Ampersand
Event-Condition-Action Rules

Software Requirement Specification

Version 0 Revised

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
J Guo	04/ 02/ 2016	Revision 0

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	19/01/2016
11/02/2016	Software Demonstration	11/02/2016
01/04/2016	EFA Complete	—

Chapter 1

Ampersand As A System

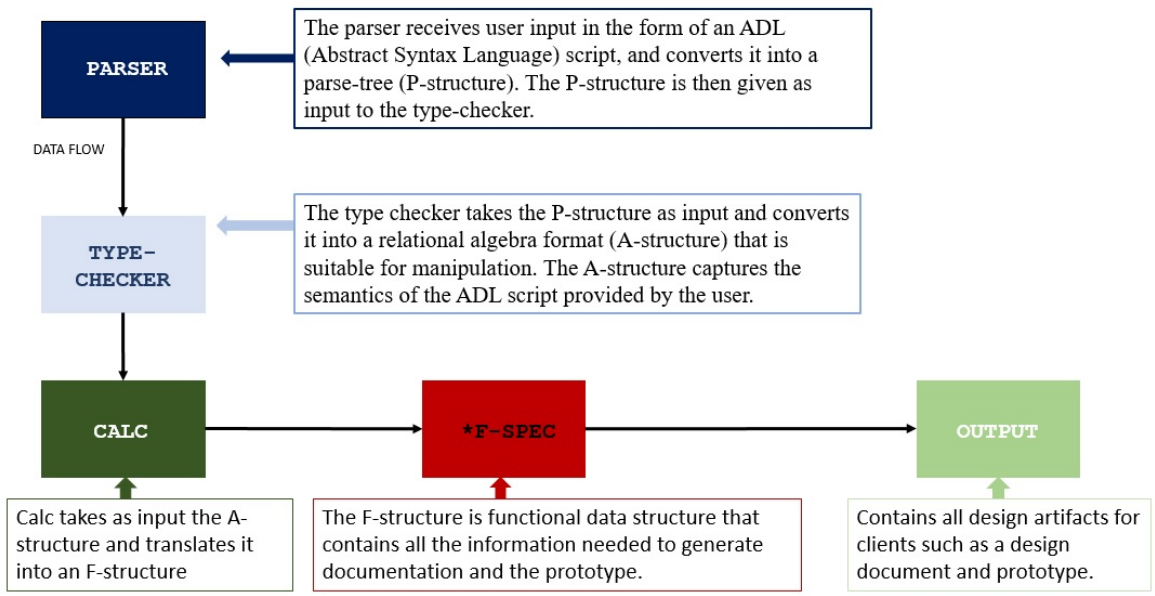

Figure 1.1: Components of Ampersand System

Ampersand is an open-source project that produces design artifacts based on business rules. One of the essential functions of Ampersand is to maintain all the business rules that keep transactions valid during the course of each cycle, where multiple transactions could take place. Figure 1.1 breaks down the major components of ampersand ignoring specification details. This project focuses mainly on one component of Ampersand, which is the F-spec which contains ECA (Event-Condition-Action Rules) that each process must obey. These rules are essential to the functionality of ampersand, and purpose of this project is to correctly translate ECA Rules into type safe SQL queries.

1.1 The Ampersand Environment

Ampersand is a on-going project with an increasing number of modules being added to it on a weekly basis. Since this project focuses on a component of ampersand, it must be built to fit within the ampersand environment and co-exist with other modules. Due to this restriction, many design decisions are predetermined such as types of data that are used and programming language used to build them.

1.1.1 Project Purpose

The purpose of this project is to correctly translate ECA rules into type-safe SQL queries using Haskell. These ECA rules are used to maintain business constraints.

1.1.2 Project Goals

The goals of this project can be divided into two components. The first component consists of satisfying the condition necessary for the completion of an undergraduate capstone project. The second component is to design and implement maintainable code that can be absorbed to the ampersand open-source project.

1.2 The Stakeholders

The stakeholders are separated into two sections, those that directly benefit from this projects contribution and those that indirectly benefit.

1.2.1 Ampersand Designers

Ampersand designers are our client, and they directly benefit from this project as it bring Ampersand one step closer to completion. This project EFA (ECA for Ampersand) delivers a maintainable component for Ampersand that produces type safe SQL queries, which will be used to maintain the consistency of the data in the back-end Database.

1.2.2 End-Users

Ampersand users indirectly benefit from this project's contribution because it drastically decrease the time spent manually restoring system invariants, which are the rules that maintain the validity of each business process. As EFA is executed during

compile time, the user will not suffer any noticeable delays and can rest assured that the artifacts they received are correct according to specification.

