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" ([Joo07]). Even though the business requirements and formal specification are written in entirely different languages, the "compiler guarantees compliance between the two" ([Joo07, 2]).

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 ([Joo07]).

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

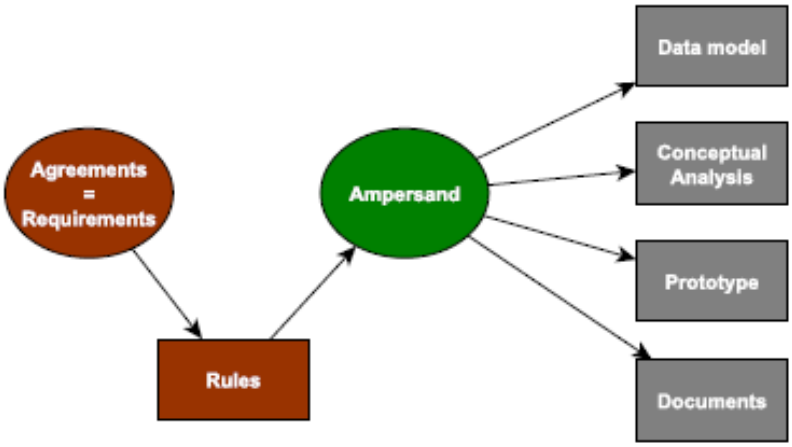

Figure 1.1: Ampersand produces a set of artifacts based on user's requirements.

> *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 & Software Engineers

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.

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 [xam] 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 ([Mic10, 8–9]):

```
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 Stands for "A Description Language" ([Joo07, 13]). From a given set of formally defined business requirements, Ampersand 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. An ADL file consists of a plain ASCII text file.

Ampersand Ampersand is the name of this project. It is used to refer to both the method of generating functional specification from formalized business requirements, and the software tool which implements this method.

2.3 Relevant Facts and Assumptions

Chapter 3

Functional Requirements

3.1 The Scope of the Work

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. Note that it is isolated to a process completely internal to Ampersand. It should not affect the interface to Ampersand.

Table 3.1: Description of entities present in figure 3.1

Entity	**Type**	**Description**
Real World	Actor	Everything outside of the software system
Requirements engineers	Actor	
Ampersand system	Actor	The Ampersand software tool (2.2) and all of its associated resources.
Software engineers	Actor	
ADL File	Object	The input to Ampersand, see section 2.2
Real world constraints	Object	Driving force for software system
Finished product	Object	The finished software system, including supporting documentation and training/services

Table 3.1: Description of entities present in figure 3.1

Entity	**Type**	**Description**
Business requirements	Object	Important constraints seperated into logical units, see section 2.2
Internal representation	Object	The internal representation on which all computations are done; this is never exposed to the outside world directly
Prototypes	Object	Resource generated by Ampersand
Data models	Object	Resource generated by Ampersand
Design documents	Object	Resource generated by Ampersand
Functional specification	Object	Resource generated by Ampersand
Distribute product	Process	Physical deployment of software and services
Reevaluate constraints	Process	Constraints deemed not valid and must be reevaluated
Interpret constraints	Process	Turn constraints into business requirements
Formalize	Process	Rewrite something in formal language, usually from natural language
Reevaluate requirements	Process	Requirements deemed not valid and must be reevaluated
Verify and compile	Process	Verification consists of typechecking (see 2.2)
Automatically fix violations	Process	Insert the necessary code so that generated code will not violated business constraints
Write software	Process	
Product fulfills requirements?	Decision	Is the product deemed sufficient or accepted?
Software matches specification?	Decision	Does the software meet all of the requirements set forth by the formal specification?
Design is feasible?	Decision	Does the software engineer think that the design can be implemented in the desired timeframe?

