Creating An Editor For The Implementation of WorkFlow$^+$: A Framework for Developing Assurance Cases

By

Thomas Chiang

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements for the Degree

Master of Applied Science

McMaster University

Abstract

As vehicles become more complex, the work required to ensure that they are safe increases enormously. This in turn results in a much more complicated task of testing systems, subsystems, and components to ensure that they are safe individually as well as when they are integrated. As a result, managing the safety engineering process for vehicle development is of major interest to all automotive manufacturers. The goal of this research is to introduce a tool that provides support for a new framework for modeling safety processes, which can partially address some of these challenges. WorkFlow$^+$ is a framework that was developed to combine both data flow and process flow to increase traceability, enable users to model with the desired granularity safety engineering workflow for their products, and produce assurance cases for regulators and evaluators to be able to validate that the product is safe for the users and the public. With the development of an editor, it will bring WorkFlow$^+$ to life.

Acknowledgments

A large thank you goes to Dr. Alan Wassyng, Dr. Mark Lawford, and Dr. Richard Paige for the aid in the creation of this thesis.

Contents

List of Figures

List of Acronyms

Declaration of Academic Achievement

The contributions of this thesis apply to the development of tools for safety engineering in the automotive industry, the application of model driven engineering strategies for software engineering, and the application of both in a practical industry example.

Chapter 1

Introduction

In this thesis we explore topics that are related to the development of software tools for safety engineering. This topic includes the physics of notation for graphical editors, the development of assurance cases, model management, model traceability, modeling, and Model-Driven Engineering (MDE). These topics are being explored through the development of a tool that implements the WF$^+$ framework, an approach to modeling assurance cases through safety engineering processes and workflows, developed at McMaster University [1]. The first step in the development for a tool for WF$^+$, or any language for that matter, is to develop the abstract syntax for that language. The abstract syntax is usually specified before the concrete syntax, defined using a meta-language or metamodel. In the case of development for WF$^+$, which is a modeling framwework, the abstract syntax is defined using a metamodel. To create the metamodel for this language we used the Eclipse Modeling Framework (EMF) for its rich tools that allow for the development of metamodels and the instantiation of those metamodels into usable graphical editors.

After completing a usable metamodel we then instantiated it to define the concrete syntax and the semantics of the model. This was done using Sirius, a tool that can plugin into Eclipse that enables the development of graphical editors. In this layer we can start adding in more constraints to the model such as how the abstract syntax will be represented, which is the concrete syntax. It is also where we develop the semantics for the model and allows us to implement queries that will help with model management, validation and verification, and ultimately, the derivations of assurance claims which will lead to Assurance Cases.

1.1 Motivation

The main motivation for this project was to provide a tool that could support traceability between all elements within a defined safety process when working on safety critical solutions. As this is the first step in creating a fully functioning development environment for users there are 3 main objectives: the development of the abstract syntax (the metamodel), the development of the concrete syntax (the model), and the development of constraints on the workspace of the editor (some semantics and some syntax). Furthermore we did this in conjunction with industry partners to ensure that we did not veer off of the practical world into the academic one. In this way we were also able to ensure that the tool would be more easily adaptable to industry users; this was a direct influence for the design of the concrete syntax (ease of use requirement). The tools used for development were decided upon based on industry standards and leading edge technologies for MDEs. We decided that it would be best to make this tool in an MDE as when dealing with safety crit-

ical systems it is also very likely to be working with very complex systems. In this type of environment, modeling using graphical notation has many distinct advantages over a textual notation: the ability to take advantage of parallel processing of humans, the ability to reduce the need for redundant coding, and the ability to abstract away unimportant information depending on the context. These advantages, in conjunction with already existing modeling tools, led us to create a tool using a graphical notation rather than a textual based one.

1.2 Contributions

The contributions of this thesis are as follows:

- Analysis of the theory for specifying syntax for a MDE editor.
- Test the limits of what the WF$^+$ framework can handle for modeling engineering workflows.
- Lay the foundations for the tool for WF$^+$ as it continues to grow.
- The development of a metamodel environment for WF$^+$

1.3 Outline

This thesis is organized into 7 chapters: Introduction, Previous Work, Iteration 1 of Tool, Iteration 2 of Tool, Tool Specification, Evaluation, Conclusion. As this is the beginning of development for the tool this thesis aims to cover the foundations for it to enable a smooth development path for it moving forwards. Having a proper requirements document is crucial to ensure this,

regardless of an AGILE or Waterfall development strategy (both of which were used with the development of this tool). Next the metamodel is likely the most crucial part of the implementation of the requirements as it is where the entire tool starts to take shape. If the metamodel is not correct then it becomes a practically insurmountable task to implement the requirements properly for the tool. Finally the development of the syntax is the next most crucial element for this tool as it is a graphical one, and is what will determine the usability of the tool. If the notation makes no sense to anyone other than the developer then no one will want to use the tool. In this chapter we explored many design decisions and reasoning for current and past implementations of graphical editors so as to avoid some common mistakes, while also doing our best to maintain some similarity so that users can more easily adapt to the new tool. Finally the evaluation of the tool in its current form, as well as immediate steps for its continued development are discussed in the conclusion.

Chapter 2

Previous Work

This chapter aims to introduce some key terminologies for MDE work and Domain Specific Language (DSL). This chapter also covers contributions made by other publications towards advancing techniques for the creation of DSLs, as well as explores other MDE tools that tackle the assurance of safety in safety-critical systems.

2.1 Assurance Cases

An ever growing aspect for the development of safety-critical products is how to assure regulators that a product is indeed safe to use in its intended use cases. The most common approach to this task is the use of an Assurance Case. While perhaps originally these assurance cases were done textually, there are many key issues with using a textual based approach, most notable among them being the use of imprecise language when creating a textual safety argument [2]. Consequently, where natural language may struggle, one can turn to modeling instead. Along with being able to define safety cases more precisely,

modeling also allows for the recognition of patterns in safety cases, allowing for a more mechanical approach to the development of an assurance case as opposed to textual based assurance cases which rely more heavily upon the intuition of the engineers in charge of development. There are two widely adopted methods for modeling assurance cases, one of them being Goal Structured Notation (GSN), while the other is Claim-Argument Evidence (CAE) [3]. However, though the use of modeling does enable pattern recognition, there is still a lot of experience and intuition required to develop an assurance case. This problem was also identified by Ewen Denny et al. [4] as they implemented their own solution in the form of AdvoCATE. In their publication for their tool they specify the formal methods done to show how they formulate their safety arguments so that they are well-founded.

Modeling strategies for assurance cases are constantly growing as the demand for more rigorous testing of software and embedded systems increases. However assurance cases themseleves are still relatively young compared to safety-critical industry, as such the tools available for development, while growing, are still relatively young. A survey done by the University of Toronto [5] took a look at 37 currently available tools for this domain and attempted to grade them on 6 metrics: creation, maintenance, assessment, collaboration, reporting, and integration.

- Creation measures the ability to develop assurance cases.
- Maintenance measures how easy it is to maintain an assurance case during a product life-cycle.
- Assessment measures the ability to assess syntactic and semantic elements of the assurance case.

- Collaboration measures the ability for multiple users to work on an assurance case concurrently.
- Reporting measures the ability to create a report of the assurance case for stakeholder.
- Integration measures the ability for the tool to interact with third-party tools.

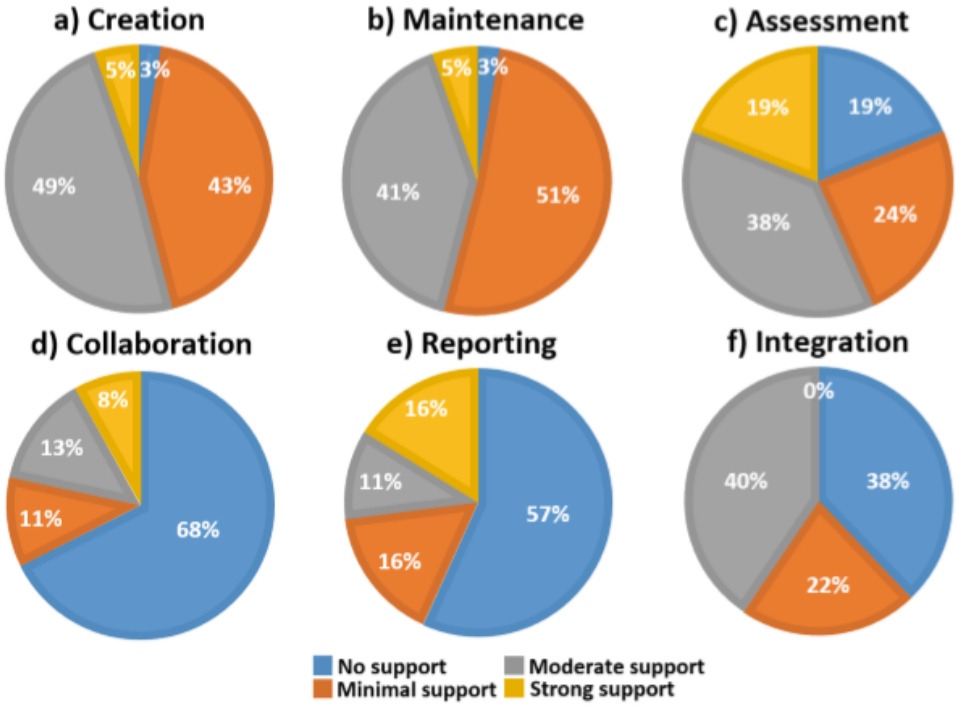

Figure 2.1: In this figure from [5] we can see the general trends of strengths and weaknesses of the surveyed tools when compared against the specified metrics

Each metric got an alphabetical score from D to A, with D being the lowest and A being the highest. A look at the results from this survey shows that most of the tools have some ability to create and maintain assurance cases, however there was little to no support for collaboration, reporting, and integration,

with some support for assessment. Thus, as concluded in the survey, it is clear that there is still a lot of room for improvement for tool support in this domain.

A different approach to the creation of assurance cases was explored in a publication by Ran Wei et. al [6]. Structured Assurance Case Metamodel (SACM) is a standard specified by the OMG and while it boasts a more mechanical approach to the creation of GSN and CAE assurance cases, there is no tool support currently for it to enable users to create metamodels that adhere to SACM. It is currently dependent on the user to ensure that the metamodels created are correct and conform to the SACM framework. SACM uses a similar approach as WF^+; it attempts to simplify the method to creating assurance cases so that it can be done more mechanically as opposed to intuitively. SACM however is not a metamodel for GCN or CAE, it is an independent metamodel for assurance case creation and argumentation that can be used as a reference for creating GSN and CAE assurance cases. As a result, what SACM currently lack is tool support; a concrete syntax for the creation of SACM models that can then be transformed to GSN or CAE assurance cases. The main difference between SACM and WF^+ lies in the way that the argumentation is created. SACM is a metamodel for the creation of arguments; a structure for users to follow to create argument. The goal of WF^+ is to derive argumentation directly from the modeled Safety Engineering Process (SEP) that the user defines.

2.2 WorkFlow$^+$ (WF$^+$)

WF$^+$ is a formal mathematical framework, developed at McMaster University, which introduces a formal method for the development of assurance cases for software and embedded systems [1]. The WF$^+$ framework allows for the modeling and development of assurance cases by providing the mechanisms to model all aspects of SEPs. This is done through multiple levels of modeling strategies. The editor that was created during this thesis is in fact an editor that allows for the development of WF$^+$ metamodels. The reason this is important is because while WF$^+$ is aiming to address the complexity of safety cases and engineering across not just a product, but a product family. Thus, some of the goals beyond this thesis include the instantiation of WF$^+$ metamodels into executable models, each one representing a specific product, while all sharing the same parent WF$^+$ metamodel. This will likely have a higher upfront cost, as users will have to define their SEP template as a WF$^+$ metamodel, with the aim being to reduce the cost of generating assurance cases further down the production line as they will be able to instantiate their templates as many times as they want. In addition, it presents the SEP in a model that unifies both the process-flow and data-flow, allowing for a more complete picture compared to when they are disjoint. The reason we believe this makes a more complete model is because of the increased traceability that can be modeled with the input/output relationship that is modeled between process and data elements within a single model. This is where the "+" comes from in WF$^+$. The advantage of this type of modeling is that we can have traceability at a much higher granularity between all work-products and the processes that produce them. This increased traceability, in turn, enables WF$^+$ to derive its

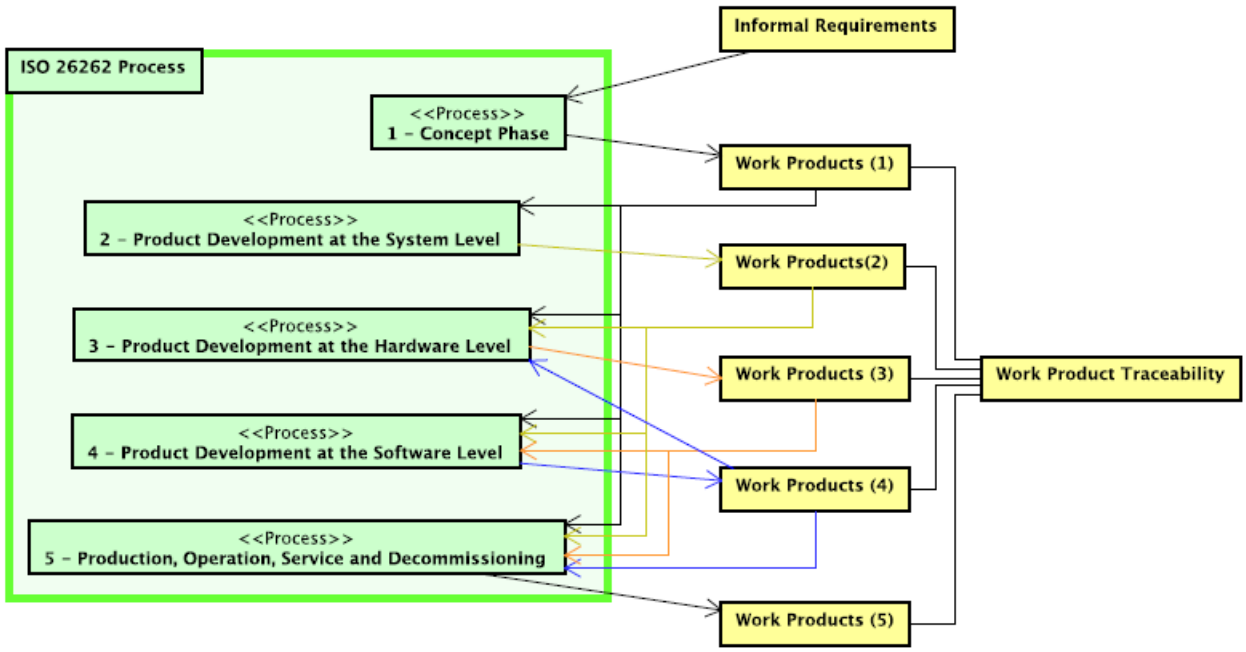

Figure 2.2: A high level view of an example WF$^+$ model from [1]

safety arguments directly from the SEPs that are defined, thus ensuring that the safety-claims are directly tied to the evidence that produces them. Thus, rather than having to form the safety arguments after the SEPs are executed, we are able to generate them directly from the SEPs themselves. This reduces the intuition required to both develop and maintain assurance cases by providing a more mechanical approach, therefore reducing some of the subjectivity that is involved in determining when a solution is "safe enough".

The idea of having a generic model from which assurance cases can be iteratively worked on and improved is not a novel idea. However the management of those models, with everything from maintenance to reuse, is a much more complex task than just the generation of an assurance case for a specific instance. A prior exploration of the topic of the management of assurance cases, among other criteria, was conducted by Kokaly et. al [7]. In the publication they used a formal approach to introduce a generic modelling framework, within which they were able to specify a model management reuse algorithm that used know model management operators to produce a solution to the issue of incremental assurance and the reuse of portions of an assurance case.

