# System and Software Design Description (SSDD)

**Code Generator – Template based**

**Team #10**

|  |  |  |
| --- | --- | --- |
| **Hung-Fu Chang** |  | **Project Manager** |
| **XYZ** |  | **Requirement Analyst** |
| **ABC** |  | **System Architect** |
|  |  |  |

**Prepared by Hung-Fu Chang**

**01/01/2021**

# Version History

| Date | Author | Version | Changes made | Rationale |
| --- | --- | --- | --- | --- |
| 09/30/05 | Hung-Fu Chang | 1.2 | * Completed Section 1-2 | * Complete Section 1-2 |
| 10/06/05 | Hung-Fu Chang | 1.3 | * Fixed errors in version 1.2 | * Contexts in section 1-2 |
| 10/09/05 | Hung-Fu Chang | 1.4 | * Adding new content for interactions | * Add new contents |
| 10/22/05 | Hung-Fu Chang | 1.5 | * Adding new content for interactions | * Add new contents |
| 12/04/05 | XYZ  ABC | 2.0 | * Revised system design and description | * Based on new guideline |
| 02/25/06 | XYZ  ABC | 3.2 | * Revised system design and description | * Based on new guideline |
| 01/01/21 | XYZ  ABC | 4.1 | * Revised system design and description | * Based on new guideline |

# Table of Contents

[System and Software Design Description (SSDD) 1](#_Toc67407789)

[Version History 3](#_Toc67407790)

[Table of Contents 4](#_Toc67407791)

[Table of Figures 6](#_Toc67407792)

[Table of Tables 8](#_Toc67407793)

[1. Introduction 10](#_Toc67407794)

[1.1 Purpose 10](#_Toc67407795)

[1.2 Overview 10](#_Toc67407796)

[2. System Overview 11](#_Toc67407797)

[2.1 Design Consideration 11](#_Toc67407798)

[2.1.1 Assumptions 11](#_Toc67407799)

[2.1.2 Constraints 11](#_Toc67407800)

[2.2 System Analysis 12](#_Toc67407801)

[2.2.1 System Modeling 12](#_Toc67407802)

[2.2.2 Modes of Operation 13](#_Toc67407803)

[3. Architecture 16](#_Toc67407804)

[3.1 System Composition 16](#_Toc67407805)

[3.1.1 Hardware Composition 17](#_Toc67407806)

[3.1.2 Software Composition 18](#_Toc67407807)

[3.2 Design Rationale 19](#_Toc67407808)

[3.2.1 Software Component Design 19](#_Toc67407809)

[3.3 Architectural Styles, Patterns and Frameworks 38](#_Toc67407810)

[4. System Design 40](#_Toc67407811)

[4.1 Structural View 40](#_Toc67407812)

[4.2 Logical View 41](#_Toc67407813)

[4.2.1 List of Logical Designs 41](#_Toc67407814)

[4.3 Interaction 46](#_Toc67407815)

[4.4 Interface 47](#_Toc67407816)

[4.4.1 User Interface Design 47](#_Toc67407817)

[4.4.2 Component Interface Design 50](#_Toc67407818)

[4.4.3 External Interfaces 50](#_Toc67407819)

[4.5 Information 51](#_Toc67407820)

[4.5.1 Data 51](#_Toc67407821)

[4.6 Algorithm Design 58](#_Toc67407822)

# Table of Figures

[Figure 1 – System Analysis 12](#_Toc67407823)

[Figure 2 System Structure - Static-Structure Diagram 12](#_Toc67407824)

[Figure 3 System Structure - Collaboration Diagram 13](#_Toc67407825)

[Figure 4 Models of Operations 14](#_Toc67407826)

[Figure 5. System Structure 16](#_Toc67407827)

[Figure 6: Hardware Component Diagram 17](#_Toc67407828)

[Figure 7 – Hardware Classifier Model 18](#_Toc67407829)

[Figure 8 Software Model 18](#_Toc67407830)

[Figure 9 – Interface(s) -- User Interface 20](#_Toc67407831)

[Figure 10 Process 1.1 Select Input File Use Case Diagram 20](#_Toc67407832)

[Figure 11 Process 1.2 Select Output Destination Use Case Diagram 23](#_Toc67407833)

[Figure 12 Process 1.3 View Error Log Use Case Diagram 25](#_Toc67407834)

[Figure 13 - Source File Parser – Interfaces 26](#_Toc67407835)

[Figure 14 Process 2.1 Parse Source File Use Case Diagram 27](#_Toc67407836)

[Figure 15 Template Handler – Interfaces 29](#_Toc67407837)

[Figure 16 Process 3.1 Parse Template File Use Case Diagram 30](#_Toc67407838)

[Figure 17 Script Builder – Interfaces 32](#_Toc67407839)

[Figure 18 Process 4.1 Build Script Use Case Diagram 33](#_Toc67407840)

[Figure 19 - Error Handler – Interfaces 35](#_Toc67407841)

[Figure 20 Process 5.1 Log Error Use Case Diagram 36](#_Toc67407842)

[Figure 21 Process 5.1 Log Error Activity Diagram 38](#_Toc67407843)

[Figure 22: UML Class Diagram 40](#_Toc67407844)

[Figure 23: UML Class Diagram 41](#_Toc67407845)

[Figure 24 Build Code – UCR-3.1 Build Script Sequence Diagram for LoadRunner 43](#_Toc67407846)

[Figure 25 Build Code – UCR-3.1 Build Script Sequence Diagram for WinRunner 45](#_Toc67407847)

[Figure 26 View Error Log – UCR-3.2 Sequence Diagram 45](#_Toc67407848)

[Figure 27: Interactions Between Converter and Actors 46](#_Toc67407849)

[Figure 28: Interactions Between Components and Actors 47](#_Toc67407850)

[Figure 29: Select Multiple Files 47](#_Toc67407851)

[Figure 30 : Process Status Indicator 48](#_Toc67407852)

[Figure 31 : Successful Conversion Process 48](#_Toc67407853)

[Figure 32 : Error Occurs During The Conversion Process 49](#_Toc67407854)

