Ampersand
Event-Condition-Action Rules

Software Requirement Specification

Version 0

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
Yuriy Toporovskyy	30 / 09 / 2015	Project drivers, description and added project diagram and project flow chart
J Guo	09 / 10 / 2015	Update: Non-Functional first half 4.1-4.3, added to 1.2.2, completed 2.2
J Guo	13 / 10 / 2015	Update: Figures added for Non-Functional 4.1-4.7, Non-Functional second half 4.4-4.7 half, added Functional 3.3 - System requirements and diagram figure, & Section 5.8
Yash Sapra	12/ 09 / 2015	Non-Functional - legal requirements, Functional - User Requirements, tasks, risks and chapter 5.
Yuriy Toporovskyy	13 / 10 / 2015	Initial round of editing

Contents

List of Figures

List of Tables

Project Time Table		
Projected Finish Date	**Milestone**	**Actual Finish Date**
09/10/2015	EFA SRS version 0	13/10/2015
19/10/2015	EFA SRS version 1	22/10/2015
23/10/2015	Proof of Concept Demonstation	—
—	—	—
—	—	—
01/04/2016	EFA Complete	—

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

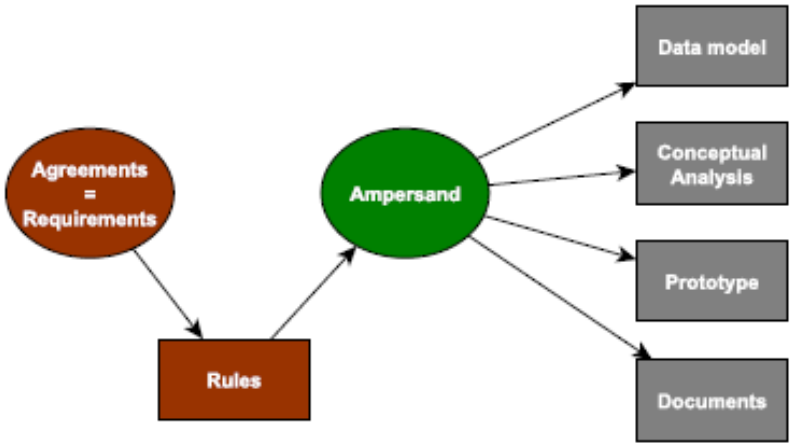

Figure 1.1: Ampersand produces a set of artifacts based on user's requirements.

> *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.*

The requirements engineer formalizes this requirement as a rule in Ampersand, which is a constraint on the data in the database. These constraints are called invariants, because they are meant to remain satisfied at all times during runtime.

The Ampersand compiler translates each rule into Event-Condition-Action Rules (referred to as ECA rules hereafter). An ECA-rule specifies the action to be taken by the system when an event takes place under a given condition. In Ampersand, the purpose of an ECA-rule is to restore invariants, i.e. to change data in the database such that the constraint becomes true again, after having been violated. Currently, an information system generated by Ampersand produces runtime events that signal violations of the constraints.

It also contains the ECA rules that are generated by the Ampersand compiler – information about ECA rules is propagated ["propagated" —DS] through the pipeline. ["pipeline"? —DS] of Ampersand. However, these ECA rules are not being called in order to restore violations automatically in the current version of Ampersand. This is due to the fact that it is currently unclear how to incorporate the functionality of ECA rules into the information system generated by Ampersand.

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 (i.e. ["i.e." —DS] 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 aforementioned ["aforementioned" —DS] 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.

[You may want to simplify the introduction a bit to make the purpose more clear. —DS]

1.2 The Stakeholders

1.2.1 Ampersand Designers

Ampersand designers are Dr. Stef Joosten and Dr. Sebastian Joosten, who in addition to supervising the development of EFA ("Event-Condition-Action" rules for Ampersand, see 2.2) are also Ampersand contributors. ["contributors" —DS] Dr. Stef Joosten is the designer of the Ampersand method which have been successfully implemented in various public sectors. These two individuals have not only professional interest in Ampersand, but also personal interest.Ampersand has been nurtured by a great deal of patience and dedication. The designers of EFA 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 our undergraduate degrees and is designed to test the capabilities of an undergraduate team.

