Ampersand
Event-Condition-Action Rules

Software Requirement Specification

Version 0

Yuriy Toporovskyy, Yash Sapra, Jaeden Guo

CS 4ZP6
February 13th, 2016
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
11/02/2016	Software Demonstration	11/02/2016
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

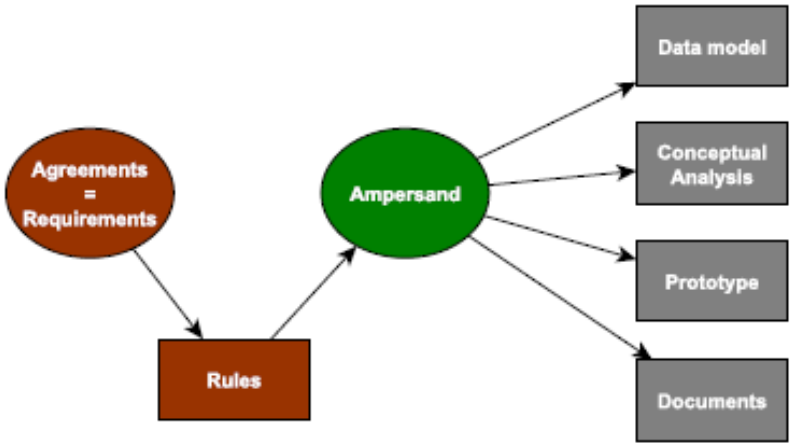

Figure 1.1: Ampersand produces a set of artifacts based on user's requirements.

> *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 through the pipeline 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. 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 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 The Client: Ampersand Designers

Ampersand designers are our clients and hold a stake in EFA's completion because it brings Ampersand one step closer to completion. The Ampersand designers have made provable correctness and maintainability as a part of the necessary conditions for EFA to be incorporated into Ampersand. The completion and acceptance of EFA would mean a permanent maintainable component for Ampersand to replace their current solution (i.e. the exec-engine) which acts as a temporary stand-in until a more maintainable solution can be found.

1.2.2 The Customer: Ampersand Users

EFA users are Ampersand users. EFA's contribution decreases the amount of time Ampersand users spend manually inserting PHP code to restore system invariants. This would save time for Ampersand users and make the Ampersand system more efficient over all.

[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

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

2.1 Mandated Constraints

Due to the long-term nature of this project, external dependencies must be minimized. Since EFA is meant to fit within the current Ampersand system, many design decisions are predetermined, such as the choice of implementation language, what is taken as input (i.e. ECA rules) and produced as output (i.e. SQL queries). Additional constraints for EFA include the need for backwards compatibility and agreement with other modules.

2.1.1 Solution Constraints

GHC 7.10 and the Cabal build system 4.22 2.1.6 must be used to compile all implemented code, and all implemented code must be in Haskell. Backwards compatibility is tested on older versions of Ampersand, and agreement with other modules are tested with system tests.

2.1.2 Partner or Collaborative Applications

This section provides a brief description of the modules that EFA uses. More details are provided in the Module Guide for EFA.

Ampersand Core Modules	
Module Name	**Description**
AbstractSyntaxTree	Ampersand's abstract representation of input from ADL.
FSpec	A module that represents the F-Spec structure.
Basics	An Ampersand internal module that provides basic functions for data manipulation.
ParseTree	Ampersand parse tree for ADL script; concrete representation of the input, and retains all information of the input.

