Ampersand
Event-Condition-Action Rules

Test Plan

Yuriy Toporovskyy, Yash Sapra, Jaeden Guo

CS 4ZP6
October 30th, 2015
Fall 2015 / Winter 2016

Table 1: Revision History

Author	**Date**	**Comment**
Yuriy Toporovskyy	27 / 10 / 2015	Reorganized document
Yuriy Toporovskyy	27 / 10 / 2015	Initial version - template

Contents

List of Tables and figures

Chapter 1

General Information

1.1 Purpose

This document outlines the test plan for ECA for Ampersand, including our general approach to testing, system test cases, and a specification of methodology and constraints. This test plan specifically targets our contribution to Ampersand, namely ECA – elements of Ampersand, such as design artifact generation, will not be tested.

1.2 Objectives

Preparation for testing

The primary objective of this test plan is to collect all relevant information in preperation of the actual testing process, in order to facilitate this process.

Communication

This test plan intends to clearly communicate to all developers of ECA for Ampersand their intended role in the testing process.

Motivation

The testing approach is motivated by constraints and requirements outlined in the Software Requirements Specification. This document seeks to clearly demonstrate this motivation.

Environment

This test plans outlines the resources, tools, and software required for the testing process. This includes any resources needed to perform automated testing.

Scope

This test plan intends to better describe the scope of our contribution, ECA, within Ampersand.

1.3 Acronyms, Abbreviations, and Symbols

SRS Software Requirements Specification. Document regarding requirements, constraints, and project objectives.

ECA Rule Event-Condition-Action Rule. A rule which describes how to handle a constraint violation in a database. See SRS for details.

HUnit A Haskell library for unit testing. See TODO: ref test tools

QuickCheck A Haskell library for running automated, randomized tests. See TODO: ref test tools

Chapter 2

Plan

2.1 Software Description

Ampersand is a software tool which converts a formal specification of business entities and rules, and compiles it into different design artifacts, as well as a prototype web application.

This prototype implements the business logic in the original specification, in the form of a relational database with a simple web-app front-end.

A particular class of relational database violations can be automatically restored; the algorithm for computing the code to fix these violations is called AMMBR [Joo07]. This class of violations is realized within Ampersand as ECA rules – our contribution to Ampersand will add support for ECA rules, in both the Ampersand back-end and the generated prototype.

2.2 Test Team

The test team which will execute the strategy outline in this document is comprised of

- Yuriy Toporovskyy
- Yash Sapra
- Jaeden Guo

2.3 Test Schedule

Chapter 3

Methods and constraints

3.1 Methodology

3.2 Test tools

3.2.1 Static Typing

Programming languages can be classified by many criteria, one of which is their type systems. One such classification is static versus dynamic typing. Our implementation language, Haskell, has a static type system. Types will be checked at compile-time, allow us to catch errors even before the code is run, reducing the errors that need to be found and fixed using testing techniques.

3.2.2 Formal verification

A part of our project deals with generating source code annotated with the proof of derivation of that source code, which will act as a correctness proof for the system. In particular, when we generate code to restore a database violation using ECA rules, then the generated code will have a proof associated with it, which details how that code was derived from the original specification given by the user.

3.2.3 Random Testing

Random testing allows us to easily run a very large number of tests without writing them by hand, and also has the advantage of not producing biased test cases, like a programmer is likely to do.

We will be using QuickCheck for random testing. The existing Ampersand code base using QuickCheck for testing, therefore, using QuickCheck has the added benefit of easier integration with the existing Ampersand code base.

QuickCheck allows the programmers to provide a specification of the program, in the form of properties. A property is essentially a boolean valued Haskell function of any number of arguments. QuickCheck can test that these properties hold in a large number of randomly generated cases. QuickCheck also takes great care to produce a large variety of test cases, and generally produces good code coverage.

3.2.4 Unit Testing

Unit testing is comprised of feeding some data to the functions being tested and compare the actual results returned to the expected resultd. We will be using HUnit for unit testing of the new source code in Ampersand. HUnit is a library providing unit testing capabilities in Haskell. It is an adaption of JUnit to Haskell that allows you to easily create, name, group tests, and execute them.

3.3 Requirements

3.3.1 Functional requirements

The functional requirements for ECA for Ampersand are detailed in the SRS; they are also briefly summarized here. Our implementation must

F1 provably implement the desired algorithm.

F2 accept its input in the existing ADL file format.

F3 produce an output compatible with the existing pipeline.

F4 be a pure function; it should not have side effects.

F5 not introduce appreciable performance degradation.

F6 provide diagnostic information about the algorithm to the user, if the user asks for such information.

3.3.2 Non-Functional requirements

The functional requirements for ECA for Ampersand are detailed in the SRS; they are also briefly summarized here. Our implementation must

N1 produce output which will be easily understood by the typical user, such as a requirements engineer, and will not be misleading or confusing.

N2 be composed of easily maintainable, well documented code.

N3 compile and run in the environment currently used to develop Ampersand.

N4 annotated generated code with proofs of correctness or derivations, where appropriate.

N5 automatically fix database violations in the mock database of the prototype.

3.4 Data recording

3.5 Constraints

3.6 Evaluation

Chapter 4

System Test Descriptions

4.1 Black Box Test

4.1.1 Control

4.1.2 Inputs

4.1.3 Outputs

4.1.4 Procedures

4.2 White Box Test

4.2.1 Control

4.2.2 Inputs

4.2.3 Outputs

4.2.4 Procedures

Bibliography

[Joo07] Stef Joosten. AMMBR: A Method to Maintain Business Rules. 2007.