1.2.2 Requirements engineers, Architects and Functional Specification Designers

Requirement engineers such as Dr. Joosten, are responsible of translating business requirements into project specifications. Ampersand provides requirement engineers with the freedom to explore their creativity rather than be tied down down by technical minutea ([Mic10]). These job titles listed consist of individuals who are end-users and hold stake in this project because it acts similar to an editor that will oversee the minutia of creating an information system. Furthermore, the editor will have the ability to prove if the system that the designer has built is correct according to client specification ([vdw11]).

[Break up this wall of text into paragraphs. —DS]

[This entire section (1.2) has been unnecessarily verbose. A simple explanation of who the stakeholder is and why they hold a stake would be enough. —DS]

Chapter 2

Project Constraints

The main constraints of this project is time and the programming language that this project must be written in. The maximum allotted time for this project is 8 months and during this time we must create a program from a partial algorithm, test it externally for all possible bugs and then implement it within a much bigger system. The other constraint is the use of Haskell as it is the language choosen by Ampersand designers, any additions to Ampersand must be in Haskell.

[Lack of a constraint is not a constraint. —DS]

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. 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.

Primarily, our code must be well-documented, backwards-compatible and mathematically provable. 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.

[So the code must be well-documented and backwards-compatible? —DS]

2.1.2 Implementation environment

Haskell

The Ampersand code base is written almost entirely in Haskell ([Joo]). Part of the prototype software is written in PHP and Javascript. However, we 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 ([GHC]) , or GHC, and the associated Cabal build system (see `ampersand.cabal` from [Joo]). Ampersand also uses many open source Haskell packages, all available on the Hackage package archive. If we are to introduce dependencies on additional packages, [This doesn't sound like a constraint —DS] these must be well-justified and negotiated with the Ampersand developers.

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

Graphviz is an open source graph visualization software, which can 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 Graphviz 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 recommended way [Convenient, but not the only way? Then is it really a constraint? Is there a specific reason you *must* use XAMPP? —DS] to set up a working database for Ampersand and access .php pages. XAMPP is the only bundled software that has all the

necessary components that Ampersand needs. Alternatively, users can install individual components, however it is very tedious to adjust individual ports and Ampersand may not automatically detect these components to use as part of its internal system. Experienced programmers who wish to install individual components are welcome to do so.

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]): [A lot of these are far too specific. I do not need to know implementation details to understand the abstract requirements. —DS]

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.

Business requirements Requirements which exist due to some real world constraints; i.e. ["i.e." —DS] financial, logistic, physical or safety constraints.

Business rules See *Business Requirements*.

EFA Stands for "ECA (see above) for Ampersand". This term is used to refer to the contribution of this project.

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

Natural language Language written in a manner similar ["similar" —DS] to that of human communication; language intended to be interpreted and understood by humans, as opposed to machines.

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

2.3 Relevant Facts and Assumptions

[Why is this section here and empty? —DS] This project assumes that those who may use Ampersand and by extension this project, are computer literate and work in the field of business or engineering. We assume our users are knowledgeable about computer systems and able to understand and follow error messages. Furthermore, we assume that any system that can run Ampersand, can run EFA.

2.3.1 Error Detection

Ampersand has a built in type-error detector that will reject scripts if they are not written in the appropriate order (e.g. misalignment of syntax, missing logical inconsistencies, missing references to objects, missing rules, missing headers, etc). Thus the error-detection system for this project only focuses on error detection within the internal operations of EFA. It does not extend beyond EFA output.

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 artifacts ["artifacts" —DS] 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.

[This table needs to have its columns adjusted. The description should be the widest column. —DS]

Entity	**Type**	**Description**
Real World	Actor	Everything outside of the software system
Requirements engineers	Actor	Stakeholder: end-users
Ampersand system	Actor	The Ampersand software tool (2.2) and all of its associated resources.
Software engineers	Actor	Stakeholder: end-users

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
Business requirements	Object	Important constraints separated ["separated" —DS] 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
Re-evaluate ["Re-evaluate" —DS] constraints	Process	Constraints deemed not valid and must be ["re-evaluated" —DS] re-evaluated
Interpret constraints	Process	Turn constraints into business requirements
Formalize	Process	Rewrite something in formal language, usually from natural language

Re-evaluate ["Re-evaluate" —DS] requirements	Process	Requirements deemed not valid and must be re-evaluated ["re-evaluated" —DS]
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	
Error detection	Process	Is this product able to detect inconsistent data errors with new contributions? Is the error detection robust enough to catch process associated bugs such as cycling, and deadlocks?
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?