External Haskell Modules	
Module Name	**Description**
GHC.TypeLits	Internal GHC module used in the implementation of type-level natural numbers [hac]
Data.List	A module that provides support for operations on list structures.
Data.Char	A module that provides support for characters and operations on characters.
Data.Coerce	Provides safe coercions between data types; allows user to safely convert between values of type that have the same representation with no run-time overhead [hac].
Debug.Trace	Interface for tracing and monitoring execution, used for investigating bugs and other performance issues [hac].
GHC.TypeLits	Internal GHC module that declares the constants used in type-level implementation of natural numbers [hac].
SimpleSQL.Syntax	The AST for SQL queries [hac]
Text.PrettyPrint .Leijen	A pretty printer module based off of Philip Wadler's 1997 "A prettier printer", used to show SQL queries in a readable manner to humans [hac].
GHC.Exts	This module provides access to types, classes, and functions necessary to use GHC extensions.
Unsafe.Coerce	A helper module that converts a value from any type to any other type.This is used in the translation of ECA rules to SQL using user-defined data types.
System.IO.Unsafe	IO computation must be free of side effects and independent of its environment to be considered safe. Any I/O computation that is wrapped in unsafePerformIO performs side effects.

EFA Modules	
Module Name	**Description**
ECA2SQL	The top-level module that takes ECA rules from FSpec and converts it into SQL queries.
Equality	Various utilities related to type level equality
PrettyPrinterSQL	Prints SQl queries in human-readable format
Singletons	Module for Singleton datatypes, the module defines singleton for a kind in terms of an isomorphism between a type and a type representation
Trace	Provides trace messages and various utilities used by ECA2SQL
TSQLCombinators	Uses an overloaded operator for indexing and implements SQL as a primitive data type.
TypedSQL	Contains basic SQL types represented in Haskell.
Utils	Provides utility functions for ECA2SQL and contains type families

2.1.3 Off-the-Shelf Software

?? Open source software used to implement EFA modules include the Glasgow Haskell Compiler with the Cabal system (2.1.6).

2.1.4 Schedule Constraints

- To meet software completion deadline by April 2016
- To meet all necessary deadlines for Capstone project

2.1.5 Project philosophy

Any code submitted must be well-documented, backwards-compatible and mathematically provable. Project code must satisfy existing tests, and additional tests should be written for the new algorithm being implemented.

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

2.1.6 Implementation environment

Haskell

The Ampersand code base is written almost entirely in Haskell ([Joo]) with the exception of user interfaces for the generated prototypes written in PHP and Javascript. Additional PHP code may be required for the prototype, however Haskell is the main programming language we use to build modules for Ampersand.

The Glasgow Haskell Compiler & Cabal build

The Glasgow Haskell Compiler 7.10 ([GHC]) together with the Cabal build system (see `ampersand.cabal` must be used to compile Ampersand [Joo]). Ampersand is not designed to used with other Haskell compilers.

[This doesn't sound like a constraint —DS]

GitHub

Github hosts the most current version of the Ampersand system. Github is used to maintain consistency between the main Ampersand branch and this project.

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, thus Graphviz is one of the basic components required to produce Ampersand artifacts.

The Cabal System

The Cabal system is for building and packaging Haskell libraries and programs. Cabal describes what a Haskell package is, how these packages interact with the language, and what Haskell implementations must do to support the packages. It is part of a larger infrastructure to distributing, organizing, and cataloguing libraries and programs. [hac]

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.

ADL Stands for "Abstract Data 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. 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 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.

Prototype Ampersand generates a prototype for the user that provides a front-end interface that connects to a back-end database.

2.3 Relevant Facts and Assumptions

We assume that all Ampersand users are EFA users. Furthermore, we assume that these individuals are comprised of mostly industry professionals.

2.3.1 Error Detection

The Ampersand system undergoes sentential tests on a regular basis.

Users Errors

Users errors will appear as part of EFA's front-end interface and will clearly indicate to the user what errors have been countered as well as the ECA rules that have been violated and EFA has mended.

Developers Errors

In addition to the user errors available, there are traceable error protocols built specifically for programmers; these will appear in console. Furthermore, if necessary, the Cabal build system provides compilation errors to aid the developer.

