

[Naam van leerling]  
[Naam van klas]  
[Naam van onderwijzer]

[Documenttitel]

# Inhoudsopgave

Hoofdstuk 1 #

Deel 1.1 #

Deel 1.2 #

Deel 1.3 #

Hoofdstuk 2 #

Deel 2.1 #

Deel 2.2 #

Deel 2.3 #

Hoofdstuk 3 #

Deel 3.1 #

Deel 3.2 #

Deel 3.3 #

# Overall Requirements

We use the that Railroad Diagram Generator on <http://bottlecaps.de/rr/ui> to generate syntax diagrams. For that reason, the syntax discussed here is written in the the [W3C](http://www.w3.org/)'s EBNF notation. Upon delivery of the result, the syntax must be documented on the Ampersand-wiki (<http://wiki.tarski.nl/index.php/Ampersand_Syntax>) with syntax diagrams that are usable for students and other users of Ampersand.

# ENFORCE statement

## Current situation

Rule enforcement in Ampersand is implicit. A rule can be attached to a role, e.g.

ROLE Tutor, Coach MAINTAINS attendance

If a rule has one or more roles attached to it, that rule gets ‘deferred enforcement’. If a rule has no roles attached to it, it gets ‘immediate enforcement’.

## PROBLEM

The enforcement of rules is too coarse for practical purposes. First, we need room for multiple enforcement strategies. Secondly, we need to apply different strategies for different roles.

## Desired situation

We want the following syntactical structure

Enforce-Statement

::= 'ENFORCEMENT' Rule-id ( ',' Rule-id )\*

'BY' ( 'REPORTING'

| 'REJECTING'

| 'DEFERRING TO' ( Role-id ( ',' Role-id )\* )

| 'RESOLVING'

| 'IGNORING'

)

An enforce statement is allowed anywhere in a context, and it is valid throughout that context. To ensure that each rule has precisely one enforcement strategy, IGNORING is the default and multiple strategies per rule are a compile time error.

The semantics are as follows:

1. REPORTING means that violations are reported. This is currently not implemented in Ampersand
2. REJECTING means that the rule must be satisfied in order to commit a transaction. This is implemented in the current compiler.
3. DEFERRING means that the rule may remain violated. The violation is treated as a signal. This too is currently implemented.
4. RESOLVING means that the rule must be resolved automatically. This requires the presence of ECA-rules, either derived by the system or specified by the user. This is not implemented currently.
5. IGNORING means that no enforcement is done. The reason to specify rules without enforcement is to help the normalizer, which may use this rule in its derivations.

An enforce statement may also be written inside an interface, where it overrules the enforcement of the larger context.

## Design considerations

This syntax has five strategy identifiers, each with a different syntax. That creates room for future refinement, either by adding new alternatives, but also by adding more syntax after each strategy identifier. This is illustrated in the DEFERRING strategy, which requires one or more roles to defer to. In the future, we might easily add syntax at the end of the ENFORCE statement in a similar manner.

This proposal makes the ROLE statement obsolete. However, it may be retained for a while to allow a smoother transition path.

# MULTIPLICITY syntax

## Current situation

Multiplicity properties are rules in their own right. For instance, if relation r is univalent, the Ampersand modeler can write:

RULE Uni\_r : r~;r |- I

RULE Tot\_r : I |- r;r~

In the current syntax, the declaration of relation r allows a shorter, more convenient notation for this:

RELATION r[A\*B] [UNI,TOT]

The effect of specifying UNI, TOT, or any other property is that the rule corresponding to that property is generated as if it were written in full. So the shorthand is pure syntactic sugar. Notice that the enforcement is implicitly defined as immediate.

## PROBLEM

It is useful to have a shorthand, but it would be nice to specify enforcement for the corresponding rules. Currently, if you want deferred enforcement, the rule must first be specified in full and then allocated to the appropriate role.