[Figure 33: Error Log 49](#_Toc67407855)

[Figure 34 Artifact diagram 50](#_Toc67407856)

[Figure 35 – Artifacts & Information 51](#_Toc67407857)

[Figure 36: Sample XML Input File 53](#_Toc67407858)

[Figure 37: Sample XSD Input File 54](#_Toc67407859)

[Figure 38: Sample Velocity Template File 55](#_Toc67407860)

[Figure 39: Sample LoadRunner Output File 57](#_Toc67407861)

[Figure 40: Sample WinRunner Output File 58](#_Toc67407862)

# Table of Tables

[Table 1 – GUI Mode state 14](#_Toc67407863)

[Table 2 – Command Line Mode State 15](#_Toc67407864)

[Table 3 – Waiting State 15](#_Toc67407865)

[Table 4 – Attribute Setting State 15](#_Toc67407866)

[Table 5 – Generating State 15](#_Toc67407867)

[Table 6: Hardware Component Description 17](#_Toc67407868)

[Table 7: Software Component Description 19](#_Toc67407869)

[Table 8 Process 1.1.1 Use Case Description 21](#_Toc67407870)

[Table 9 Process 1.1.2 Use Case Description 21](#_Toc67407871)

[Table 10 Process 1.1.1 Typical Course of Actions 22](#_Toc67407872)

[Table 11 Process 1.1.2 Typical Course of Actions 23](#_Toc67407873)

[Table 12 Process 1.2 Use Case Description 23](#_Toc67407874)

[Table 13 Process 1.2 Typical Course of Actions 24](#_Toc67407875)

[Table 14 Process 1.3 Use Case Description 25](#_Toc67407876)

[Table 15 Process 1.3 Typical Course of Actions 26](#_Toc67407877)

[Table 16 - Source File Parser - Parameters 27](#_Toc67407878)

[Table 17 Process 2.1 Use Case Description 28](#_Toc67407879)

[Table 18 Process 2.1 Typical Course of Actions 28](#_Toc67407880)

[Table 19 - Template Handler - Parameters 29](#_Toc67407881)

[Table 20 Process 3.1 Use Case Description 30](#_Toc67407882)

[Table 21 Process 3.1 Typical Course of Actions 31](#_Toc67407883)

[Table 22 - Script Builder - Parameters 32](#_Toc67407884)

[Table 23 Process 4.1 Use Case Description 34](#_Toc67407885)

[Table 24 Process 4.1 Typical Course of Actions 35](#_Toc67407886)

[Table 25 - Error Handler - Parameters 36](#_Toc67407887)

[Table 26 Process 5.1 Use Case Description 36](#_Toc67407888)

[Table 27 Process 5.1 Typical Course of Actions 37](#_Toc67407889)

[Table 28: Architectural Styles, Patterns, and Frameworks 38](#_Toc67407890)

[Table 29 Artifacts interacting table 50](#_Toc67407891)

### Introduction

#### Purpose

This document contains the Software Design Description of the Template-based Code Generator that is under development by Team T10 for the company Sophoi Inc.

#### Overview

The Template-based Code Generating System is constructed to generate testing scripts automatically in order to reduce the complexity and testing workload in software product developments for Sophoi Inc.

### System Overview

The proposed Template Based Code Generating System is constructed to generate testing scripts (WinRunner and LoadRunner scripts) automatically in order to reduce the complexity and testing workload in software product developments for Sophoi Inc. The services provided by the system are listed as below:

1. Provide both drag-and-drop and browsing methods for selecting input files
2. Parse and process source data files (XML and XSD Files)
3. Parse and process the template files
4. Error logging and reporting for exceptional operations
5. Script codes generation (in WinRunner or LoadRunner format, based on inputted template file)
6. Multiple input files concession
7. Provide process status indicator

#### Design Consideration

##### Assumptions

* The end user will be the employee inside Sophoi, Inc.

##### Constraints

Software Constraints

1. Component Usage:

The system uses the template-based code generator; hence, related information is provided below.

The following are some URLs of code generator – template based <http://www.jajakarta.org/velocity/velocity-1.2/docs/>

<http://codegenerator.sourceforge.net/>

1. Language

The system must use JAVA programming language.

Hardware Constraints

1. This will be a tool running on a personal computer.

Data Format Constraints

1. The data format for the template must be XML.
2. The source code file must be JAVA.

#### System Context



Figure 1 – System Analysis

##### System Modeling

Template Based Code Generator is a stand-alone application. It would be interacted with the System Operator, which consists of system User and Maintainer.

The following Static-Structure and Collaboration Diagrams describe the structure of the Template Based Code Generator:



Figure 2 System Structure - Static-Structure Diagram



Figure 3 System Structure - Collaboration Diagram

##### Modes of Operation

This section defines the modes of the system, and the mode defines a set of states that the system can be in and another that it cannot. It also defines the event that causes a mode change.

The system includes the following states:

1. Waiting: The system starts, waiting for user’s action.
2. WinRunner Mode: The system is going to generating a WinRunner script.
3. LoadRunner Mode: The system is going to generating a LoadRunner script.



Figure 4 Models of Operations

###### GUI Mode State

Table 1 – GUI Mode state

|  |  |
| --- | --- |
| **Process** | **Mode Impact** |
| **UC-1** | The system is set to GUI mode, so there is a GUI provided to user to allow user to manipulate, the GUI will provide buttons and list to allow user to select input files and generate output script. |

###### Command Line Mode State

Table 2 – Command Line Mode State

|  |  |
| --- | --- |
| **Process** | **Mode Impact** |
| **UC-1** | The system is set to Command Line Mode, no GUI is provided, user has to input parameters by typing in a console. |

###### Waiting State

Table 3 – Waiting State

|  |  |
| --- | --- |
| **Process** | **Mode Impact** |
| **UC-1** | The system is waiting for user to decide what kind of script he wants to generate. |

###### WinRunner Mode State

Table 4 – Attribute Setting State

|  |  |
| --- | --- |
| **Process** | **Mode Impact** |
| **UC-1, UC-2** | The system is going to generating a WinRunner Script. In this mode, it allow user to select XSD files, XML files and a Template file as the input. |