2.3 Domain-Specific Languages (DSL) and Graphical Editors

While there are different tools that support the creation and, to varying degrees, the maintenance, assessment, collaboration, reporting, and integration of assurance cases, the one thing that they all have in common is that they are domain specific languages. That is, they are languages and tools that have

been created as a solution of a very specific problem that general purpose languages (such as python, Java, or C++) cannot easily satisfy. While these DSLs may inherit from the larger the more general languages from which they are created, they are specialized enough to the point that they hardly resemble their parent language. This is even more true with the specification of a graphical language and editor, which comes with its own challenges.

Work done by Moody in [8] sets about laying a scientific foundation based on work done in cognitive psychology, graphic design, software engineering, and many more. Based on the previous work, he has defined design principles for creating a meaningful graphical syntax. These principles are:

- Semantic Transparency
- Complexity Management
- Cognitive Integration
- Graphic Economy
- Cognitive Clarity
- Semiotic Clarity

While the work done by Moody was just the start, there have been some attempts to define implementation strategies by Harald Störrle and Andrew Fish [9]. They discovered that while the Physics of Notation is a good start, there is much more work to be done to formalize it before it can be used as a concrete tool for the evaluation and creation of graphical syntax.

Another approach to the development of graphical syntax can be reasoned from the work of Nicolas Genon, Patrick Heymans, and Daniel Amyot [10].

They use the concepts introduced in the Physics of Notation to analyze the cognitive effectiveness of the syntax of BPMN 2.0. In their conclusion they surmised that the task of defining a perfect syntax for everyone, while nice, is nigh impossible. While much of the analysis looks at the problem from a purely syntactic point of view, this also ignores how heavily semantics and syntax are tied together. Thus it requires some major creative critical thinking to generate a syntax that meets the concepts introduced in the Physics of Notation.

Another aspect of WF$^+$ that makes it unique is that it is used to specifically model engineering processes and their related data. While for the sake of assurance we are using it to model safety engineering processes, the domain of modeling engineering processes is one that has limited tool support. In a publication by Teng et. al [11], they introduce a framework for benchmarking risk and safety programs to achieve compliance with a regulatory guideline using Business Process Modeling. While this is an example of modeling programs to check compliance, it does not fall within the same domain as WF$^+$, which is to model executable processes and generate assurance. While there are other publications that explore the topic of modeling business processes, there are not many of the modeling of engineering processes. One other such metamodel that exists is [12], a process engineering metamodel developed by OMG. This meta-model focuses primarily on process engineering, with some references to work-products and data elements. The reason we decided to develop WF$^+$ was to be more specific to the domain of assurance cases and safety engineering, focusing equally on process process engineering and data engineering.

2.4 Eclipse Modeling Framework (EMF), Epsilon Object Language (EOL), and Sirius

While developing in a MDE environment, it is not just important to have tools that can support the development, it would be impossible without tool support. EMF [13] is an industry leader for the development of model driven DSLs and Epsilon Object Language (EOL) is an excellent tool from the framework. EMF is a framework famed for its ability to implement model management solutions, as well as the creation of metamodels and their instantiations using its built-in metamodel; Ecore. EMF implements a custom metamodel that was partially inspired by the Meta-Object Facility (MOF) which is a standard model management framework created and set by the Object Management Group (OMG). EOL [14] is a custom language that was created to aid in the development. It was inspired by OCL but replaced the type system and added features more specific for the development of metamodels and their instances. EOL is also reused as a base layer for more domain specific languages depending on the required use case (model validation, model transformation, etc).

As WF$^+$ matures, the plan is to leverage the Epsilon Transformation Language (ETL) for its model to model transformation abilities, which can help address the lack of integration support. The Epsilon Merging Language (EML) will be useful for model merging which can be used to help address collaboration support for WF$^+$.

As the development of WF$^+$ progresses, all the facilities provided by EMF will be crucial to addressing the areas where current tool support is lack-

ing. This was main driving force behind the decision to use EMF. Furthermore, in order to specify the editor for this DSL we use Sirius [15], an open sourced specification tool that was partially inspired the now less widely used Graphical Modeling Framework (GMF). The main strengths of Sirius, as stated by Vladimir et al, [15] are as follows:

- Foundation on the open and widely used industry standard - EMF
- Adaptability to any EMF-compatible Domain Specific Model (DSM) (especially helpful when it comes to addressing integration)
- A strong separation between semantic and syntactic models
- Easy to use and allows for rapid development
- A high level of extensibility

On top of the features listed above, Sirius is also being used to develop other tools such as Papyrus, a modeling environment that allows for development in both UML and SysML. Sirius also now has a web implementation based on cloud services that allows users to interact with Sirius through their web browser.

2.5 Deep Metamodeling

Another part of the problem creating and maintaining assurance cases appears when trying to maintain assurance cases across a product family. While we have mentioned some methods for assurance case reuse, these do not solve the main issue of having to redo an assurance case for every product within a product family. One approach that we are proposing with the development

of WF$^+$ as a framework and tool is the use of deep metamodeling, a concept that is explored by Juan de Lara and Esther Guerra [16]. In their research they used the EMF framework to show the effectiveness of deep metamodeling and the effective use of inheritance of actions and constraints through the multiple model layers. While we are not using the framework they proposed, the concept of using more than just 2 layers of models is one that we believe is an effective approach to solving the issue of applying an assurance case across a product family. In the case of WF$^+$ specifically, it is currently working within 3 layers; the Ecore metamodel, the WF$^+$ metamodeling editor, and eventually the instantiation of the WF$^+$ metamodel into a WF$^+$ model editor.

Another benefit to deep metamodeling, and modeling in general, comes with the ability to aid with software interoperability. A topic discussed across several chapters in Model-Driven Software Engineering in Practice, Second Edition [17], they discuss the benefits that come from using models in practice, and one of the benefits comes with the interoperability of software. Using a model to model transformation implements a semantic mapping between 2 domains, allowing for the equivalent expression of one model using the syntax of another. Metamodels A and B can be manually generated, derived from the corresponding format description (e.g., an XML schema when the input or output are XML documents), or automatically created if the formats to bridge conform to a meta-format for which a bridge at the metametalevel is already available." [17]pg35.

Chapter 3

Iteration 1 of Tool

As with any large software development project, the first step to address is the requirements of the tool. The development strategy that we used for this project was an Agile development methodology. The reason for this was to allow for iteration on every aspect of the project, from the requirements to the WF^+ itself as we suspected implementation of the mathematical framework would provide many challenges. Thus, no matter what shape the first version of requirements would take, there was no way that it would be correct on the first attempt. Naturally as mentioned in previous chapters, the first requirement and constraint was that we develop this tool in EMF.

3.1 The Requirements

The derivation of the requirements for the tool was difficult at first as our industry partners did not have a clear concept of what they would want out of a tool that could support something like WF^+. As a result the requirement had to be defined as how we would like to use a tool that would support

WF$^+$. While this may have introduced some confirmation bias with regards to how the tool should behave, we figured it was the best starting point for the creation of the first prototype so that we could then show it to our industry partners and get more concrete feedback in terms of their requirements for the tool that we may not have been able to generate ourselves. We chose to use natural language to define the requirements for the tool. This was because we already had a formal specification for the framework of WF$^+$, thus the main use for the tool was to provide a method for engineers to interact with the framework. As a result we took a business event approach to the requirements since that would allow us to most easily define what actions the user (for this tool the assumed user is an engineer) can take and how the tool will respond. Version one of the requirements was then roughly based on this use case diagram.

3.1.1 Business Event 1: Engineer wants to create new WF$^+$ definition

- Engineer Viewpoint

    1. Engineer shall be able to create new project for WF$^+$. Names project as desired process definition
    2. Engineer shall be able to browse WF$^+$ GUI palette for Process Definition Class. Selects diagram element and places it in the project canvas
    3. Engineer shall be able to fill out attributes of Process Definition Class such as name, process rules etc.

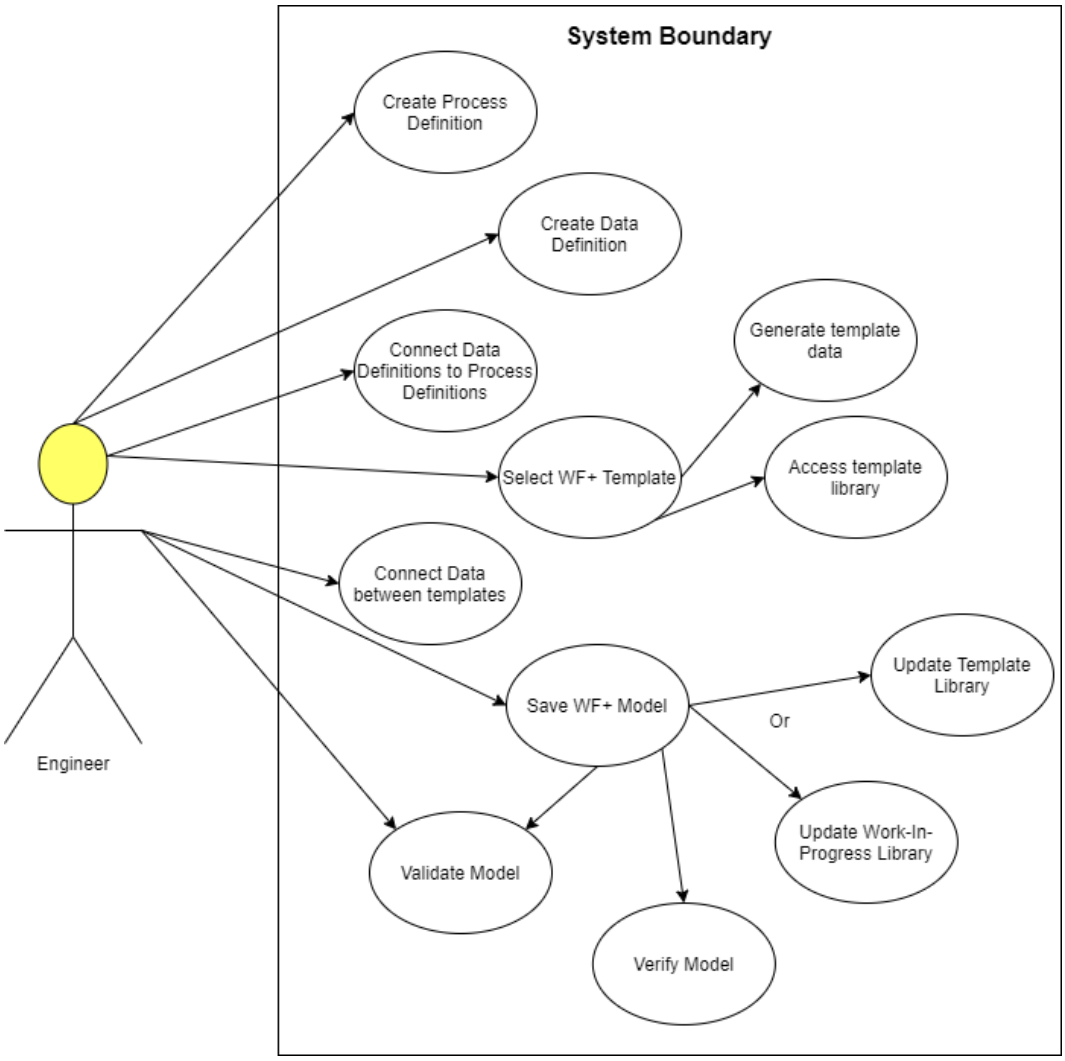

Figure 3.1: This figure was the first rough draft we used to map out how the user can interact with the tool and was further refined as we learned from implementing these use cases.

4. Engineer shall be able to browse WF$^+$ GUI palette for Data Definition Class. Selects diagram element and places it in the project canvas
5. Engineer shall be able to connect the data definition as input to the process definition.
6. Engineer shall be able to define input data definition attributes. (Is this to be done graphically with elemental classes or is this going to be text based attributes?)
7. Engineer shall be able to select another Data Definition Class from the palette and places it in the canvas. Connects the new data definition as the output from the process definition
8. Engineer shall be able to define output data definition attributes (Same question from point 6)
9. Engineer shall be able to save WF$^+$ model into template library so that it can be reused in other WF$^+$ models and go through V&V at a later date

- Tool Viewpoint (n/a)

The first requirement was centered around the creation of a new WF$^+$ model from scratch. This is under the assumption that there are no prior defined models to work from. The requirements here conflict with one another as some of the are high level, while other are defining specific actions taken by the user. Some of them are still incomplete as we did not have all the answers that we wanted yet to concretely decide on how to approach the requirement and were intentionally left as a question for further discussion within the research group.

As we will see with the second business event, a strong emphasis was being placed on the re-usability of models created by the user and how they can be saved and go through V&V for later re-use.

3.1.2 Business Event 2: Engineer wants to create new WF$^+$ definition from WF$^+$ templates

- Engineer Viewpoint
    1. Engineer shall be able to create new WF$^+$ project
    2. Engineer shall be able to browse WF$^+$ template library for desired process templates, and select the templates desired and place them in the project
    3. Engineer shall be able to specify how the output data of one process relates to the input data of another process
    4. Engineer shall be able to connect all the data between processes together into a cohesive WF$^+$ model
    5. Engineer shall be able to save WF$^+$ model into template library so that it can be reused and so that it can go through V&V at a later date (manual V&V)
- Tool Viewpoint
    1. Templates shall be able to auto generate all of their required input and expected output data templates
    2. Tool shall be able to ensure that the WF$^+$ model is syntactically and semantically correct before being able to be saved into the template library

3. WF$^+$ template library is updated to reflect addition of a new template

This is the business event where we started to look at re-use a little more closely. We wanted to be able to browse all of the already made WF$^+$ models in a sort of library catalog so that the models themselves could behave as modules and could be plugged into one another in order to create a larger WF$^+$ model. Our thought process behind this requirement was that this way the user could develop all of the SEPs separately and connect them together at a later date. Alternatively, the user could develop smaller subsections of an SEP and then put the SEP together afterwards. The biggest problem with this business event however, was that it forced a bottom-up development strategy on the user. It is unrealistic to expect the user to have a full idea of the SEP they wish to define from the bottom up all the time. In fact, it is much more common to use a top-down approach as many users will define a black box process in an SEP to be further refined at a later date. This was recognized as a problem during the development of the first version of the metamodel for WF$^+$ as it added unintended restrictions to users and how they can interact with WF$^+$.

3.1.3 Business Event 3: Engineer wants to create new WF$^+$ model mixing both templates and new definitions

- Engineer Viewpoint

1. Engineer shall be able to create new WF$^+$ project

2. Engineer shall be able to browse WF$^+$ template library for desired process templates
3. Engineer shall be able to specify how the output data of one process relates to the input data of another process
4. Engineer shall be able to connect all the data between processes together into a cohesive WF$^+$ model
5. Engineer shall be able to notice that a process is missing from model and is not available as a template within the library
6. Engineer shall be able to browse WF$^+$ GUI palette and select a Process Definition Class
7. Engineer shall be able to fill out attributes of Process Definition Class such as name, process rules etc.
8. Engineer shall be able to browse WF$^+$ GUI palette for Data Definition Class
9. Engineer shall be able to connect the data definition as input to the process definition.
10. Engineer shall be able to define input data definition attributes. (Is this to be done graphically with elemental classes or is this going to be text based attributes?)
11. Engineer shall be able to select another Data Definition Class from the palette and places it in the canvas
12. Engineer defines output data definition attributes (Same question from point 11)
13. Engineer shall be able to save WF$^+$ model into template library so

that it can be reused and so that it can go through V&V at a later date (manual V&V)

- Tool Viewpoint

    1. Templates shall be able to auto generate all of their required input and expected output data templates
    2. Tool shall be able to ensure that the WF^+ model is syntactically and semantically correct before being able to be saved into the template library
    3. WF^+ template library is updated to reflect addition of a new template

With this business event we tried to mitigate another issue that we had with how the first two business events were defined; at this point the user could only create a new model, or connect already existing models together. This disjoint development method seemed needlessly restrictive as if a user was halfway through connecting some already existing templates together, and then found that one was missing from the template library, they would have to start a whole new project to create that template. Then upon completing the missing template, they would have to return to their original project and then import it. This business event was an attempt to define the requirements around letting a user mix and match between creating new WF^+ models on the fly while also being able to use existing templates.

