

**Coverity Professional Service System Architecture**

**Professional Services Team**

Last Revised: October 31, 2011

Table of Contents

[Document Control 4](#_Toc307855143)

[1. Introduction 5](#_Toc307855144)

[1.1 Purpose 5](#_Toc307855145)

[1.2 References 5](#_Toc307855146)

[1.3 Glossary 5](#_Toc307855147)

[2. Architecture Overview 5](#_Toc307855148)

[2.1 System Rational 6](#_Toc307855149)

[2.2 Architectural Representation 6](#_Toc307855150)

[3. Architectural Goals and Constraints 6](#_Toc307855151)

[4. Use Case View 7](#_Toc307855152)

[4.1 Overview 7](#_Toc307855153)

[4.2 Actors 7](#_Toc307855154)

[4.3 Architecturally Significant Use Cases 8](#_Toc307855155)

[4.3.1 Local Configuration 8](#_Toc307855156)

[4.3.2 Remote Configuration 8](#_Toc307855157)

[4.3.3 Generate Report 8](#_Toc307855158)

[4.3.4 Export Defect 8](#_Toc307855159)

[4.3.5 Clean Before Check-In 8](#_Toc307855160)

[4.3.6 3rd Party SA Integration 8](#_Toc307855161)

[5. Component View 9](#_Toc307855162)

[5.1 Overview 9](#_Toc307855163)

[5.1.1 Business Model 9](#_Toc307855164)

[5.1.2 PS Configuration Manager 9](#_Toc307855165)

[5.1.3 CIM Web APIs 9](#_Toc307855166)

[5.1.4 Remote Node Manager 9](#_Toc307855167)

[5.1.5 Local Node Manager 10](#_Toc307855168)

[6. Implementation View 10](#_Toc307855169)

[6.1 Implementation Patterns 10](#_Toc307855170)

[6.2 Tools and Technologies 10](#_Toc307855171)

[7. Data View 10](#_Toc307855172)

[8. Deployment View 11](#_Toc307855173)

[9. Outstanding Issues 11](#_Toc307855174)

Document Control

Version

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Version | Author | Comments | Approver name |
| 10/22/2011 | 0.1 | Randy Hollines | Initial draft | Anton Kim |
|  |  |  |  |  |

Distribution list

|  |  |  |
| --- | --- | --- |
| Name | Title | Role |
| Anton Kim | Professional Services Manager | Approver |
| Frederic Langlet | Professional Services Engineer | Reviewer |
|  |  |  |

# Introduction

This document provides a high-level overview of the Coverity Professional Services architecture. It outlines the tools and technologies that will be used to deliver current and future state service offerings. The Professional Services (PS) technology platform compliments capabilities that exist in core Coverity product offerings.

The document outlines the business drivers that have influenced the Coverity Services architecture and the supporting use cases. The use cases have been decomposed into a set of technical views that express the system architecture.

## Purpose

This architecture allows customers to customize the functionality of core Coverity product offerings by making these products part of the development workflow. By integrating into the development workflow Coverity tools become an integral part of the software development lifecycle (SDLC). Service integrations allow customers to address defects early in the software development lifecycle through the use of reporting, metrics, and integrations with existing development tools such as bug tracking, software configuration management (SCM) and code review tools.

## References

* Professional Services Feature Sets
* Survey of Legacy PS Technologies
* GitHub SCM document

## Glossary

* CI - Continuous Integration
* CIM – Coverity Integrity Manager
* CRUD - Create, read, update and delete
* GUI - Graphical user interface
* JEE - Java Enterprise Architecture
* PS - Professional Services
* SA - Static analysis
* SCM - Software Configuration Management
* SDLC – Software Development Lifecycle
* WAR – Java web archive

# Architecture Overview

The Coverity PS Technical Architecture is expressed using UML models that represent conceptual abstractions. The intent of this document is to capture architecturally significant elements that together comprise the system architecture. When appropriate, detailed technical descriptions may be described in supporting design documents.

The following sections will describe the goals/constrains, stakeholders and technical structure of the PS platform architecture:

* Architectural Goals and Constraints
* Use Case View
* Component view
* Data View
* Implementation View
* Deployment View

## System Rational

The legacy Professional Services architecture is mainly a collection of command-line Perl scripts with recently added Java-based solutions. While these Perl scripts were highly flexible each engineer had their version of the scripts. With a number of scripts in the field they have become hard to manage and extend. Like many script-based offerings solutions were developed organically based upon customer demand. Some solutions leveraged common infrastructure while others duplicated existing functionality.