###### LoadRunner Mode State

Table 5 – Generating State

|  |  |
| --- | --- |
| **Process** | **Mode Impact** |
| **UC -01, UC-2** | The system is going to generating a LoadRunner Script. In this mode, it allow user to select XML files and a Template file as the input. The selection of XSD file will be disabled in this mode. |

### Architecture

#### System Composition

Block diagram of the system architecture is shown as below, while the red block identify the outside system users:



Figure 5. System Structure

##### Hardware Composition



Figure 6: Hardware Component Diagram

Table 6: Hardware Component Description

|  |  |
| --- | --- |
| **Hardware Component** | **Description** |
| Work Station | This is a tool running on a single personal computer. There is no network connection. |

The following diagram indicates the hardware model:



Figure 7 – Hardware Classifier Model

* The Template Based Code Generator is a stand-alone application which would be run in independent WorkStation.
* The System Operator would directly operate and interact with the system on the WorkStation.

##### Software Composition

This section shows the kinds of software components, the interfaces, the relations, and the classes and objects that reside on the component. The software model shows the interaction between components and actors.



Figure 8 Software Model

* User would operate and get responses from the system through the User\_Interface component, which is the only interface connecting other internal components of the system.
* The internal components include Error Handler, Source File Parser, Template Handler and Script Builder, all of which are transparent to the system users.
* Details of the Software would be described in the Software section.

This section shows the software components composed of system spreading over different layers. It covers the high-level component classifiers without getting into the details of the subsections such as parameters and interfaces. Inside the business controller, there are script builder and error log components.

Table 7: Software Component Description

|  |  |
| --- | --- |
| **Software Component** | **Description** |
| User interface | Provide system users with a graphical user interface in application dialogs/forms that enable users to interact (input/output) with the system. |
| Converter | This is the conversion engine. |
| Script Builder | Provides the entire system with a conversion engine which could convert the script files with Parameters and Template File that offered by Business Objects Component. |
| Error Logger | Show and write the error log when any error happens. |

#### Design Rationale

##### Software Component Design

This section shows the software components that composed of system spreading over different layers. It covers the high level component classifiers without getting into the details of the subsections such as parameters and interfaces, which will be described in Software Component Section.

###### Component Classifier 1 – User\_Interface

Purpose

Provide system operators with a graphical user interface, including application dialogs/forms which enable users to interact (ex: input/output) with the system.

Interface(s)



Figure 9 – Interface(s) -- User Interface

Behavior

User Interface is the only component classifier through which the System User would interact with the Template Based Code Generator. Such interactions include: browsing paths of the input files (XML and XSD files), template file and the proposed output files; examining error log generated in the code generating process



Figure 10 Process 1.1 Select Input File Use Case Diagram

Table 8 Process 1.1.1 Use Case Description

|  |  |
| --- | --- |
| Identifier | Process 1.1.1 |
| Use-Case Name | Select Input File (for generating LoadRunner script) |
| Abstract | No |
| Purpose | Provide the System Operator with a control panel in which the operator could select input files relevant to the LoadRunner script generation process |
| Priority | Must Have |
| Actors | System Operator |
| Importance | High |
| Requirements | CR-3, IR-1 |
| Risks | NONE |
| High–Risk? | No |
| Architecturally Significant? | Yes |
| Development Status | IOC |
| Overview | System Operator can select XML and template files |
| User Interface | Prototype v.01.04 |
| Pre-conditions | System successfully initialized |
| Post-conditions | Template Handler and Source File Handler are ready to process selected XML and Template files) |
| Specializes | None |
| Includes | Select XML File, Select Template File |
| Extends | None |
| Extension Points | None |

Table 9 Process 1.1.2 Use Case Description

|  |  |
| --- | --- |
| Identifier | Process 1.1.2 |
| Use-Case Name | Select Input File (for generating WinRunner script) |
| Abstract | No |
| Purpose | Provide the System Operator with a control panel in which the operator could select input files relevant to the WinRunner script generation process |
| Priority | Must Have |
| Actors | System Operator |
| Importance | High |
| Requirements | CR-3, IR-1 |
| Risks | NONE |
| High–Risk? | No |
| Architecturally Significant? | Yes |
| Development Status | IOC |
| Overview | System Operator can select XML, XSD and template files |
| User Interface | Prototype v.01.04 |
| Pre-conditions | System successfully initialized |
| Post-conditions | Template Handler and Source File Handler are ready to process selected XML, XSD and Template files |
| Specializes | None |
| Includes | Select XML File, Select XSD File, Select Template File |
| Extends | None |
| Extension Points | None |

Table 10 Process 1.1.1 Typical Course of Actions

|  |  |  |
| --- | --- | --- |
| **Seq. #** | **Actor Actions** | **System Response** |
| 1. | System Operator select “LoadRunner Script Generation” in the User Interface |  |
| 2. | System Operator push relevant “browse button” for selecting XML, Template files |  |
| 3. |  | System pop out a File Chooser dialogs and wait for user’s operation |
| 4. | System Operator select XML, Template files in the pop-out dialogs |  |

Table 11 Process 1.1.2 Typical Course of Actions

|  |  |  |
| --- | --- | --- |
| **Seq. #** | **Actor Actions** | **System Response** |
| 1. | System Operator select “WinRunner Script Generation” in the User Interface |  |
| 2. | System Operator push relevant “browse button” for selecting XML, XSD and Template files |  |
| 3. |  | System pop out File Chooser dialogs and wait for user’s operation |
| 4. | System Operator select XML, XSD and Template files in the pop-out dialogs |  |



Figure 11 Process 1.2 Select Output Destination Use Case Diagram

Table 12 Process 1.2 Use Case Description

|  |  |
| --- | --- |
| Identifier | Process 1.2 |
| Use-Case Name | Select Output Destination |
| Abstract | No |
| Purpose | Provide the System Operator with controls through which the operator could select output directory for generated scripts |
| Priority | Must Have |
| Actors | System Operator |
| Importance | High |
| Requirements | CR-1, IR-3 |
| Risks | NONE |
| High–Risk? | No |
| Architecturally Significant? | Yes |
| Development Status | IOC |
| Overview | System Operator can select output directory |
| User Interface | Prototype v.01.04 |
| Pre-conditions | System successfully initialized |
| Post-conditions | Path of output directory is recorded by the User Interface component |
| Specializes | None |
| Includes | None |
| Extends | None |
| Extension Points | None |

