EPSILON DRIVER FOR ASTAH GSN

CHALLENGES IN IMPLEMENTING AN EPSILON DRIVER FOR ASTAH GSN

By

BARAN KAYA, M.Eng.

A Thesis
Submitted to the Department of Computing & Software
And School of Graduate Studies
of McMaster University
in Partial Fulfillment of the Requirements
for the degree
Master of Engineering

McMaster University

Master of Engineering (2020)	McMaster University
(Computing & Software)	Hamilton, Ontario, Canada

TITLE:	Challenges in Implementing an Epsilon Driver for Astah GSN
AUTHOR:	Baran Kaya M.Eng. (Software Engineering) McMaster University, Hamilton, Canada
SUPERVISOR:	Dr. Richard Paige
NUMBER OF PAGES:	

ABSTRACT

CONTENTS

ABSTRACT 4

1. INTRODUCTION 6
2. RELATED WORK 6
3. PROJECT REQUIREMENTS 6
4. DESIGN 8
5. IMPLEMENTATION 10
6. TESTING/EVALUATION 14
7. CONCLUSION 19
8. FUTURE WORK 19
9. REFERENCES 20

1. INTRODUCTION

Intro

Explain sections!!!!

2. RELATED WORK

SAFETY CASES…

MDE…

EPSILON…

Epsilon Astah-GSN driver project is my McMaster University Master of Engineering project. The main goal of the project is to use the Eclipse Epsilon Model-Driven Engineering tool to modify Astah GSN models. The integration between these two programs made with the Epsilon Model Connectivity Layer (EMC). This project is developed for General Motors.

Epsilon is an Eclipse plugin for Model-Driven Engineering processes. Epsilon website defines it as “Epsilon is a family of mature languages for automating common model-based software engineering tasks, such as code generation, model-to-model transformation, and model validation, that work out of the box with EMF (including Xtext and Sirius), UML, Simulink, XML and other types of models.” [1].

GSN stands for Goal Structuring Notation. GSN Working Group defines it as a visualization of safety argument elements and relations. Requirements, claims, evidences, and contexts are some of the safety argument elements [2]. GSN aims to show these elements and the connection/relationship between each element. Astah GSN is one of the modeling programs that specifically designed for creating and modifying GSN models.

3. PROJECT REQUIREMENTS

The main goal of the project is to be able to use Epsilon on Astah GSN models so that users can work on the Astah GSN models. In order to do that, the driver to be able to parse the Astah GSN files correctly. Astah GSN saves its model files with ‘.agml’ extension; however, it provides XMI import/export features for GSN models. Astah GSN encodes AGML files so users cannot reach these files content. All EOL functionality can be used within other Epsilon languages as well. Therefore, in this driver project XMI files used. Also, there was Plain XML driver for Epsilon and its used as a based version of the project driver.

Epsilon has one core language (Epsilon Object Language, EOL) and ten task specific languages. Task specific languages derived from EOL thus, integrating models to EOL is sufficient for all other language support. The first requirements show the EOL integration and the later ones show task specific language integrations.

1. Users should be able to load Astah GSN models into the Epsilon in Eclipse IDE.

2. Users can read/access the GSN models with EOL.
   - 2.1. Users should be able to access the entire GSN model
   - 2.2. Users should be able to access certain types of elements
     - 2.2.1. Users should be able to access all nodes
     - 2.2.2. Users should be able to access all links
     - 2.2.3. Users should be able to access certain types of nodes (e.g. goals)
     - 2.2.4. Users should be able to access certain types of links (e.g. asserted evidence)
   - 2.3. Users should be able to access specific element
     - 2.3.1. Users should be able to access specific node
     - 2.3.2. Users should be able to access specific link
   - 2.4. Users should be able to access elements' attribute values
     - 2.4.1. Users should be able to access element's content
     - 2.4.2. Users should be able to access element's ID
     - 2.4.3. Users should be able to access element's type
     - 2.4.4. Users should be able to access element's XMI:ID
     - 2.4.5. Users should be able to access element's XSI:TYPE
     - 2.4.6. Users should be able to access link's target
     - 2.4.7. Users should be able to access link's source

3. Users can update the GSN models with EOL.
   - 3.1. Updating element content
   - 3.2. Updating element ID
   - 3.3. Updating element type
   - 3.4. Updating element XMI:ID
   - 3.5. Updating element XSI:TYPE
   - 3.6. Updating element target link
   - 3.7. Updating element source link

4. Users can create new elements and append these elements into the GSN model.

5. Users can delete elements in the GSN model.

4. DESIGN

As mentioned before, Epsilon Plain-XML driver used as a base for this project. Plain-XML driver can parse XML files and users could load, read and update XML models in Epsilon with this driver. However, this driver is not useful for Astah GSN XMI files. Because, GSN XMI files store every element in the GSN model with the same tag but Plain-XML driver parses files via different tag names. Astah GSN uses XML attributes to store every elements' values. Therefore, Astah GSN driver heavily modified for attribute values instead of tag names.

Astah-GSN commercial tool, XMI export feature

Explain XMI elements' each attribute

Node Elements	Link Elements
• Goal • Strategy • Solution • Context • Assumption • Justification	• Inference (Asserted Inference) • Evidence (Asserted Evidence) • Context (Asserted Context)

Table 1: GSN Element Types

Why did I choose Epsilon HTML driver?

There are multiple built-in drivers for different models in the Epsilon. For instance, Plain-XML driver is coming with Epsilon download. However, I needed to create a new driver plugin for Astah GSN models. Therefore, I searched for other model drivers for Epsilon that isn't built-in Epsilon. EpsilonLab GitHub page [3] has a few EMC drivers for several models such as HTML, JDBC, JSON and more. The most similar model driver to Astah GSN was HTML due to XML structure.

After cloning EMC-HTML git repository, I tried to run it on Epsilon source code. The first step was importing the 2 HTML project packages into Eclipse Epsilon workspace. Even though HTML repository have 6 different packages, only 2 of them are necessary for running the HTML driver. Other 4 packages are test and example packages. After that step, Epsilon source code rebuilds itself. Then running Epsilon on the new Eclipse application is sufficient for HTML driver. The new driver is an Epsilon model in the Run Configuration of every Epsilon languages. For testing purposes, I created a new EOL file. Then I created new run configuration for this EOL file and HTML Document was one of the model options in the Model Selection tab. After selecting it, EOL script will use HTML file as a model.

