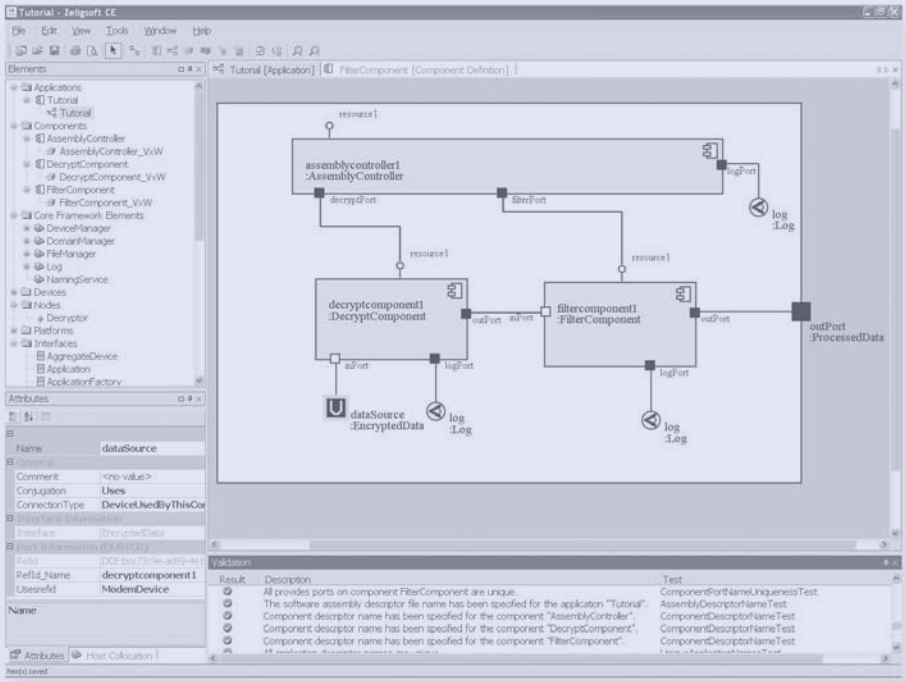
- **JTRS**
  http://jtrs.army.mil/

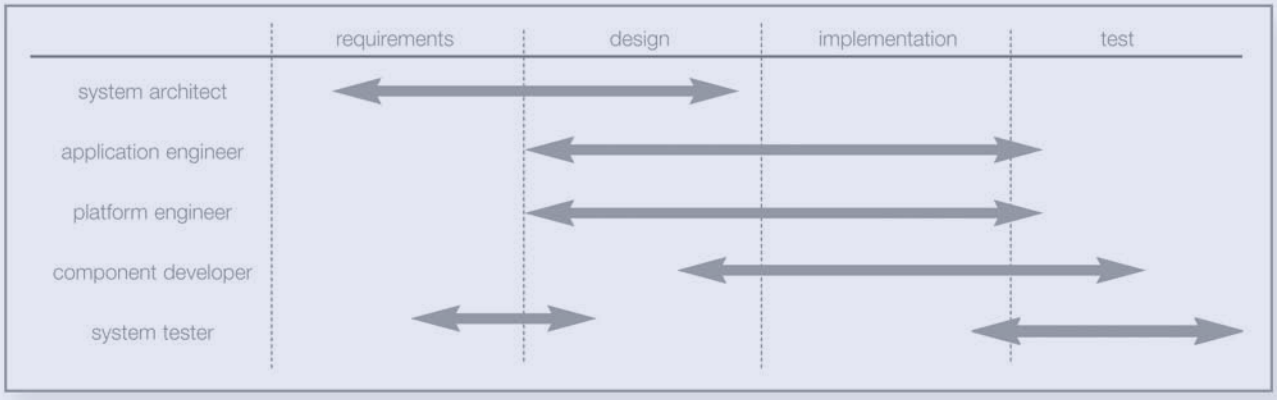
- **UML**
  Unified Modeling Language
  http://www.uml.com

- **CORBA**
  http://www.corba.org

- **ZSFT**
  Zeligsoft Inc.
  http://www.zeligsoft.com

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
Platforms
Interfaces
Tutorial [Application]
FilterComponent [Component Definition]
resource1
assemblycontroller1
:AssemblyController
logPort
log
:Log
decryptPort
filterPort
decryptcomponent1
:DecryptComponent
outPort
inPort
filtercomponent1
:FilterComponent
outPort
:ProcessedData
dataSource
:EncryptedData
Attributes
Name
dataSource
Comment
<no value>
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

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