Table 13 Process 1.2 Typical Course of Actions

|  |  |  |
| --- | --- | --- |
| **Seq. #** | **Actor Actions** | **System Response** |
| 1. | System Operator push the “browse button” |  |
| 2. |  | System pop out a File Chooser dialog and wait for user’s operation |
| 3. | System Operator select output directory with the pop-out dialog |  |



Figure 12 Process 1.3 View Error Log Use Case Diagram

Table 14 Process 1.3 Use Case Description

|  |  |
| --- | --- |
| Identifier | Process 1.3 |
| Use-Case Name | View Error Log |
| Abstract | No |
| Purpose | Provide the System Operator with a dialog displaying the error log generated in the code generation process |
| Priority | Must Have |
| Actors | System Operator |
| Importance | High |
| Requirements | CR-2, CR-4, IR-5 |
| Risks | NONE |
| High–Risk? | No |
| Architecturally Significant? | Yes |
| Development Status | IOC |
| Overview | System Operator can view error log after code generation |
| User Interface | Prototype v.01.04 |
| Pre-conditions | All relevant input files selected; output directory selected; code generation process ended |
| Post-conditions | None |
| Specializes | None |
| Includes | None |
| Extends | None |
| Extension Points | None |

Table 15 Process 1.3 Typical Course of Actions

|  |  |  |
| --- | --- | --- |
| **Seq. #** | **Actor Actions** | **System Response** |
| 1. | System Operator push the “generate” button |  |
| 2. |  | Code generation process start |
| 3. |  | Display the status of code generation along the process; record errors occurred during the generation |
| 4. |  | Code generation process end; render and display the error log |
| 5. | System Operator view the displayed Error Log |  |

###### Component Classifier 2 – Source File Parser

Purpose

Parse input files (XML and XSD files) provided by User Interface component and generate relevant data structures which would store necessary parameters for the code generation.

Interface(s)



Figure 13 - Source File Parser – Interfaces

Parameters

Table 16 - Source File Parser - Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter Name | Type/Signature | Default Value | Purpose |
| **XML Source File** | **FileStream** | **NULL** | **The file includes all necessary data which would be served as attributes for Script Builder component in order to generate both LoadRunner and WinRunner Scripts.** |
| **XSD Source File** | **FileStream** | **NULL** | **The file includes part of data which would be served as attributes for Script Builder component in order to generate WinRunner Scripts.** |

Behavior

This is an internal component of the Template Based Code Generator and it is transparent to the System Operator. This component classifier will parse the source files specified in User Interface component, and then convert the data to system recognizable data structures which would be attributes for Script Builder component in order to generate the scripts.

Processes

*Process 2.1 Parse Source File*

Figure 14 Process 2.1 Parse Source File Use Case Diagram



Table 17 Process 2.1 Use Case Description

|  |  |
| --- | --- |
| Identifier | Process 2.1 |
| Use-Case Name | Parse File (for XML and XSD files) |
| Abstract | No |
| Purpose | Parse the source files specified in User Interface component, and then convert the data to system recognizable data structures (XMLValue Container, XSDValue Container) which would be attributes for Script Builder component in order to generate the scripts. |
| Priority | Must Have |
| Actors | NONE |
| Importance | High |
| Requirements | CR-5, IR-6 |
| Risks | NONE |
| High–Risk? | Yes |
| Architecturally Significant? | Yes |
| Development Status | IOC |
| Overview | Parse source files and convert the data to specific data structures |
| User Interface | NONE |
| Pre-conditions | All relevant input files selected in the User Interface |
| Post-conditions | Data structure has been provided which is ready for gernerating codes. |
| Specializes | None |
| Includes | None |
| Extends | Parse XML File; Parse XSD File |
| Extension Points | None |

Table 18 Process 2.1 Typical Course of Actions

|  |  |  |
| --- | --- | --- |
| **Seq. #** | **Actor Actions** | **System Response** |
| 1. | System Operator push the “generate” button |  |
| 2. |  | Open and load source files (XML and XSD files) |
| 3. |  | Parse files and fill the data structures (XMLValue Container, XSDValue Container) that serve as parameters for Script Builder component |
| 4. |  | Log errors occurred in the parsing process |

###### Component Classifier 3 – Template Handler

Purpose

Template Handler will first check syntax errors which would be recorded in Error Log. While the template file contains a parameter definition field and script structure definition field, the Template Handler will parse the template file in two different ways, which are:   
1. Abstracts parameters and have them stored in special data structures which would serve as attributes for Script Builder component;

2. Validate the script structure definition.

Interface(s)

Figure 15 Template Handler – Interfaces



Parameters

Table 19 - Template Handler - Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter Name | Type/Signature | Default Value | Purpose |
| **Template File** | **FileStream** | **NULL** | **Provide script structure and necessary parameters for the Script Builder component.** |

Behavior

This is an internal component of the Template Based Code Generator and it is transparent to the System Operator. This component classifier will parse the template files specified in User Interface component by:

1. Abstracts parameters and have them stored in special data structures (ParameterMap) which would serve as attributes for Script Builder component;

2. Validate the script structure definition.



Figure 16 Process 3.1 Parse Template File Use Case Diagram

Table 20 Process 3.1 Use Case Description

|  |  |
| --- | --- |
| Identifier | Process 3.1 |
| Use-Case Name | Parse File (for Template file) |
| Abstract | No |
| Purpose | Parse the template files specified in User Interface component, and then convert the contained parameters to system recognizable data structures (ParameterMap). |
| Priority | Must Have |
| Actors | NONE |
| Importance | High |
| Requirements | CR-3, CR-4 |
| Risks | NONE |
| High–Risk? | No |
| Architecturally Significant? | Yes |
| Development Status | IOC |
| Overview | Parse template files and convert the data to specific data structures (ParameterMap) |
| User Interface | NONE |
| Pre-conditions | Template file selected in the User Interface |
| Post-conditions | Script Builder component is ready to generate codes based on data structures provided by this component |
| Specializes | None |
| Includes | None |
| Extends | Parse Template File |
| Extension Points | None |