Table 3.1: Description of entities present in figure 3.1

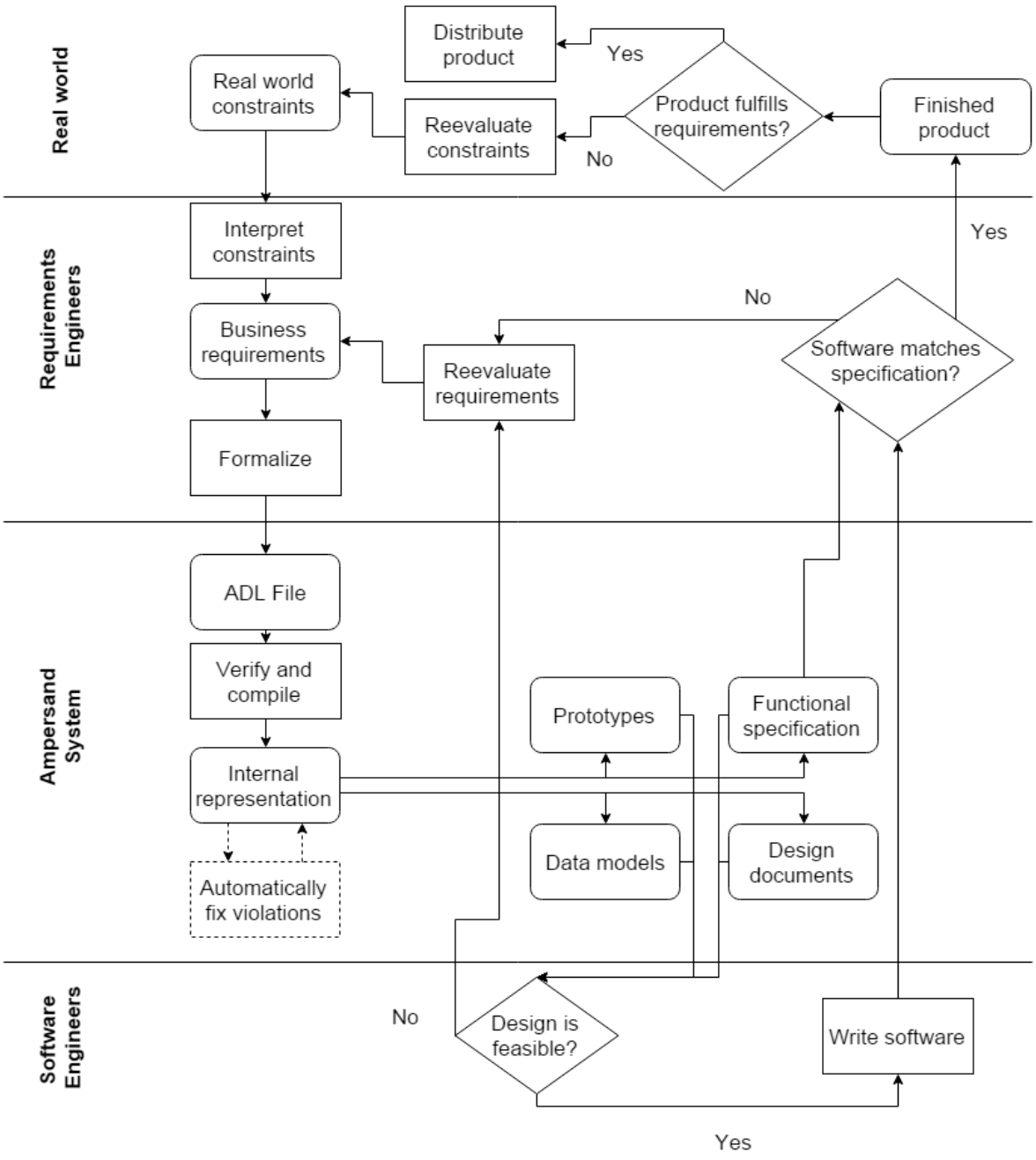

Figure 3.1: Business process diagram representing the role of Ampersand in the software design cycle

3.2 The Scope of the Product

EFA require formal proofs and documentation that are provable in addition to its implementation; the implementation requires the final product to be able to implement an algorithm in Haskell that restores rules made by the user (i.e., invariants) and correct any data which have violated these rules. A secondary component of this project is the implementation of ECA (Event-Condition Action) Rules used for reactive rule-based systems which has the ability to discover and systematically respond ["respond" – I'm no longer looking at spelling or grammar. It is up to you to find the rest of the mistakes (if any) before the next revision is submitted. —DS] to triggers for events in a timely manner. ([Bon07, p.182]) Currently, implementation and architecture of ECA rules systems are intensively studied in Active Database Management Systems. ([Bon07, p.202]) There are various problems that may arise during implementation of the ECA rules translation, what those problems may be are unforeseeable at this stage of development.