Chapter 2

Project Constraints

The current Ampersand system is the main limitation of this project; everything that is built, must be built to fit within its current constraints. These constraints include the language used to build Ampersand (i.e. Haskell). Anything incorporated into Ampersand must be implemented in the Haskell language. An Additional constraint placed on the acceptance of this project by our clients is to produce maintainable code; this includes the use of dependable libraries and any support modules generated by this project to help the translation of ECA rules to SQL queries.

2.1 Mandated Constraints

All code must be well documented, backwards compatible and fit seamlessly into the current Ampersand project. Due to the long-term nature of this project, we must minimize the number of external dependencies.

2.1.1 Implementation Environment of the Current System

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. Haskell is the only programming language we can use to build modules for ampersand.

The Glasgow Haskell Compiler & Cabal Build System

As most of Ampersand's base code is written in Haskell, a compiler must be used to compile it. The Glasgow Haskell Compiler ([GHC]) 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.

GitHub Repository

Ampersand's main repository is hosted on Github, we must also host our project on Github to be able to maintain consistency with the Ampersand source code.

Graphviz

Graphviz is an open source graph visualization software, which can visually represent information in the form of charts and graphs. Graphviz is used to create visuals in ampersand artifacts and is an essential to running ampersand.

2.1.2 Partner of Collaborative Applications

Name	Type	Description
AbstractSyntaxTree	Ampersand module	A module designed specifically for Ampersand, data from this module is manipulated in EFA.
Control.Applicative	Library module	An interface that provides an intermediate structure between a monad and a functor. This interface is used to embed pure expressions, sequence computations and combine their results.
Control.Exception	Library module	An interface that provides support for raising and catching build-in and user-defined exceptions.

Control.DeepSeq	Library module	This module is used to fully evaluate data structure and is used to prevent resource leaks in lazy IO programs.
Data.Proxy	Library module	A concrete proxy type, used to represent the value of something else.
Data.Type.Equality	Library module	This module offers pattern-matching on types and provides a proof, it is used as a definition of propositional equality.
Data.List	Library module	A module that provides support for operations on list structures.
Data.Char	Library module	A module that provides support for characters and operations on characters.
Data.Coerce	Library module	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.
Debug.Trace	Library module	Interface for tracing and monitoring execution, used for investigating bugs and other performance issues.
GHC.TypeLits	Library module	Internal GHC module that declares the constants used in type-level implementation of natural numbers.
GHC.Exts	Library module	This modules allows the use of pointers to an object or array of objects.
Language.SQL.SimpleSQL	Library module	Syntax: provides the AST for SQL queries.

		Pretty: provides pretty printing functions that formates output for human reading.
Numeric.Natural	Library module	Natural number type
Prelude	Library module	A standard module that is imported by default and provides support for basic data types, comparison functions, and methods used for data manipulation.
System.IO.Unsafe	Library module	This module allows IO computation to be performed at any time, the IO computation must be free of side effecets and independent of its environment to be considered safe. Any I/O computation that is wrapped in unsafePerformIO performs side effects.
Text.PrettyPrint.Leijen	Library module	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.
Unsafe.Coerce	Library module	A helper module that converts a value from any type to any other type, the user must assure that the old data type and the new data type have identical internal representations, else runtime corruption occurs. This is used in the translation of ECA rules to SQL using unique data types.

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 a method and the name of the open source project.

⇒ The Ampersand method is used to generate functional specification from formalized business requirements.

⇒ The Ampersand software is a tool that implements this method.

Business rules Rules that exist to represent real world constraints that the virtual world does not naturally possess, such as resource and social limitations. Examples of constraints include but are not limited to financial, logistic, physical or legal constraints.

EFA Stands for "ECA (see above) for Ampersand". This term is used to refer to 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.

2.3 Relevant Facts and Assumptions

This project makes the assumption that Ampersand users are using it according to its intended purposes and have all the necessary software dependencies installed for it properly function. This project is designed with the assumption that no direct interaction is necessary between the design component and the user. All interactions that could take place is buffered by Ampersand. Furthermore, we assume that Ampersand users are industry professionals that are capable of tracing error messages and fixing them.

2.3.1 Error Detection

EFA provides traceable error messages for the developer, however on a user level these trace error messages would be absorbed into Ampersand and its various error detection mechanisms situated on every level of compilation. Ampersand is equipped with friendly error detection for the user beginning with syntax detection for ADL scripts to assure that there are no missing or out of place components. To logical error detections that the user might have missed during the creation of their information system. The error messages inform the user what line the error has been found, what the error pertains to, and what is expected typically in the script structure. The script structure provides the user clues for how they may wish to fix the error by adjust their script to fit the appropriate format.