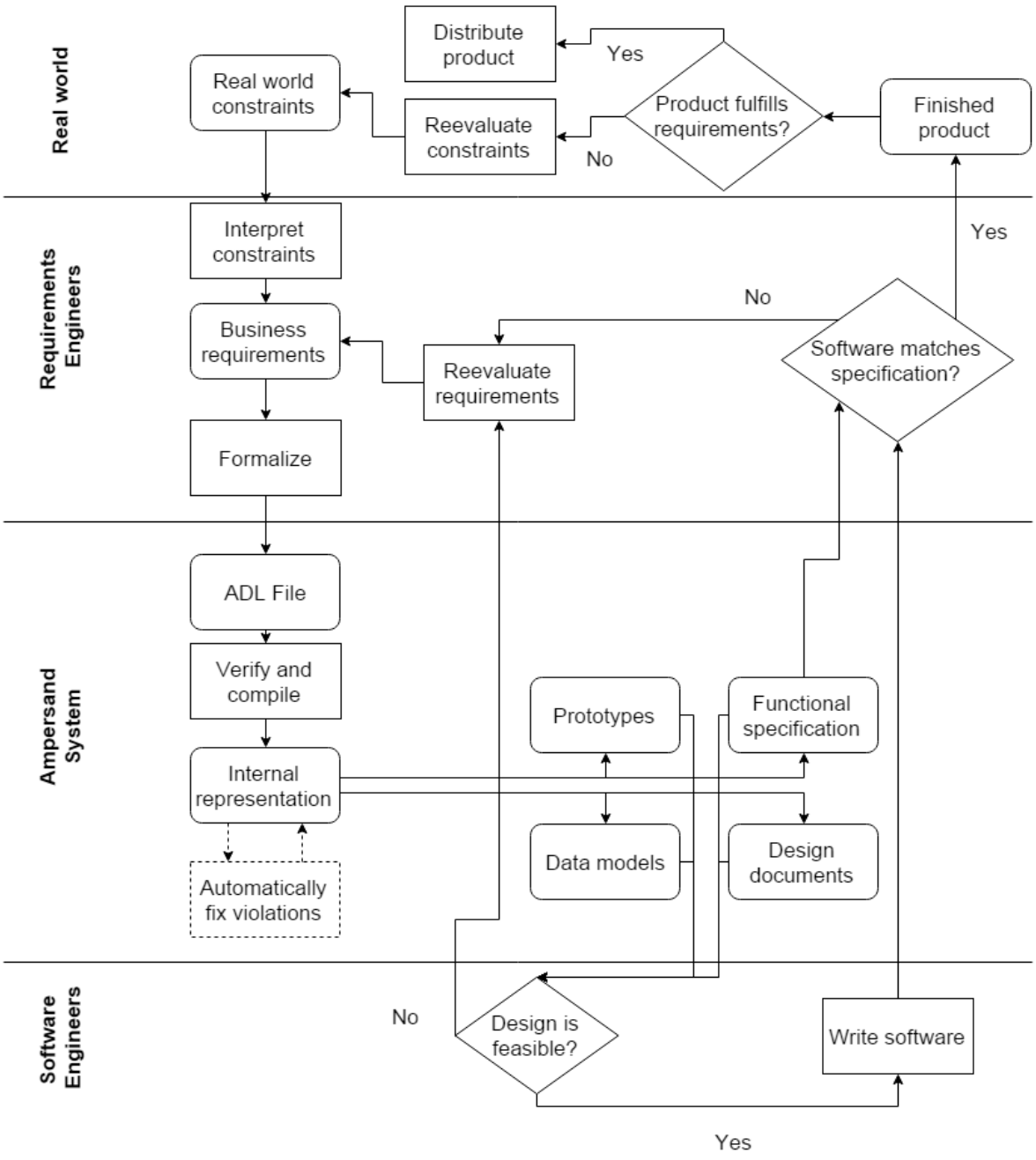

Figure 3.1: Business process diagram representing the role of Ampersand in the software design cycle

3.2 Functional Requirements

Our code contribution must provably implement the desired algorithm.

Our implementation must accept its input in the existing ADL format.

Our implementation must produce an output compatible with the existing pipeline.

Our implementation must be a pure function; it should not have side effects.

Our implementation must not introduce appreciable performance degredation.

Our implementation must provide diagnostic information about the algorithm to the user, if the user asks for such information.

Requirement # : 1 **Priority:** 4
Description: Users of Ampersand shall be able to insert and modify ECA rules at any time.
Rationale: To allow user to change the requirements in accordance to the changing needs.
Fit Criterion: The Data in the database preserves these rules and maintains consistency.