3.3 Functional Requirements

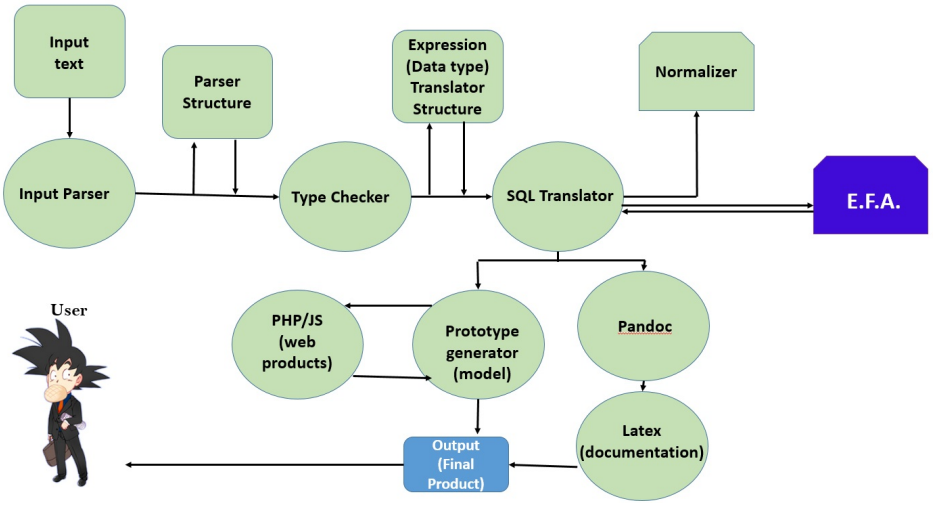

Figure 3.2: Diagram of how EFA fits into Ampersand

Ampersand as a whole is a large system with many moving parts. However, for the most part, these different subsystems have little influence on our contribution. As highlighted in figure 4.1, our contribution will be implemented as a program transformation on the internal data structures of Ampersand. Our contribution will not interact with the parsing of input; or with the generation of design artifacts.

Therefore, we omit functional requirements that are not related to our contribution from this section; that is, those requirements which are related only to subsystems with which we will not be interacting. These requirements are included in section 5.9.

Our functional requirements can be summarized as follows. Our implementation must

- provably implement the desired algorithm.
- accept its input in the existing ADL file format.
- produce an output compatible with the existing pipeline.
- be a pure function; it should not have side effects.
- not introduce appreciable performance degredation.
- provide diagnostic information about the algorithm to the user, if the user asks for such information.

3.3.1 User Requirements

Requirement	U1
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.
Originator	Users of Ampersand
Fit Criterion	The Data in the database preserves these rules and maintains consistency.
Priority	4 - High
Supporting Materials	—

Requirement	U2
Description	Error reports for users
Rationale	To give users a detailed report of bugs at the users request
Originator	Users of Ampersand
Fit Criterion	Usablity and Understandability
Priority	4 - High
Supporting Materials	—

3.3.2 Product and System Requirement

Requirement	S1
Description	Error handling on new contributions
Rationale	Must produce user-friendly error messages that are specific to the error it has caught and able to catch errors with new additions to the information system.
Originator	Client/Designers
Fit Criterion	Able to compile using GHC
Priority	5 - Highest
Supporting Materials	Ampersand Documentation (Rule Based Design [Mic10])

[This is a constraint, not a functional requirement. —DS]

Requirement	S2
Description	The code contribution must be a patch to the existing Ampersand system
Rationale	Cohesion of new product into existing system; necessary for EFA to be added to source code of Ampersand
Originator	Client/Designers
Fit Criterion	Able to submit pull request on GitHub
Priority	5 - Highest
Supporting Materials	Ampersand Documentation (Rule Based Design [Mic10])

Requirement	S3
Description	Able to recognize input in the existing Ampersand format
Rationale	Necessary for ECA rules to be added; the algorithm needs an input, which must take the form of existing Ampersand datatypes
Originator	Client/Designers
Fit Criterion	Able to take input from existing Ampersand datatypes and return a value
Priority	5 - Highest
Supporting Materials	Ampersand Documentation (Rule Based Design [Mic10])