While this makes sense in theory, at this stage in the product life-cycle for WF^+ we later determined that this added an exponential amount of complexity to the tool that we were ill-equipped to implement. While this is a business

event that we want to implement at some point for the tool, it was decided to be added to an expected change to the requirements to be implemented in a later version of the tool.

3.1.4 Business Event 4: Engineer wants to edit an already existing WF$^+$ model that has been saved as a template

- Engineer Viewpoint

    1. Engineer shall be able to open the WF$^+$ model that is to be edited
    2. Engineer shall be able to select individual elements of the model that are to be changed
    3. Engineer shall be able to save changes made to model as a template only if changes are syntactically and semantically correct.
    4. Engineer shall be able to save changes made to model as a diagram that is a work in progress if changes are not complete or model is not syntactically and semantically correct.

- Tool Viewpoint

    1. Tool shall not allow WF$^+$ template that is open for editing to be used in other WF$^+$ projects.
    2. Tool shall not allow the engineer to use other WF$^+$ templates that are partially composed of the template that is being edited in any WF$^+$ projects.

3. Tool shall lockdown instances of a WF^+ template that is being edited to prevent instantiation.

This business event was to decided on for the situation where there is a change to the environment around which an SEP was originally defined. Thus the user would have to be able to edit already defined WF^+ models. Ideally this change would also propagate through all of the project that use the template that has been updated. This set of requirements was set to tackle the issue of maintenance with assurance cases and WF^+ specifically. This business event was never properly completed as it was quickly decided that this would need a complete reworking of this set of requirements. The reasoning for this was because of the unclear requirements that were created around this business event.

3.1.5 Business Event 5: Engineer wants to edit an already existing WF^+ model that has been saved as a diagram

- Engineer Viewpoint

1. Engineer shall be able to open WF^+ model that is to be edited.
2. Engineer shall be able to select individual elements in diagram that are to be edited.
3. Engineer shall be able to save changes to model as a template only if changes are syntactically and semantically correct.
4. Engineer shall be able to save changes to model as a diagram that is a work in progress if changes are not complete or model is not

syntactically and semantically correct.

- Tool Viewpoint

    1. Tool shall not allow WF^+ template that is open for editing to be used in other WF^+ projects.
    2. Tool shall not allow the engineer to use other WF^+ templates that are partially composed of the template that is being edited in any WF^+ projects.
    3. Tool shall shall lockdown instances of a WF^+ template that is being edited to prevent instantiation.

3.1.6 Business Event 6: Engineer wants to instantiate a model to create a safety case

- Engineer Viewpoint

    1. Engineer shall be able to compile model in order to create an instance of said model.
    2. Engineer shall be able to input concrete data values (as specified by the model the instance was compiled from) into data entries for/from processes.
    3. Engineer shall be able to complete the process rules of the instance as specified by the model the instance was compiled from.
    4. Engineer shall be able to validate data in the instance through custom validation rules.

- Tool Viewpoint

1. Tool shall be able to populate the GUI palette based on the specifications from the model that was instanced.

3.2 The Metamodel

Once a rough idea of the requirements for the tool were laid out we started with the development of the metamodel for the tool. As at this stage in the project we were still new to Ecore, we decided to start with developing a platform independent model in UML. This was done for 3 main reasons; UML was easily understood by everyone thus improving collaboration between team members, it would be easily translated to Ecore when we decided we were starting development in the EMF environment, and there would be no bias towards implementation tool or constraints.

3.2.1 The UML Models

3.2.1.1 Platform Independent Model 1

From 3.2, we determined that there was insufficient detail with regards to how Process and Data interact with one another, as well as the hierarchy between Processes. This lack of hierarchy makes it difficult to ensure that the users would be able to specify the WF^+ models at their desired level of granularity for their SEPs. Thus this was a point of focus for development that we were sure we needed to improve. The Correspondence Span class was created as a means to identify overlap between different WF^+ models. Thus it would become apparent to the user what work was redone in different WF^+ models (Perhaps some SEP uses the same input data, but it is named differently

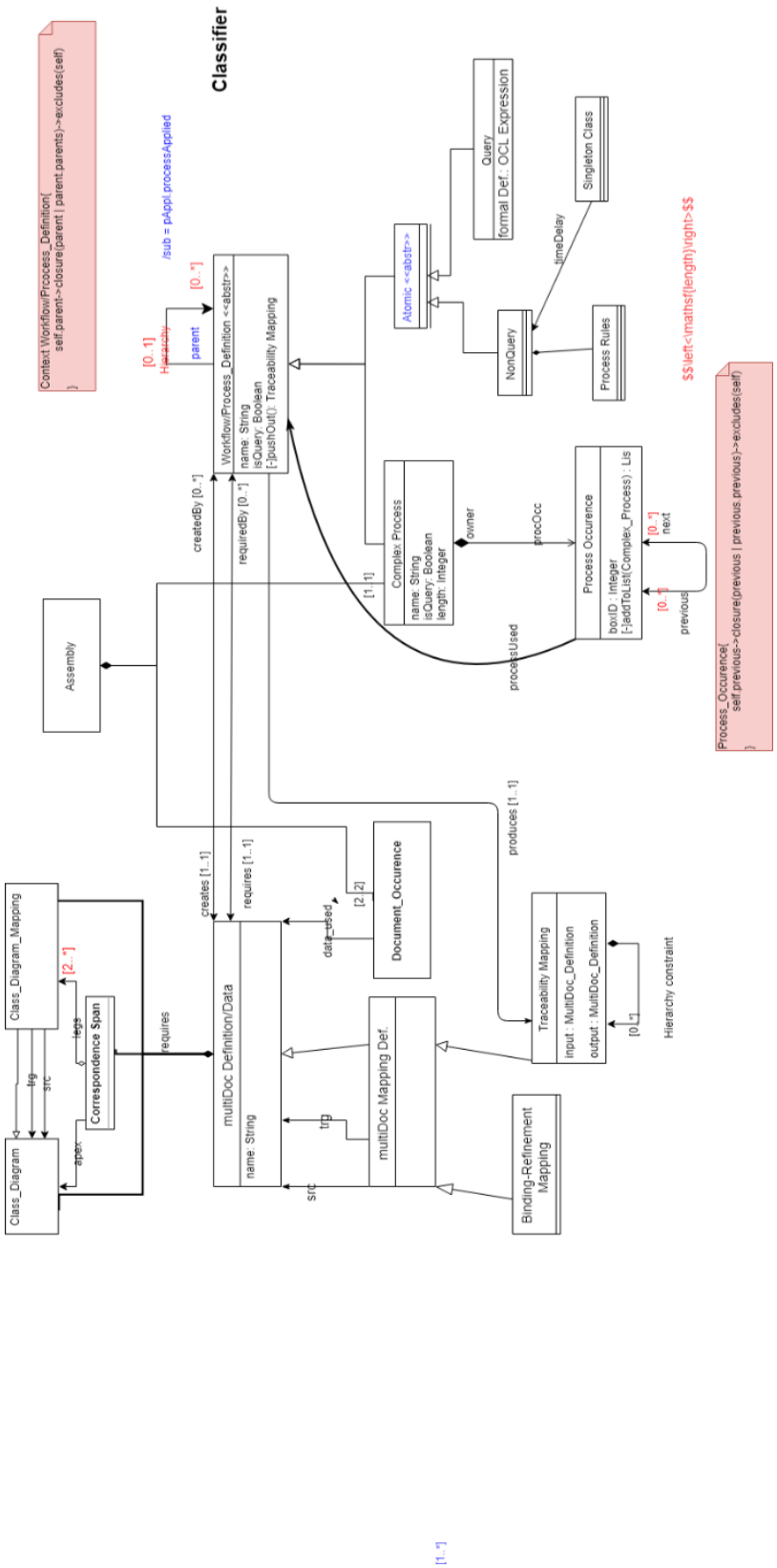

Figure 3.2: In this figure is the very first attempt to create a meteamodel for WF$^+$ that can be used for tool implementation

by different teams). As this was all still being done at a high level, actual implementation of this class was still unclear, but while we were still unsure of how to implement it, we recognized the usefulness that it could provide to users for collaboration across different teams.

3.2.1.2 Platform Independent Model 2

At this point in the development lifecycle we had created a PIM for how Process should behave independent of Data. The reason we did this was to simplify and focus purely on the behavior we want to achieve with how Processes are constructed and relate to one another. It is known that there needs to be a hierarchy, as Processes can be composed of other Processes, and there also needs to be a way of sequencing those Processes. This model achieves the former through association composition property. While the sequencing of Processes was still somewhat vague in this model, we decided that we had enough of a base to begin development in Ecore, our tool of choice.

3.2.2 The Ecore Models

Rather than directly importing the UML models, we decided to recreate them in Ecore so that we could fully explore what functionality was present when developing in Ecore. This was to determine the ease of developing directly in Ecore versus in UML. Another benefit of recreating the metamodel rather than importing was that we weren't sure if we wanted to import UMLs infrastructure and superstructure as it may have had unexpected consequences as development continued. The exercise quickly showed that Ecore had a very intuitive interface to work with that enabled quick and comprehensive devel-

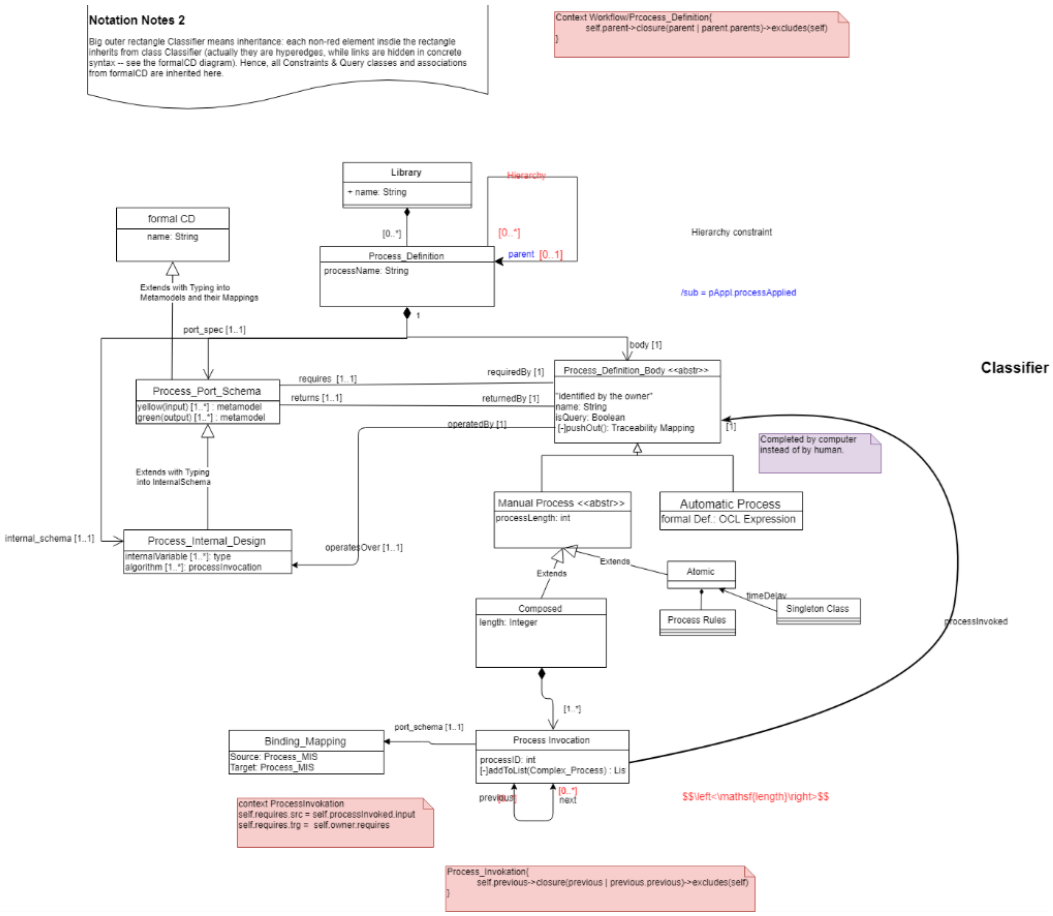

Figure 3.3: This figure show specifically how Processes in WF$^+$ should behave independent of how Data behaves

opment. It was also easy to understand by everyone involved due to its similar syntax to UML Class Diagrams.

3.2.2.1 Ecore Model Iteration 1

At this point in the development lifecycle of WF$^+$, it was determined that we have lost the ability to do top-down designing of instances. With this current iteration, shown in 3.4, it relies heavily on the assumption that a full library of Data and Process elements to be able to construct models and instances. This goes against the intuitive method of building a black box with inputs and outputs, then later refining that black box to see what it would actually be built of. Furthermore, in this model how to implement Data_Library and Process_Library classes as we intend. Rather than being a container that is present within the editor, we intended them to behave more like a workspace folder. The concept of the libraries is to allow the re-usability of WF$^+$ models.

The next steps taken were to edit this model so that we can have Process and Data refinement, have a clearer picture of how Data and Process should interact with each other (likely through the Data_Metamodel class), And also ensure that we can have explicit sequencing through process elements, as well as traceability between Data elements, especially as they have been refined.

3.2.2.2 Ecore Model Iteration 2

In this iteration of the Ecore model for WF$^+$ we identified that Data_Metamodels are in fact separate from other data nodes. This is because the Data_Metamodel class is the effective bridge between the Data and Process halves of WF$^+$. The idea of Data_Metamodel is to allow the user to pic and choose with whatever