Table 21 Process 3.1 Typical Course of Actions

|  |  |  |
| --- | --- | --- |
| **Seq. #** | **Actor Actions** | **System Response** |
| 1. | System Operator push the “generate” button |  |
| 2. |  | Open and load template file |
| 3. |  | Parse files and create relevant data structures (ParameterMap) that serve as parameters for Script Builder component |
| 4. |  | Log errors occurred in the parsing process |

###### Component Classifier 4 – Script Builder

Purpose

Build and output LoadRunner or WinRunner scripts based on data structures created by Source File Parser and Template File Parse components; record errors occurred in the script building process.

Interface(s)



Figure 17 Script Builder – Interfaces

Parameters

Table 22 - Script Builder - Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter Name | Type/Signature | Default Value | Purpose |
| **XML Value Container** | **Hash Table** | **NULL** | **Provide the Script Builder component with parameters that are necessary for both LoadRunner and WinRunner scripts generation process.** |
| **XSD Value Container** | **Hash Table** | **NULL** | **Provide the Script Builder component with parameters that are necessary for the WinRunner script generation.** |
| **ParameterMap (from template file)** | **Hash Table** | **NULL** | **Provide variables, which were extracted from Parameter Definition Field in template file, for Script Builder component to generate script files** |
| **Script structure definition (from template file)** | **String/Stream** | **NULL** | **Provide script structure and necessary parameters for the Script Builder component.** |

Behavior

This Script Builder component classifier use script structure definition, which is defined in the template file, as well as parameters parsed from XML and XSD file, to build script codes in either LoadRunner or WinRunner format. This is an internal component of the Template Based Code Generator and it is transparent to the System Operator.



Figure 18 Process 4.1 Build Script Use Case Diagram

Table 23 Process 4.1 Use Case Description

|  |  |
| --- | --- |
| Identifier | Process 4.1 |
| Use-Case Name | Build Script |
| Abstract | No |
| Purpose | Use relevant data structure that has been parsed from template file (Script structure definition) and source files (XML Value Container and XSD Value Container) to generate LoadRunner or WinRunner scripts |
| Priority | Must Have |
| Actors | NONE |
| Importance | High |
| Capabilities | OC-1, OC-2, OC-6 |
| Requirements | CR-3, CR-5, IR-4, IR-6 |
| Risks | NONE |
| High–Risk? | No |
| Architecturally Significant? | Yes |
| Development Status | IOC |
| Overview | Build script codes with parameters from source files and template file |
| User Interface | NONE |
| Pre-conditions | Source Files (XML and XSD files) and template file are successfully parsed; relevant data structures are created |
| Post-conditions | LoadRunner or WinRunner script(s) is/are generated and outputted to user-specified output directory |
| Specializes | None |
| Includes | Parse File |
| Extends | None |
| Extension Points | None |

Table 24 Process 4.1 Typical Course of Actions

|  |  |  |
| --- | --- | --- |
| **Seq. #** | **Actor Actions** | **System Response** |
| 1. | System Operator push the “generate” button |  |
| 2. |  | Parse relevant input files and create necessary data structures |
| 3. |  | Build script codes with relevant data structures from Source File Parser and Template File Parser components |
| 4. |  | Log errors occurred in the parsing process |

###### Component Classifier 5 – Error Handler

Purpose

Record the errors that occurred in source files, template file parsing and code building process. The recorded error would be displayed to the System Operator when the generation ended.

Interface(s)



Figure 19 - Error Handler – Interfaces

Parameters

Table 25 - Error Handler - Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter Name | Type/Signature | Default Value | Purpose |
| **Error log entry** | **object** | **NULL** | **The minimum data structure processed by Error Handler that records errors occurred in the code generation process** |

Behavior

The error handler will record errors occurred in file parsing and script generating process. The errors recorded by this component classifier would be transferred to the User Interface to display after the completion of code generation.



Figure 20 Process 5.1 Log Error Use Case Diagram

Table 26 Process 5.1 Use Case Description

|  |  |
| --- | --- |
| Identifier | Process 5.1 |
| Use-Case Name | Log Error |
| Abstract | No |
| Purpose | When exceptions occurred in the file parsing or code building process, the Error Handler records the errors which would be displayed through the User Interface after the code generation ended. |
| Priority | Must Have |
| Actors | NONE |
| Importance | High |
| Requirements | CR-2, CR-4, CR-5, IR-5 |
| Risks | None |
| High–Risk? | No |
| Architecturally Significant? | Yes |
| Development Status | IOC |
| Overview | Record errors occurred in file parsing and script generating process. |
| User Interface | NONE |
| Pre-conditions | File parsing or code generating process started and system encounters exceptions |
| Post-conditions | Errors are recorded in relevant data structures; error information is transferred to User Interface and displayed |
| Specializes | None |
| Includes | None |
| Extends | None |
| Extension Points | None |

Table 27 Process 5.1 Typical Course of Actions

|  |  |  |
| --- | --- | --- |
| **Seq. #** | **Actor Actions** | **System Response** |
| 1. | System Operator push the “generate” button |  |
| 2. |  | Parse relevant input files and generate output scripts |
| 3. |  | Record all errors encountered in Seq.#2 |
| 4. |  | Transfer recorded errors to User Interface and display |



Figure 21 Process 5.1 Log Error Activity Diagram

#### Architectural Styles, Patterns and Frameworks

Table 28: Architectural Styles, Patterns, and Frameworks

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Benefits, Costs, and Limitations** |
| Model-View-Controller | Using three layers to separate the logic. The view layer will handle all the user operations on the interface. The controller will handle all the business logic. The model handles the data model | Flexibility to change. |
| Singleton | Using this pattern in order to make sure only one error log and one script builder instances exist. | Avoid errors from duplicate instances. Data can be handled only by one instance. |
|  |  |  |

### System Design

#### Structural View



Figure 22: UML Class Diagram



Figure 23: UML Class Diagram

#### Logical View

##### List of Logical Designs

The following sections will discuss two logical views.