Chapter 3

Functional Requirements

3.1 The Scope of the Work

3.1.1 The Current Situation

The current Ampersand system relies on the use of an exec-engine to restore system invariants; this requires users to manually update PHP source code. The primary goal of this project is to replace the exec-engine with EFA. By doing so, it will eliminate the need for users to manually update PHP source code and provide stable, type-safe SQL queries, that are provably correct.

3.1.2 The Context of the Work

The context of the work involves investigating how the exec-engine connects to the other major components. Additional work goes into determining what degree of the old program must be replaced by EFA.

Figure 3.2 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 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.

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 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 constraints	Process	Constraints deemed not valid and must be re-evaluated
Interpret constraints	Process	Turn constraints into business requirements
Formalize	Process	Rewrite something in formal language, usually from natural language
Re-evaluate requirements	Process	Requirements deemed not valid and must be re-evaluated
Verify and compile	Process	Verification consists of type-checking (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 3.2

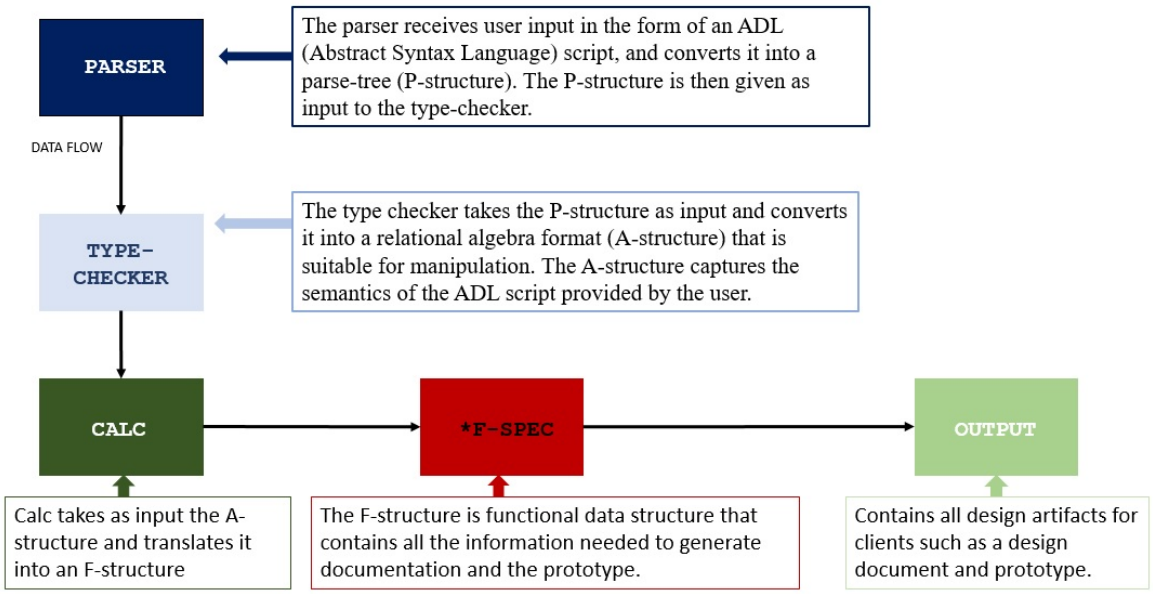

Figure 3.1: Components of Ampersand System

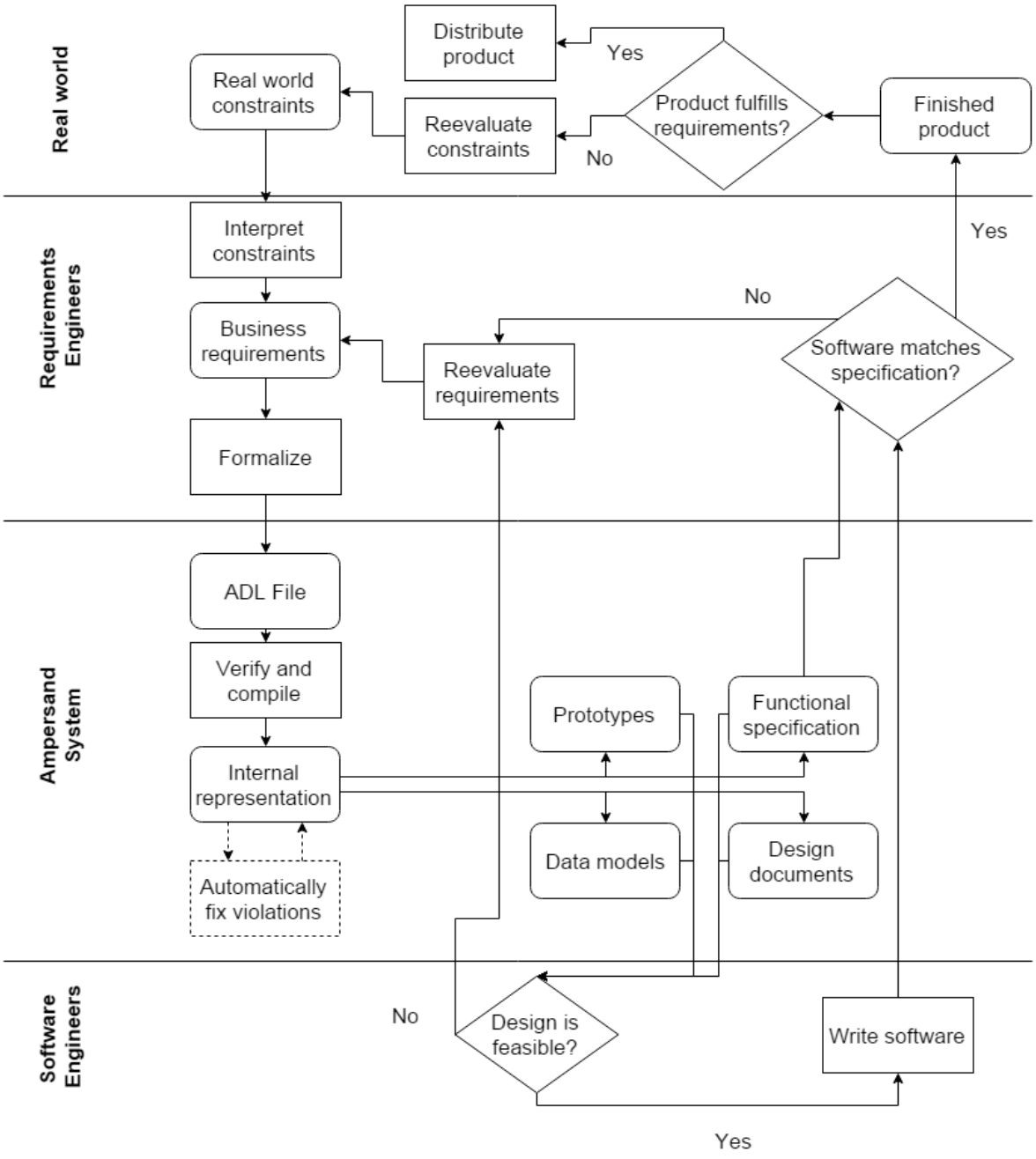

Figure 3.2: Business process diagram representing the role of Ampersand in the software design cycle

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.