Since I have no experience with Epsilon plugin development, I used HTML driver as a base plugin project. I changed all names in the project and used new image for Astah GSN model selection tab. After that, I examine the HTML driver classes. The main class is called HTML model and it invokes getter and setter classes for different functionalities. However, HTML getter and setter classes functions only calls Plain-XML getter and setter functions.

I just used HTML driver for plugin features and changed all model, getter and setter classes/methods.

Java Jsoup library for HTML parse…

How did I use Plain-XML driver?

HTML plugin was the most similar example to Astah GSN driver that I am going to work on.

Java W3C Dom library for Node and Element…

Plugin features didn't change…

Using ID for determining element type changed…

Determining element types such as Goal, Solution or Asserted Evidence is one of the important parts of the Astah GSN Driver. In early design, each user queries were parsed before accessing the element in the XMI file. Thus, if the given query element wasn't in the GSN type, it would return an empty result. For example, a user can only get elements with IDs like G1, Sn4, C2, … These IDs gave by Astah GSN by element's type. Goal elements' ID starts with G, Solution with Sn and Context with C. However, element IDs don't have to start with element type letters. In this case, GSN type parse for custom IDs such as CA1, AR-C1 returned null and Epsilon returned error message for couldn't finding the element with given custom ID.

In later designs, custom ID access added to the driver. Because GSN standards don't require IDs to start with element type letters. That's why instead of returning null for custom ID queries, the driver compares every ID in the model and if it couldn't find it then it returns null. For custom IDs, GSN type parser still returns null but after that, it checks every element for custom ID probability.

Another update for not determining types with element ID would be the ".gsntype" query. Before this change, the node elements' type were found by element IDs and link elements' IDs were found by *xsi:type* attribute. After discovering custom IDs, *gsntype* function has to change. However, there is a problem with determining element type without IDs. The only way to finding element types is *xsi:type* attribute. But, some of the node elements use the same xsi:type attributes. For instance, Goal, Assumption, and Justification elements all use "ARM:Claim" value for *xsi:type* attribute. So, for determining types, I have to use other attributes. The only difference between Goal-Justification pair and Assumption elements is assumed attribute. For Assumption elements this attribute's value is true but for Goal and Justification elements this attribute's value is false. Now we can determine Assumption elements from Goal and Justification elements. For determining between Goal and Justification elements, there aren't any attributes. The only difference between these 2 elements is their connections. Goal elements can connect all 3 types of link elements but Justification elements can only connect to the Asserted Context element's target side. Thus, if the given element's *xmi:id* stored in one if the Asserted Context attributes' target attribute, that means the element's type is Justification.

The Goal-Justification situation is the same for Solution-Context pairs. Instead of "ARM:Claim" *xsi:type* attribute Solution-Context elements use "ARM:InformationElement". Similarly, Context elements can only be connected to Asserted Context links' target side. Hence, the same function used for determining Justification and Context elements. All 5 element examples could be seen below.

- Goal element:

```
<argumentElement xsi:type="ARM:Claim" xmi:id="_fvLpEJq4EeqyzooT9RpXrQ"
id="G1" description="" content="Control System is acceptably safe to
operate" assumed="false" toBeSupported="false"/>
```

- Assumption Element:

```
<argumentElement xsi:type="ARM:Claim" xmi:id="_fvLpI5q4EeqyzooT9RpXrQ"
id="A1" description="" content="All hazards have been identified"
assumed="true" toBeSupported="false"/>
```

- Justification element:

```
<argumentElement xsi:type="ARM:Claim" xmi:id="_fvLpJJq4EeqyzooT9RpXrQ"
id="J1" description="" content="SIL apportionment is correct and
complete" assumed="false" toBeSupported="false"/>
```

- Solution element:

```
<argumentElement xsi:type="ARM:InformationElement"
xmi:id="_fvLpE5q4EeqyzooT9RpXrQ" id="C1" description=""
content="Operating Role and Context" url=""/>
```

- Context element:

```
<argumentElement xsi:type="ARM:InformationElement"
xmi:id="_fvLpH5q4EeqyzooT9RpXrQ" id="Sn1" description=""
content="Formal Verification" url=""/>
```

5. IMPLEMENTATION

There are 5 Java classes that implemented or changed from Plain-XML driver by me. These 5 classes are: GsnModel, GsnProperty, GsnPropertyType, GsnPropertyGetter, and GsnPropertySetter. Each class and its methods will be explained in this section below.

Link elements' target and source attributes are reverse…

5.1. GSN Model

GSN Model is the Astah GSN driver's main class. It consists of file operations, model load and store operations, all elements collector, new element creator, and element deletion methods.

XMI uses XML structure therefore, Plain-XML driver's file operations weren't changed in the Astah GSN driver. Moreover, model loading and storing functions as well as removing an element and collecting all elements functions are the same as Plain-XML driver. Most of the changes done within new element creation, and owns the function.

Plain-XML element creator function was using tag name for new elements but the Astah GSN XMI file uses the same tag name for all elements except root tag. The new function takes as a type parameter and parses it in the GsnProperty class. It returns element's type such as Goal, Strategy, or root. Then new attributes and values are created with type data. All node and link elements have 5 common attributes: *xsi:type, xmi:id, id, content, and description. Content* and *description* attributes created empty. *Xsi:type* value comes from the GsnProperty class's parser function. *Id* value also comes from GsnProperty parse function but it only consists of the new element's type prefix such as G for Goal typed element. The hardest part was generating new *xmi:id* for the new element. Each element unique value and Astah GSN uses the root element's *xmi:id* to generate new *xmi:id* values for each element. Since I don't know which parameters Astah GSN uses for ID generation, I couldn't implement a working ID generator. So, instead of empty *xmi:id* values, current function puts type prefix letter + "MustBeUnique" string in the *xmi:id* attribute. For example, a new goal element's *xmi:id* attribute value will be "GMustBeUnique".

Another difference between Plain-XML and Astah GSN drivers would be appending new elements into the model. Plain-XML driver appends new elements into the model when they are created but Astah GSN doesn't append them to model directly. Appending requires another command which is ".append". This function will be explained in the setter class.

Owns function in model class responsible for calling the right class functions. For instance, if *owns* function returns falls for given input then it calls the superclass of the GsnProperty which are JavaProperty. JavaProperty class doesn't have any XML parser so, for correct elements, the *owns* function has to return true so that model class can call GsnProperty class. Two more conditions added into *owns* function. The first one is added for getting the root element and the second one is added for list elements. These list elements are generated for getting all elements with a specified type like all goal elements. This element list should call the GsnProperty methods thus it returns true for element array lists.