1. Build code
2. View ErrorLog

###### Build Code



Figure 24 Build Code – UCR-3.1 Build Script Sequence Diagram for LoadRunner



Figure 25 Build Code – UCR-3.1 Build Script Sequence Diagram for WinRunner

###### View Error Log



Figure 26 View Error Log – UCR-3.2 Sequence Diagram

#### Interaction



Figure 27: Interactions Between Converter and Actors



Figure 28: Interactions Between Components and Actors

#### Interface

##### User Interface Design

User selects single file or multiple files by either click shift or ctrl with mouse clicking.

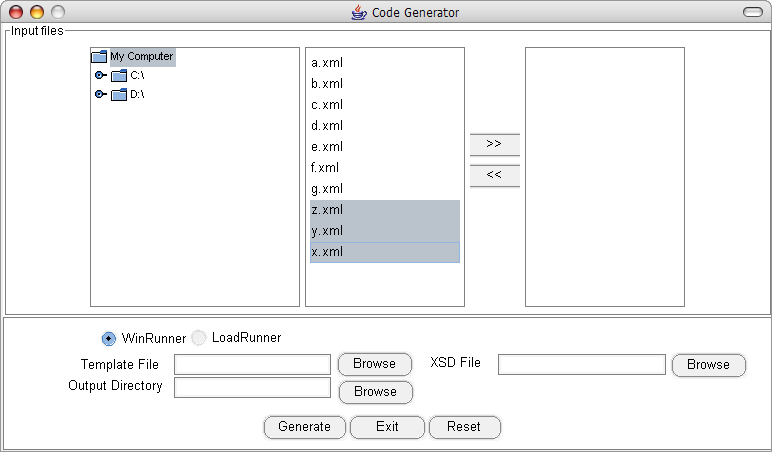


Figure 29: Select Multiple Files

Progress bar shows up after user clicks on the “Generate” button

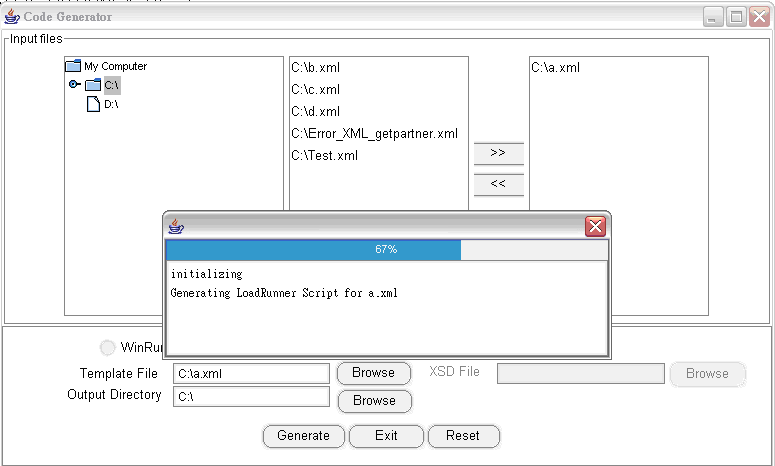


Figure 30 : Process Status Indicator

Successful message pops up when conversion completes without error.

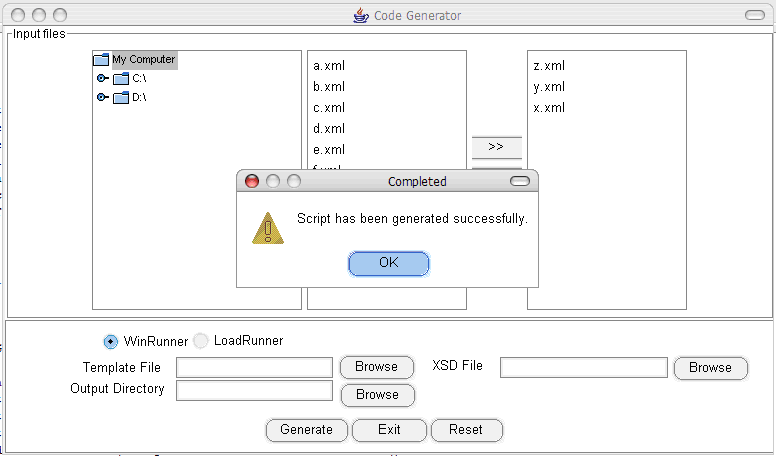
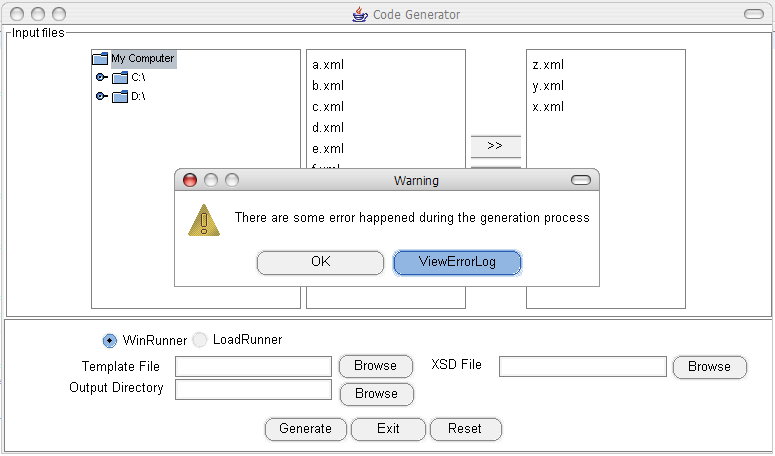


Figure 31 : Successful Conversion Process

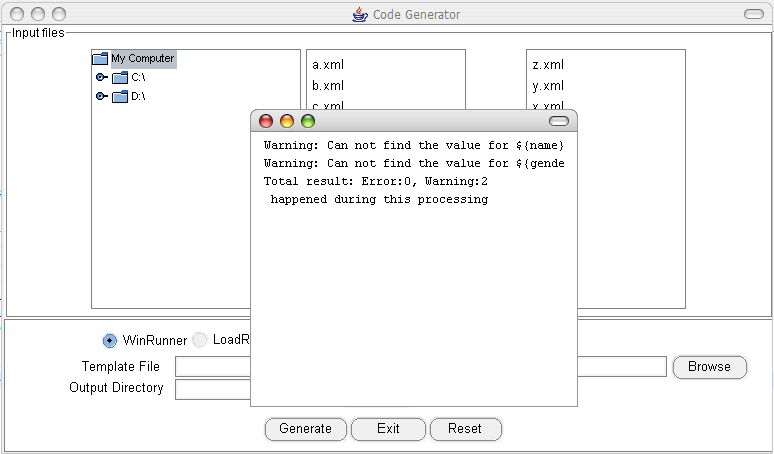
Error message pops up when conversion completes with error.