Chapter 4

Non-functional Requirements

Non-functional requirements are fundamental to the success of any product on the market. This type of requirement is especially important to this project as it heavily relates the needs of the clients to the changes that need to be made to the final product before it is ready for the public. The human element guides the principals introduced in non-functional requirements, and for this project, it is motivated by a simple perspective. Tools should make a users job easier to do, and for that to happen the user must be able to under how to use it.

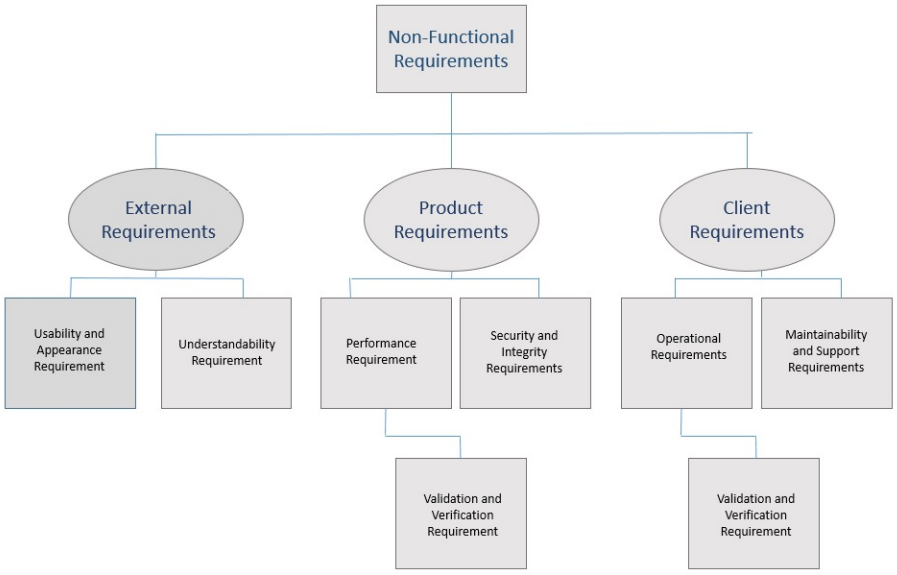

Figure 4.1: Tree of non-functional requirements as it relates to E.F.A.

4.1 Look and Feel Requirements

4.2 Usability and Humanity Requirements

E.F.A. is built as an internal structure of Ampersand and does not have a graphical interface for users. Ampersand uses the command prompt to compile user prototypes which includes various graphical representations and webpages. It is very unlikely that users will be able to distinguish E.F.A. from Ampersand, it would be similar to someone attempting to distinguish a persons hand as a separate entity from that person. E.F.A. may not look very personable, or user friendly, but it will hopefully relieve some frustration and make the user feel more at ease with their design because if things do not line up, there is something that can automatically fix it for them. It would be another thing users would not have to worry about when designing a new system, this feeling of ease could extend to smooth the interactions between the designer and the client as it goes through various cycles of overhaul. The user will be notified of any changes that have been made to their design and from there Ampersand provides options of how much of the changes the user would like to keep, fix themselves, or add to a repository of changes to keep in mind for future instances. Thus E.F.A. can return corrections according to what the user specifies.

This product shall be used by any individual who is willing to dedicate the time to learning the basics of the Ampersand model and wishes to design an information system. The time it take to learn Ampersand is truly up to the user, a normal individual may take anywhere from a week to a month. While an engineer who has a strong background in mathematical logic or minor experience in programming will be able to use Ampersand efficiently within a week. The most pronounced problem concerning Ampersands usability is that its limited to two languages: English and Dutch. Those who are not fluent in neither of these languages will have great difficulty learning the Ampersand system.

However, since Ampersand is a successor of CC, [Mic10, p.10] and similarly it could be used to create consensus between groups of stakeholders with strong diverging opinions on the same matter. Once the prototype is generated, it can be presented to different individuals who may wish to conduct their own investigation concerning the best course of action. After the creation of the prototype, common users are able to manipulate various aspects of a project due to an autofill feature.

Figure 4.2: An example of what Ampersand users see when they compile a prototype, each tab and figure they have created can be modified.

