Module Guide for ECA Rules for Ampersand

Yuriy Toporovskyy (toporoy)
Yash Sapra (sapray)
Jaeden Guo (guoy34)

February 24th, 2016

Table 1: Revision History

Author	**Date**	**Comments**
Yash Sapra	24 / 02 / 2016	Initial draft
Jaeden Guo	27/ 02 / 2016	Update and merge

Contents

1 Introduction

1.1 Description

This document follows the principle set by Parnas and Clements(DP86). EFA is currently in development where changes occur frequently, a commonly accepted practice for this situation is to decompose modules based on the principle of abstraction, where unnecessary information in hidden for the benefit of designers and maintainers(DP84; Par72).

Our design follows the principles laid out by (DP84), as follows:

- Unnecessary design details are omitted for simplicity
- Each module is broken down based on hierarchy with no overlap of functionality
- All our modules are Open Modules i.e they are available for extension in the future
- Reference material are provided for external libraries

The language of implementation is Haskell.

1.2 Scope

The purpose of this document is to outline the implementation details of the EFA project described in the Problem Statement. EFA is responsible for generating SQL Statements from ECA rules that will be used to fixed any data inconsistencies in the Ampersand Database. The document will serve as a referral document for future Software Development in the Ampersand project.

1.2.1 Intended Audience

This document is designed for:

New project members: This document designed to be a guide to introduce new Ampersand users to EFA (ECA rules for Ampersand). It provides a basic structure that allows individuals to quickly access what they are looking for.

Maintainers & Designers: The structure of this module guide will help maintainers rationalize where changes should be made in order to accomplish their intended purpose. Furthermore, the design document will act as a guide to EFA for future designers of Ampersand.

2 Anticipated and Unlikely Changes

2.1 Anticipated Changes

It is likely that EFA will require changes to the front-end interface and an addition that will connect the front-end interface to back-end functions, which will give the user more control. In addition, ECA rules are not static and may change over time, if changes do take place those changes will need to be incorporated into EFA's future versions.

Thus far anticipated changes include:

AC1: New front-end interface.

AC2: Addition or elimination of ECA rules.

AC3: The algorithm used for EFA.

AC4: The format of output.

AC5: The format of input parameters.

AC6: Integration of front-end interface to back-end modules.

AC7: Testing for individual modules and internal systems.

2.2 Unlikely Changes

These unlikely changes include the things that will remain unchanged in the system, and also changes that would not affect EFA.

UC1: There will always be a source of input data external to the software.

UC2: Results will always be provably correct.

UC3: Output data must exist.

UC4: The implementation language must be the same as that which is used for building the Ampersand system.

UC5: The format of initial input data and associated markers for data association.

UC6: Type of output data will always be a SQL procedure.

3 Module Hierarchy

This section provides an overview of the module design. Modules are decomposed based on their hierarchy from top to bottom. The modules are broken down into three sections, the first section consists of the main module used for EFA, the second section contains support modules and finally the last section contains external libraries.