## Desired situation

We can enhance the ENFORCE syntax to provide Ampersand with a nice shorthand notation. We might write, for example:

ENFORCEMENT UNI, TOT r BY DEFERRING TO Tutor

This yields a slightly more complex ENFORCE statement:

Enforce-Statement

::= 'ENFORCEMENT' Prop-rul ( ',' Prop-rul)\*

'BY' ( 'REPORTING'

| 'REJECTING'

| 'DEFERRING TO' ( Role-id ( ',' Role-id )\* )

| 'RESOLVING'

| 'IGNORING'

)

Prop-rul

::= Prop-id ( ',' Prop-id )\*

Relation-ref (',' Relation-ref)\*

| Rule-id ( ',' Rule-id )\*

Prop-id ::= 'UNI' | 'TOT' | 'SUR' | 'INJ'

| 'SYM' | 'ASY' | 'RFX' | 'IRF' | 'TRN'

## Design considerations

This syntax would make the current shorthand notation redundant. However, it may be retained for a while to allow a smoother transition path.

# Population

## Current situation

Population may be specified in two ways: either at the end of a relation declaration (ancient syntax) or by means of a POPULATION statement (current syntax). Both ways allow the user to specify the population of binary relations. This population is used by the software generator to build the initial population of the database.

## PROBLEM

Specifying population in the form of binary tables produces many lists with lots of quotes and parentheses, in which it is difficult to edit. It is highly desirable to have a syntax in which (rectangular) contents of tables, for instance from a spreadsheet, can be pasted.

## Desired situation

In order to make specifying populations more friendly, a syntax proposal is required that allows us to specify rectangular tables, wider than 2 columns.

## Design considerations

Momentarily, we have a program called parsCSV, which translates an excel-CSV file into Ampersand source format. At the same time, we use a PHP-program to read CSV files directly into the database. Ideas exist to import XML-messages for this purpose.

There is a proposal by Lex Wedemeijer for more user friendly syntax:

Set-of-atoms

::= '{' Atom ( ',' Atom )\* '}'

Atom

::= String | Quoted-string

Set-of-tuples

::= '{' Atom-tuple (',' Atom-tuple)\* '}'

Set-or-atom ::= Atom | Set-of-atoms

Atom-tuple ::= Set-or-atom '\*' Set-or-atom

This proposal solves the problem of too many quotes, but does not provide the option to specify rectangular tables.

# Context and Patterns

## Current situation

Currently, both a PATTERN and a PROCESS must be defined inside a CONTEXT.

## PROBLEM

Patterns are meant to be re-used in different contexts. For that reason it makes more sense to define patterns outside a context, and import that pattern in each context where you want the rules of that pattern to be valid.

A complication arises with name spaces. If multiple patterns are imported in a context, conflicting names might cause havoc. For that reason, an aliasing mechanism is required that creates a consistent namespace throughout the context, without the need to edit any pattern.

Another complication might be that the meaning of rules in a pattern may change as it is used in different contexts. That is a result of the current binding process, which binds unbound relations to declarations in the namespace of the context. For this reason, patterns must not have unbound relations if defined outside the scope of a context. I.e. all relations must be declared inside the pattern itself.

## Desired situation

## Design considerations

# Order Independence

## Current situation

Some of the statements in the current syntax prescribe the order in which details are specified. Consider for example this relation declaration

RELATION aan[Bericht\*Persoon] [UNI]

PRAGMA "" " is gericht aan "

MEANING "Elke persoon wordt geregistreerd."

PURPOSE RELATION aan[Bericht\*Persoon]

{+Een bericht kan aan meerdere personen gericht zijn. Zolang het bericht niet verzonden is, hoeft een geaddresseerde niet geregistreerd te staan.

-}

In this statement, the order of things is prescribed: first the name, then the signature, etcetera. The relation declaration currently has the following syntax:

Relation-Declaration

::= 'RELATION' Relation-id Signature

('BYPLUG'?)

(Properties?)

(Pragma?)

(Meaning?)

(Content?)

The reason for this is that the modeler can specify a lot of information about a relation and still mention it only once. Splitting these things in smaller parts gives more freedom, but makes it necessary to write a reference to the relation for every detail written in the model. The purpose statement shows that.