Chapter 3

Functional Requirements

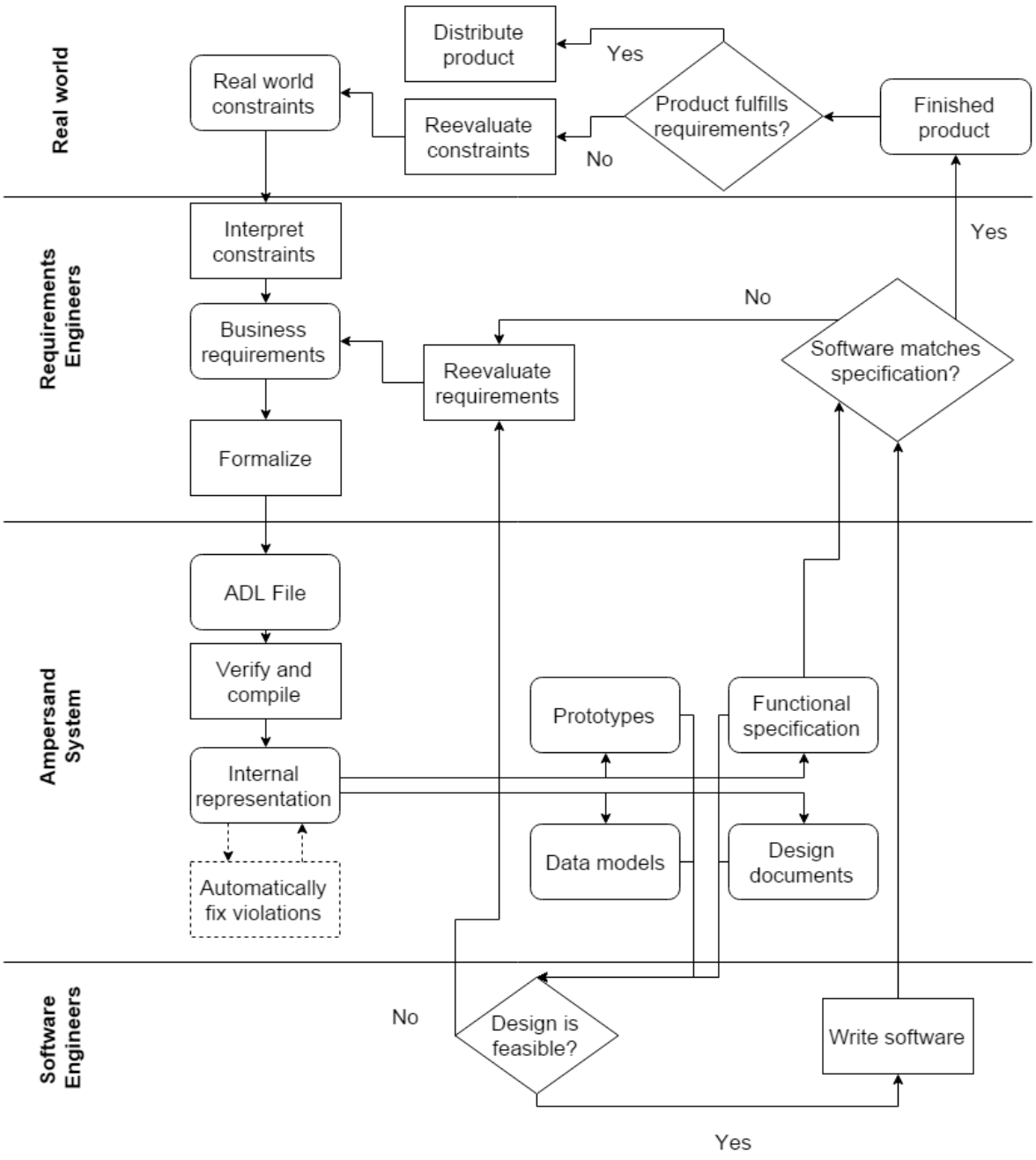

Figure 3.1: Business process diagram representing the role of Ampersand in the software design cycle

The diagram 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.