5.2. GSN Property

GsnProperty is a parser class. It parses given elements and returns a new created GsnProperty object. This class consists of a few protected attributes.

- ***GsnPropertyType* gsnPropertyType:** Type of the element
- **String idPrefix:** Given type's ID prefix (e.g. G for Goal).
- **String xsiType:** *xsi:type* attribute value for given type (e.g. ARM:Claim for Goal).
- **boolean isNode:** Given element is node or not.
- **boolean isLink:** Given element is link or not.
- **boolean isRoot:** Given element is root or not.

There are also three functions in this class. Two of them are parser and the last one is element type determiner. The first parser class gets string input and parses it. This string input could be ID like G1, A4, J5 or a type name like solution, strategy, … With these inputs, it creates a new GsnProperty object, assigns the above variables according to the element type, and returns it. If it couldn't parse the given string properly, it returns null. Elements with custom ID values return null by this parser function so they use second parser function. This parser function gets an element object as an input and parses it by *xsi:type* attribute. Nonetheless, some elements have the same values for *xsi:type*. In this case, the parser function calls the third function which is "isJustificationOrContext". This function determines if the given element Goal or Justification and also Solution or Context. As mentioned before, Context and Justification elements only connect to the Asserted Context link's target side. So, this function checks every Asserted Context elements' target attribute and if it finds the given element's *xmi:id* in them, it returns true. Otherwise, it returns false.

5.3. GSN Property Type

This class only consists of element types enumaration. Six node types plus three link types and the total 9 GSN types.

5.4. GSN Property Getter

Getter class is used for all element access queries. There are several different getter commands in the Astah GSN driver and all of them are in the getter classes invoke function.

- All elements (.all)

This command's input parameter is root element. Function parses child elements and return them as an Epsilon's sequence type.

- All node elements (.nodes)

Nodes command parses root element and create a new list with only node elements. Node elements have non-empty ID attributes. It loops over all child elements and only adds elements with non-empty ID attribute into the result list.

- All link elements (.links)

Links command works similar to nodes command. It parses root element, loop overs every child and only adds elements with empty ID into the list. Then it returns the result list.

- Element by ID (.G1, .c4)

There are two types of search by ID methods: proper ID and custom ID. Getting element by ID is the last case in the invoke function. The proper ID part (e.g. G4, S2, C1) calls GsnProperty parser with string ID variable. If it can parse it. It will return a new GsnProperty object. Then, it checks every elements' ID and if it can find it, it returns the element. Custom ID part works the same bu it only invokes this part after returning null from GsnProperty parser. If there wasn't any element with given custom ID, invoke function returns null.

- Element by type (.strategy, .assumption)

Element types are determined via GsnProperty parser. This part loops over every child element and calls parser to determine element's type. If the types are match, it adds element into the result list. Finally, it returns the result list.

- Link element with source and target IDs (s_G1_t_C2, t_a1_s_s3)

This case takes string input like "s_G1_t_C2" and parses it to get two node ID values. The reason I used "_" characters between each part is, Epsilon doesn't work with "–" character. After parsing sting, it finds node elements with given two IDs. If both of the nodes are found, it loops over every link element and tries to find given nodes *xmi:id*s in the target and source attributes. Finally, it returns the link element or null depending on a search result.

- Element's type (.gsntype)

Gsn type case's input could be root element, list or just an element. It uses GsnProperty element parser to determine given element/s type and returns it.

- Element's target (Node and link) (.target)

Target getter works different for node and link elements. If the element is a node, it returns given node's all links that are targeted to given node. If the element is a link, it returns link's targeted node element.

- Element's source (Node and link) (.source)

Source case works the same as the target case. The only difference is, it checks source attribute instead of the target attribute.

- Element's content (.content)

This case directly returns given element/s content string.

- Element's ID (.id)

Returns given element/s ID value/s.

- Element's *xmi:id* (.xmiid, .xmi_id)

Returns given element/s *xmi:id* value/s.

- Element's *xsi:type* (.xsitype, .xsi_type)

Returns given element/s *xsi:type* attribute value/s.

Find element by attribute name and value

Get element attribute

Find link by node IDs

5.5. GSN Property Setter

Every element value update calls use this setter class. Similar to getter class, setter class only uses one invoke function. Since all elements' values cannot change, setter class doesn't have that much different commands.

- Set element's content (.content)

Sets given element's content attribute value.

- Set element's ID (.id)

ID command sets given element's ID attribute.

- Set element's *xmi:id* (.xmiid, .xmi_id)

This command updated given element's *xmi:id* attribute value.

- Set element's *xsi:type* (.xsitype, .xsi_type)

Updated given element's *xsi:type* attribute.

- Set link element's target (.target)

Changes given link element's target attribute. It takes ID as a string, finds the node with given ID, gets node's *xmi:id* attribute and sets given link element's target attribute to the new nodes *xmi:id*.

- Set link element's source (.source)

Works like target setter case.

- Set element's GSN type (.gsntype)

Takes new type string as an input. It calls GsnProperty parser to find given type's *xsi:type* value and sets it to the given element's *xsi:type* attribute.

- Append a new element into the model (.append)

Takes element object as an input. If the new element object doesn't have an ID with digits (e.g. G, S), it finds the highest ID number for new element's type and assigns highest/largest ID to the new element. Then it adds the new element into the root element as a children tag.