4.3 Understandability

What E.F.A. does is simple to understand, it fixes logical inconsistencies with minimal if any input from the user. It looks for contradictions and violations, an overly simplistic but accurate example would be: a chair is red, a chair is blue, and there is only one chair. That chair cannot be blue and red simultaneously, to fix the problem either another chair is added or one of the colour conditions is eliminated. What is eliminated may be based on what is more important for the system and the user. In the previous example, if it is most important that there is only one chair, then one of the colours statements is erased. Otherwise, E.F.A. could add a new chair, now ever if adding a new chair decreases the efficiency of the system because it must keep track of two chairs now. It may pick a chair colour depending on another criteria that is not directly part of this problem. Such as if most individuals who may use the chair prefer it to be red rather than blue. That may be an external condition that Ampersand has been programmed to keep in mind.

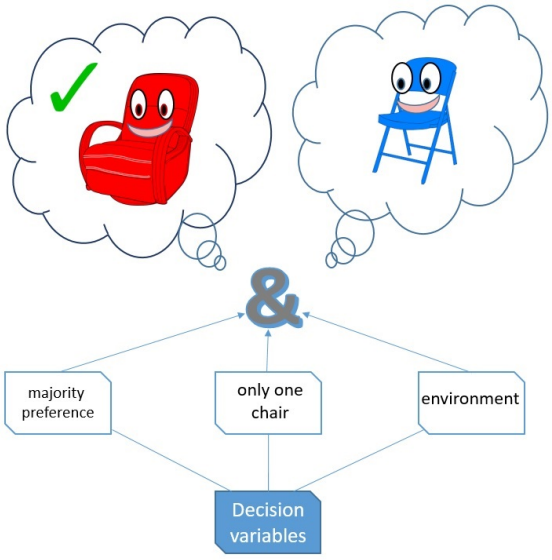

Figure 4.3: Simplistic diagram of decision making process, for full example of decision making process see. Figure 5.

4.4 Performance Requirements

Ampersand can efficiently process a large amount of information in a short amount of time, the process that E.F.A. requires will be added onto Ampersands process time. The duration of time it would take to verify data inconsistencies would depend on the complexity of the system design and the number of corrections it may take to update the system to a valid prototype according to specification. The current upper bound is expected at 120 seconds, or 2 minutes, the correction process is terminated if exceeds beyond this threshold for simple Ampersand models. The prototypes termed simple models are tasks that can be accomplished by an individual within a short amount of time, such as selecting the best actor/actress to play lead according to experience and audience appeal. Simple models are able to take into consideration not only the task at hand but the context and environment to which the task must be completed in. Performance is measured on speed, but also on validation of limitations set forth by the user; it will not matter how quickly E.F.A. can do something if it does it incorrectly and creates more work for the user. Currently, the quantification of the desired accuracy of the results produced is unknown, as E.F.A. is in its initial stages. Furthermore, the allowable time between failures is unknown and untested.

E.F.A. can be a great tool, but in case of failure, in the unlikely event that errors are propagated through the system and no one through any cycle of development catch it. If the designer puts full faith in E.F.A. it could be catastrophic, especially if mistakes are not caught and products proceed into the production stage. However, Ampersand

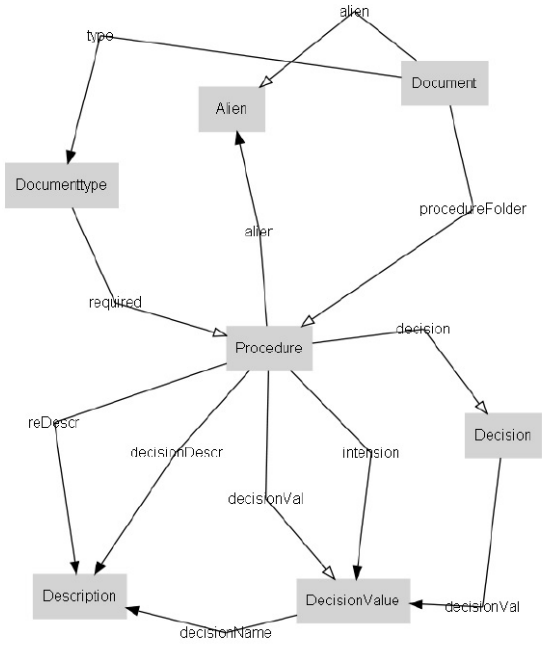

Figure 4.4: This is an example of Ampersand's complete decision process. Source : Rule Based Design

possess an internal structure to detect inconsistencies and allows for manual correction. E.F.A. although important is not the last line of defense, this secondary system greatly minimizes the chances of complete failure. The likely if E.F.A. completely fails, is that the secondary system detects errors and compels users to manually correct data ([Mic10, 153]).