Requirement	S4
Description	Able to translate input into ECA rules
Rationale	Major portion of project and EFA is useless without it
Originator	Client/Designers
Fit Criterion	Outputs ECA rules in proper format for F-Spec (i.e., SQL translator)
Customer Satisfaction	5 - Highest
Priority	5 - Highest
Supporting Materials	Rule Based Design ([Mic10]); ECA for Web Development

Requirement	S5
Description	Use of Haskell
Rationale	The use of a functional programing languages requires that this program be a pure function and does not have side effects
Originator	Stakeholder
Fit Criterion	This is necessary for our system design and the incorporation of core programming principles
Customer Satisfaction	5 - Highest
Priority	5 - Highest

Requirement	S5
Supporting Materials	—

[What about error handling on the new contributions? Where's the functional requirement related to: "be a pure function; it should not have side effects." and "provide diagnostic information about the algorithm to the user, if the user asks for such information."? —DS]

Chapter 4

Non-functional Requirements

[You just explained what NFRs are, but did not contribute anything. Perhaps just introduce the diagram. —DS]

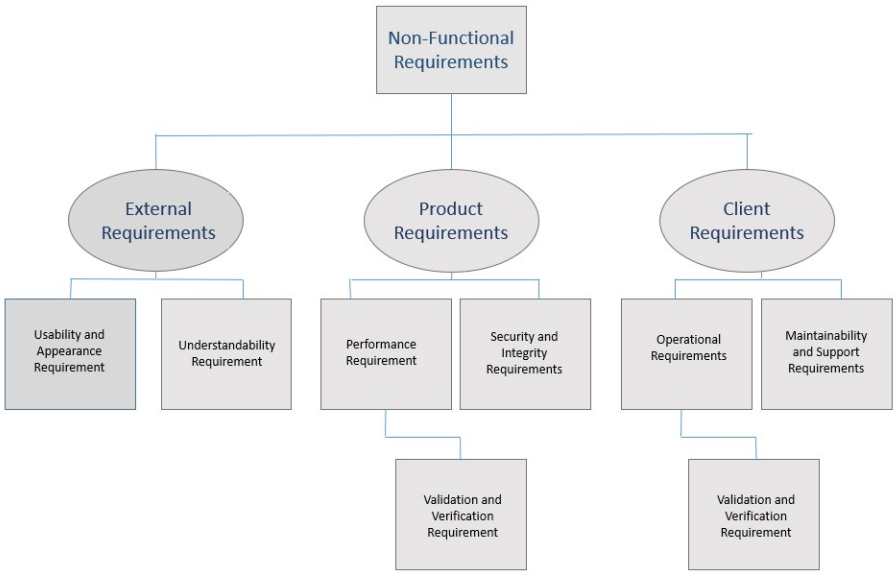

Figure 4.1: Tree of non-functional requirements as it relates to EFA

4.1 Usability and Humanity Requirements

This product can be used by anyone who uses Ampersand, but it is geared to individuals with an engineering or business background such as university students and graduates. Anyone who is 16 years of age and older is able to use this product as they

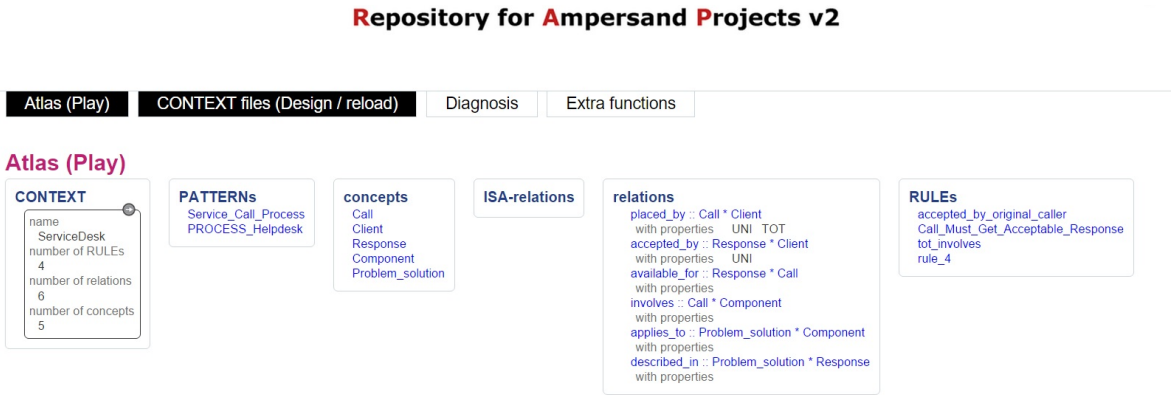

Figure 4.2: An example of what Ampersand users see when they compile a prototype

Note: each tab and figure the user has created can be modified.
Created by: Dr. Stef Joosten
Source: Ampersand-model repository