The primary deliverable

3.3 Functional Requirements

3.3.1 System Requirements

Requirement	S1
Description	Create pure functions with no unintended side effects
Rationale	The use of a functional programing languages requires that this program be a pure function and does not have side effects, however certain portions of the code requires the execution of side effects to match the behaviour presented by external programs. In these specific instances, the side effects are an intended behaviour.
Originator	Stakeholder/Developer
Fit Criterion	This behaviour is necessary to produce the results the stakeholders desire
Test Case	Desired results can be confirmed as they will be reflected in changes that take place in the Ampersand database.
Customer Satisfaction	5 - Highest
Priority	5 - Highest

Requirement	S1
Supporting Materials	(Rule Based Design [Mic10])

Requirement	S2
Description	Added modules must fit within Ampersand's current framework
Rationale	As Ampersand is a huge system that has weekly additions to prevent conflict and breaking of existing packages/modules, an effort should be made to minimize external dependencies. As EFA will be an internal component of Ampersand, if a package that EFA depends on to function properly is no longer maintained and breaks, it will in turn break Ampersand.
Originator	Ampersand Creators (i.e. our client)
Fit Criterion	Functionality of EFA as an Ampersand internal component.
Test case	Added modules are tested with cabal build inside of the Ampersand system as an internal component (i.e. System testing)
Customer Satisfaction	4 - High
Priority	4 - High
Supporting Materials	Hackage, Dr. Kahl

Requirement	S3
Description	Machine-checked proofs

Requirement	S3
Rationale	EVA data-types are indexed on Haskell types, and indices are selected in a way that impossible values are eliminated by Haskell's strong type system.
Originator	Stakeholder/Developer
Fit Criterion	Program compiles and provides an inductive proof that the SQL generated by ECA2SQL is correct
Test Case	Verified by examination
Customer Satisfaction	5 - Highest
Priority	5 - Highest
Supporting Materials	Requirement solicitation.

3.3.2 Project Requirements

Requirement	P1
Description	Provable Correctness: Haskell like other functional programming languages have a strong type system which can be used for machine-checked proofs.
Rationale	Curry-Howard correspondence which states that the return type of the function is analogous to a logical theorem, that is subject to the hypothesis corresponding to the types of the argument values that are passed to the function and thus the program uysed to compute that function is analogous to a proof of that theorem.
Fit Criterion	Provable correctness of the program that is generated.

Requirement	P1
Test Cases	Internal structure of ECA rules can be compared to SQL queries through a series of datatype tests, each of which will result in a traceable result or error message
Priority	4 - High
Supporting Materials	Programming language theory, Dr. Kahl

Requirement	P2
Description	Generated SQL queries must preserve the semantics of ECA rules.
Rationale	The translation would otherwise not be correct, as the rules would be meaningless if their semantics are lost.
Originator	Ampersand Developers
Fit Criterion	Generated queries must be provably correct as per client's request.
Test Cases	Internal structure of ECA rules can be compared to SQL queries through a series of datatype tests, each of which will result in a traceable result or error message
Priority	4 - High
Supporting Materials	Requirements solicitation, Dr. Kahl

Requirement	P3
Description	Be able to process and handle errors on new additions.

Requirement	P3
Rationale	For possible future changes to EFA, errors must be processed and handled correctly. Otherwise it could introduce a variety of problems to the Ampersand system and make it difficult to retain EFA as a maintainable component.
Originator	Ampersand Developers
Fit Criterion	Errors must be handled correctly
Test Cases	Manual Testing of individual functions
Priority	4 - High
Supporting Materials	Hackage [hac], Dr. Kahl