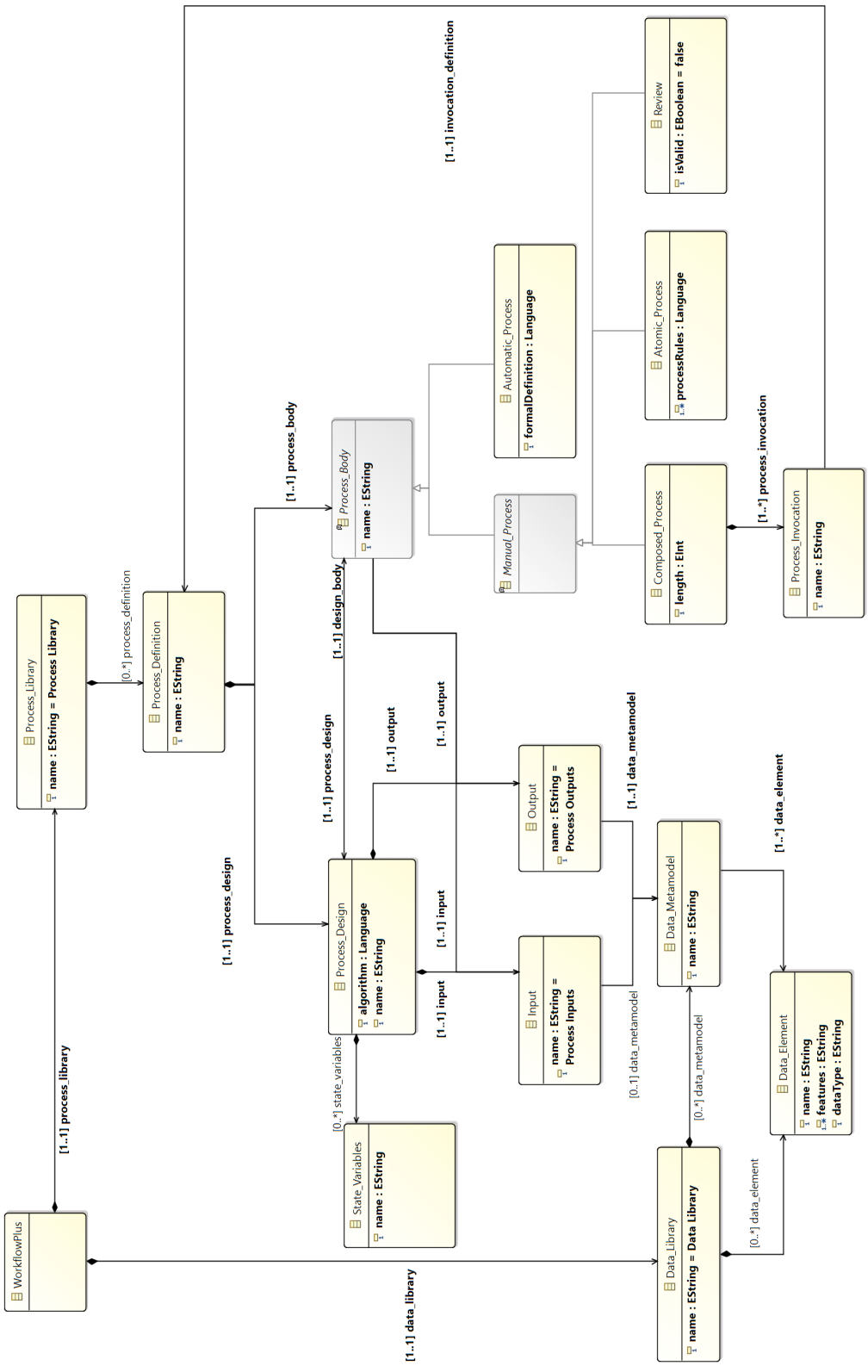

Figure 3.4: The first attempt at modeling in Ecore

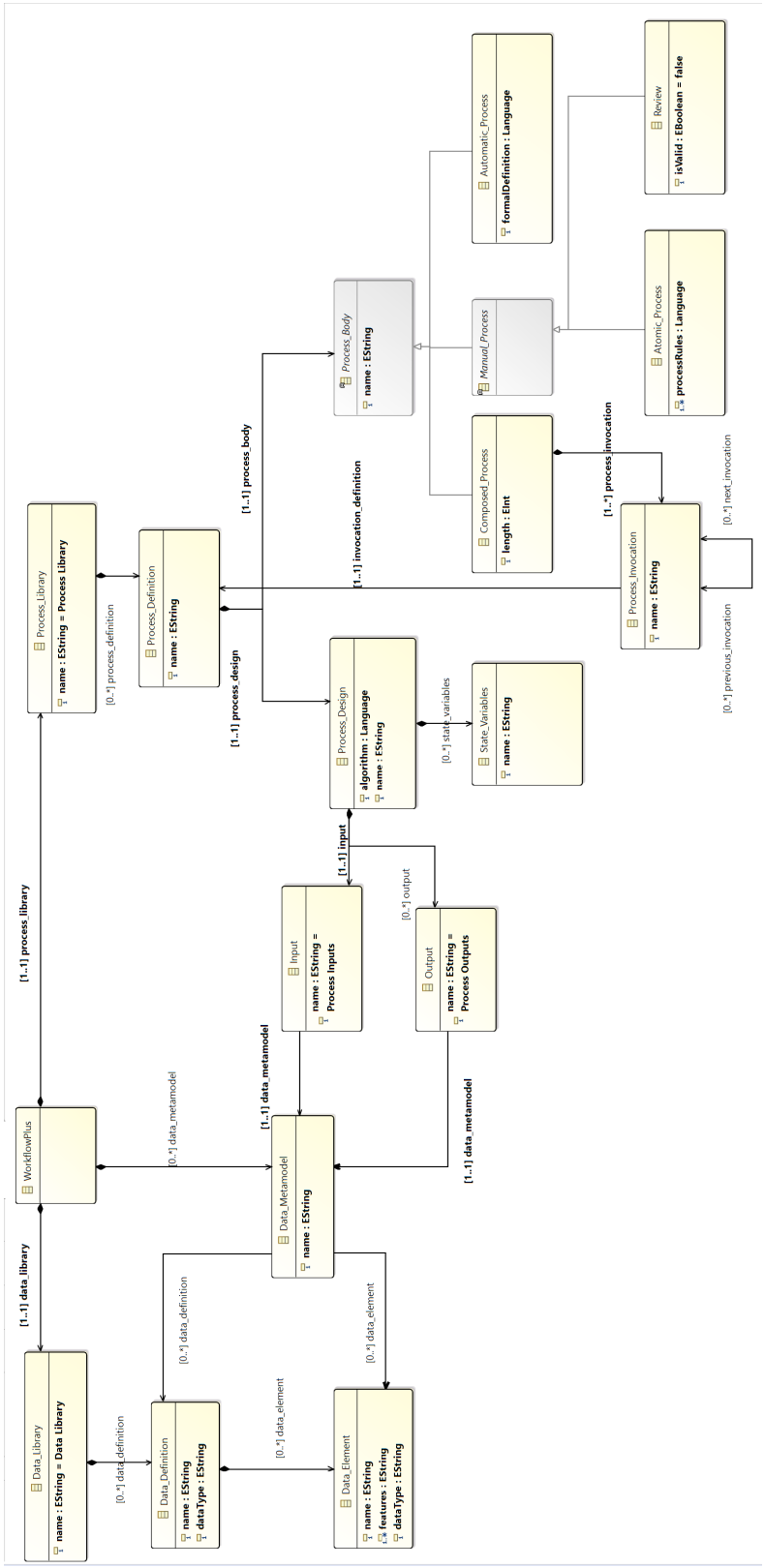

Figure 3.5: The attempt to make it more clear how the Process and Data can interact with one another

granularity they want the type, size, and amount of data that they want to define and inputs and outputs of processes. As such it cannot be owned by Data as it behaves much more like a pointer to nodes that Data contains.

Using this version of the metamodel I have tried modeling some parts of the SEP from GM. I hope to glean some ideas of what kind of constraints might be needed. The first constraint that comes to mind is on Data_Elements and how they can be put together.

At this point in development we determined that we needed to define a structure that can allow users to create formal class diagrams in Data structures so that they can define how the data should look when it is instantiated. By enabling such a feature the user would then be able to define exactly what the data should look like when it is produced as an output from a process or used as an input to a process. This resulted in the creation of the model in 3.6

With the addition of the Arrow abstract class we have the basis for recreating Associations, Compositions, Inheritances, and other UML based reference classes as needed to define Data more formally.

3.2.2.3 Ecore Model Iteration 3

In this iteration we added the composite pattern to both Data_Library and Data_Metamodels. As they were added, we realized this reflects how Process_body is defined as well and this addition improved the symmetry of the model. However, it was discovered that there was a crucial failing for this iteration. There is a very large bias in this iteration for a bottom up approach to creating WF^+ models. It requires the user to first define the libraries (process and data) to create templates and how they connect with each other. Next it

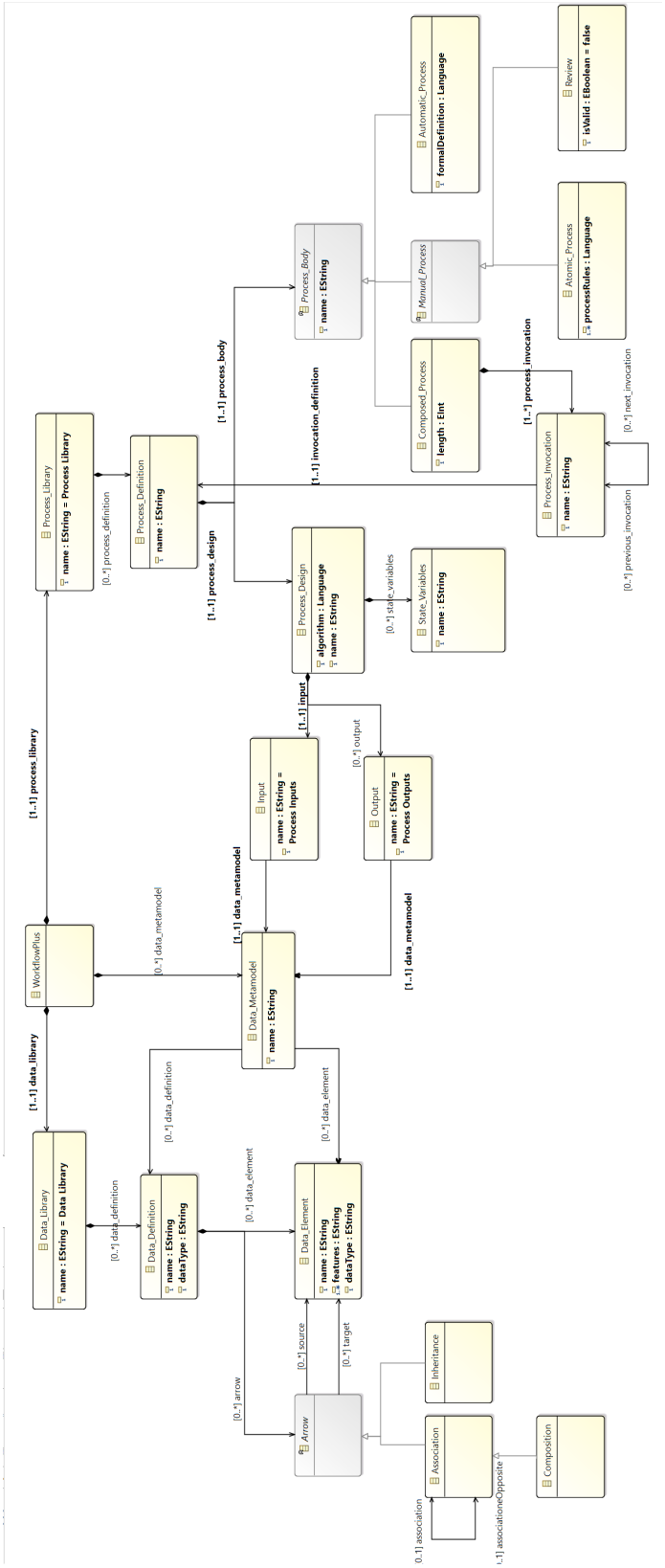

Figure 3.6: Third iteration on the model to add some structure for Data classes to allow for more formal descriptions

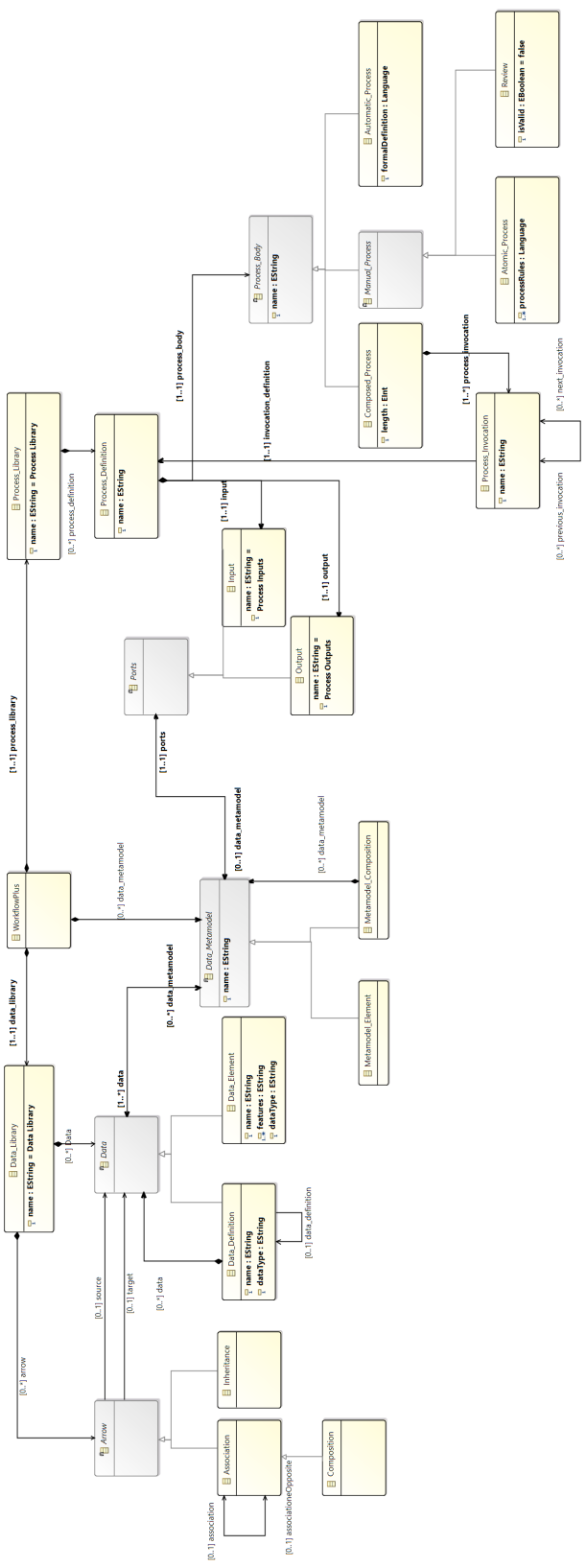

Figure 3.7: The final attempt with this iteration of the requirements and metamodel before we gave up.

requires the user to instantiate those libraries to then be able to start creating WF^+ models. While mathematically it makes complete sense to do it that way, as a user it introduces many more stages to the workflow that they would need to complete for their engineering process. This runs against the natural efficiency that WF^+ is seeking to provide; using a modeling tool that enhances the engineers workflow. And so it was at this point that we returned to the requirements stage of the development process to be more concise with the requirements. We decide to think more critically on what the core functionality was to first enable the tool to create WF^+ models, and then after that has been achieved, work towards of functionality for the tool.

3.3 Sirius Implementation

During this stage of the tooling, as the metamodel created in Ecore was extremely volatile it was difficult to maintain pace with the Visual Specification Model (VSM) in Sirius. As such the VSM was only updated to test specific functionality that was created in the Ecore model. This did enable us to explore the functionality that is provided by Sirius when it comes to rapid deployment and maintainability of VSM when dealing with a volatile Ecore metamodel. The biggest hurdle when implementing in Sirius is the lack of documentation on the more custom features that are available in the tool. When trying to deploy something like an element based edge relationship (such as the Association class that we have defined) there is very little documentation on how implement the creation tool for it.

3.4 Final Thoughts

With this first attempt many issues became apparent with our naive approach to the development.

1. Complete and concise requirements are a must to ensure that we take into account all the stakeholders when developing this tool. This includes user workflows and user interfaces.

2. Due to constraints with modeling in Ecore and Sirius, some compromises to the mathematical model might have to be made in order to allow implementation and ease of use for the users. These compromises were not to be done in a way to forsake the formal approach that is being done to the development and maintainability of assurance cases.

3. A feature diagram or product family tree is critical to determining what are need-to-haves for the tool and what are nice-to-haves.

Chapter 4

Iteration 2 of Tool

While keeping in mind all the lessons learned from 3, we restarted the development cycle, beginning with the requirements for the tool, with an emphasis on core functionality and what requirements can be implemented at a later date.

4.1 The Requirements for Iteration 2

4.1.1 Engineer wants to create new WF+ metamodel

- Engineer Viewpoint

    1. Engineer shall be able to create new project for WF+. Names project as safety engineering workflow.
    2. Engineer shall be able to browse WF+ GUI palette for desired elements to build WF+ model (process, data, arrows).
    3. Engineer shall be able to save WF+ model.

4. Engineer shall be able to validate syntax of created WF+ metamodel.
5. Engineer shall be able to add custom constraints to WF+ metamodel (syntactic and semantic, BE2).

- Tool Viewpoint

1. Tool shall have unique syntax for all of its elements.
2. Tool shall have a bijective relationship between its syntax and semantic elements.
3. Tool shall provide basic syntax checking for errors.
4. Tool shall provide useful error codes to aid the user in debugging model.

The first requirement for this second iteration was defined around the functionality of the tool; it should allow users to create a WF+ metamodel. During the first iteration we lost sight of this in favor of other features and as a result the Ecore models grew increasingly complex without a benefit to the editor. This is the main requirement where we started development for iteration 2 of the tool.

4.1.2 Engineer wants to derive assurance from WF+ metamodel

- Engineer Viewpoint

1. Engineer shall be able to create both syntactic and semantic constraints on WF+ metamodel(s).

2. Engineer shall be able to create review checks on processes.
3. Engineer shall be able to generate assurance claims when combining at least 1 syntactic and at least 1 semantic constraint together.
4. Engineer shall be able to generate assurance claims from other assurance claims.