Get highest number of given typed element ID

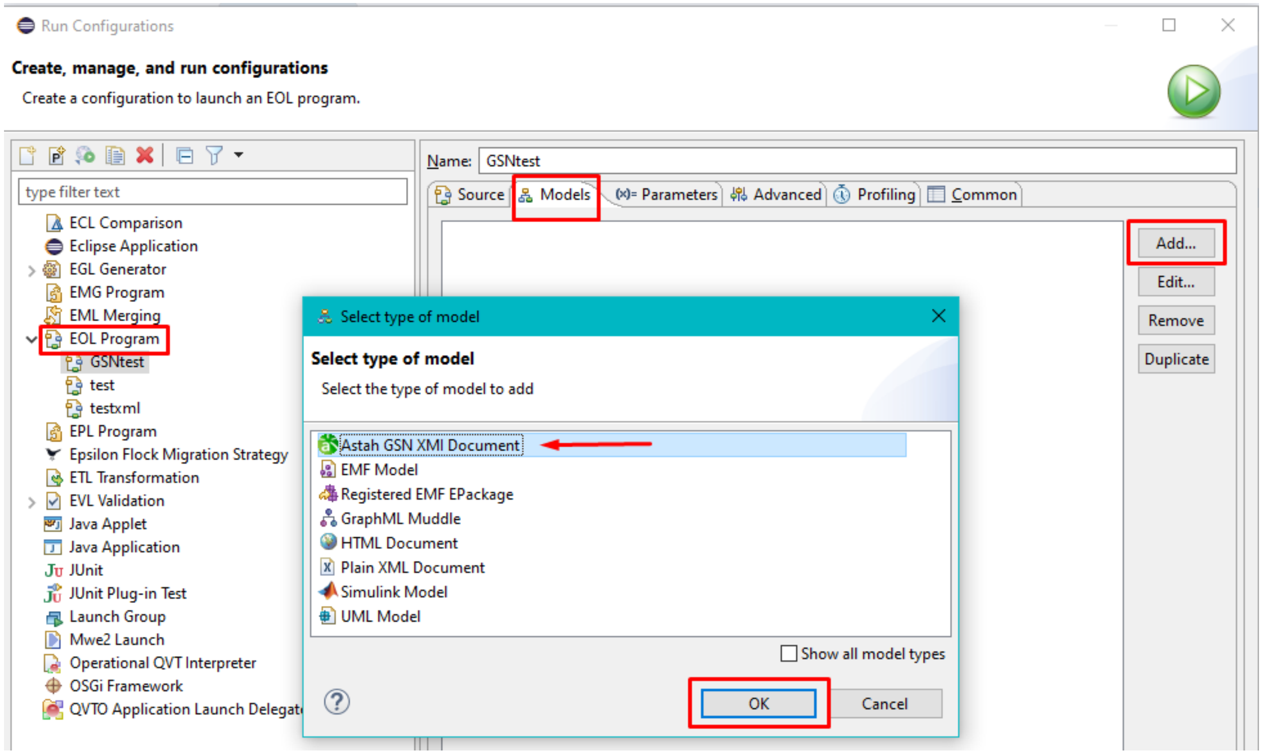
6. TESTING/EVALUATION

In this section usage examples are given. Each requirement would be explained with examples.

1. Loading the Astah GSN model into Epsilon

Since this driver based on Plain-XML driver, loading model file code is the same. The only difference between these 2 drivers is names. For instance, “Plain-XML Document” changed to “Astah GSN XMI Document”. Other than names rest of the model loading code is work like Plain-XML driver. Below steps explains how to load an Astah GSN model into Epsilon.

a. Create a new EOL file in Eclipse IDE with Epsilon.
b. Right click the EOL file and click *Run As* > *Run Configuration*.
c. Choose *EOL Program* and create new Run Configuration.
d. Choose your EOL files in the *Source* tab.
e. Go to *Model* tab and click *Add* button
f. Choose *Astah GSN XMI Document* and click *OK* (Figure 1).


Run Configurations

Create, manage, and run configurations

Create a configuration to launch an EOL program.

Name: GSNtest

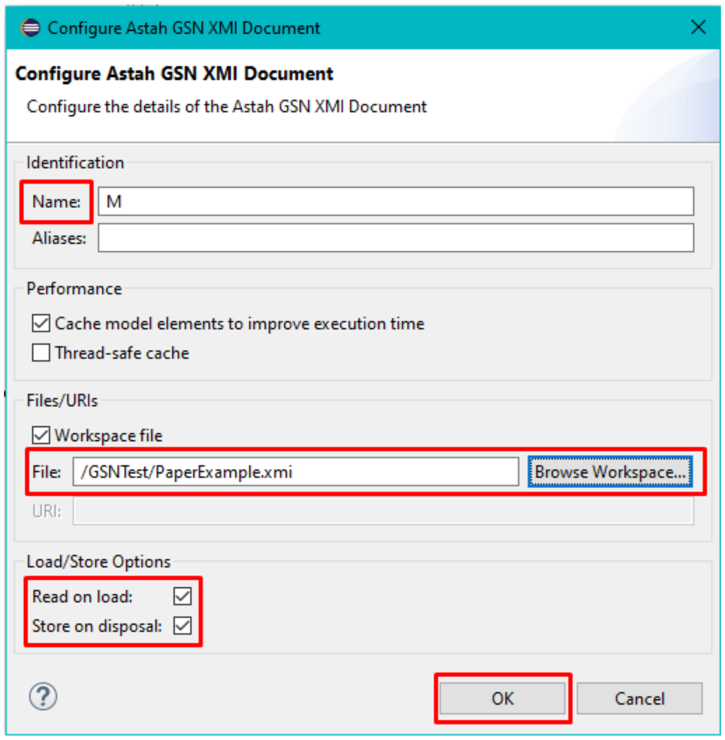
Select type of model

Select the type of model to add

- Astah GSN XMI Document
- EMF Model
- Registered EMF EPackage
- GraphML Muddle
- HTML Document
- Plain XML Document
- Simulink Model
- UML Model

Figure 1: Loading Astah GSN model into EOL

g. Give a name to your model, it is not very important if you don’t want to use multiple models in the same EOL script.
h. Choose your XMI file if it’s already in the workspace. If not, add your model file into workspace.
i. If you are going to update/modify the GSN model, choose both *Read on load* and *Store on disposal* options (Figure 2).


Configure Astah GSN XMI Document

Configure Astah GSN XMI Document

Configure the details of the Astah GSN XMI Document

Identification

Name: M

Aliases:

Performance

☑ Cache model elements to improve execution time

☐ Thread-safe cache

Files/URIs

☑ Workspace file

File: /GSNTest/PaperExample.xmi

URI:

