Ampersand
Event-Condition-Action Rules

Software Requirement Specification

[**JG: insert:** Version 1]

Yuriy Toporovskyy, Yash Sapra, Jaeden Guo

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

[**JG: replace:**

Chapter 1

Introduction

with:

Chapter 2

Project Drivers

]

[**JG: replace:**

2.1 Project Description

with:

2.2 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 [**YT:** *This is 'proven' because someone told me we have a proof/presentation of the algorithm, but not its implementation. Can anyone find this? We absolutely need a reference to this.*] 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 [**YT:** *Should have a name to refer to the algorithm somehow?*] 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.

2.3 The Stakeholders

2.3.1 Ampersand

2.3.2 Business requirements

Chapter 3

Project Constraints

3.1 Mandated Constraints

3.2 Naming Conventions and Terminology

3.3 Relevant Facts and Assumptions

Chapter 4

Functional Requirements

4.1 The Scope of the Work

4.2 Business Data Model and Data Dictionary

4.3 The Scope of the Product

4.4 Functional Requirements

Chapter 5

Non-functional Requirements

5.1 Look and Feel Requirements

5.2 Usability and Humanity Requirements

5.3 Performance Requirements

5.4 Operational and Environmental Requirements

5.5 Maintainability and Support Requirements

5.6 Security Requirements

5.7 Cultural Requirements

5.8 Legal Requirements

Chapter 6

Project Issues

6.1 Open Issues

6.2 Off-the-Shelf Solutions

6.3 New Problems

6.4 Tasks

6.5 Migration to the New Product

6.6 Risks

6.7 Costs

6.8 User Documentation and Training

6.9 Waiting Room

6.10 Ideas for Solutions