- Tool Viewpoint

1. Tool shall provide traceability from assurance claims to the process and data used to derive it.

This requirement definition is to focus solely on how we can derive assurance that a product is safe from the SEPs that are defined in the WF+ metamodel. The idea is that so long as the WF+ metamodel is complete and well defined, we can derive the assurance case claims and evidence from it. Then by the transitive property of instantiating the WF+ metamodel, the instances will also have the assurance claims and evidence baked into them.
This is one aspect that go lost during the first iteration of the tool. Because the original requirements were not well defined nor complete we lost track of some of the core functionality for WF+ which is ultimately a tool to enable more efficient development of assurance cases for safety critical solutions.

4.1.3 Engineer wants to connect WF+ metamodels to each other

- Engineer Viewpoint

1. Engineer shall be able to connect the outputs of one WF+ metamodel as the input to another WF+ metamodel(s).

2. Engineer shall be able to create WF+ metamodels within another WF+ metamodel.
3. Engineer shall be able to create WF+ metamodels separately and connect them together at a later time.
4. Engineer shall be able to view how all the WF+ metamodels interact with each other (whether they are connected or isolated).

- Tool Viewpoint

1. Tool shall provide an interface to view an abstract level of WF+ metamodels.

This requirement is focused on the modularization of WF+ to allow for easier development. Rather than having one giant WF+ metamodel that might be extremely intimidating to deal with, being able to develop the WF+ metamodel is smaller chunks is much easier, not to mention standard developing practice. This is also reminiscent but not exactly the same as the concept of using templates. While both allow for a modular design, there are no templates in this iteration that allow for re-use of WF+ models. Instead this modularization is meant to be an approach to abstraction; a way of zooming out to get a look at the big picture to make sure that the user is able to keep track of what is currently being done and how it relates to what has been done, and what needs to be done.

4.1.4 Engineer wants to transform WF+ metamodel to a GSN-type viewpoint

- Engineer Viewpoint

1. Engineer shall be able to select an option to generate GSN viewpoint of WF+ metamodel.

- Tool Viewpoint

1. Tool shall provide useful errors if the WF+ metamodel is not complete enough to generate GSN viewpoint.
2. Tool shall provide traceability between GSN viewpoint and WF+ metamodel.
3. Tool shall be able to transform back from GSN to WF+ after user edits GSN version of WF+.

This requirement is a first attempt at defining how we want inter-operation to work for this tool. As GSN is a commonly used tool for the development and maintenance of assurace cases, having some way of transforming WF+ to GSN would be a natural advantage for users to adapt to working in WF+. This would also allow for easy visualization of how the derived is working based on the defined SEPs. It would also make it easier for regulators to understand what has been done as they wouldn't need to learn a whole new modeling methodology to be able to judge whether a product is safe enough.

4.1.5 Engineer shall be able to edit GSN viewpoint

- Engineer Viewpoint

1. Engineer shall be able to edit generated GSN viewpoint of WF+ metamodel.
2. Engineer shall be able to transform in the reverse direction back to WF+ metamodel after making changes

- Tool Viewpoint (n/a)

This requirement is a follow-up to the previous one as this would allow the user to further edit the generated GSN version of the assurance case. Being able to transform back also ensures that there is no loss in traceability between the two versions, ensuring that they can remain reflections of one another. By enabling this bi-directional transformation it also allows for multiple assurance case development strategies.

4.2 The Metamodel

At this point in the project, rather than develop in UML we decided to develop right away in Ecore. This was because at this point in the development life-cycle I was more comfortable in the Ecore environment, it allowed for a faster implementation time, and the rest of the research team was more accepting of the technology.

4.2.1 Ecore Model Iteration 1

For this iteration the main goal was to implement the most core requirement; being able to create WF+ models. Thus we had to consider the rules about how WF+ is built; data input and output to process. Since WF+ is a kind of unified modeling framework between data modeling and process modeling we had to consider how those two very different types would interact with one another. Thus we focused solely on developing the abstract syntax for that in this iteration of the tool. In this iteration, shown in 4.1, of WF+ a few decisions have been made. For one the issue of re-usability of WF+ graphs

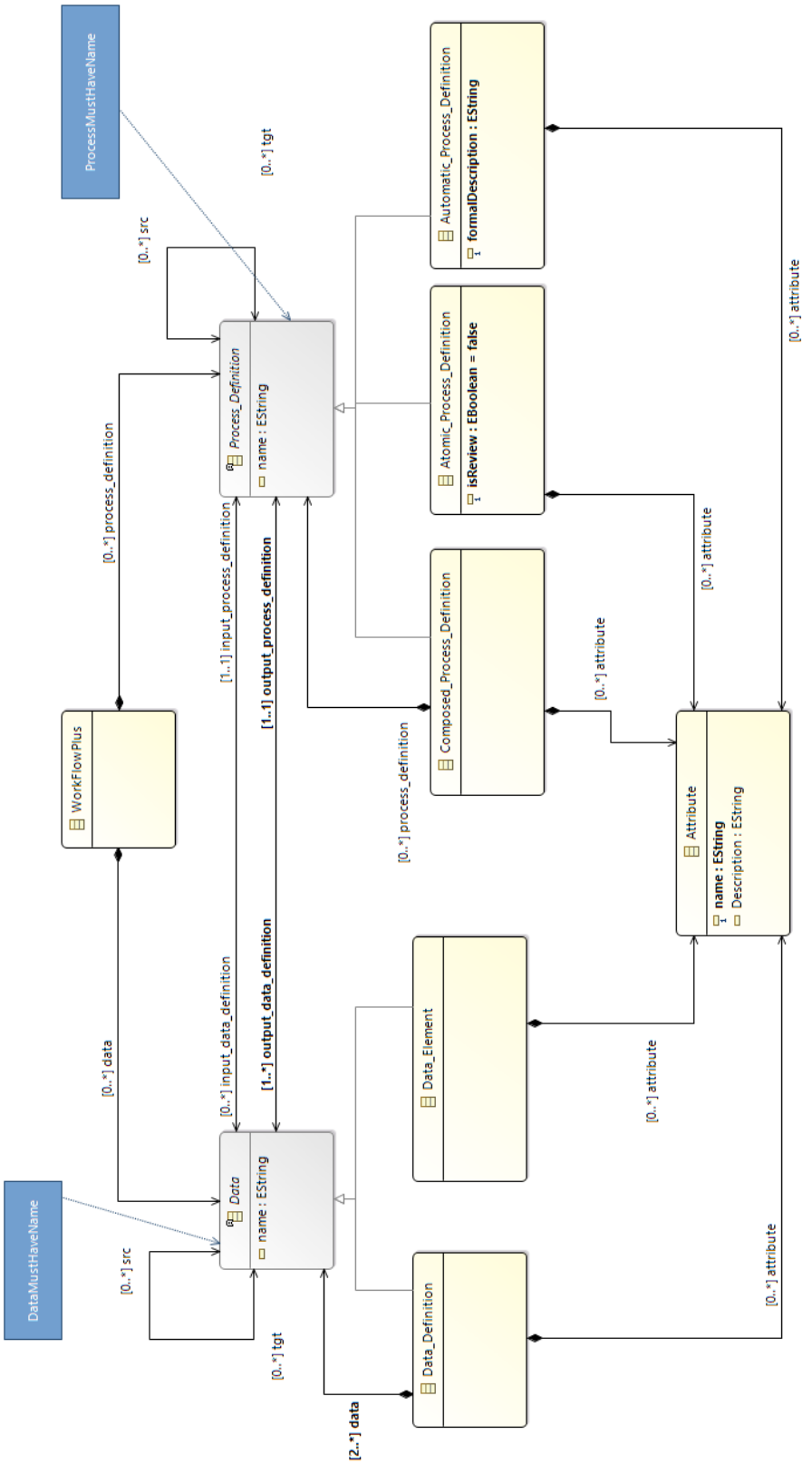

Figure 4.1: This is the first version of the tool after redefining the requirements. This iteration focused solely on the abstract syntax of how data and process can be connected to one another.

and elements has been shelved for a future change (thus getting rid of the library classes). In this metamodel we focused on core functionality of the editor; being able to create data and process elements while grouping them and connecting them to form a workflow. This metamodel allows that precise functionality. We kept the three types of process definitions that existed in the previous version (composed, atomic, and automatic) while getting rid of the process invocation class. This now creates ownership of a process in the container that it is defined it but does not allow for its re-usability by other processes. This also makes defining the hierarchy much easier as we now have clear leaf nodes that can be used for the transitive closure (atomic and automatic). We also have a flag in atomic process to define if the process is a review or not. This may be changed later if the reviews are too complex to be held within an atomic process element.

On the data side we got rid of the data metamodel class entirely and just kept data elements and data definitions. As defined in the metamodel, the data element class is the leaf node for the transitive closure. Finally, an addition here that was not present in any other iterations is the Attribute class. This class was added to allow us to treat the instantiation of the metamodel more similarly to class diagrams. As of right now every class can have attributes but this may be amended in future iterations.

The references in this metamodel are also quite simple. The only references between data and processes are handled by the abstract classes. This simplifies defining the input and output relationship. The reference definitions are also bidirectional to allow for the instantiation models to allow defining the relationship from either side of the reference. The multiplicities may change as the project progresses. Also, there are association that are capable between

data classes and process definition classes. This was to enable reified associations later if still desired.

There are also some basic constraints in this iteration as we wanted to see how they affect the instantiation. We predict that constraints will likely be added in Sirius as opposed to Ecore, but some of the advantages of having it in the Ecore level would be the centralizing of everything for one level in one model. This iteration is much simpler than the previous ones, but also much more focused. As we redefined the requirements it allowed us to determine what features are critical to the tool and what features can be pushed off to a later date. This metamodel was specifically designed to enable the immediate creation of WF+ models and to allow for the definition of constraints for the editor. At this stage we were still developing the concrete syntax for the editor and figuring out what other constraints might be needed before we could begin on figuring out how to evaluate the editor.

4.2.2 Ecore Model Iteration 2

In the iteration shown in 4.2, we removed some of the associations to the Attribute class that we discovered were unnecessary while developing the concrete syntax, specifically to Data_Definition and Composed_Process_Definition. Since those classes are intended to be used as containers for the hierarchy we decided to confine the Attributes class to the Element classes that are contained so as not to pollute the concrete syntax. We have also added the Constraint class in this version so that the user will be able to place constraints into the instantiated model. Currently the EType that is used to define the constraints is EString so it will have to be interpreted by something in order to function

as an actual constraint; ideally OCL since it is the most commonly used by the modeling community.

Finally we have also added the Reference and Node abstract classes. The Reference class was created in order to define the Associations class and the Reify_Associations class. This was done so that in the instantiation the user can define associations with multiplicities that can be displayed in the editor. The Reify_Association class was also created so that as the name implies, the users will be able to draw associations from Reference elements to Node Elements. This brings up the Node class. This class was created so that it would be easier to have Constraints be able to constrain both Data and Process with one reference in the metamodel. It also has the bonus of doing the same thing for the Reference class and the classes that inherit from it, thus making the model easier to maintain for future revisions. For now it is easier to maintain the editor by having the root element of this metamdoel, WorkFlowPlus, be composed directly of Process and Data rather than Node. As a result we did not change the composition relationship.

The introduction of the Reference class also leaves the metamodel open to adding other types of references in the future if they are deemed necessary for creating WF+ models. Thus creating a metamodel in Ecore that is robust with respect to anticipated changes. During the analysis of some examples with this version of the metamodel it was discovered that there were a couple of flaws with previously defined WF+ models due to a conflict of the ownership of classes due to different ways of defining it. This brought up the discussion of whether or not to implement Aggregation in the metamodel.

This brought up an interesting point because there is an inherent limitation

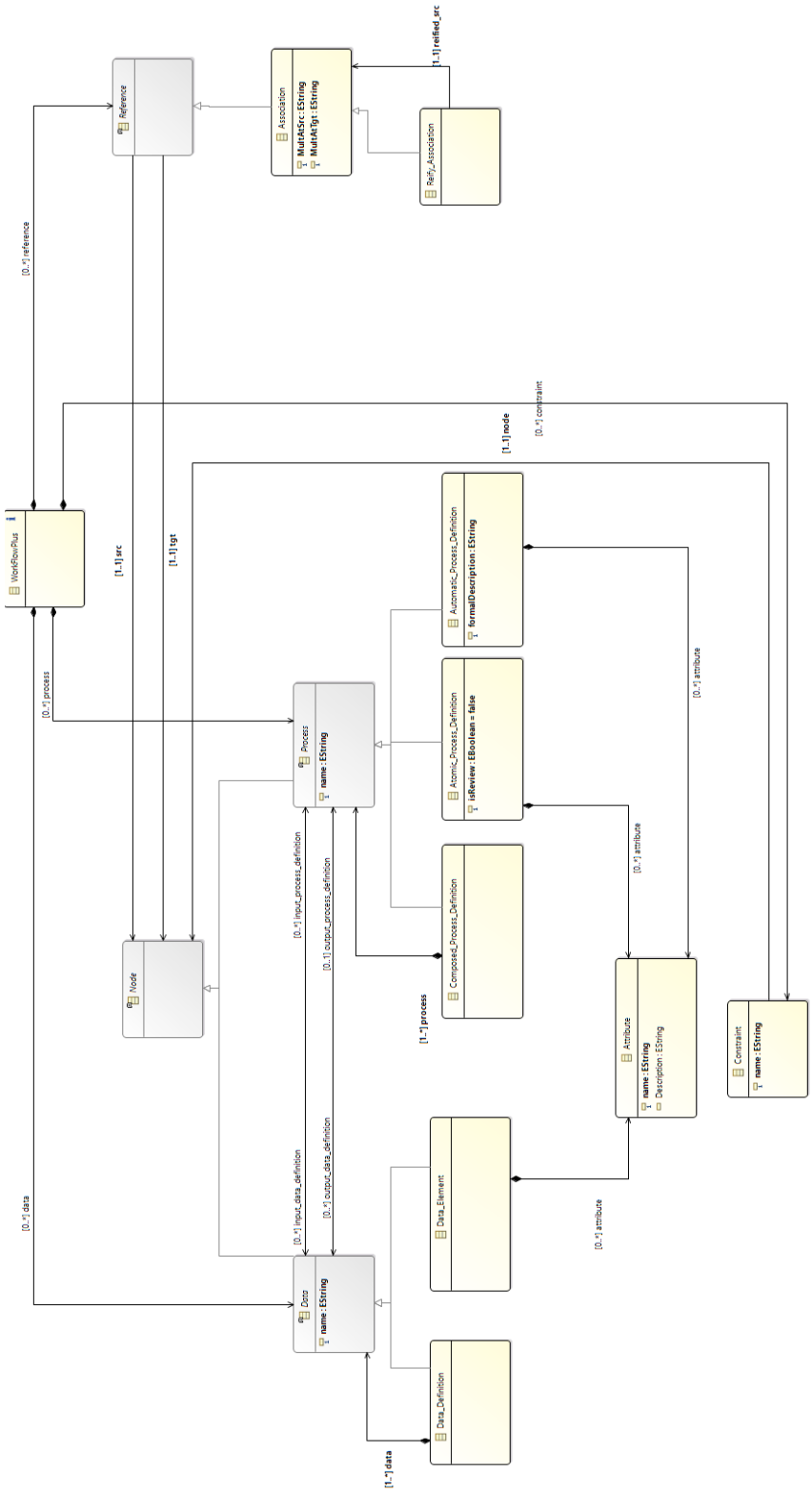

Figure 4.2: The addition of formal definitions for references being created

in the tools that are being used (Ecore, Sirius). This is because there is no Aggregation currently defined in these tools, thus it would have to be manually defined by us. Further analysis will have to be done on the WF+ framework to determine if Aggregation is indeed needed, which is why having the Reference abstract class is so important to the robustness of this metamodel.

Another thing worth noting here is that we removed the Ecore constraints from this metamodel. We decided that we would instead implement constraints in Sirius, in the specification file. This way we can avoid adding to the complexity of the metamodel and keep all the definition for the editor in one place, thus reducing the coupling between Ecore and Sirius. This also increases the cohesion in Sirius as it means we can use the native language of Sirius as well which is Acceleo Querying Language (AQL). While this does mean that we will need to learn another flavor of OCL, we believe this is the right design decision to maintain low coupling and high cohesion between the two tools.