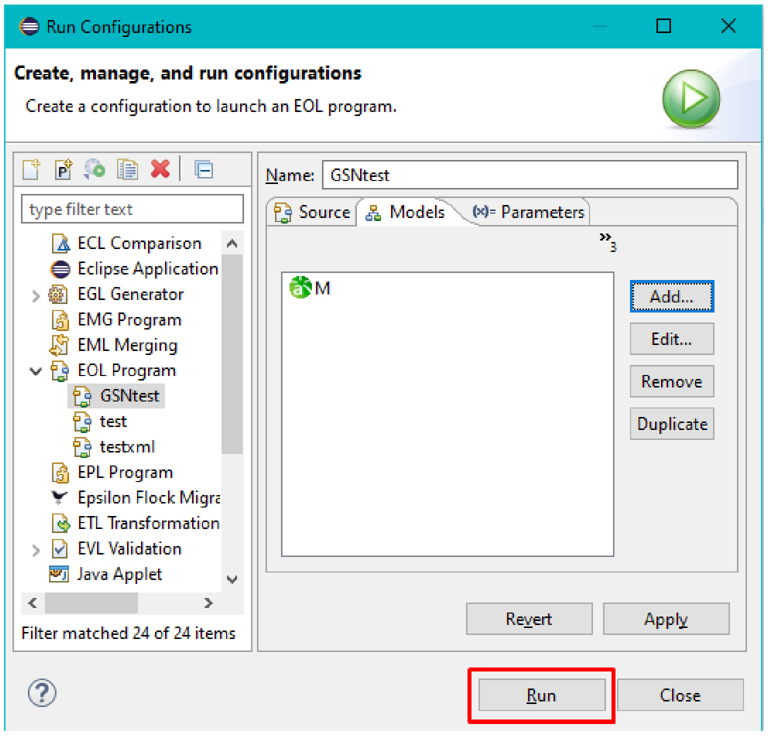
Load/Store Options

Read on load: ☑

Store on disposal: ☑

Figure 2: Loading model configurations

j. After that, you can run EOL script with *Run* button. EOL will run on your Astah GSN model (Figure 3).


Run Configurations

Create, manage, and run configurations

Create a configuration to launch an EOL program.

type filter text

- ECL Comparison
- Eclipse Application
- EGL Generator
- EMG Program
- EML Merging
- EOL Program
  - GSNtest
  - test
  - testxml
- EPL Program
- Epsilon Flock Migra
- ETL Transformation
- EVL Validation
- Java Applet

Filter matched 24 of 24 items

Name: GSNtest

Source | Models | Parameters

M

Figure 3: Running EOL with Astah GSN model

2. Reading/accessing GSN models with EOL

Plain-XML driver has 2 classes for getters and setters. Reading and accessing model calls getter class functions. Astah GSN driver's getter class completely changed and it has minimal similarities with Plain-XML getter class.

Most of the changes made for getting attribute values rather than tag names. Plain-XML getter class has method for getting tags, child tags, attribute values, etc. In Plain-XML driver users can only select different tag named elements but Astah GSN XMI document requires getting different attribute valued elements.

Plain-XML documents can have several layered elements so driver can get the root element or any child element. Astah GSN driver have two different options: getting the root element or root's child elements. Because GSN XMI document only has the root element (tag name: ARM:Argumentation) and its child elements (tag name: argumentElement). Some of the methods get root element and some of them only get children. For instance, *gsn.all* call parses the document and returns root element. On the other hand, *gsn.goal* or *gsn.S4* calls parse the document, find the specified child elements and return them as a list or a single element.

Getter Commands	Command Explanation
`gsn.all`	Returns entire model and all elements
`gsn.nodes`	Returns all node elements
`gsn.links`	Returns all link elements
`gsn.goal`	Returns all goal elements
`gsn.assertedcontext`	Returns all asserted context (link) elements

`gsn.all.content`
`gsn.context.content`
`gsn.Sn13.content`
	Returns specified element/s' content attribute value/s. The results could be Sequence or string depending of an element. Link elements' don't have content attribute so it returns empty string ("").

`gsn.G1.gsntype`	Returns given element/s' GSN type
`gsn.strategy.id` `gsn.c23.id`	Returns given element/s' ID. Link elements' don't have ID thus it returns empty string.
`gsn.S5.xmiid` *OR* `gsn.a9.xmi_id`	Returns given element/s' xmi:id attribute. Each element has unique xmi:id values.
`gsn.j5.xsitype` *OR* `gsn.goal.xsi_type`	Returns given element/s' xsi:type attribute. Same elements (e.g. goal and assumption) have the same xsi:type values.
`gsn.Sn3.target` `gsn.context.target`	Returns link elements that have target value as given node element/s.
`gsn.g2.source` `gsn.strategy.source`	Returns link elements that have source value as given node element/s.
`gsn.t_g1_s_g2`	Returns link element that has target value is G1 and source value is G2.
`gsn.all.last`	Returns last element of given elements list
`gsn.solution.first`	Returns first element of given elements list
`gsn.S3.content.println()`	Prints given value/s

Table 2: Astah GSN Driver Element Getters

Some of the getters could be combined differently. For example, *`gsn.goal.last.content.println()`* and *`gsn.goal.content.last.println()`* prints the same result. The difference between these 2 commands is simple. The first command gets all goal elements list, then finds the last goal element and prints its content attribute value. The second command gets goal elements list, then gets all goal elements contents and creates new list later it prints the last content in that list. Thus, the first command is faster than the second one because, it doesn't get all goal elements' content attribute, it just gets one goal element's content attribute and prints it.

3. Updating GSN models with EOL

Updating elements commands call setter class functions. Most of the element attributes can be set via below commands. Getter and setter commands are the same. The only difference is setters requires "=" character and new value after equals character.

Setter Commands	Type	Command Explanation
`gsn.sn13.content = "New content";`	String	Updates given element's content value
`gsn.g3.id = "G6";`	String	Updates given element's ID (ID attributes must be unique!)
`gsn.j15.xmiid = "fvLpH5q4Eeqyz11T9RpXrQ";`	String	Updates given element's xmi:id attribute. However, Astah GSN generates unique xmi:id values based on model and element location. Therefore, using this command can corrupt the model file.
`gsn.c7.xsitype = "ARM:ArgumentReasoning";`	String	Updates given element's xsi:type attribute. Changing xsi:type without changing id might corrupt model file.
`gsn.t_s1_s_g1.target = gsn.sn7;`	Node element	Updates given link element's target attribute to new node element's xmi:id. New value must be node element.
`gsn.t_J1_s_G13.source = gsn.J2;`	Node element	Updates given link element's source attribute to new node element's xmi:id. New value must be node element.
`gsn.a12.gsntype = "goal";`	String	Updates given element's type attribute. Changing xsi:type without changing id might corrupt model file.

Table 3: Astah GSN Driver Element Setters

4. Creating new elements in GSN model with EOL

Creating new element command uses *`new`* keyword. Using *new* keyword only creates new element object but it doesn't append this new object into the model.

Creator Command	Command Explanation

`var newElement = new goal;`	*New* keyword creates new element with given type.
`gsn.all.append = newElement;`	*Append* command attaches given element into the model file. New element would be the last element in the model file.