have access to a computer that has a linux or windows operating system and is capable of running Ampersand. This product will help users correct mistakes, produce easy to understand error messages, and help users to understand the logical construction of the information system they are building.

[So what are the usability and humanity requirements? All you did was explain what constitutes one of these requirements and some of Ampersand's current system. —DS]

4.2 Understandability

What EFA does is simple to understand: it fixes logical inconsistencies with minimal, if any,input from the user. It does this by looking for contradictions, or violations. This product will use natural language that is easy to understand when error reporting and hide the details of its construction from the user. Any symbols that are not standard mathematical symbols will be explained and a full range of mathematical symbols are explained in detail in the Ampersand User Guide.

An example is given as to how a decision making process takes place: there is a red chair, there is a blue chair, but 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.

It may pick a chair colour depending on another criteria that comes from another part of the information system. For example, if majority of customers' favourite colour is red,then most individuals who may use the chair would prefer it to be red rather than blue. Ampersand is built to keep external conditions in mind and those conditions are passed along to EFA. EFA can recognize if any of these conditions are violated and may return a message to the user concerning what was modified based on what conditions.

[What are your understandability requirements for this project? —DS]

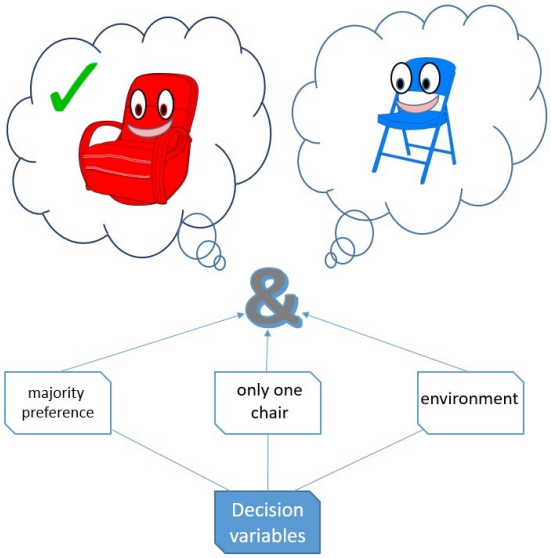

Figure 4.3: Simplistic diagram of decision making process

Note: For this process, the blue chair is discarded, we can think of it as the deletion of a dataset that contradicted one or more of the conditions that the user specified. In this case, a message may be returned such as:
Based on condition : "can only have one chair", "higher precentage of users prefer the use of red chair".
modified: deletion of blue chair, retain red chair.
Each condition is seperated by a comma, and each action taken to modify the existing system is also seperated by a comma.
The original dataset is always preserved thus any changes can be reverted if they are deemed undesirable, however if the user decides that they would like to keep the changes that are made. They are prompted (e.g. Would you like to keep current changes?) and the most current copy is saved over previous copies. In either circumstance the original input data created by the user is kept as "before" version.) **??**

4.3 Performance Requirements

Ampersand can efficiently process a large amount of information in a short amount of time; and the process that EFA requires will be added onto Ampersands process time. The 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. Any interface between a user and the automated system shall have a maximum response time of 1 minute. The response time shall be faster after the initial compilation to avoid breaking the user's flow of thought. The system shall prevent deadlock and cycling by flagging areas that it has already corrected.

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 EFA 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 EFA is in its initial stages. Furthermore, the allowable time between failures is unknown and untested.

EFA 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 EFA it could be catastrophic, especially if mistakes are not caught and products proceed into the production stage ([Mic10, 153]). However, our contribution will include the propogation of proofs as well as feedback to the user. These proofs will be generated based on the algorithm used by EFA. These proofs will be human-readable, so that if an engineer suspects that EFA has made a mistake, she can examine the proof to be convinced that EFA is indeed sound. If the proof contains a mistake, it will be much easier to catch than a code error.