4.2.3 Ecore Model Iteration 3

In 4.3 we see some significant changes, primarily in how all the elements can be connected together. It was determined that in order to remain faithful allow for easy translation and understanding a composition arrow is required along with inheritance. At this point, all the reference classes are connected at the Node abstract class, but are only drawable between Data classes due to how it is specified in Sirius. Having the references connect to Node helps to improve robustness if it was decided in the future that we wanted to allow some of the references to work the Process classes as well.

Next there are 2 new classes on the left of the model: Work_Product and

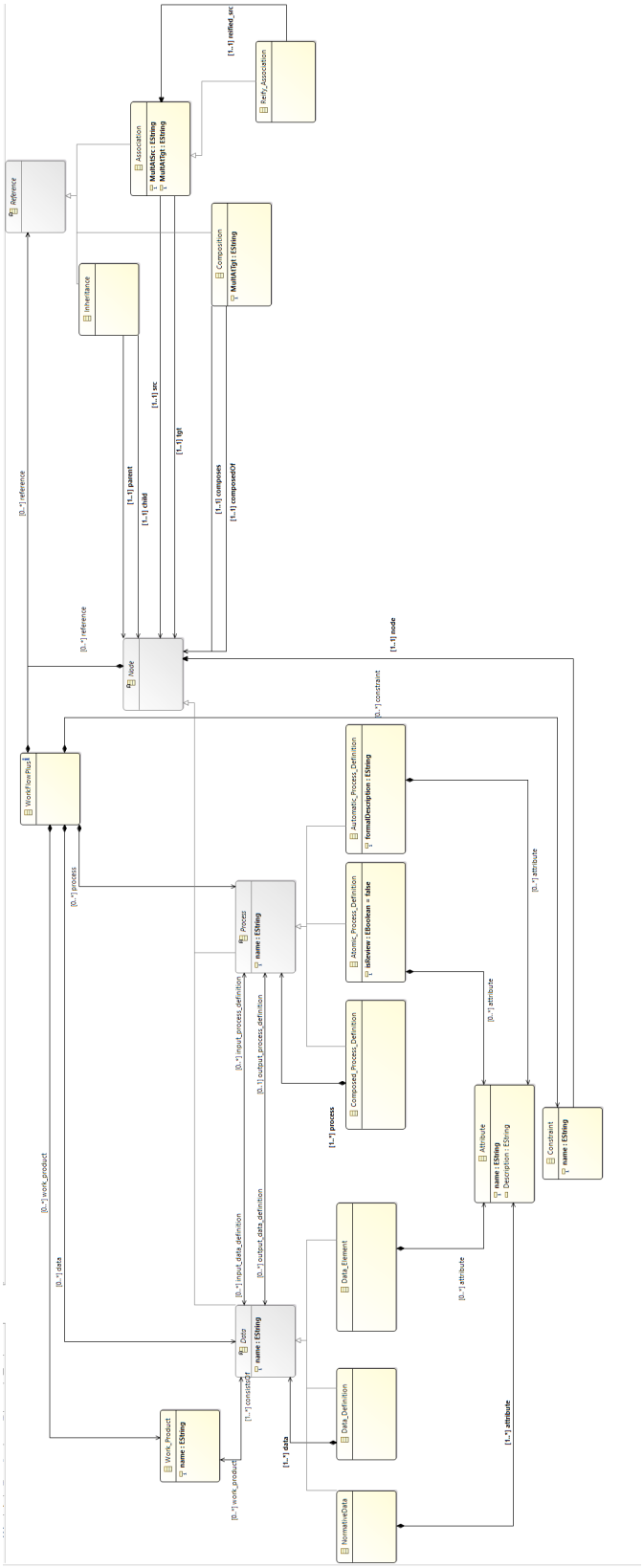

Figure 4.3: More formal definitions of reference classes as well a a parent abstract class of Node

Normative_Data. Work_Product was introduced as we ran into ownership issues with Data classes as they were produced from Process classes. It was possible for an output piece of data to be owned by both the input to a process, as well as a work product, which was a collection of output data pieces that would be used for V&V or other tools to check milestones and quality. As such, Work_Product was created, with a weak ownership association to the abstract Data class; Aggregation. This is reserved solely for Work_Product as we only want to show that it is a special type of Association and don't want to open the option to use it up to the rest of the model at this stage. Finally there is Normative_Data. This class is to allow for the definition of standards and other pieces of data that are necessary for the SEP, but are not in direct control of the SEP. For now it is defined as a sub-part of the Data abstract class so that it can be easily defined, though this will need to be reviewed moving forward in order to ensure that it actually makes sense. From this stage there are a few key steps to be completed; a transformation method to elevate the instances of this model into something that can be executed, a way to query the models, a way to explicitly sequence Process items, and a way to derive assurance from the model.

4.2.4 Ecore Model Iteration 4

In 4.4 there were some minor adjustments made after some preliminary testing. The first difference is that now the Work_Product class inherits from Data so that it can be a part of the hierarchy if needed. This was done because we are still struggling to figure out how we want ownership to work syntactically. Aggregation was also made into a formal class so that the possibility to make

it apply to more than just Work_Product is feasible. The ability to explicitly state the sequence that processes should be executed in was also introduced with the self-referencing association to/from Process with next/previous. This also allows for the ability to define parallel processes. Finally, I defined the Input and Output classes between Data and Process so that we can have multiplicities on those connects, as well as labels if desired later in the project as well. Furthermore, by making them into classes it simply allows for an easier way of editing those classes if needed in the future, thus making the metamodel more robust. They inherit directly from the Reference abstract class instead of from Association because we want them to remain independent of the Association class so as not to pollute them with whatever changes might be made to the Association class in the future. Aggregation still inherits from the Association class as it is simply a special form of Association with a stronger sense of ownership. Moving forward, we will be working on testing what models can be made from this metamodel. This is to test the editor to make sure that we can at least make some base level WF+ models. This will provide us with the baseline that we need so that we can start to make more decisions with the direction that we want to take this tool in. The likely directions are in the constraints and review nodes. Currently, they are a single class, respectively, but in discussions we have come to the conclusions that there is a lot of information that needs to be sorted through with those classes. Constraints will eventually be used as the building blocks of assurance, while reviews have much to do with how those constraints come to fruition. Thus, there is a high level of dependency between reviews and constraints, and ultimately, with how assurance can be generated from a WF+ model.

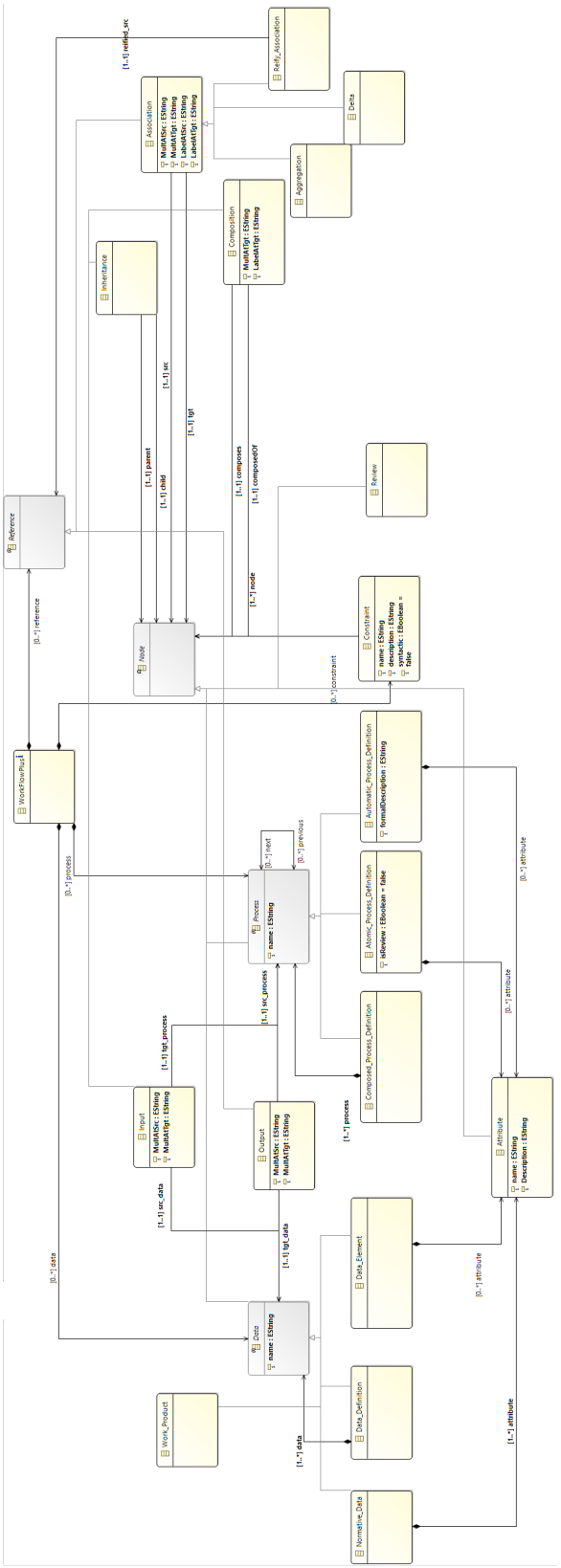

Figure 4.4: More attributes for the reference classes and some minor refinements

4.2.5 Ecore Model Iteration 5

In figure 4.5 we have attributes specified for all of the reference classes so that we can specify the multiplicities and end labels within the VSM. They are also unique so that there is no inheritance issues with redundant information and naming being inherited by children classes. Comments have been added to help keep track of design decisions that have been made, as well as keep track of future work. This version is also the one that is used to show the Specification of the editor and the evaluation of the tool.

4.2.6 Ecore Model Iteration 6 - February 11, 2021

Finally in figure 4.6 we have the latest revision of the metamodel at the time of writing the main body of this thesis. There are two major changes in this revision, both to the data side of the model. The first change was the removal of the Normative_Data class as it was replaced witha boolean attribute in both Data definition and element. This was done to declutter both the metamodel and the editor since normative data is effectively no different from any other data; the only difference being the source of the data. The second change was the addition of the Performance_Data class. This is a unique class as it is used to show the resources that will be required/used for the execution of a process; personnel, their education, experience etc. This class is unique as it does not fall into the usual input/output relationship that all other data has with process. It is meant to be used more for the sake of reviewing executions to generate assurance.

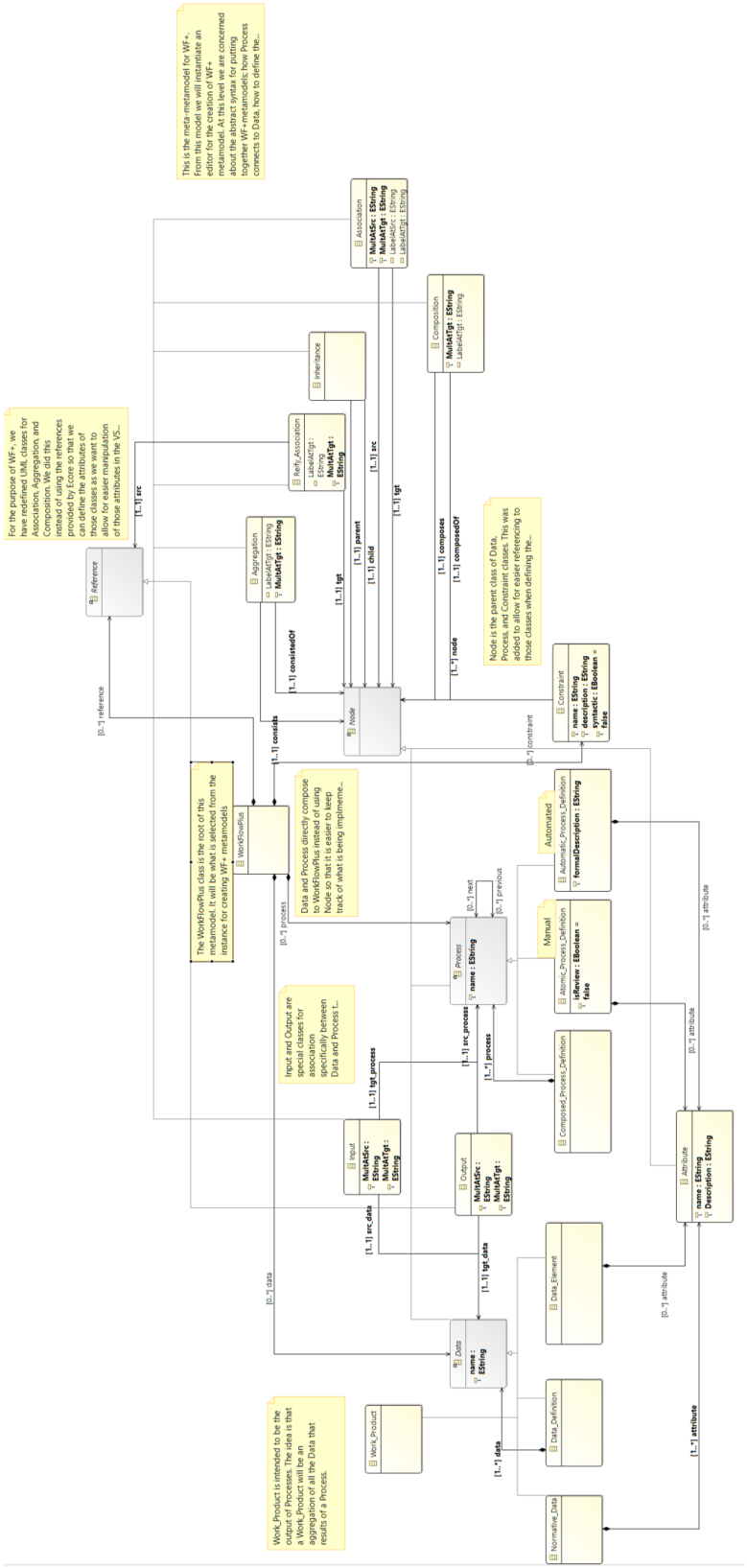

Figure 4.5: Comments added and more attributes added for the reference classes.

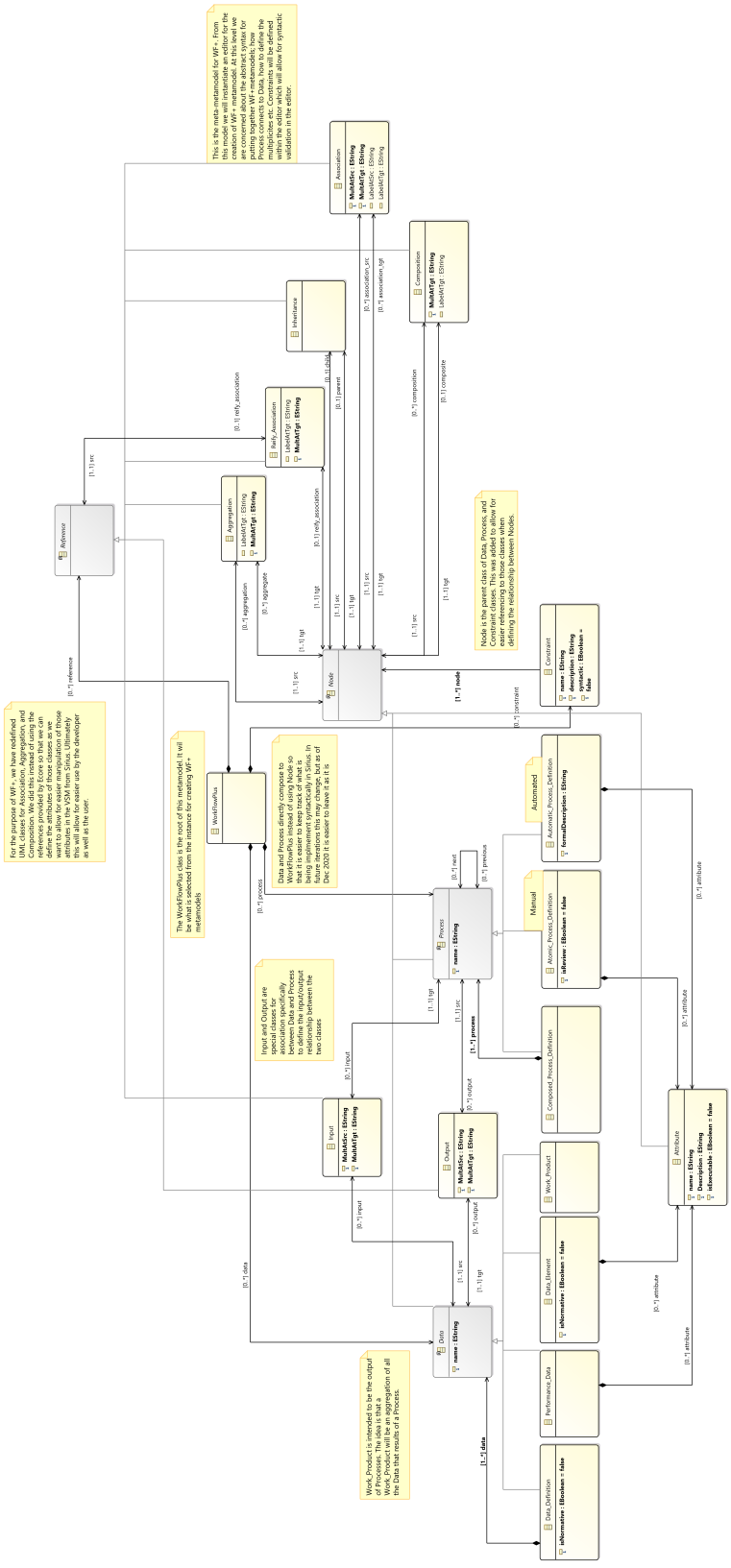

Figure 4.6: At the time of this thesis this is the latest working version of the metamodel.