Figure 32 : Error Occurs During The Conversion Process



The log message shows after user clicks “ViewErrorLog” button.

Figure 33: Error Log



##### Component Interface Design



Figure 34 Artifact diagram

##### External Interfaces

The following shows that the artifacts (files) that the external interface uses.

Table 29 Artifacts interacting table

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Name | Role, Purpose and Responsibility | Interacting Module |
| IF-01 | Java Class File | Alternative source file, containing all data elements required for constructing a script file, which is described in standard Java format | Source File Interpreter |

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Name | Role, Purpose and Responsibility | Interacting Module |
| IF-02 | XML File | Alternative source file, containing all data elements required for constructing a script file, which is described in standard XML format | Source File Interpreter |

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Name | Role, Purpose and Responsibility | Interacting Module |
| IF-03 | Template File | Input file, written in format appointed by Sophoi Software, describing the file structure (WinRunner or LoadRunner) of the script file | Template File Interpreter |

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Name | Role, Purpose and Responsibility | Interacting Module |
| OF-01 | Script Files | Output file, in WinRunner or LoadRunner format, utilized directly by WinRunner or LoadRunner for software testing | Output Interface |

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Name | Role, Purpose and Responsibility | Interacting Module |
| OF-02 | Error Log/Report | Log file, recording error of input files or conversion process | Error Logging and Reporting |

#### Information

##### Data

###### Data Description

The Velocity use XML as the template. Inside the Velocity, WinRunner will be used. The following section will show the details for each input files.



Figure 35 – Artifacts & Information

This diagram shows all artifacts that will be used or generated during the code generation process.

* The XML, XSD and template file are the input files that will be used to generate a script, and there are two types of script file: WinRunner script and LoadRunner script.
* During the code generation process, the system will produce an error log if there are some exception happened in the processing.

###### Data Dictionary

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <PartnerWrapper>  <PartnerCommentDetailValueObj>    <PartnerUserId>Partner\_User\_Id5</PartnerUserId>    <PartnerSysId>20266</PartnerSysId>    <PartnerName>Satyam-Hyd5</PartnerName>    <Comment>Comment</Comment>    </PartnerCommentDetailValueObj>  <PartnerChannelDetailValueObjs>  <PartnerChannelDetailValueObj>    <PartnerChannelSysId>89</PartnerChannelSysId>    <MasterMediaSysId>45</MasterMediaSysId>    <MasterMedia>HomeVideo</MasterMedia>    <PartnerSysId>20266</PartnerSysId>    <Partner>Satyam-Hyd5</Partner>    </PartnerChannelDetailValueObj>    </PartnerChannelDetailValueObjs>    </PartnerWrapper> |

Figure 36: Sample XML Input File

|  |
| --- |
| **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**  **<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">**  **<xs:element name="PartnerWrapper">**  **<xs:complexType>**  **<xs:sequence>**  **<xs:element name="PartnerChannelDetailValueObjs" minOccurs="0">**  **<xs:complexType>**  **<xs:sequence>**  **<xs:element name="PartnerChannelDetailValueObj" minOccurs="1" maxOccurs="unbounded">**  **<xs:complexType>**  **<xs:sequence>**  **<xs:element name="TerritoryArray" minOccurs="0">**  **<xs:complexType>**  **<xs:sequence>**  **<xs:group ref="TerritoryArray" maxOccurs="unbounded"/>**  **</xs:sequence>**  **<xs:attribute name="size" type="xs:string"/>**  **</xs:complexType>**  **</xs:element>**  **<xs:element name="PartnerChannelSysId" type= "integer\_10" minOccurs="0"/>**  **<xs:element name="MasterMediaSysId" type= "integer\_10" minOccurs="0"/>**  **<xs:element name="MasterMedia" type= "MASTER\_MEDIA" minOccurs="0"/>**  **<xs:element name="PartnerSysId" type= "integer\_10" minOccurs="0"/>**  **<xs:element name="Partner" type= "xs:string" minOccurs="0"/>**  **<xs:element name="PartnerChannelNumberDetailValueObjs" minOccurs="0">**  ( omit the rest of the content ) |

Figure 37: Sample XSD Input File

|  |
| --- |
| #foreach (${valueObjs} in ${valueObjsMap}.get(${wrapper.Name}))  ${wrapper.Name}${valueObjs.Name}Action(){  int i;  web\_submit\_data(  "${valueObj.name}Action.do”,  "Action=http://localhost:7001/web/marketing/{wrapper.lowerCaseName}/{wrapper.upperCaseName}DetailsNavBarAction.do", “Method=POST”,  “RecContentType=text/html”,  “Refer=http://localhost:7001/web/jsp/marketing/{wrapper.upperCaseName}Details/{wrapper.lowerCaseName}NavBar.jsp”  “Snapchot=t”  “Mode=HTML”,  ITEMDATA,  “Name=navToPage”, “value=${valueObjs.lowerName}”, ENDITEM, LAST);  For(i=0; i<NO\_OF\_LIST\_ENTRIES; i++){  lr\_start\_transaction(“${valueObjs.upperName}Action”);  web\_submit\_data(  “${valueObjs.upperName}Action.do”,  “Action=http://localhost:7001/web/marketing/${wrapper.lowerName}/${valueObjs.upperName}Action.do”,  “Method=POST”,  “RecContentType=text/html”,  “Referer=http://localhost:7001/web/jsp/marketing/${wrapper.lowerName}/${valueObjs.lowerName}.jsp”,  “Snapchot=t”,  “Mode=HTML”,  ITEMDATA,  “Name=actionValue”, “Value=new”, ENDITEM, LAST  );  lr\_end\_transaction(“${valueObjs.upperCaseName}”, LR\_AUTO);  }  } |