Table 4: Astah GSN Driver Element Creator Commands

New element's attributes could be set via two different ways. Either updating the *newElement* object like *`newElement.content = "test";`* or accessing the last element and updating its attributes such as *`gsn.all.last.content = "test";`*.

5. Deleting elements in GSN model with EOL

Deleting an element in EOL uses *delete* keyword. One or multiple elements could be deleted via *delete* command.

Delete Command	Command Explanation
`delete gsn.G10;`	Deletes given element/s.

Table 5: Astah GSN Driver Element Delete Commands

6. Epsilon Validation Language usage
7. Epsilon Code Generation Language usage
8. Epsilon Transformation Language usage

7. CONCLUSION

Conclusion…

8. FUTURE WORK

Element create requires unique xmi:id…

GSN COMMUNITY STANDARD VERSION 1

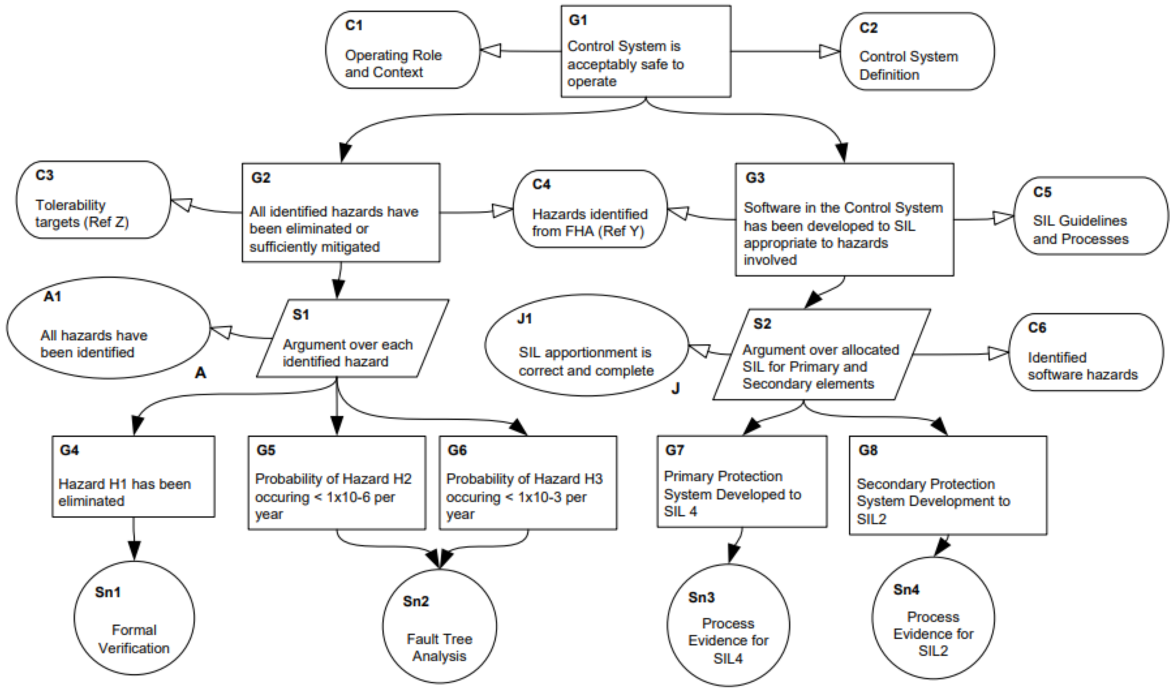
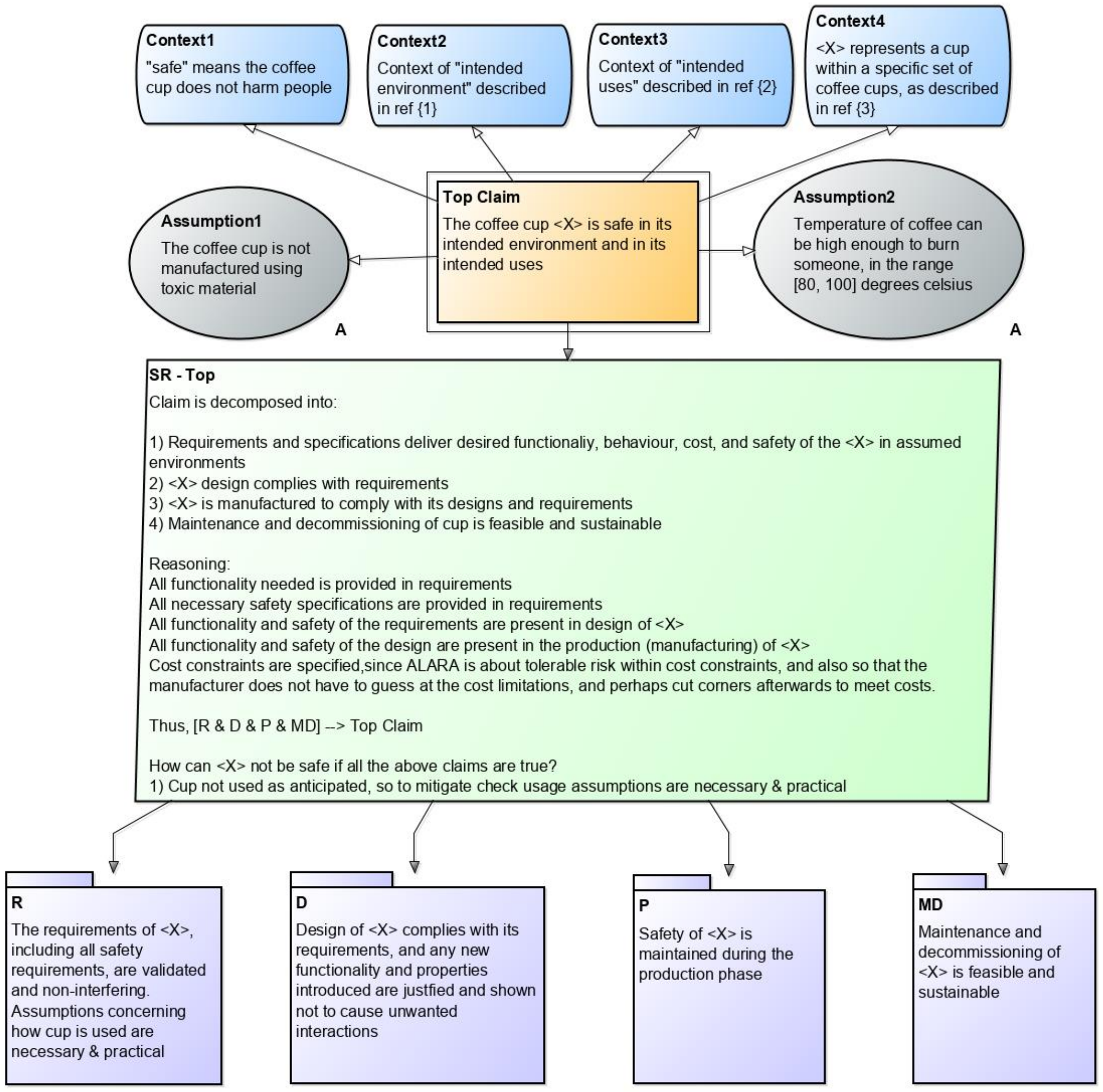