## PROBLEM

The order of things in a relation declaration is difficult to remember and therefore a continuous source of frustration to novice Ampersand modelers. Some modelers would like to isolate the meanings of all relations in a separate section of their model, just as they do with purposes. However, since meanings are linked syntactically to the declaration, they can put the purposes together but the meanings have to stick with the declarations. This problem occurs for other statements as well, whenever the order in syntax is prescribed.

## Desired situation

We would like a way to have a separate statement for each thing, yet without the penalty of repeated references if these things are put together.

Relation-Declaration

::= 'RELATION' Relation-id Signature

Properties

::= 'PRAGMA' Relation-ref? pProps

Pragma

::= 'PRAGMA' Relation-ref? pPragma

Meaning-Declaration

::= 'MEANING' Relation-ref?

Language-ref?

TextMarkup?

String

The meaning of this syntax is that omission of a relation-reference takes the relation reference of the previous statement, if defined. In this way, we could still write:

RELATION aan[Bericht\*Persoon] [UNI]

PRAGMA "" " is gericht aan "

MEANING "Elke persoon wordt geregistreerd."

PURPOSE

{+Een bericht kan aan meerdere personen gericht zijn. Zolang het bericht niet verzonden is, hoeft een geaddresseerde niet geregistreerd te staan.-}

But we may also write:

RELATION aan[Bericht\*Persoon] [UNI]

PRAGMA aan[Bericht\*Persoon] "" " is gericht aan "

MEANING aan[Bericht\*Persoon]

"Elke persoon wordt geregistreerd."

PURPOSE aan[Bericht\*Persoon]

{+Een bericht kan aan meerdere personen gericht zijn. Zolang het bericht niet verzonden is, hoeft een geaddresseerde niet geregistreerd te staan.-}

## Design considerations

This proposal comes from the desire to have

1. A compact notation if one relation is specified together with all its details.
2. An order-independent notation, so the user does not have to remember the prescribed order;
3. A notation that allows separating out some of the details, for instance for a modeler who wants to put all meanings together.

# Pragmas

## Current situation

Each relation declaration can be provided with a PRAGMA. This serves as a natural language template, from which statements in natural language are assembled. A pragma has three strings: l, m, and r. Two atoms A and B are assembled in a sentence by concatenating l++A++m++B++r.

## PROBLEM

This way of assembling strings is way too restrictive, when compared to the way we assemble text for violations.

## Desired situation

The Ampersand modeler wants to assemble natural language sentences using the same mechanism as used in VIOLATION.

## Design considerations

This makes the assembly with three strings obsolete. It may be retained, though, to create a smooth transition path.

# Language

## Current situation

## PROBLEM

Default context languages are not handled correctly on includes. For example, consider the inclusion of this context:

CONTEXT Includee IN DUTCH

...

PURPOSE .. Iets {+ Nederlandse uitleg -}

in an English context:

CONTEXT Includer IN ENGLISH

...

The language assigned to Iets will now be English and not Dutch. The reason is that the defaults are assigned in P2A\_Converters, where the individual contexts (with their languages) have already been merged into one context.

## Desired situation

To do it well, either the P structure needs to support nesting, or we can add a state to the parser, and use this to fill in the language when we encounter elements without a language. The latter solution should be easy to implement using the UU.Parsing.StateParser module.

## Design considerations

# Upper and lower case identifiers

## Current situation

Concept identifiers start with an uppercase letter and relation identifiers with lowercase letter. This is enforced by the parser.

## PROBLEM

The problem is that there is no formal necessity for this distinction. So, the parser is overrestrictive.

## Desired situation

We would like the parser to use one definition of identifier, which is used throughout the language. The Ampersand modeler gets the choice how to keep concept names and relation names apart.

## Design considerations

# BIBLIOGRAfie

**Eerste referentie.** De tekst begint bij de linkermarge van het papier. Er wordt een dubbele regelafstand gebruikt. Wanneer de vermelding langer is dan één regel, springt de tweede regel automatisch in.

**Meer referenties.**