The goal of the target architecture is to develop a core framework that will support reusable components. The new component architecture will allow engineers to leverage existing solutions without having to make significant code modifications. The new architecture will be configuration driven versus code driven allowing engineers (and customers) to wire together solutions. In addition, the new architecture will leverage the CIM web container allowing customers to configure integrations from a common user interface that’s part of the CIM.

## Architectural Representation

The architecture of the platform is represented using a modified version of the Rational “4+1” model. The following views express the system architecture in its entirety:

* Use Case View: describes a set of actions or steps between actors in the software system. The use case view describes architecturally significant use cases, which will be broken down into detailed feature sets.
* Component View: describes the major structural components within the architecture and their relationships between one another. A component represents an implementation item, which may encapsulate a number of implementation classes.
* Implementation View: describes the technical design and implementation patterns that were used to realize the system architecture along with support tools and technologies.
* Data View: describes data sources that the system architecture uses. Data sources include web services, configuration files, database systems, etc.
* Deployment View: describes the tools and technologies that comprise the system architecture. Moreover, this view describes how the system should be deployed at customer sites.

# Architectural Goals and Constraints

This section describes the business and technology objectives that have significant impact on the system architecture. In addition special considerations may apply to the design and implementation strategy, team structure and delivery schedule.

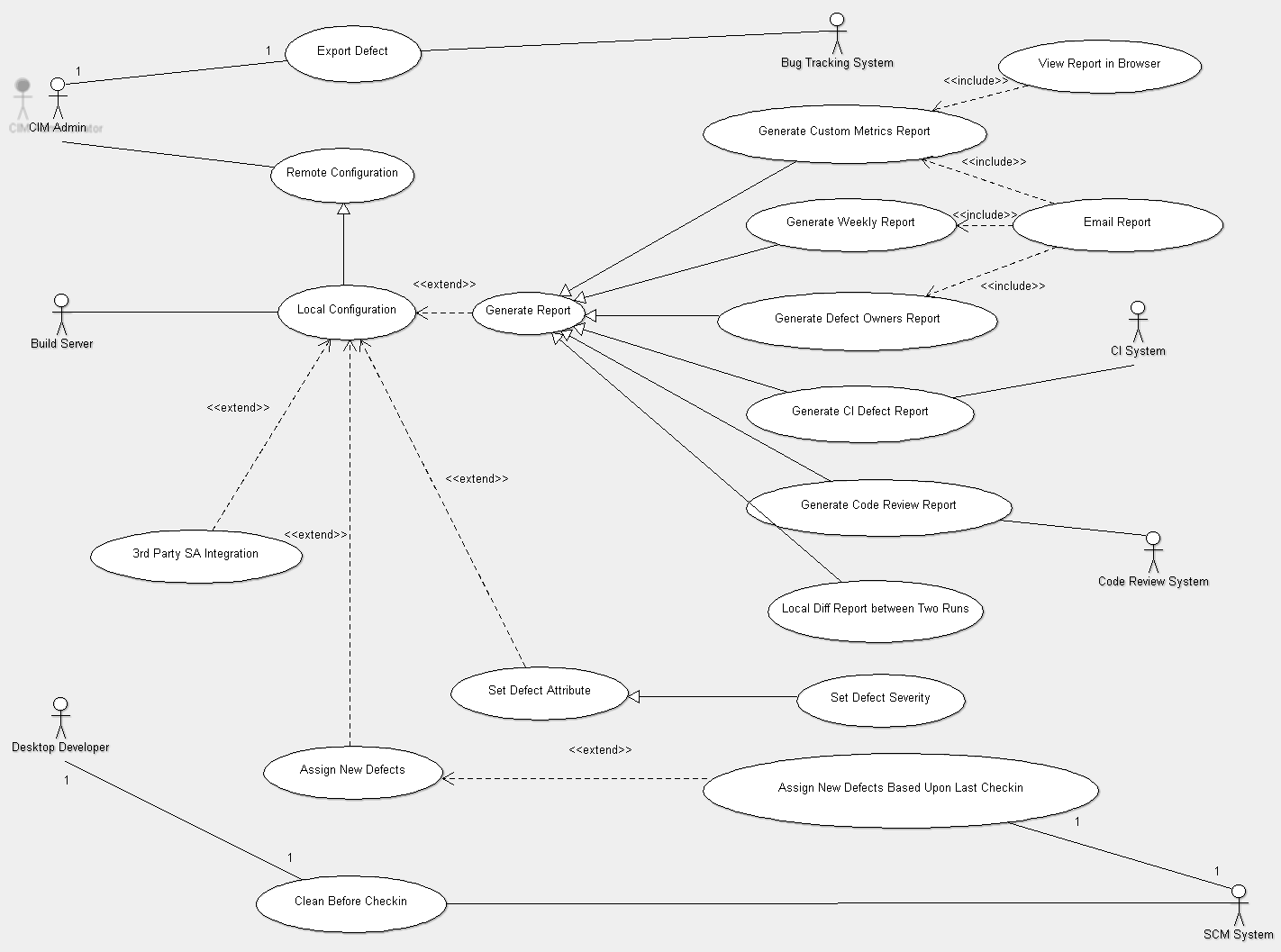
Goals

* Support legacy service offerings (please refer to the [Use Case View](#_Use_Case_View))
* Create a platform that will support new service offerings
* Enforce loose coupling between components
* Reuse ideas and solutions that worked well in the legacy architecture
* Provide a web interface that allows customers to configure integrations

Constraints

* Each customer has unique needs and this architecture may not meet all integration requirements
* Existing customers are used to scripted solutions, providing binaries without source code may not satisfy the needs of legacy customers
* The Coverity professional services team is distributed worldwide and only has time allocated to work on the target architecture between deliveries

# Use Case View



## Overview

The use case diagram above outlines system boundaries, actors and behaviors. The actors on the left-hand side of the diagram represent systems that initiate tasks. The actors on the right-hand side of the diagram represent systems that are the recipients of data or required for internal data processing. Where possible, common use case behaviors have been expressed using UML extension and specialization notations.