M1: eca2SQL

M2: TypedSQL

M3: Equality

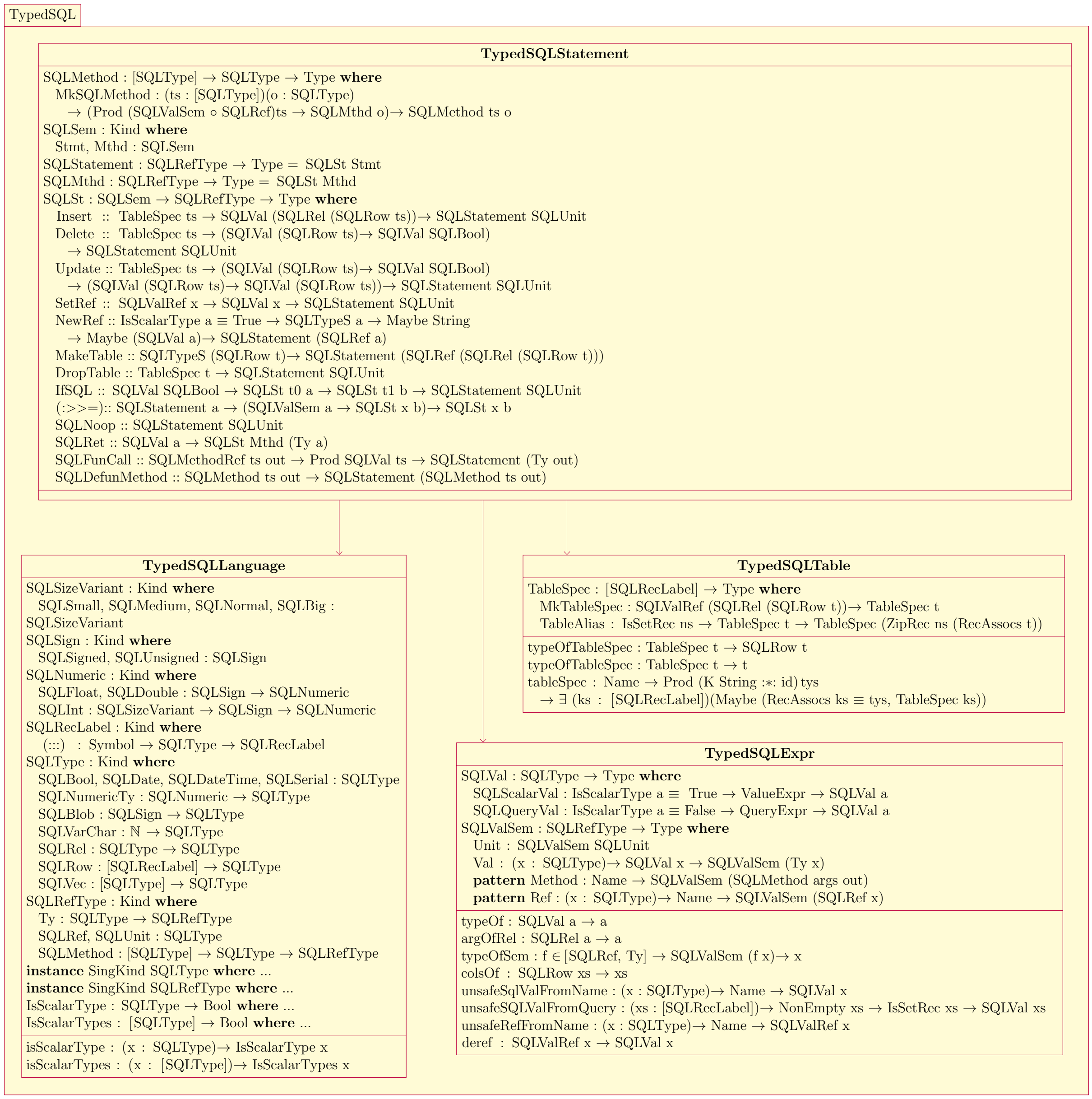
M4: Singletons

M5: Utils

M6: Trace

M7: Combinators

M8: Pretty


TypedSQL
TypedSQLStatement
SQLMethod : [SQLType] → SQLType → Type where
MkSQLMethod : (ts : [SQLType])(o : SQLType)
→ (Prod (SQLValSem ∘ SQLRef)ts → SQLMthd o)→ SQLMethod ts o
SQLSem : Kind where
Stmt, Mthd : SQLSem
SQLStatement : SQLRefType → Type = SQLSt Stmt
SQLMthd : SQLRefType → Type = SQLSt Mthd
SQLSt : SQLSem → SQLRefType → Type where
Insert :: TableSpec ts → SQLVal (SQLRel (SQLRow ts))→ SQLStatement SQLUnit
Delete :: TableSpec ts → (SQLVal (SQLRow ts)→ SQLVal SQLBool)
→ SQLStatement SQLUnit
Update :: TableSpec ts → (SQLVal (SQLRow ts)→ SQLVal SQLBool)
→ (SQLVal (SQLRow ts)→ SQLVal (SQLRow ts))→ SQLStatement SQLUnit
SetRef :: SQLValRef x → SQLVal x → SQLStatement SQLUnit
NewRef :: IsScalarType a ≡ True → SQLTypeS a → Maybe String
→ Maybe (SQLVal a)→ SQLStatement (SQLRef a)
MakeTable :: SQLTypeS (SQLRow t)→ SQLStatement (SQLRef (SQLRel (SQLRow t)))
DropTable :: TableSpec t → SQLStatement SQLUnit
IfSQL :: SQLVal SQLBool → SQLSt t0 a → SQLSt t1 b → SQLStatement SQLUnit
(:>>=):: SQLStatement a → (SQLValSem a → SQLSt x b)→ SQLSt x b
SQLNoop :: SQLStatement SQLUnit
SQLRet :: SQLVal a → SQLSt Mthd (Ty a)
SQLFunCall :: SQLMethodRef ts out → Prod SQLVal ts → SQLStatement (Ty out)
SQLDefunMethod :: SQLMethod ts out → SQLStatement (SQLMethod ts out)
TypedSQLLanguage
SQLSizeVariant : Kind where
SQLSmall, SQLMedium, SQLNormal, SQLBig :
SQLSizeVariant
SQLSign : Kind where
SQLSigned, SQLUnsigned : SQLSign
SQLNumeric : Kind where
SQLFloat, SQLDouble : SQLSign → SQLNumeric
SQLInt : SQLSizeVariant → SQLSign → SQLNumeric
SQLRecLabel : Kind where
(:::) : Symbol → SQLType → SQLRecLabel
SQLType : Kind where
SQLBool, SQLDate, SQLDateTime, SQLSerial : SQLType
SQLNumericTy : SQLNumeric → SQLType
SQLBlob : SQLSign → SQLType
SQLVarChar : ℕ → SQLType
SQLRel : SQLType → SQLType
SQLRow : [SQLRecLabel] → SQLType