[Explicitly state your performance requirements. Also, fix the odd break in the last paragraph. —DS]

4.4 Operational and Environmental Requirements

Any system that is currently running Ampersand will be able to run this product under a new verion and thus no new requirements have been introduced. [Either explicitly state the current requirements for running Ampersand, or mention that any system currently running Ampersand will be able to run the new version and thus no

new requirements have been introduced —DS].

4.5 Maintainability and Support Requirements

All code submitted for this project must be maintainable, which mean it is well documented and comes with mathematical proof. This is an essential requirement for the future maintenance of Ampersand. EFA must make sure that each specification/error is traceable ([Joo07, 2]).

[Pull out the explicit requirements and remove the fluff. —DS]

4.6 Security and Integrity Requirements

As an open source project any individual who clones the Ampersand Github has access to source code and can manipulate it as they wish. Irrelevant of what changes the user makes locally, it does not effect the Github respository, as only developers of Ampersand (i.e., our client) has permission to change source code in the respository. Access to the database and software which supports the various functions of Ampersand are run locally and subjected to the security system the user has in place on his or her work station.

[The git repository is not the issue here, are there any security/integrity concerns for the data you will be handling or for the way your contribution will handle data? —DS]

4.7 Validation and Verification Requirements

Validation and verification are important parts of non-functional requirements used to test the quality of the final product. Validation is a human activity and requires user input; the rules for the software system to follow are provided by the user, and only the user can judge if the rules are correct for the system that they are attempting to create. The validation that EFA offers is a logical test to confirm that all data sets do not violate the rules that the system follow for the model to properly mimic realistic conditions.

Verification test the accuracy of EFA, and how well it can replicate what the use has in mind according to an ECA rule system. Verification is an automated process that confirms correctness of the project in accordance to previously validated rules.

Furthermore, verification confirms the success of ECA rule adoption, and is able to quantify the level of success or failure encountered by EFA ([Mic10]).

[What are your explicit verification/validation requirements? —DS]

4.8 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

Our contribution consists entirely of adding a new feature to Ampersand. There have been no significant previous efforts to implement this feature within Ampersand. There are no open issues outside of implementing this feature.

5.2 Off-the-Shelf Solutions

No off the shelf solutions exist for this project.

5.3 New Problems

Our contribution will be entirely internal to Ampersand; it will not affect end users in terms of the interface to Ampersand or any of the resources that Ampersand generates.

The current Ampersand environment will not be affected by our contribution, since EFA is an additional feature whose functionality will exist on top of existing functionality; that is, EFA will not affect old functionalities of Ampersand.

Our contribution will be smoothly intergrated into Ampersand using git and GitHub. This transition is not expected to create any problems.

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 EFA (GHC, LaTeX, etc.) are open source as well; there is no cost requirement for any component used. [You can also mention time costs. —DS]

5.8 User Documentation and Training

User documentation will accompany EFA which will give a brief explaination concerning how EFA works and how users can use EFA to maximize their efficiency. EFA requires minimal input from user, but each possible input will be explained thoroughly and various examples will be provided in a step by step tutorial to help familiarze the user with EFA. Furthermore, a practice simulation will be in place for the user. The test will include a well documented default script which the user can compile to try out different priority options. The manual will include pictures as a type of visual guide concerning what the screen should look like from beginning to end. The documentation will likely be part of the Ampersand documentation, but for the purposes of this project, it will be submitted independently.

5.9 Waiting Room

Our project consists of implementing a single feature. There are no other features that even touch the scope of this project. Therefore, there are no requirements which will not be satisfied as a part of the initial release.

[Overall comment: Too verbose. Throughout the document you use a lot of words to say very little. —DS]

Bibliography

[Bon07] Bruno Berstel; Philippe Bonnard. Reactive rules on the web. *Reasoning Web, LNCS*, 2007.

[GHC] Glasgow Haskell Compiler. `https://www.haskell.org/ghc/`. Accessed: 2015-10-13.

[Joo] Stef Joosten. Ampersand software tool. `https://github.com/AmpersandTarski/ampersand.git`.

[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.

[vdw11] Stef Joosten; Sabastiaan Joosten; Gerard Michaels; Jaap van der woude. Ampersand - Applying Relational Algebra in Practice. *Conference: Relational and Algebraic Methods in Computer Science - 12th International Conference*, 2011.

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