Chapter 5

Tool Specification

Due to our choice in technology, the specification of the editor, as defined in the VSM, can only happen after a metamodel has been defined. Sirius is heavily dependent on an Ecore model to be defined in order to generate all of the files needed for specifying an editor. One of the great things about Sirius and Ecore is that this is a one click process and so we were able to start creating the editor immediately after specifying the Ecore model. For the sake of remaining concise we will only be looking at the evolution of the VSM for the second iteration of the metamodel. 5.1 shows a look at the whole VSM file for the tool. In this section we will take a deeper look at all the aspects of this file to see how the tool been specified and to explore design decisions were made to implement the requirements of the tool and previous research that has been done on graphical editors and DSLs.

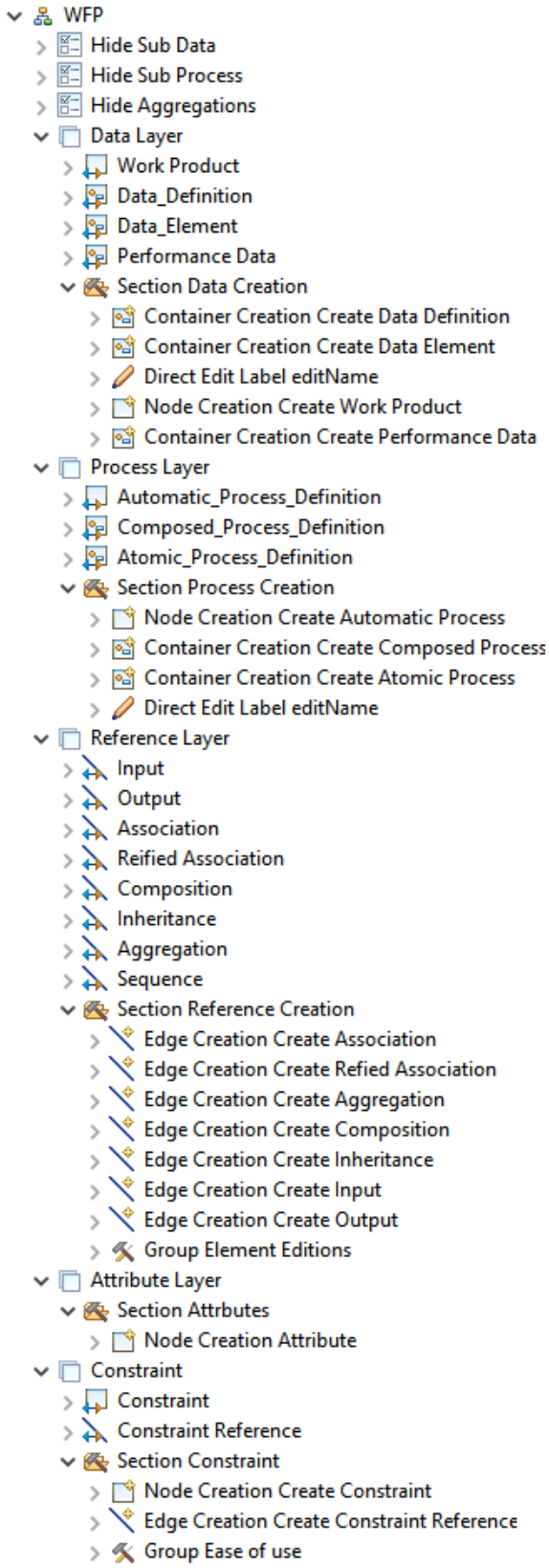

Figure 5.1: A top level view of the VSM project in Sirius

5.1 The Visual Specification Model

The basic functionality for the tool was built based on the metamodel from 4.1. Despite how the metamodel has changed over the different iterations, this specified the basic functionality for being able to define Process and Data classes and how they connect with one another. In figures 5.2 and 5.3 we show a few more details about how the Data and Process classes have been specified for their concrete syntax. Despite the Physics of Notation design principles still being relatively young, we used those principles as guiding question when deciding on the concrete syntax for the tool. The principles that we paid the most attention to were semiotic clarity, semantic transparency, and graphic economy.

Before beginning the specification for the model however, it is important to remember what we are trying to achieve with this editor. Keeping in mind that with iteration 2 of the metamodel our main requirements that were guiding our design decisions were defining the input/output relationship between the process classes and the data classes, and usability. Since those are our main 2 requirements, our goal with the editor was to make an easily usable editor that allowed the end user to be able to specify the workflows that they want from the beginning input of the workflow to the final output.

To begin we will look at the different types of data classes. There are 4 main data classes: Data_Definition, Data_Element, Work_Product, and Performance_Data. The Work Product class has a different shape than all the other Data classes, while all the other Data classes that share a similar shape have different colors depending on what they mean; Sub_Data_Element is light yellow, and Data_Definition is a gradient from white to light yellow for exam-

ple. This allows for extra differentiation despite them all have a rectanglular shape. The color yellow was chosen for data primarily because we thought it is still easy to see black text on top of it, while the grey color was chosen because it can contrast easily from the yellow. Another simple difference is rounded corners vs. sharp corners. The main reason why decided to keep them all as rectangles and had to look for other ways to differentiate the syntax was so that when users use the tool they would be able to intuitively understand what the shapes mean based on previous experience with other graphical languages, such as UML. This design decision was made so that implementing the requirement of ease-of-use was reflected in the design of the editor, while trying to also minimize the propagation of unexplained design decisions that were made in previous graphical languages.

The Data_Definition class has the option of being nested within it itself while Data_Element classes can only be nested within Data_Definition classes. This enables the user to draw hierarchies as defined in the metamodel. This is reflected in the Composed_Process_Definition class as well. Furthermore all of these classes have been defined as containers so that the user can create attributes within them to allow for those classes to own those attributes. We will show how these behave in the evaluation of the tool.

Second after the data classes we have the process classes. In figure 5.6 one can see the rounded corners for the Data classes compared to figure 5.7 where the process classes have sharp ones, along with also have different colors. This goes to show how we strived to reduce the syntax redundancy for the unique classes, while still remaining similar enough to the UML syntax so that users with experience in modeling can still understand that they are class diagrams. The Work_Product class is unique in how it is an object that act

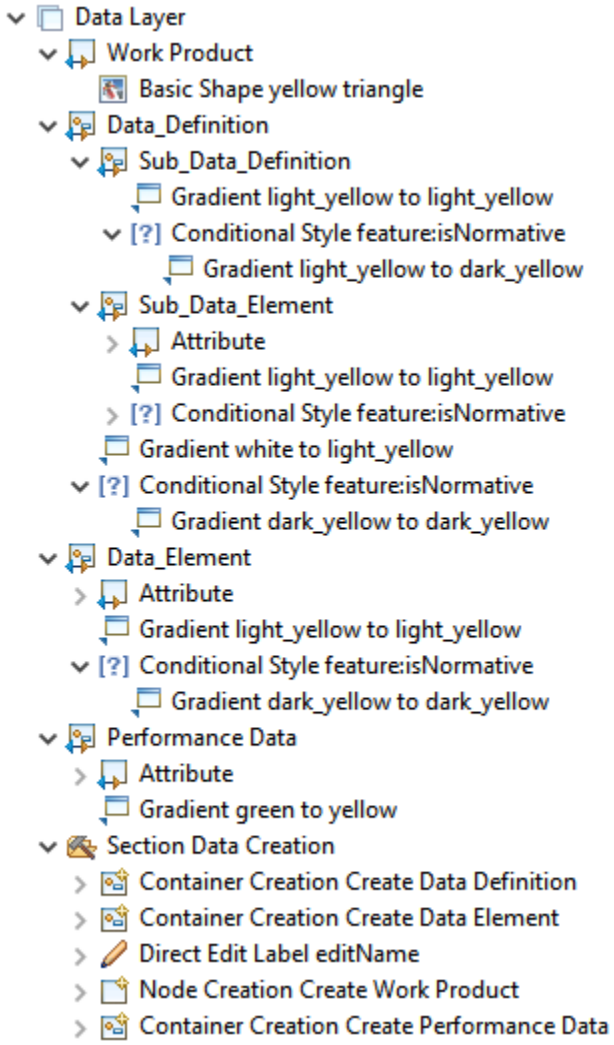

Figure 5.2: A more detailed view of how all the Data classes have been specified

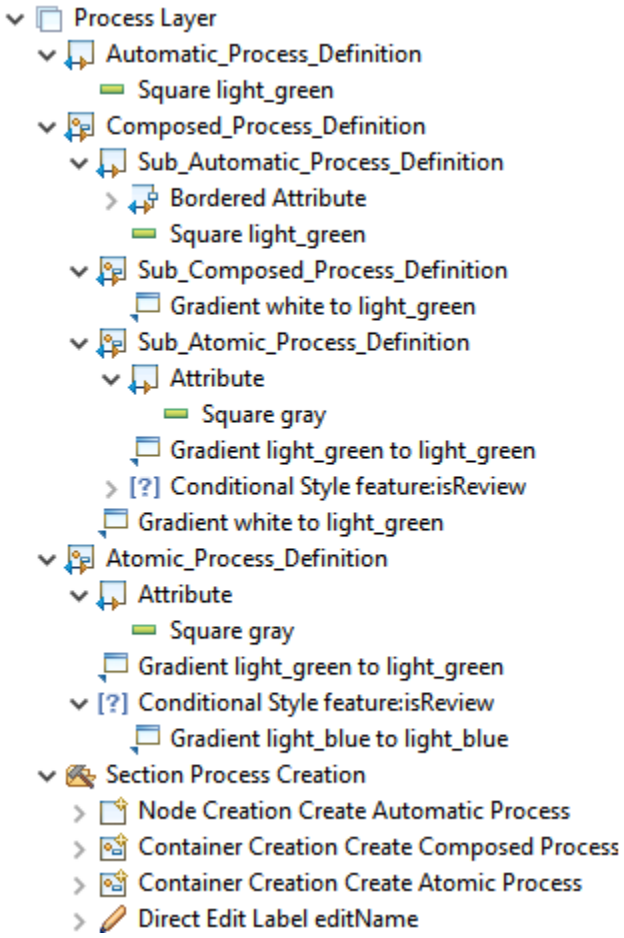

Figure 5.3: A more detailed view of how all the Process classes have been specified

similar to a head to aggregate a bunch of other Data classes together, hence it has an entirely different shape from the rest of the classes, further showing its difference as it is not a class diagram. Data classes that are a darker shade of yellow have the boolean attribute 'isNormative' set to true to signify that it is normative data. This difference is then highlighted by the color change. This is done via the Conditional Style specification in the Data classes. The same approach is used for Atomic_Process. If the process is meant to be a review, by setting the boolean attribute 'isReview' to true, the editor triggers a change in color for the concrete syntax of the class from light green to light blue as shown in 5.7. The last class of the process classes is the Automatic_Process. The concept for this process is to allow engineers to specify processes that would not require human intervention. Currently this class does not have the ability to hold commands for automation, however the hope is that moving forward in the project we will be able to specify a simple editor that would allow for a user to place OCL commands so that the process could be formally specified, and executed by a machine.

Next we have the Reference classes as shown in 5.4. As shown in the last metamodel in 4.5 we have defined classes for the most commonly used UML edge constructs; Association, Aggregation, Composition, Inheritance. This allows us to be able to have more control over the semantics and syntactic behavior of these classes, decoupling from the built in semantics and syntax of the edges available in Ecore and Sirius. This decision was made to once again make the tool a little easier to use as most users would already be familiar with UML syntax and can thus be instantly familiar with what the syntax mean, increasing the semantic transparency of the syntax. This does however come with a cost as we now have some syntactic redundancy in how we define

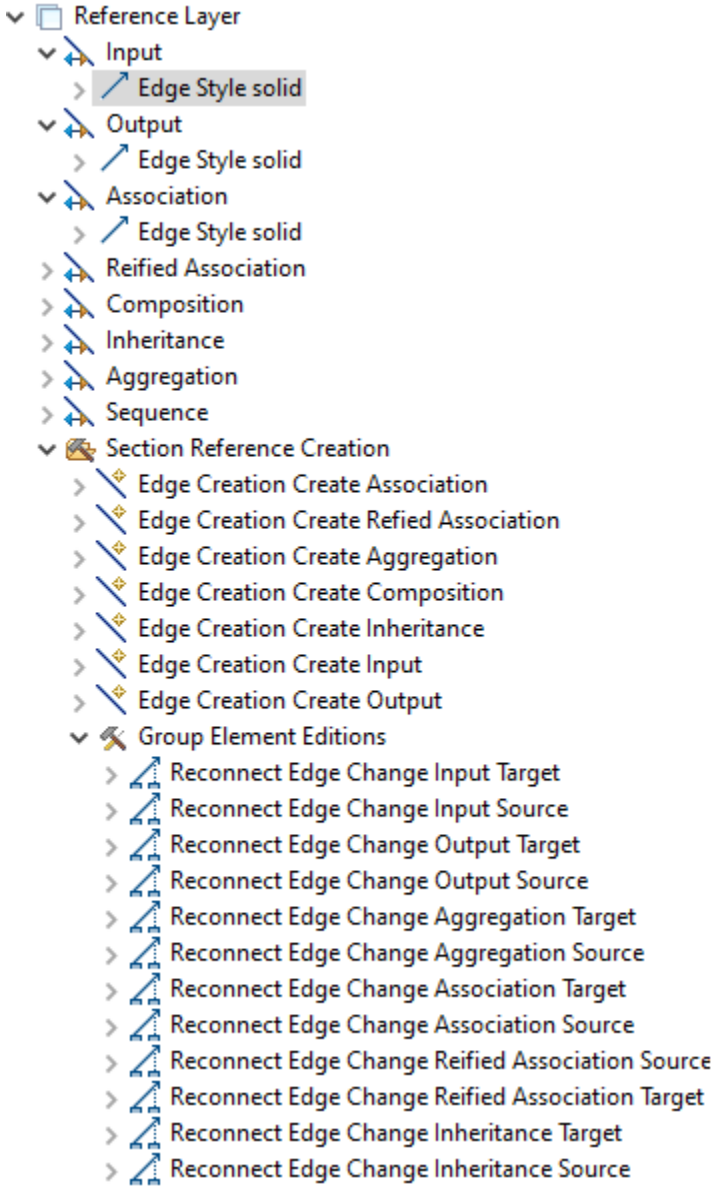

Figure 5.4: A more detailed view of how all the Reference classes have been specified

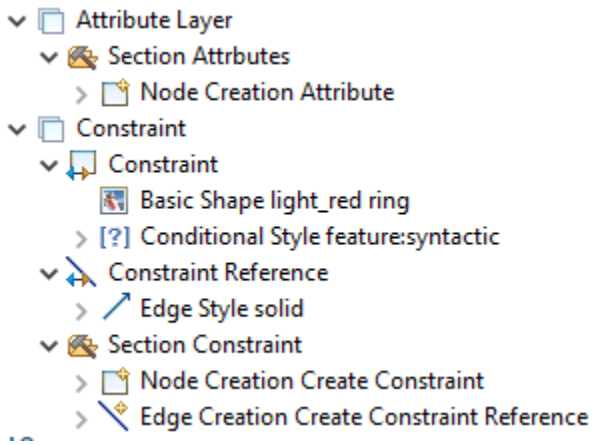

Figure 5.5: A more detailed view of how the Attribute and Constraint classes have been specified

ownership/composition of classes.