SQLVec : [SQLType] → SQLType
SQLRefType : Kind where
Ty : SQLType → SQLRefType
SQLRef, SQLUnit : SQLType
SQLMethod : [SQLType] → SQLType → SQLRefType
instance SingKind SQLType where ...
instance SingKind SQLRefType where ...
IsScalarType : SQLType → Bool where ...
IsScalarTypes : [SQLType] → Bool where ...
isScalarType : (x : SQLType)→ IsScalarType x
isScalarTypes : (x : [SQLType])→ IsScalarTypes x
TypedSQLTable
TableSpec : [SQLRecLabel] → Type where
MkTableSpec : SQLValRef (SQLRel (SQLRow t))→ TableSpec t
TableAlias : IsSetRec ns → TableSpec t → TableSpec (ZipRec ns (RecAssocs t))
typeOfTableSpec : TableSpec t → SQLRow t
typeOfTableSpec : TableSpec t → t
tableSpec : Name → Prod (K String :∗: id) tys
→ ∃ (ks : [SQLRecLabel])(Maybe (RecAssocs ks ≡ tys, TableSpec ks))
TypedSQLExpr
SQLVal : SQLType → Type where
SQLScalarVal : IsScalarType a ≡ True → ValueExpr → SQLVal a
SQLQueryVal : IsScalarType a ≡ False → QueryExpr → SQLVal a
SQLValSem : SQLRefType → Type where
Unit : SQLValSem SQLUnit
Val : (x : SQLType)→ SQLVal x → SQLValSem (Ty x)
pattern Method : Name → SQLValSem (SQLMethod args out)
pattern Ref : (x : SQLType)→ Name → SQLValSem (SQLRef x)
typeOf : SQLVal a → a
argOfRel : SQLRel a → a
typeOfSem : f ∈ [SQLRef, Ty] → SQLValSem (f x)→ x
colsOf : SQLRow xs → xs
unsafeSqlValFromName : (x : SQLType)→ Name → SQLVal x
unsafeSQLValFromQuery : (xs : [SQLRecLabel])→ NonEmpty xs → IsSetRec xs → SQLVal xs
unsafeRefFromName : (x : SQLType)→ Name → SQLValRef x
deref : SQLValRef x → SQLVal x

4 System Architecture

4.1 Key Algorithm

The key Algorithm for the EFA project is AMMBR (Joo07). AMMBR is a method that allows organization to build information systems that comply to their business requirements in a provable manner. This Algorithm is implemented in Ampersand and is responsible for translating the business requirements into ECA rules. These ECA rules contain information on how to fix any data violation and are translated into SQL in our EFA project.