Figure 38: Sample Velocity Template File

|  |
| --- |
| #include "as\_web.h"  const int NO\_OF\_LIST\_ENTRIES = 10;  PartnerAction()  {  PartnerCommentAction();  PartnerChannelAction();  lr\_rendezvous("PartnerSave");  PartnerSaveAction();  return 0;  }  PartnerCommentAction() {  web\_submit\_data("PartnerCommentDetailAction.do",  "Action=http://localhost:7001/web/marketing/partner/PartnerCommentDetailAction.do",  "Method=POST",  "RecContentType=text/html",  "Referer=http://localhost:7001/web/jsp/marketing/partnerDetails/partnerCommentDetail.jsp",  "Snapshot=t3.inf",  "Mode=HTML",  ITEMDATA,  "Name=valueObject.partnerUserId", "Value=Partner\_User\_Id5", ENDITEM,  "Name=valueObject.partnerSysId", "Value=20266", ENDITEM,  "Name=valueObject.partnerName", "Value=Satyam-Hyd5", ENDITEM,  "Name=valueObject.comment", "Value=Comment", ENDITEM,  LAST);  lr\_think\_time( 1 );  }  }  PartnerChannelAction() {  int i;  web\_submit\_data("PartnerDetailNavBarAction.do",  "Action=http://localhost:7001/web/marketing/partner/PartnerDetailNavBarAction.do",  "Method=POST",  "RecContentType=text/html",  "Referer=http://localhost:7001/web/jsp/marketing/partnerDetails/partnerNavBar.jsp",  "Snapshot=t1.inf",  "Mode=HTML",  ITEMDATA,  "Name=navToPage", "Value=partnerChannel", ENDITEM,  LAST);    for(i=0; i<NO\_OF\_LIST\_ENTRIES; i++) {  lr\_start\_transaction("PartnerChannelAction");  web\_submit\_data("PartnerChannelAction.do",  "Action=http://localhost:7001/web/marketing/partner/PartnerChannelAction.do",  "Method=POST",  "RecContentType=text/html",  "Referer=http://localhost:7001/web/jsp/marketing/partnerDetails/partnerChannel.jsp",  "Snapshot=t2.inf",  "Mode=HTML",  ITEMDATA,  "Name=actionValue", "Value=new", ENDITEM,  LAST);  lr\_end\_transaction("PartnerChannelAction",LR\_AUTO);  lr\_start\_transaction("PartnerChannelDetailAction");  web\_submit\_data("PartnerChannelDetailAction.do",  "Action=http://localhost:7001/web/marketing/partner/PartnerChannelDetailAction.do",  "Method=POST",  "RecContentType=text/html",  "Referer=http://localhost:7001/web/jsp/marketing/partnerDetails/partnerChannelDetail.jsp",  "Snapshot=t3.inf",  "Mode=HTML",  ITEMDATA,  "Name=valueObject.partnerChannelSysId", "Value=89", ENDITEM,  "Name=valueObject.partnerSysId", "Value=20266", ENDITEM,  "Name=valueObject.partner", "Value=Satyam-Hyd5", ENDITEM,  "Name=valueObject.mediaName", "Value=HomeVideo", ENDITEM,  "Name=valueObject.masterMediaSysId", "Value=45", ENDITEM,  LAST);  lr\_end\_transaction("PartnerChannelDetailAction",LR\_AUTO);  lr\_think\_time( 1 );  }  } |

Figure 39: Sample LoadRunner Output File

|  |
| --- |
| load("C:\dev\qa\ipls2Testscripts\Automation\WinRunner\iPLS 3.0\Compiled\Login");  Login();  set\_window("My JSP 'iplsAdmin.jsp' starting page",108);  button\_press("Product Maintenance");  web\_sync(20);    load("C:\dev\qa\ipls2Testscripts\Automation\WinRunner\iPLS 3.0\Compiled\TextValidation");  load("C:\dev\qa\ipls2Testscripts\Automation\WinRunner\iPLS 3.0\Compiled\DateValidation");  load("C:\dev\qa\ipls2Testscripts\Automation\WinRunner\iPLS 3.0\Compiled\EditDeleteButton");  load("C:\dev\qa\ipls2Testscripts\Automation\WinRunner\iPLS 3.0\Compiled\NumberTextValidation");  load("C:\dev\qa\ipls2Testscripts\Automation\WinRunner\iPLS 3.0\Compiled\YearValidation");  load("C:\dev\qa\ipls2Testscripts\Automation\WinRunner\iPLS 3.0\Compiled\popUpTerritorySearch");  load("C:\dev\qa\ipls2Testscripts\Automation\WinRunner\iPLS 3.0\Compiled\popUpLanguageSearch");  load("C:\dev\qa\ipls2Testscripts\Automation\WinRunner\iPLS 3.0\Compiled\SelectionDialogPage");  load("C:\dev\qa\ipls2Testscripts\Automation\WinRunner\iPLS 3.0\Compiled\popUpProductSearch");  load("C:\dev\qa\ipls2Testscripts\Automation\WinRunner\iPLS 3.0\Compiled\popUpPartnerSearch");    set\_window("headermenu",2);  web\_event("Partner", "click", 14, 18);    set\_window("buttons",22);  web\_event("New", "click", 5, 11);    ValidateText("partnerDetailValueObj.partnerUserId",500, "User Partner ID", "ID 789", "body");  ValidateText("partnerDetailValueObj.partnerName",100, "Partner Name", "SHIVA", "body");  ( omit the rest of the content ) |

Figure 40: Sample WinRunner Output File

#### Algorithm Design

No special algorithm is used in this system.