C1
Operating Role and Context
G1
Control System is acceptably safe to operate
C2
Control System Definition
C3
Tolerability targets (Ref Z)
G2
All identified hazards have been eliminated or sufficiently mitigated
C4
Hazards identified from FHA (Ref Y)
G3
Software in the Control System has been developed to SIL appropriate to hazards involved
C5
SIL Guidelines and Processes
A1
All hazards have been identified
A
S1
Argument over each identified hazard
J1
SIL apportionment is correct and complete
J
S2
Argument over allocated SIL for Primary and Secondary elements
C6
Identified software hazards
G4
Hazard H1 has been eliminated
G5
Probability of Hazard H2 occuring < 1x10-6 per year
G6
Probability of Hazard H3 occuring < 1x10-3 per year
G7
Primary Protection System Developed to SIL 4
G8
Secondary Protection System Development to SIL2
Sn1
Formal Verification
Sn2
Fault Tree Analysis
Sn3
Process Evidence for SIL4
Sn4
Process Evidence for SIL2

Figure 6: An Example Goal Structure

© 2011 Origin Consulting (York) Limited, on behalf of the Contributors 5

Coffee cup safety cases GSN diagram:


Context1
"safe" means the coffee cup does not harm people
Context2
Context of "intended environment" described in ref {1}
Context3
Context of "intended uses" described in ref {2}
Context4
<X> represents a cup within a specific set of coffee cups, as described in ref {3}
Top Claim
The coffee cup <X> is safe in its intended environment and in its intended uses
Assumption1
The coffee cup is not manufactured using toxic material
A
Assumption2
Temperature of coffee can be high enough to burn someone, in the range [80, 100] degrees celsius
A
SR - Top
R
The requirements of <X>, including all safety requirements, are validated and non-interfering. Assumptions concerning how cup is used are necessary & practical
D
Design of <X> complies with its requirements, and any new functionality and properties introduced are justfied and shown not to cause unwanted interactions
P
Safety of <X> is maintained during the production phase
MD
Maintenance and decommissioning of <X> is feasible and sustainable

Claim is decomposed into:

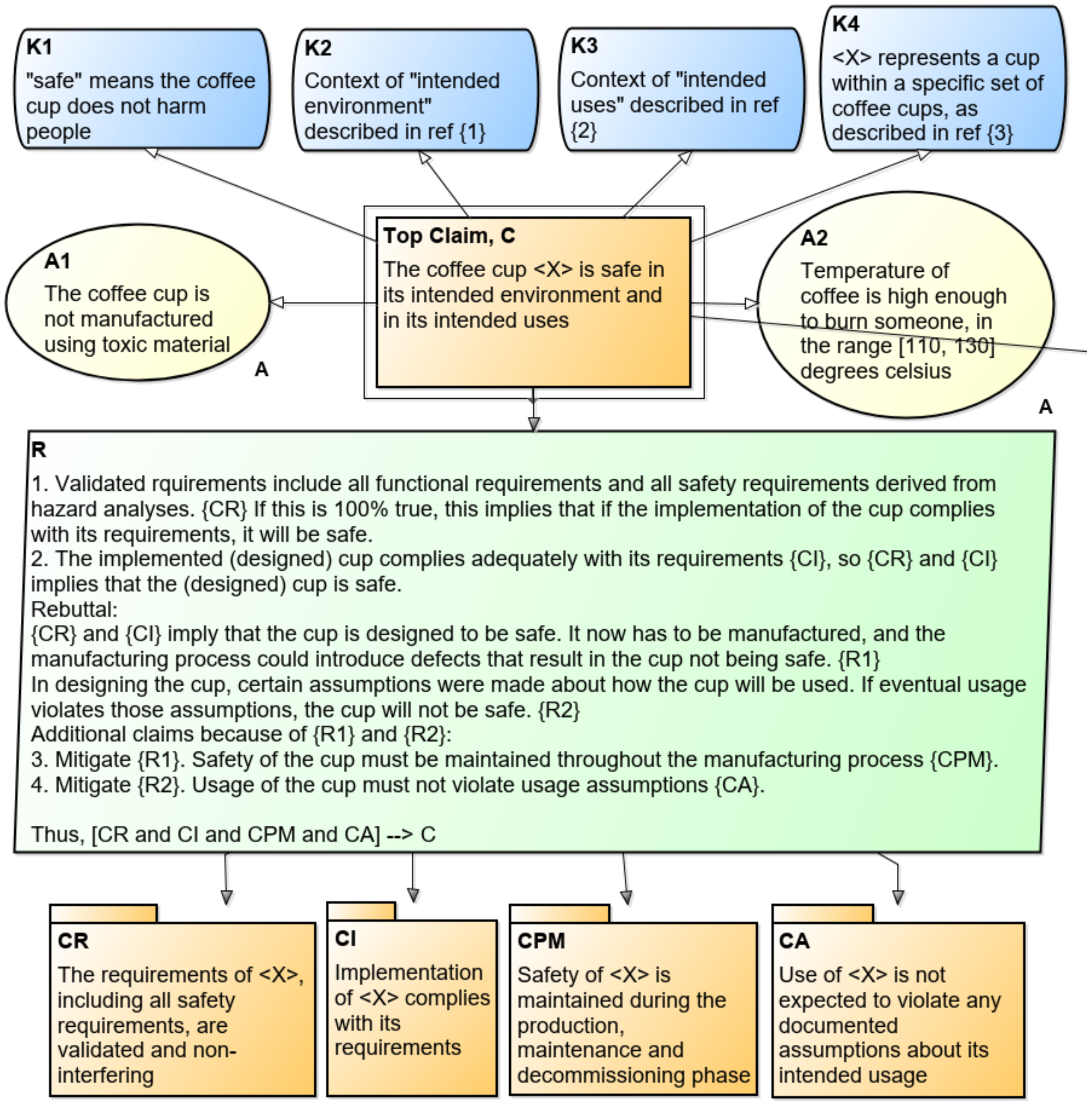
1) Requirements and specifications deliver desired functionaliy, behaviour, cost, and safety of the <X> in assumed environments
2) <X> design complies with requirements
3) <X> is manufactured to comply with its designs and requirements
4) Maintenance and decommissioning of cup is feasible and sustainable