4.2 Communication Protocol

The EFA implementation needs to communicate with the front end to be able to run the generated SQL when a violation occurs.

- **Old communication protocol - PHP engine**
  In in existing version, Ampersand depends on PHP code to run the generated SQL on the database. However this comes at the cost of human intervention and maintenance cost for the development.

- **New Communication protocol - Stored Procedures**
  The developments teams of EFA has come to a conclusion that the best way of communicating with the front-end will be to use Stored Procedures(Ora14). The Stored Procedures provide the extra benefit of query optimization at compile time which results in better performance. While this is a suggested change, it will require changes to the existing Ampersand software in order to successfully implement this idea. This is an anticipated change and will be implemented in the near future.

4.3 Error handling

Most of the error handling is done by the underlying Ampersand software. Any human errors (syntactic or semantical) in the input ADL file are handled during the generation of the ECA. The resulting ECA rules which are fed into the EFA project are provably correct. The language of implementation (i.e Haskell) guarantees type level correctness of the EFA project at compile time.
If any error occurs during runtime, the state of the database will be checked before and after the SQL statement is run. In case of an inconsistency in the data, the SQL

rollback command will be issued that will change the database to its previous state. In such a scenario, the user will be notified about the event and can take necessary actions to fix the issue.

4.4 Main Module

M1	eca2SQL
Main function	eca2SQL
Input	Options, FSpec, ECARule
Output	Doc
Services	Produces the final product by using supporting modules as tools in translating ECA rules to SQL statements which can be applied to a database.

Internal Description

eca2SQL :: Options → FSpec → ECArule → Doc

4.5 Support Modules

M2	eca2SQL
Requires Modules	ProofUtils, Trace, Singleton
Services	Implements a type language for SQL through pattern matches where the representation of SQL references types which can appear in SQL statements. Contains Base which implements a typed SQL query language and a type SQL statement.

Internal Description

Read as: *function: input → input2 → output*

Where a function may require multiple inputs of different types to produce the necessary output type.

function (type and value level): (x:A) → output This indicates the function of a certain type and its value level, this is seen on SQL types. (x:A) is used to indicate

a variable x of type A (e.g. x=9, (x:A) is a type of integer). (function::) is used to define a function and its input types
(:::) : Symbol $\rightarrow$ SQLType $\rightarrow$ SQLRecLabel

SQLSizeVariant : Kind
SQLSmall, SQLMedium, SQLNormal, SQLBig :SQLSizeVariant

SQLSign : Kind
SQLSigned, SQLUnsigned : SQLSign

SQLNumeric : Kind
SQLFloat, SQLDouble : SQLSign $\rightarrow$ SQLNumeric

SQLInt : SQLSizeVariant $\rightarrow$ SQLSign $\rightarrow$ SQLNumeric
SQLRecLabel : Kind

SQLType : Kind
SQLBool, SQLDate, SQLDateTime, SQLSerial : SQLType

SQLNumericTy : SQLNumeric $\rightarrow$ SQLType
SQLBlob : SQLSign $\rightarrow$ SQLType

SQLVarChar : N $\rightarrow$ SQLType SQLRel : SQLType $\rightarrow$ SQLType

SQLRow : [SQLRecLabel] $\rightarrow$ SQLType SQLVec : [SQLType]$\rightarrow$ SQLType

SQLRef, SQLUnit: SQLType
SQLRefType: Kind

SQLMethod: [SQLType] $\rightarrow$ SQLType $\rightarrow$ SQLRefType
SQLVal: SQLType $\rightarrow$ Type

SQLScalarVal: IsScalarType a $\equiv$ True $\rightarrow$ Sm.ValueExpr $\rightarrow$ SQLVal a
SQLQueryVal: IsScalarType a $\equiv$ False $\rightarrow$ Sm.QueryExpr $\rightarrow$ SQLVal a