## Actors

The following use case actors have been identified:

* Bug Tracking System - a system that is used to manage software defects
* Build Server – a server that is running Coverity SA software
* CIM Administrator – an administrator who manages one or more Coverity instances
* Continuous Integration (CI) system - an automated build system that used to enforce CI best practices
* Desktop Developer – a software engineer who’s running Coverity SA software on their workstation
* Software Configuration Management System (SCM) - a system that is used to manage software codebases

## Architecturally Significant Use Cases

Architecturally significant use cases are those use cases that directly impact the realization of the system architecture. These use cases may have impact on system design, performance and implementation. Please note, that not all use cases are architecturally significant. Architecturally significant use cases should be used to guide the design and implementation of the system architecture.

## Local Configuration

This use case supports the ability to configure a local build server node. Various integrations will be configured using this common infrastructure. In addition, global setting such as the location of the CIM server will be managed by this use case.

## Remote Configuration

This use case is a specialization of the “Local Configuration” use case and provides the ability to configure nodes remotely. A web interface will be provided that will allow administrators to configure remote agents. Moreover, the user interface will provide the ability to share common configuration information among multiple nodes.

## Generate Report

This use case describes a system's ability to generate defect reports. The specialized reporting use cases, which depend upon this use case, describe specific types of reports that the system must support. The reporting use cases will interact with the CIM system in order to obtain defect information. The system will have the ability to generate reports in different formats such as HTML and plaintext. Reports may be delivered directly to end-users via e-mail or stored in the CIM for viewing or web-based retrieval.

## Export Defect

This use case describes the ability to export defect information to existing bug tracking systems. The system will have support for different bug tracking systems. A common user interface will be provided that will allow administrators to configure the system interface to a given bug tracking system. In addition, the user interface will allow administrators to map fields that are available in the CIM to fields that are available in the given bug tracking system.

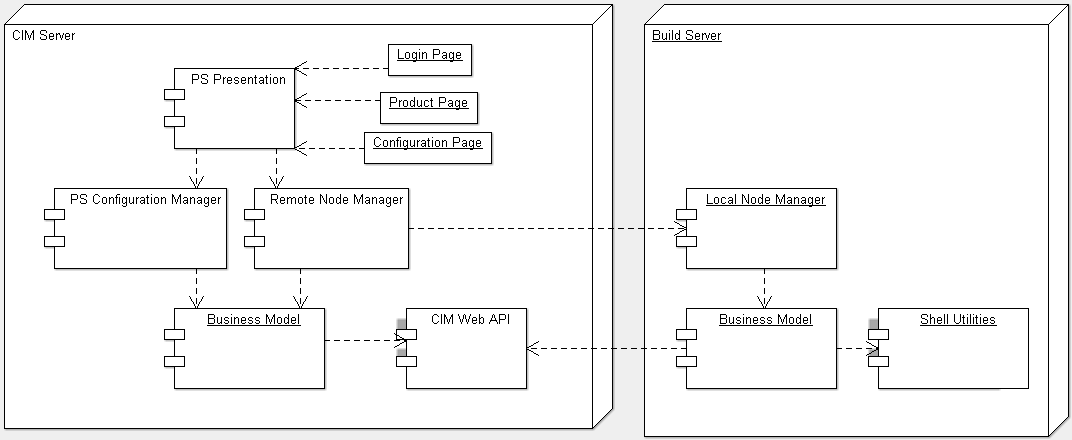
## Clean Before Check-In

This use case describes how the system will support a “clean before check-in” process. The system will provide the ability for a developer to run local analysis and triage any defects that are detected. All defects must be triaged before the developer is allowed to check-in their source code changes. The system will provide the ability to interact with different SCM systems.