3.1 The Scope of the Work

The following sections focuses specifically on EFA and how it will function in the Ampersand environment. EFA is an automated process internal to Ampersand, and as a part of the ampersand system it works in collaboration with other internal components such as the F-spec. The purpose of EFA is to replace the current exec-engine, creating a permanent solution for the implementation of ECA rules. EFA also provides extensive functionalities that the exec-engine is missing, such as the ability to manipulate data in a database beyond the basic level of creating and dropping tables, and basic select queries. It provides the fundamental datatypes that are crucial for the expansion and maintenance of Ampersand as it grows. Due to the way that queries are generated in its present state, large number of projects will bog down the system until it becomes unmanageable, and if Ampersand is used in practice.

3.1.1 The Current Situation

Ampersand currently has an exec-engine that passes SQL queries which are triggered by the prototype user interface implemented in PHP. Though the exec-engine functions, it is a temporary solution for translating ECA rules into SQL queries. Any changes made to the information system after its initial generation require manual maintenance. This project will create a permanent solution that is provably correct and will automate the correction of system invariants so that manual maintenance of system invariants is no longer necessary once EFA has been successfully incorporated into Ampersand.

3.2 The Scope of the Product

The translation of ECA rules into SQL queries require unique data types that preserves the semantics the user provides in the ADL script. ECA rules are generated from the conditions the user specifies in the ADL script. The SQL queries generated from ECA rules can be thought of as a sequence of changes made to the data. This sequence of actions are made through specific event triggers, and the actions only take place if all conditions are satisfied and are valid. An example of this, would be attempting to delete a person who does not exist. This actions cannot be completed because the person does not exist, this action would be invalid.

3.3 Functional Requirements

[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]

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
Supporting Materials	(Rule Based Design [Mic10])

Requirement	S2
Description	The use of Haskell to implement EFA modules
Rationale	The source code of Ampersand is written completely in Haskell, and thus Haskell must be used for any modules created by this project to be absorbed into the pre-existing source code.
Originator	Ampersand Creators (i.e. our client)
Fit Criterion	Primary ability to write code compatible with Ampersand as it is.
Test case	Added modules are tested with cabal build inside of Ampersand
Customer Satisfaction	5 - Highest
Priority	5 - Highest
Supporting Materials	Dr. Joosten, Joosten and Kahl

Requirement	S3
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)

Requirement	S3
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

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.
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

Requirement	P1
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	Hackage, Dr. Kahl

Requirement	P3
Description	Generating traceable results and error messages for handling new contributions from this project
Rationale	Saves time by allowing the program to inform the programmer where the errors are located.
Originator	Ampersand Developers

Requirement	P3
Fit Criterion	Errors must be traceable and have a standard format that can be easily followed.
Test Cases	Error message will print to screen.
Priority	4 - High
Supporting Materials	Hackage, Dr. Kahl

3.3.3 Non-Functional Requirements

Requirement	N1
Description	EFA must be available at anytime ampersand is running.
Rationale	To provide the user with unlimited access to EFA within Ampersand.
Originator	Ampersand Developers
Fit Criterion	Ampersand can detect when internal components are non-responsive
Test cases	Ampersand is subject to sentential tests on a daily basis as part of its maintenance.
Priority	4 - High
Supporting Materials	Useful feedback in Ampersand parser [DSS15]

Chapter 4

Non-Functional Requirements

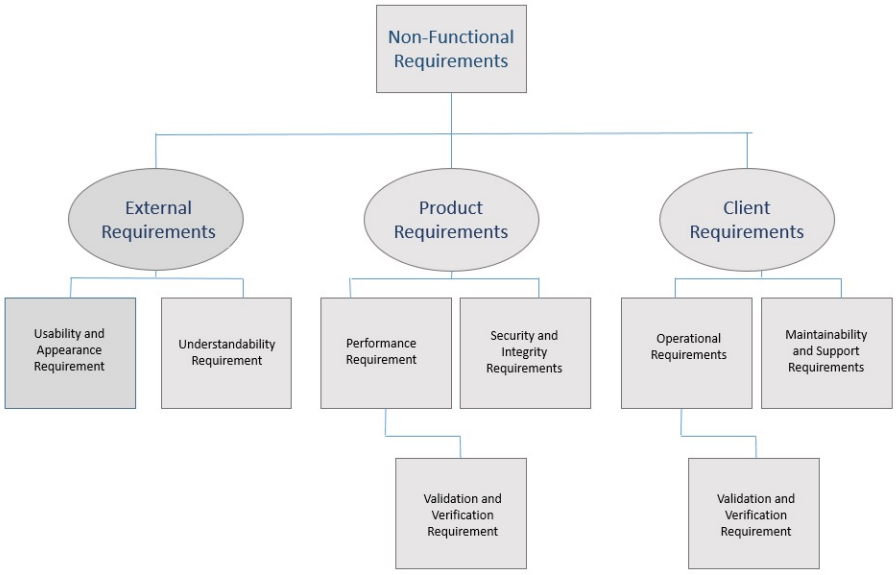

Figure 4.1: Tree of non-functional requirements as it relates to EFA

Each Non-functional requirement must be traceable in the Ampersand system with the addition of EFA. *Note: Missing sections in this chapter are not applicable for this project*

4.1 Performance Requirements

EFA is designed to optimize the Ampersand system by automating the tedious task of restoring system invariants when broken using ECA rules. Ampersand will perform as it use to with less maintenance required on behalf of the user.