4.5 Operational and Environmental Requirements

It is not feasible to predict an accurate workload capacity for E.F.A. for its state, however, E.F.A. is made to handle Ampersand loads interactively and is not designed to accumulate stored data but rather return the user a modified version of the input they submitted. E.F.A. is meant to operate locally within Ampersand, there no limit to the number of users that can use this product. Extensive maintenance is not required, although minor adjustments and additions may be necessary for further improvement. As Ampersand grows in popularity, it will not be necessary for E.F.A. to grow in proportion to Ampersand to remain useful. Due to this it is difficult to determine the longevity of this project, as it could remain until the client determines the current model is no longer useful.

Any system that is capable of running Ampersand is automatically compatible with E.F.A., currently those system are limited to Windows OS 7 and up. There are very few physical conditions required to maintain E.F.A. and they are the basic conditions required to keep most computer system running. This includes minimal moisture, and temperatures no lower than 35 (degree farenheight) or 1.7 (degree celcius) as computer hardware of

4.6 Maintainability and Support Requirements

Maintenance is a key requirement from our clients as non-maintainable code takes far too long to decipher and upgrade. The code is required to be well documented and easy to understand so that any programmer who wishes to make additions or expand upon what is already there, can do so easily without struggle. Well documented code creates a sense of cohesion among individual parties participating on the same project. In order to meet the maintainability, E.F.A. must all requirements, such that each specification is traceable back to the motivation for its existence and the purpose it drives ([Joo07, 2]). Although there are various groups, such as Dr. Joosten and his team who are available to assist individuals and businesses with any part of Ampersand, our team believes that given a specific enough error message, an individual may not require outside assistance. Moreover, E.F.A. will be accompanied with documentation that will have various examples for the user to explore. Part of the reason that non-functional requirements as essential to E.F.A. concerns the requirements for interfacing with Adjacent Systems. Thus features concerning how implementation is done which is normally based on the choice of the designers is a functional requirement when it comes to E.F.A.

4.7 Security and Integrity Requirements

4.8 Validation and Verification Requirements

4.9 Legal Requirements

The implementation must eventually be included in Ampersand, which is licensed under GPL3. To comply with this license, all of the implementation code must be either written by us so we may license it under GPL, or must already be licensed under GPL, or a compatible license, by its original author. We do not plan to use existing code, other than as a reference.

Chapter 5

Project Issues

5.1 Open Issues

N/A

5.2 Off-the-Shelf Solutions

No off the shelf solutions exist for this project.

5.3 New Problems

N/A

5.4 Tasks

- Analyze and the existing software code base and do an impact analysis of our project on the existing software.
- Propose a solution to the supervisor and product owner.
- Implement the solution and provide the annotated source code to the supervisor and the product owner for a review.
- Incorporate any changes suggested by them and create a pull request in the main Ampersand repository upon successful completion of the task.

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. This process is quite simple due to the nature of the project; the core development team of Ampersand is quite small, and the project is hosted on GitHub; so our migration will consist of submitting a pull request to the Ampersand repository.

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 Ampersand software system is open source and maintained by Tarski Systems. All the software subsystems 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. All software used in the development of E.F.A (GHC, LaTeX, etc.) are open source as well; there is no cost requirement for any component used.

5.8 User Documentation and Training

5.9 Waiting Room

5.10 Ideas for Solutions

Bibliography

[Joo07] Stef Joosten. Deriving functional specifications from business requirements with ampersand. *CiteSeer*, 2007.

[Mic10] Stef Joosten; Lex Wedemeijer; Gerard Michels. *Rule Based Design*. Open Universiteit Nederland, December 2010.

[xam] Xampp. `https://www.apachefriends.org/index.html`. Accessed: 2015-10-11.