Ampersand
Event-Condition-Action Rules

Software Requirement Specification

Version 1

Yuriy Toporovskyy, Yash Sapra, Jaeden Guo

We acknowledge that this document uses material from the Volere Requirements Specification Template, copyright 1995 - 2012 the Atlantic Systems Guild Limited.

CS 4ZP6
October 9th, 2015
Fall 2015 / Winter 2016

Table 1: Revision History

Author	**Date**	**Comment**
Yuriy Toporovskyy	26 / 09 / 2015	Initial skeleton version

Contents

List of Figures

List of Tables

Chapter 1

Project Drivers

1.1 The Purpose of the Project

A large part of designing software systems is requirements engineering. One of the greatest challenges of requirements engineering is translating from business requirements to a functional specification. Business requirements are informal, with the intention of being easily understood by humans; however, functional specifications are written in formal language to properly capture the attributes of the information system unambiguously. Typically, this translation of business requirements to a formal specification is done by a requirements engineer; this can be an error prone process.

Ampersand is a tool which aims to address this problem in a different way; by translating business requirements written in natural language into a formal specification by means of a compilation process. Even though the business requirements and formal specification are written in entirely different languages, the "compiler guarantees compliance between the two".

Ampersand also provides engineers with a variety of aids which help them to design products that fulfill all of the needs of their clients and the end-users; including data models, service cataogues and their specifications. Requirements engineering is perhaps most important in safety-critical systems; to this end, Ampersand generates modelling aids and specifications which are provably correct.

Ampersand has proven reliable in practical situations, and there have been efforts to teach this approach to business analysts. A large portion of the Ampersand system is already in place; the primary focus of this project is to augment Ampersand with increased capabilities for automation.

For example, consider a system for ordering products online. Ampersand takes as an input statements of business requirements like

Every order must have a customer and a list of products; and the total price on the order must equal the sum of the prices of the products.

These are translated into, among other things, formal rules concerning how the information system must react to changes in its state. The information system may contain a function for manipulating orders. (Ampersand can also generate prototype software models, including functions types like these, from business requirements - but this is not the topic of our contribution). For example,

```
addToOrder : ( o : Ref Order, t : Product )
```

where `Ref x` represenets the type of references to values of type `x`; `Order` and `Product` the types of orders and products, respectively.

Ampersand can generate pre- and post-conditions for this function, based on the business requirements. This constitutes a formal specification of the information system. For example, the above function may have the following specification:

```
{ PRE: o.totalCost = t0 }
addToOrder(o,t): ...
{ POST: o.totalCost = t0 + t.cost }
```

It is proven that for the subset of processes which Ampersand can support, there is an algorithm which will generate the necessary code to satisfy the post-conditions (ie, formal specifications) of each function. However, Ampersand does not yet implement this algorithm. Currently, a user of Ampersand must manually indicate how each violation must be corrected.

In the previous example, the implementation of the function could be as follows:

```
{ PRE: o.totalCost = t0 }
addToOrder(o,t):
  o.orders.append(t);
  o.totalCost = o.totalCost + t.cost;
{ POST: o.totalCost = t0 + t.cost }
```

The first line includes the item in the order, and the second line fixes the violation of the post-condition which would occur without it. Currently this second line would have to be hand-written by the programmer, but the afformentioned algorithm can derive it from the business rules. The main contribution of this project will be to implement the algorithm which generates the code to fix violations.

1.2 The Stakeholders

1.2.1 Ampersand

1.2.2 Requirements engineers

1.2.3 Software engineers

Chapter 2

Project Constraints

2.1 Mandated Constraints

2.1.1 Project philosophy

Ampersand is an existing software project with a very sizable code base. The cost of maintaining poorly-written code can be very high and can outweigh the benefit of the contribution. In order for our code to eventually be merged into Ampersand, it must be maintable: it must be written according to coding practices of Ampersand; it must be well documented, so it can be easily understood by other programmers.

Similarly, our code must not introduce any errors or performance regressions into Ampersand. Our code must satisfy existing tests and additional tests should be written for the new algorithm being implemented. Writing maintainable and well-documented code will help with this goal as well.

2.1.2 Implementation environment

Haskell

The Ampersand code base is written almost entirely in Haskell. Part of the prototype software is written in php and javascript not have to interact with this code base. Our code contribution must be entirely in Haskell.

Haskell software

Ampersand is designed to be used with the Glasgow Haskell Compiler (from here on, GHC) and the associated cabal build system. Ampersand also uses many open source Haskell packages, all available on the Hackage package archive. We may not use additional packages.

GitHub

The Ampersand code base currently lives on GitHub. Our code contributions must also be on GitHub; this will facilitate easy integration of our code into Ampersand. This is especially useful if only parts of our code eventually become integrated into Ampersand - GitHub facilitates this especially.

2.2 Naming Conventions and Terminology

2.3 Relevant Facts and Assumptions

Chapter 3

Functional Requirements

3.1 The Scope of the Work

3.2 Business Data Model and Data Dictionary

3.3 The Scope of the Product

3.4 Functional Requirements

Chapter 4

Non-functional Requirements

4.1 Look and Feel Requirements

4.2 Usability and Humanity Requirements

4.3 Performance Requirements

4.4 Operational and Environmental Requirements

4.5 Maintainability and Support Requirements

4.6 Security Requirements

4.7 Cultural Requirements

4.8 Legal Requirements

Chapter 5

Project Issues

5.1 Open Issues

5.2 Off-the-Shelf Solutions

5.3 New Problems

5.4 Tasks

5.5 Migration to the New Product

5.6 Risks

5.7 Costs

5.8 User Documentation and Training

5.9 Waiting Room

5.10 Ideas for Solutions