Ampersand
Event-Condition-Action Rules

Software Requirement Specification

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

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 unambiguously capture attributes of the information system. Typically, this translation of business requirements to a formal specification is done by a requirements engineer, which can be prone to human error.

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 (Figure 1.1); including data models, service catalogs and their specifications. Requirements engineering is perhaps most important in safety-critical systems; to this end, Ampersand generates modeling 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

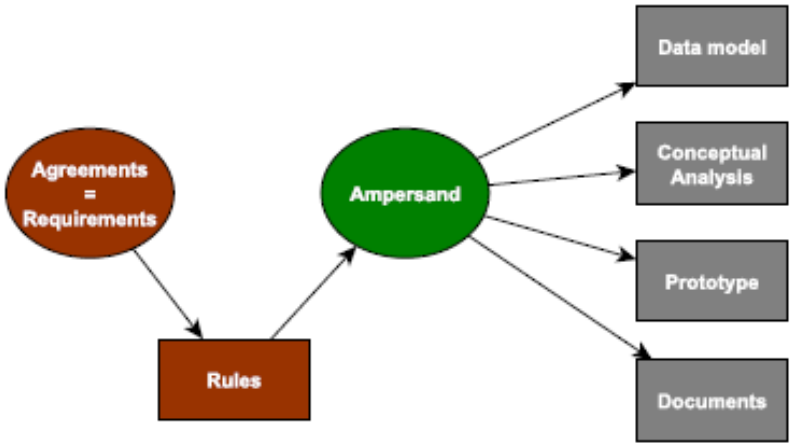

Figure 1.1: Ampersand produces a set of artifacts based on user's requirements.

> *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. Ampersand describes these rules as a set of Event-Condition-Action Rules (referred to as the E.C.A rules hereafter) that specify when an event takes place under a given condition, what Action the software must take. 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` represents 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 Designers

Ampersand designers include Dr. Stef Joosten, Dr. Sebastian Joosten and Dr. Wolfram Kahl, who in addition to supervising the development of E.F.A are also Ampersand contributers. Dr. Stef Joosten is the designer of the Ampersand method which have been successfully implemented in various public sectors. The interest stemming from these three individuals are not only professional but also personal. Ampersand has been nurtured by a great deal of patience and dedication. The designers of E.F.A by extension are Ampersand contributors and are affected by its outcome, the piece of Ampersand we will build is a necessary component required for completion of undergraduate and is designed to test the capabilities of an undergraduate team. These individuals are the foundation of the belief system by which Ampersand is built. The Ampersand model is built on the belief that technology is highly incorporated in the use of business and affects the life of every individual. It is the belief that information system can and will be designed and built at an affordable price and perfectly compliant with the functionalities that users desire. This perspective treats information systems as a commodity of the real world that can be traded, altered, and used by any individual who is willing to learn how to use it.

1.2.2 Requirements engineers

Requirement engineers are responsible of translating business requirements into project specifications. Ampersand provides requirement engineers with the freedom to explore

their creativity rather than be tried down by technical minutes. The ability to delegate time consuming tasks such as manually checking over technical specifications would increase the efficiency of the task at hand as well as decrease the number of hours required for completion. An added benefit which engineers receive is the ability to confirm that the system they have designed matching the specifications given to them by the client. E.F.A provides the ability to ensure no contradictions are found within the design by automatically fixing inconsistency that may arise.

1.2.3 Architects & Functional Specification Designers

System architects, similar to functional specification designers are responsible for various designs which affect procedure and processes in a place of business. Ampersand has the ability to confirm that the design provided by the Architect matches any building regulations that must be meet. Regardless of whether the specification is for safety or aesthetics, the product must meet the client's requirements and minimal legal regulations. Ampersand is able to compose rules and regulations for data which fully incorporates both without contradiction. The addition of E.F.A allows Ampersand to automate correction on data which may contradict rules specified for a project. Architects and functional specification designers will be able to take advantage of the numerous resources Ampersand can provide such as automatic generation of databases within prototypes along with tables and relational schema's. Furthermore, there is a full range of graphics that stakeholders can choose from in order to provide visual display of their design which user may find more friendly and easy to understand.

1.2.4 Process Innovators

Process innovators covers a broad range of individuals that would not commonly be considered stakeholders. This broad category includes anyone who provides input for business processes which includes both sides of a business agreement, in addition to individuals who play a role in seeing the completion of a business process. Ampersand allows continuous input as a living system and has the ability to reorganize requirements when new constraints are added, and E.F.A has the ability to assure the correctness of system information relative to the requirements it has been given. With E.F.A's ability to verify inconsistencies and automate their correction, changes to design and restrictions due to unforeseen circumstances can be added without a system overhaul. The efficiency in which Ampersand can provide has the potential to dramatically decrease flaws that commonly litter information system.

1.2.5 Software engineers

Chapter 2

Project Constraints

The main constraint of this project is time; this is limited to the 8 month period in which we have to build it. It is difficult to perfect an internal construction without a large system in such a short amount of time. But time restraint is an obstacle which all projects face, because it is not limitless. Another major constraint that most project face that this project does not require is the financial constraint where projects become difficult to maintain and often halt in their progress due to monetary constraints. The lack of monetary constraints due to the open source software opens up a broad base of opportunities for future improvements and limitless contributions from the community.