Reasoning:
All functionality needed is provided in requirements
All necessary safety specifications are provided in requirements
All functionality and safety of the requirements are present in design of <X>
All functionality and safety of the design are present in the production (manufacturing) of <X>
Cost constraints are specified,since ALARA is about tolerable risk within cost constraints, and also so that the manufacturer does not have to guess at the cost limitations, and perhaps cut corners afterwards to meet costs.

Thus, [R & D & P & MD] --> Top Claim

How can <X> not be safe if all the above claims are true?
1) Cup not used as anticipated, so to mitigate check usage assumptions are necessary & practical

Second coffee cup diagram:


K1
"safe" means the coffee cup does not harm people
K2
Context of "intended environment" described in ref {1}
K3
Context of "intended uses" described in ref {2}
K4
<X> represents a cup within a specific set of coffee cups, as described in ref {3}
A1
The coffee cup is not manufactured using toxic material
A
Top Claim, C
The coffee cup <X> is safe in its intended environment and in its intended uses
A2
Temperature of coffee is high enough to burn someone, in the range [110, 130] degrees celsius
A
R
1. Validated rquirements include all functional requirements and all safety requirements derived from hazard analyses. {CR} If this is 100% true, this implies that if the implementation of the cup complies with its requirements, it will be safe.
2. The implemented (designed) cup complies adequately with its requirements {CI}, so {CR} and {CI} implies that the (designed) cup is safe.
Rebuttal:
{CR} and {CI} imply that the cup is designed to be safe. It now has to be manufactured, and the manufacturing process could introduce defects that result in the cup not being safe. {R1}
In designing the cup, certain assumptions were made about how the cup will be used. If eventual usage violates those assumptions, the cup will not be safe. {R2}
Additional claims because of {R1} and {R2}:
3. Mitigate {R1}. Safety of the cup must be maintained throughout the manufacturing process {CPM}.
4. Mitigate {R2}. Usage of the cup must not violate usage assumptions {CA}.
Thus, [CR and CI and CPM and CA] --> C
CR
The requirements of <X>, including all safety requirements, are validated and non-interfering
CI
Implementation of <X> complies with its requirements
CPM
Safety of <X> is maintained during the production, maintenance and decommissioning phase
CA
Use of <X> is not expected to violate any documented assumptions about its intended usage

Workflow+ diagram:

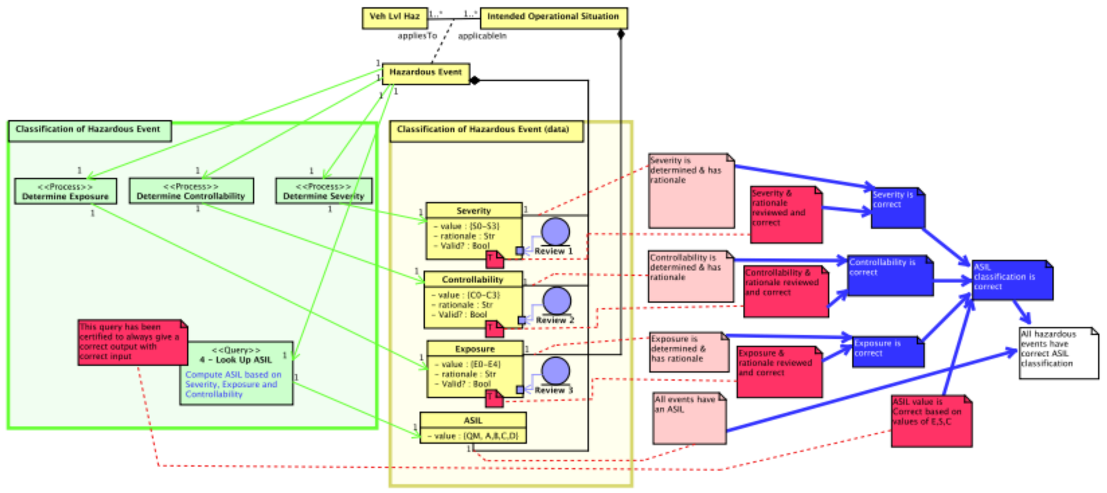

Veh Lvl Haz
1..*
1..*
Intended Operational Situation
appliesTo
applicableIn
Hazardous Event
Classification of Hazardous Event
<<Process>> Determine Exposure
<<Process>> Determine Controllability
<<Process>> Determine Severity
This query has been certified to always give a correct output with correct input
<<Query>> 4 – Look Up ASIL
Compute ASIL based on Severity, Exposure and Controllability
Classification of Hazardous Event (data)
Severity
- value : {S0–S3}
- rationale : Str
- Valid? : Bool
Review 1
Controllability
- value : {C0–C3}
- rationale : Str
- Valid? : Bool
Review 2
Exposure
- value : {E0–E4}
- rationale : Str
- Valid? : Bool
Review 3
ASIL
- value : {QM, A,B,C,D}
Severity is determined & has rationale
Severity & rationale reviewed and correct
Severity is correct
Controllability is determined & has rationale
Controllability & rationale reviewed and correct
Controllability is correct
Exposure is determined & has rationale
Exposure & rationale reviewed and correct
Exposure is correct
All events have an ASIL
ASIL value is Correct based on values of E,S,C
ASIL classification is correct
All hazardous events have correct ASIL classification

Figure 4.10: Workflow$^+$ metamodel of classification of hazardous events with argumentation as described in ISO 26262-3 [29]

9. REFERENCES

[1]. UPDATE HERE!!

[2]. Epsilon website, "Epsilon Home", https://www.eclipse.org/epsilon/

[3]. Goal Structuring Notation, "What is the Goal Structuring Notation?", http://www.goalstructuringnotation.info/archives/category/in-a-nutshell

[4]. Epsilon Lab GitHub page, https://github.com/epsilonlabs