SQLValSem: SQLRefType $\rightarrow$ Type

Ty: SQLType → SQLRefType

IsScalarType: SQLType → Bool
isScalarType: (x:SQLType) → IsScalarType x
IsScalarTypes: [SQLType] → Bool
isScalarTypes: (X:[SQLType]) → IsScalarTypes x

typeOf: SQLVal a → a
argOfRel: SQLRel a → a

Unit: SQLValSem SQLUnit
Val:: SQLVal x → SQLValSem ('Ty x)

M3	Equality
Contains functions	not, cong, elimNeg and more
Services	The module contains utility functions that are being used in the Singletons and Utils module to provide type level security for the TypedSQL with an aim to make it total. It implements a primitive strict negation type, a strict decidable equality type, and various equality proof function

M4	Singleton
Main function	No functions exported
Services	Singletons contains an implementation of singletons (types which are inhabited by precisely one value) in a kind-generic way. This is used in contrast to the 'singletons' package which makes heavy use of Template Haskell. To avoid this massive pitfall, we have reimplemented singletons without Template Haskell.

M5	Utils
Main function	
Services	The module also contains some utility functions used by the ECA2SQL module

M6	Trace
Main function	getTraceInfo
Services	The modules contains support for tracing back the ECA rule. It implements a proper error message (with line numbers and function name). Tracing is used for development and debugging purpose

M7	Combinators
Main function	No functions exported
Services	This module contains Combinatory Logic for our TypedSQL module. The module defines SQL primitives such as AND, OR, NOT and the primitive SQL functions such as the EXISTS, GROUP BY, SORT BY.

M8	Pretty
Main function	eca2PrettySQL
Services	The module contains a pretty printer for the Typed SQL types. The intended purpose of this pretty printer is to produce human-readable SQL Statements.

Internal Description

eca2PrettySQL :: Options → FSpec → ECArule → Doc

5 Module Decomposition

Modules are located in their respective subsections (i.e. main module, support modules and external library modules). Each module is decomposed based on their use while hiding implementation details.

Please see glossary for math references and clarification of uncommon terms

5.1 External Libraries

The EFA project depends on the following Libraries

Ampersand Core Libraries
: The EFA project depends on the Ampersand software for the definition of core Data Structures, i.e FSpec (which contains the definition to the underlying ECA rules). EFA also maintains the relational schema of the input and hence imports Ampersand's existing functions to fetch the table declarations while generating SQL Statements for the ECA rules. AMMBR (Joo07) which is the key algorithm responsible for translating business requirements into ECA rules is an intergral part of Ampersand.

Simple-sql-parser
: The pretty printer of the EFA project depends directly on this library for pretty printing the SQL statements in the console. This is important from a development and debugging perspective.(Whe14)

wl-pprint
: The wl-pprint library(Lei14) is a pretty printer based on the pretty printing combinators described by Philip Wadler (1997). EFA uses this library in combination with the simple-sql-pretty to output the SQL statements in a human readable format.

6 Traceability Matrix

References

[DP84] D.M. Weiss D.L. Parnas, P.C. Clements. The modular structure of complex systems. *Proceeding ICSE '84: Proceedings of the 7th international conference on Software engineering*, 1984.

[DP86] P.C. Clements D.L. Parnas. A rational design process: How and why to fake it. *IEEE Transaction on Software Engineering*, 1986.

[Joo07] Stef Joosten. Deriving Functional Specifications from Business Requirements with Ampersand. *CiteSeer*, 2007.

[Lei14] Daan Leijen. wl-pprint package. `https://hackage.haskell.org/package/wl-pprint-1.2`, 2014. Accessed: 2016-02-28.

[Ora14] Oracle. Mysql reference manual. `https://dev.mysql.com/doc/refman/5.7/en/`, 2014. Accessed: 2016-02-28.

[Par72] D.L. Parnas. On the criteria to ne used in decomposing systems into modules. *Communications of the ACM*, 1972.

[Whe14] Jake Wheat. simple-sql-parser. `https://hackage.haskell.org/package/simple-sql-parser-0.4.1`, 2014. Accessed: 2016-02-28.