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 maintainable: 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.

These motivations are very important to Ampersand and are considered primary goals alongside the obvious goal of producing code implementing the desired feature.

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 but we likely do 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.

Graphviz

Graphiv is open source graph visualization software, which has the ability to visually represent information in the form of charts and graphs. It contains various designs from which the user is able to select. This is used to take descriptions of graphs in simple text and create diagrams. Ampersand generates reports about the input system using graphs and charts, so this is one of the basic components required to run Ampersand.

XAMPP

Ampersand generates prototype websites based on business requirements; to run these, some web server software is needed. XAMPP stands for cross (i.e., X) platform Apache distribution containing MySQL, PHP and Perl. XAMPP is the most convenient way to set up a working database for Ampersand and access .php pages.

2.2 Naming Conventions and Terminology

ECA Stands for Event-Condition Action. The rule structure used for data bases and commonly used in market ready business rule engines. ECA rules are used in Ampersand to describe how a database should be modified in response to a system constraint becoming untrue. Event-Condition Action rules have the following structure:

```
when <some event has happened>
and if <some condition is fulfilled>
    do <this activity>
```

EFA Stands for "ECA (see above) for Ampersand".

Functional specification A *formal* document which details the operation, capabilities, and appearance of a software system.

Requirements engineering The process of translating business requirements into a functional specification.

Business requirements Requirements which exist due to some real world constraints; ie, financial, logistic, physical or safety constraints.

Business rules See *Business Requirements*.

Natural language Language written in a manner similair to that of human communication; language intended to be interpreted and understood by humans, as opposed to machines.

ADL Stand for "Ampersand Definition Language". From a given set of formally defined business requirements, ADL generates a functional specification consisting of a data model, a service catalog, a formal specification of the services, and a function point analysis. An ADL script acts as an input for Ampersand.

2.3 Relevant Facts and Assumptions

Chapter 3

Functional Requirements

3.1 The Scope of the Work

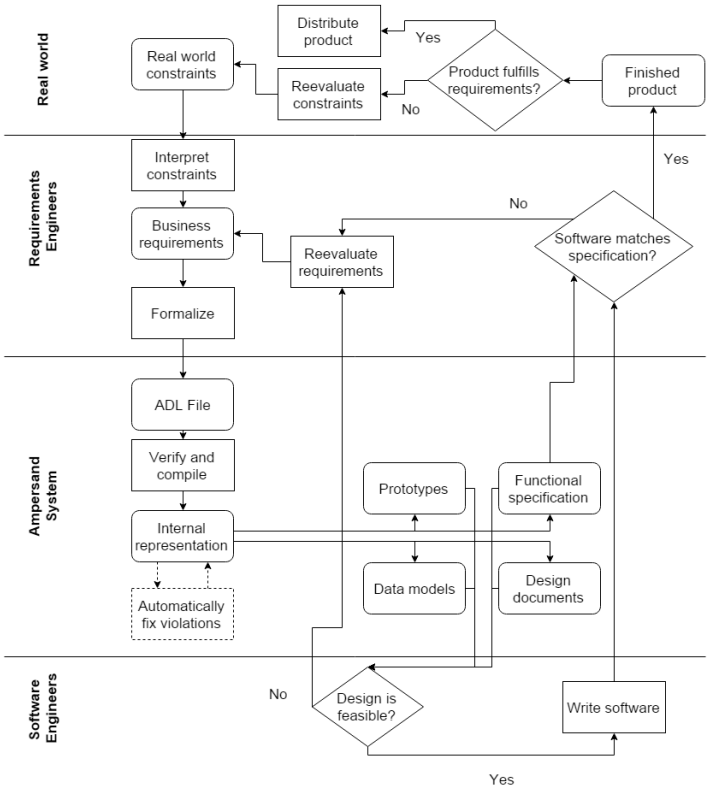

Figure 3.1: The role of Ampersand in the software design cycle

Figure 3.1 is a simplified view of the software design cycle, intended to highlight

the role of Ampersand in this cycle. This view omits many of the uses of the design artifcats generated by Ampersand; instead it focuses mainly on the primary purpose, which is to help create a finished software system. The contribution of this project is denoted with dashed lines.

3.2 Business Data Model and Data Dictionary

3.3 The Scope of the Product

3.4 Functional Requirements

3.4.1 Input

3.4.2 Correct Translation of Input

3.4.3 Output

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

The implementation should be completely the work of the group and should not copy from any existing software or open source code.

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

Upon final review by the client and intensive testing, if the client is satisfied by the quality of code and its maintainability, the implementation will be made part of the production stream.

5.6 Risks

- The new code must not introduce any errors or performance regressions into Ampersand.
- The code must satisfy existing tests and additional tests written for the new algorithm being implemented.

5.7 Costs

Currently the software is open source maintained by Tarski Systems. All the software used in Ampersand are also open source There will no change in cost as a result of our implementation. The client (Tarski Systems) will be responsible for managing the cost of maintenance of the software in the future.

5.8 User Documentation and Training

5.9 Waiting Room

5.10 Ideas for Solutions