4.2 Maintainability and Support Requirements

EFA must make sure that each specification/error is traceable ([Joo07, 2]). EFA will be upgrade and tested as a part of Ampersand does not require additional maintenance or support to perform optimally.

Chapter 5

Traceability and Proof of Concept

5.1 Validation and Verification Requirements

Notation	**Description**
X : Y	X has type Y
‘*Types*‘	Type of types
‘*Kind*‘	Type of Kinds
_ → _ Type →Type → Type	This function requires 2 Type inputs and produces a Type output. Where _ → _ can be seen as for each element x of type A defines a function F(x) where x does not exist in set Type,Kind for every x that defines F(x)
_ → _ Kind →Kind → Kind	This function requires 2 of type Kinds and produces an output of Type Kinds . Where _ → _ can be seen as for each element x of type A defines a function F(x) where x does not exist in set Type,Kind for every x that defines F(x). This a dependent type, an example of this would be SingT (x :: a) → SingT (F x)

$\mathbb{N}$, Symbol : Kind	Left of the : are the elements that represent type Kinds (right of the :)
0, 1, 2, ... $\mathbb{N}$	These are natural numbers, elements left of the : belong to the set N of natural numbers
{"", "a", "aa", ..., "b", "bb"} : Symbol	Elements left of : are elements that belong to the set Symbol
(x : A) $\rightarrow$ F x where x $\notin$ {Type,Kind}	This can also be seen as $\forall$ x $\rightarrow$ F x; where as
SingT x::a $\rightarrow$ SingT Fx	SingT is the type of singleton, a dependent type created for EFA datatypes.
$\rightarrow$: Constraint $\rightarrow$ Type $\rightarrow$ Type and _ $\Rightarrow$ _ : Type $\rightarrow$ Constraint $\rightarrow$ Type	where $\forall$x : S $\Rightarrow$ T corresponds to **G** S (λx.T) in Illative Combinatory Logic where G is the combinator [HBt]. The $\Rightarrow$ represents restraints for what on the right side of the : .
X : Y where Y $\in$ Type,Kind	X is equal in definition to 'X : 'Y and only X' : Y'.
elim-Y: (r:Type) $\rightarrow$ Y $\rightarrow$ (A_0 $\rightarrow$ r) $\rightarrow$ (A_1 $\rightarrow$ r) $\rightarrow$ $\rightarrow$ ... $\rightarrow$ (A_n $\rightarrow$ r) $\rightarrow$ r	An eliminator that corresponds to pattern matching
'P : Y $\rightarrow$ Z'	This represents 'P (Ctr x)', where 'Ctr is used for type casting
$\exists$ (x : A) (P x)	This indicates that (r:Type) $\rightarrow$ ((x:A) . (P x $\rightarrow$ r) $\rightarrow$ r)
X : Y	X has type Y

Chapter 6

Project Issues

Absent sections have been taken our as they are not applicable to this project

6.1 Tasks

- Translate ECA rules to SQL commands
- Create supporting data structures for sustainable translation and future maintenance
- Implement the solution and provide the annotated source code to the supervisor and the product owner for a review.
- Incorporate changes to project as suggested by our client and supervisor
- Provide mathematical proof to back up implemented code

6.2 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 hosted on Github.

6.3 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.
- products (i.e. SQL queries) must be provable and sustainable on large scale projects
- the system must not fail due to dependency issues

6.4 Costs

The cost is eight months of time.

6.5 User Documentation and Training

User documentation is not necessary, as EFA provides traceable error messages for developers.

Bibliography

[DSS15] Maarten Baertsoen Daniel S. Schiavini. Useful feedback in the ampersand parser. *Undergrad Thesis*, 2015.

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

[HBt] Martin Bunder Henk Barendregt and Wil Dekkers title=System of Illative Combinatory Logic Complete for First-Order Propositional and Predicate Calculus journal=The Journal of Symbolic Logic year=1993 url=http://www.jstor.org/stable/2275096.

[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.