## 3rd Party SA Integration

The system will provide the ability to import static analysis results from third-party systems into the CIM. The system will provide the ability to map data elements from third-party data sets to defect fields that exist in a CIM defect. The system will support various third-party formats.

# Component View



## Overview

The component view exposes the high-level architectural components that comprise the system architecture. These components represent the high-level building blocks that will be used for detailed design and implementation. Each of the physical components are made up of a number of implementation classes and leverage common system frameworks.

The system architecture consists of two logical deployment nodes (depicted in the diagram above). The left-hand side node is the CIM server, which provides a web based user interface and allow user to triage to defects. CIM server functionary will be extended to allow administrators to manage build server nodes. The node on the right-hand side is made up of zero or more build server instances. Each build server instance can be configured such that it can be managed by the CIM server. In addition, build server nodes may be independently configured and managed.

## Business Model

The business model (or business layer) encapsulates common CRUD operations. This layer wraps the technical interface that’s exposed by the CIM web service APIs. Operations that require one or more calls to the CIM web services APIs may be encapsulated in this model. In addition, this layer may provide functionality that is not directly supported by the CIM web services API; for example, the ability to persist configuration and reporting information.

## PS Configuration Manager

The PS Configuration Manager provides the technical model that will be used to back the configuration user interface. All configuration functionality will be provided by this component. This component will have the ability to aggregate information from remote build server agents. In addition, this component will support the ability to locally persist configuration data.

## CIM Web APIs

The CIM Web Service APIs are part of a standard CIM product. The web service APIs provide a consistent way to access defect and configuration information that resides in the CIM database. All access to data that’s stored in the CIM database must go through the CIM web services APIs.

## Remote Node Manager

The Remote Node Manager (RNM) provides the ability to interact with distributed build server agents. Configuration information that is managed within the CIM may be pushed to a given build server using this component. When a local agent is configured for remote use it will register with the RNM. The RNM will keep track of all agent nodes and provide the ability to determine if a given node is unavailable.

## Local Node Manager

The Local Node Manager (LNM) provides the ability to configure integrations on a single build server. The LNM will use a XML file to manage local configuration information. The local node manager will also support the ability to register with a RNM instance.

# Implementation View

<<introduction>>

## Implementation Patterns

<<types of design/implementation patterns used to realize the architecture>>

Notes:

* Inversion of control (IoC) – can be used to setup the properties of objects in the business layer (Spring)
* RMI – used as communication mechanism between the CIM and the build server nodes

## Tools and Technologies

The system will be implemented using JEE technologies. The CIM base components will be deployed in a WAR and leverage the CIMs Java web server application frameworks. The build server components will be deployed as standalone agents that have the ability to be remotely or locally managed.

The matrix below lists the tools and technologies that will be used to realize the PS architecture:

|  |  |  |
| --- | --- | --- |
| **Technology** | **Description** | **Version** |
| Spring? | * IoC for configuration management * Spring Remoting – RMI support * Persistence Framework - Hibernate support |  |
| Hibernate? | * Used for storing configuration information? |  |

# Data View

This view describes the internal and external data sources that the architecture uses.

<<diagram>>

Notes:

* Ability to persist configuration information
* Ability to persist reports – is this needed?
* Other possible source of data
  + XML files the build server (i.e. in the intermediate directories)
  + Error logs

# Deployment View

<<deployment diagram>>

This view describes how the system architecture is deployed on physical hardware.

Notes:

* How will the CIM WAR be deployed?
* How will the build server nodes be deployed?

# Outstanding Issues

The list below represents outstanding items that will be periodically reviewed with stakeholders.

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Description** | **Impact** | **Remediation** |
| CIM Server compatibility | What version of the CIM will the PS architecture be compatible with? | Low |  |
| Distribution of source code | How will the source be distributed to customers? Will we give them the source for individual components or the entire solution? | Medium | Discuss with Anton and/or Sunil |
| Persistence of data | How can we persistence configuration or reporting data? Can we deploy our own database like IC does? | High | Discuss with Anton; address with engineering |
| Ability to control the memory footprint of build server nodes | How/can we control the memory footprint for build server nodes (i.e. max heap size) | Medium | Investigate, should be possible via Java command line parameters |