Chapter 4

Non-functional Requirements

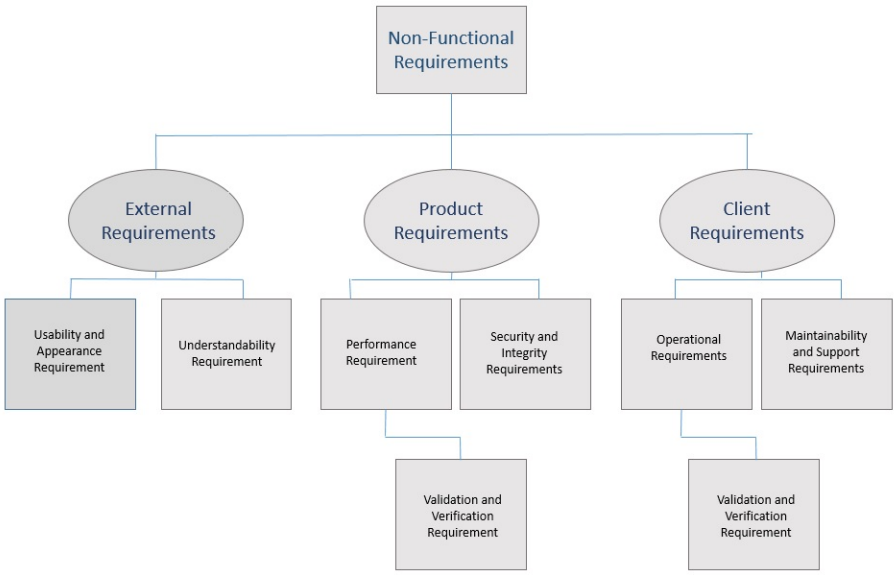

Figure 4.1: Tree of non-functional requirements as it relates to EFA

4.1 Look and Feel Requirements

- The product shall comply with Ampersand standards
- The product shall appear readable for Ampersand contributors

4.2 Usability and Humanity Requirements

- *Efficiency of use*
  - EFA is easy to use for any Ampersand user.
  - The user can use EFA with accuracy, as they are provided with the option to use EFA's automated service through the users interface.
- *Ease of remembering*
  - EFA does not require the user to memorize protocols
- *Error rates*
  - EFA eliminates ECA rule violations caused by the user, EFA's implementation is provably correct through Haskell type-checking system.
- *Feedback*
  - User receive feed-back concerning the ECA rule violations that have been resolved.
- *Overall satisfaction in using the product*
  - Users can be confident in using EFA as it is well tested and provides accurate results which the user can confirm.

4.2.1 Personalization and Internationalization Requirements

EFA is available only in English but can be adapted for other languages in later development versions.

4.2.2 Learning Requirements

EFA has a shallow learning curve, more time is spent learning the Ampersand system. No training is necessary to use EFA as long as the user can operate Ampersand.

4.3 Understandability and Politeness Requirements

Conceptually EFA is easy to understand and users will intuitively know what this product does for them. To further understandability, EFA feed back uses natural language that is familiar to the user and easy to understand. In addition, EFA hides all details of its construction from the user and provides the user.

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

4.3.1 Accessibility Requirements

EFA is unable to individually confirm to disability requirements as it is an internal component of Ampersand.

4.4 Performance Requirements

4.4.1 Speed and latency Requirements

The use of EFA should not result in a noticeable time delay. EFA shall not take more than 3 seconds to complete in the worst case scenario.

4.4.2 Safety-Critical Requirements

EFA will not expose sensitive information in the Ampersand system to outside sources or create new vulnerabilities.

4.4.3 Precision or Accuracy Requirements

SQL queries generated by EFA is correct 100% of the time with full coverage of all ECA rules that apply.

4.4.4 Reliability and Availability Requirements

EFA is available for use anytime Ampersand is used. In the event that Ampersand fails, then EFA will also be unavailable as it depends on Ampersand generated data taken from the user.

4.4.5 Robustness or Fault-Tolerance Requirements

In the absence of ECA rule violations, EFA will continue to function and be on stand-by until a rule violation is triggered.

4.4.6 Capacity Requirements

Ampersand runs on individual machines; EFA as an internal component of Ampersand will be able to deal with any amount of data that Ampersand can handle.

4.4.7 Scalability and Extensibility Requirements

EFA is capable of handling large volumes of data, and the translation of ECA rules to SQL queries requires a standard amount of time. The number of ECA rules is expected to expand as Ampersand etches closer to completion.

4.4.8 Longevity Requirements

4.5 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].

4.6 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]

4.7 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]

4.8 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]

4.9 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

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

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

[hac] Hackage.

[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.