Finally we have the specifications for the Attribute and Constraint classes. There are relatively simple specifications when compared to all the other syntax. There is only one syntax for Attributes which is to be listed in an element class (data or process), while constraints also only have a conditional syntax; they change color depending on if the constraint created is a syntactic one or a semantic one. This differentiation is up to the user and is changed with a boolean value 'syntactic', just as the two previous classes.

One thing to note is that all of the syntax relies on different colors to denote different meanings for each element. While using colors is less than ideal as it has inherent accessibility issues, such as for end users that may be color blind, it was a necessary decision to take as without using color it would have been impossible to continue use the rectangle shape while maintaining semantic differences in the syntax. Furthermore, the main reason we wanted to keep using the rectangle shape as much as possible was so that it would make the tool more familiar to end users who are already used to other languages

like UML since it would have some similarities with the syntax, thus working towards satisfying our usability requirement. One work around that we have to the issue of accessibility for people who may be color blind is the ability to change the color of any of the classes as-hoc, though we will see later on in the Conventions section that while it is feasible, we don't currently have a way to enforce consistency throughout the model if the end user decides to start changing the colors of the classes.

5.2 The Syntax

In this section we will showcase the syntax of the tool. We will show some of the basics of how the components can be combined together, as well as a simple WF+ model to show the usability of the tool at this stage of development.

5.2.1 Data, Process, and Attributes

This tool inherits much of it graphical syntax from UML despite the lack of design rationale that is present in order to reasonably satisfy the requirement of usability and intuitiveness. As a result we the Data_Element and Atomic_Process classes both look like classes that would normally appear in a UML class diagram. The differences between the two are color and shape (Data classes have rounded corners instead of sharp corners using 2 of the 3 syntactic tools to create a distance between the two classes) as shown in figures 5.6 and 5.7. The Automatic_Process class does not currently have the ability to hold attributes as for now the plan is for a process that can be automated to have a formal OCL definition instead of attributes to define how the process is supposed to run. Work_Product is a special type of Data class as it cannot

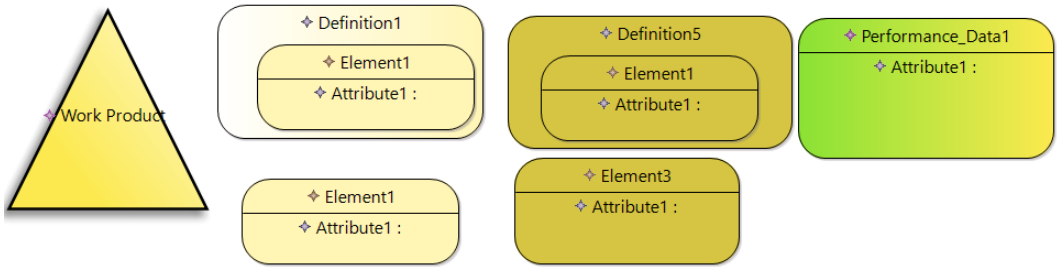

Figure 5.6: This is the concrete syntax for Data classes and their Attributes.

hold attributes either, and is instead intended to be used as a header to aggregate many data classes. This is because a work product might not necessarily own the data that it is composed of. As a result an aggregation works better here as there is a stronger sense of ownership between a work product and what it consists as opposed to a regular association.

Since the Work_Product class can not hold attributes, it doesn't qualify as a class in the same way that Data_Element and Atomic_Process do so it has an entirely different shape than both of them; a triangle. It is still yellow to show that it is part of the Data family, but has a different shade of yellow to further differentiate it from the other Data classes. Finally, the dark yellow variants of Data_Defintion and Data_Element is the way that normative data is currently represented in WF+. A data class can be considered as normative if it is used to show data that is used within a workflow that comes from an external source such as a standard (ISO 26262) or another engineering process (design documentation for example). As this is still considered to be a class diagram element it can therefore hold attributes, thus earning it the same shape as Data_Element. However as it comes from a different place (an external source to the SEP) it instead has a darker shade of yellow to show the difference between the two classes. While using only color to differentiate

classes comes with some flaws such as usability for people who are color blind, it is a compromise that makes the most sense to remain consistent with the other syntactic designs that have been made and still follows the design philosophy put forward in the Physics of Notation. Another unique Data class is Performance_Data, which has the same shape as other data classes, but a very different color scheme. For WF+, by default the Process classes are set to a green color. Performance_Data is a data class that is used to specify data about a process, rather than the input and output of a process that we have been describing so far. This unique class was created so that users could specify things like who is performing the process, how long the process is, qualifications required for the people executing the process etc. The rationale for creating this class is that it will eventually be used as part of the assurance generation from the SEP. If the process is executed properly, and done by the right people, then the output should also be valid. The way this class will be used is still subject to change with future iterations of the tool.

Finally for the Data family, we have the Data_Definition class, whose purpose is to provide a composition hierarchy for Data. Data_Definition can be composed of Data_Element or more Data_Definition, allowing for a lot of depth when creating Data classes. This allows for the use of the built in constraints for operations like the transitive closure when analyzing the created WF+ models. Further more it allows for compact designs without the needs for more edges. One downside is that it doesn't allow for the multiplicity of the composition relationship to be shown right away. It must first be defined as an attribute in the metamodel (Ecore) and then specified in the VSM for there to be a multiplicty with this relationship.

As previously mentioned, the Atomic_Process class behaves the same way

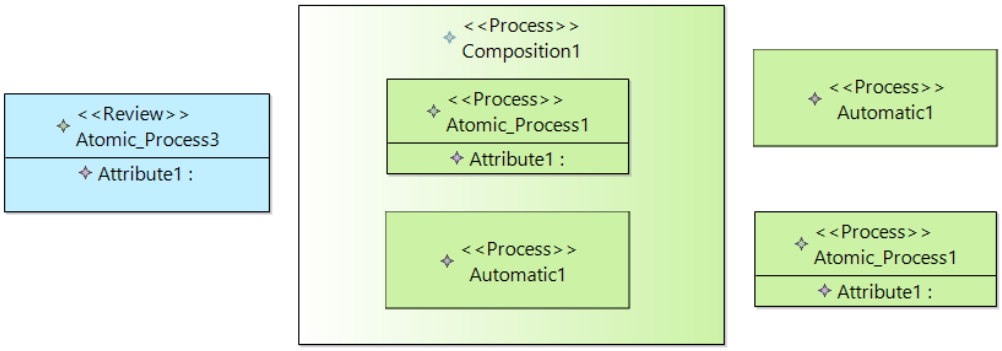

Figure 5.7: This is the concrete syntax for Process classes and their Attributes

that a class would in UML so it has the same shape as one, though it has a specific color to denote that it is a Process class specifically. Composed_Process is behaves the same way at Data_Definition except it can only hold Process classes instead of Data classes, hence it also having the Process coloration. Finally there is the Automatic_Process class. This class is a special class as it cannot hold any attributes at this time. Instead the plan for it is to hold a formal OCL definition for what the process is supposed to do. If it cannot be defined using an OCL definition then it is likely that it is in fact not an automatic process and should be changed to a different class. Furthermore, in order to help with the differentiation between the Process and Data classes, the Process classes have been stereotyped as well. That is, they automatically have the type <<Process>>or <<Review>>placed above whatever title is given to the Process class. This is to further help show the difference between a regular process and a review when the user chooses to set the boolean attribute to true; on top of the color change that occurs as well to a light blue.

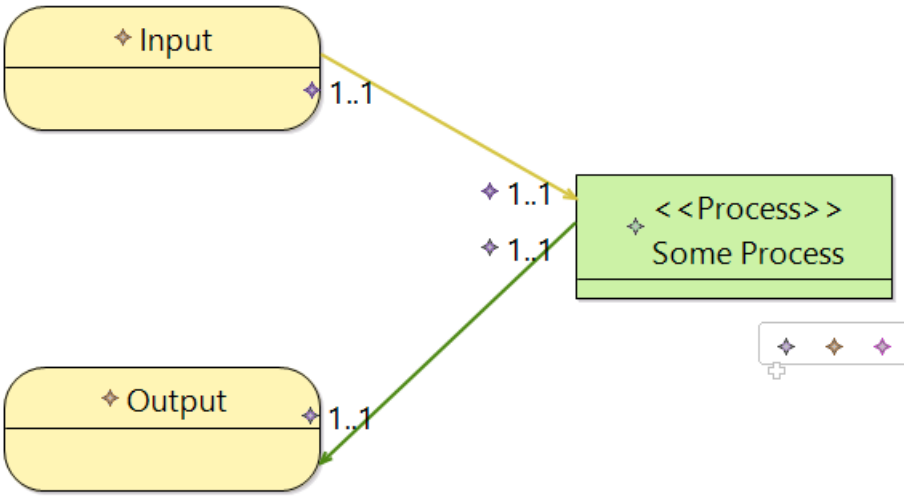

Figure 5.8: How Data and Process classes are put together as input and output

5.2.2 References and Constraints

As previously stated, within the WF+ metamodel we have defined 4 common UML edges; Association, Composition, Aggregation, and Inheritance. Along with those four classes, we also have two special types of associations call Input and Output. As one might suspect, those two special classes are used specifically for defining the input/output relationship between data and processes within an SEP. As show in figure 5.8 one can see how that syntax is represented. The Input edge can only be drawn with Data classes as the source and Process classes as the target, which is also supported by the direction of the arrow. Naturally, the Output edge is drawn opposite to the Input edge; from Process to Data. These two edges are further differentiated by the colors that they use which are the same colors as their source nodes.

Next are the more general edges that have been defined. These have been taken directly from UML and have the same syntax from UML as well. This is to ensure that it is very easy to understand the meaning of these edges which

works towards the satisfaction of the usability requirement for the tool. Some minor things to note about these edges however is that since they have been redefined for the purposes of this tool, their semantics also need to be redefined which has not been completed at this time. For example, in figure 5.9 we can see that we have a composition edge that connects Element9 to Element1, which according to the previously defined syntax, is held within Definition2. This introduces conflicting forms of syntax for the same semantics, and in this specific case, conflicting semantics as well since it is now ambiguous as to which node actually owns Element1; Definition2 or Element9. The concrete definitions of these conflicting semantics however are outside the scope of this thesis and will need to be addressed in future work for this tool. At this time the Composition edge seems to be the only Reference class that has conflicts with the built in semantics of Ecore and Sirius.

Another Reference class that was redefined to allow for flexibility with the syntax is the Inheritance class. While the syntax is the same as in UML, there are no semantics that have been defined for it yet that actually force the inheritance of the attributes and incoming/outgoing edges from the parent to the child. As a result in the current state of the tool the Inheritance class behaves in a similar way to the Association class, but with different names for the source and target nodes.

Despite the increased complexity that comes with redefining these classes, the advantages to having them defined is that it decouples the classes from Sirius. That is to say, we have more flexibility with the attributes that these classes would have, such as multiplicities and end labels shown in figure 5.12, as well as being able to define more actions that a user might want to do with them. For example, a reified association becomes possible, as show in figure

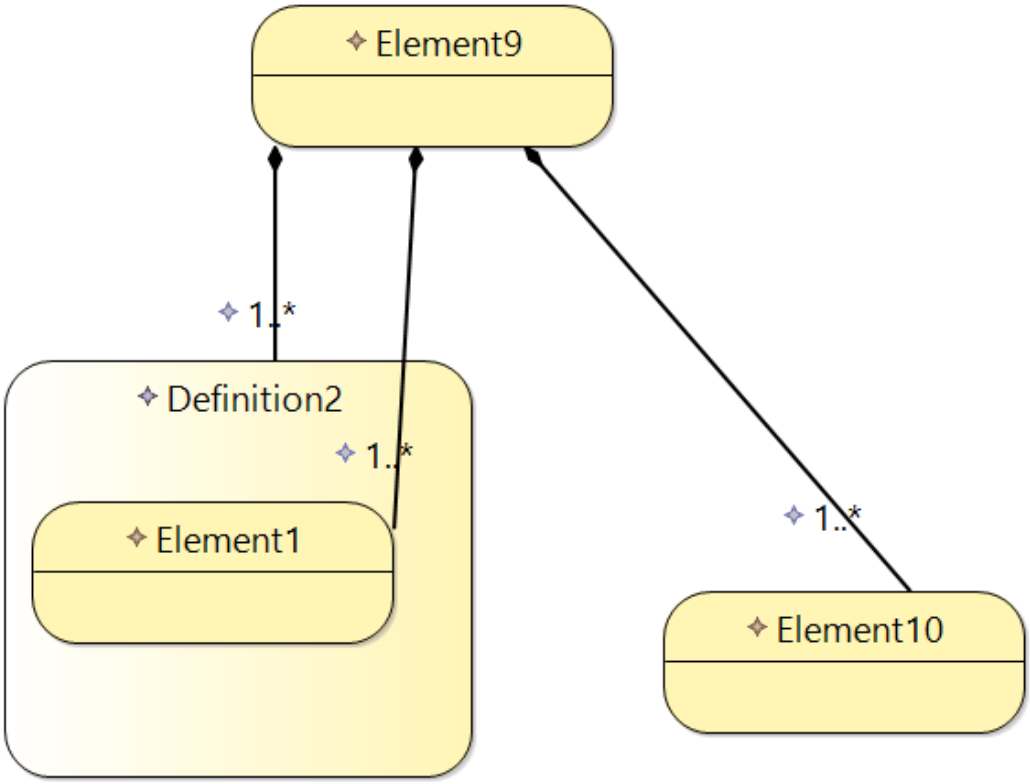

Figure 5.9: The Composition class syntax

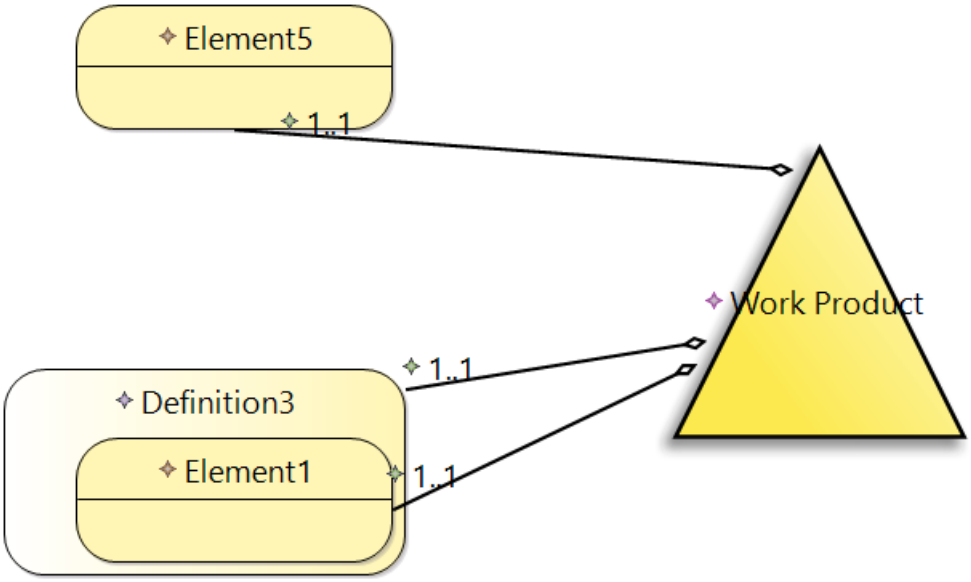

Figure 5.10: The Aggregation class syntax

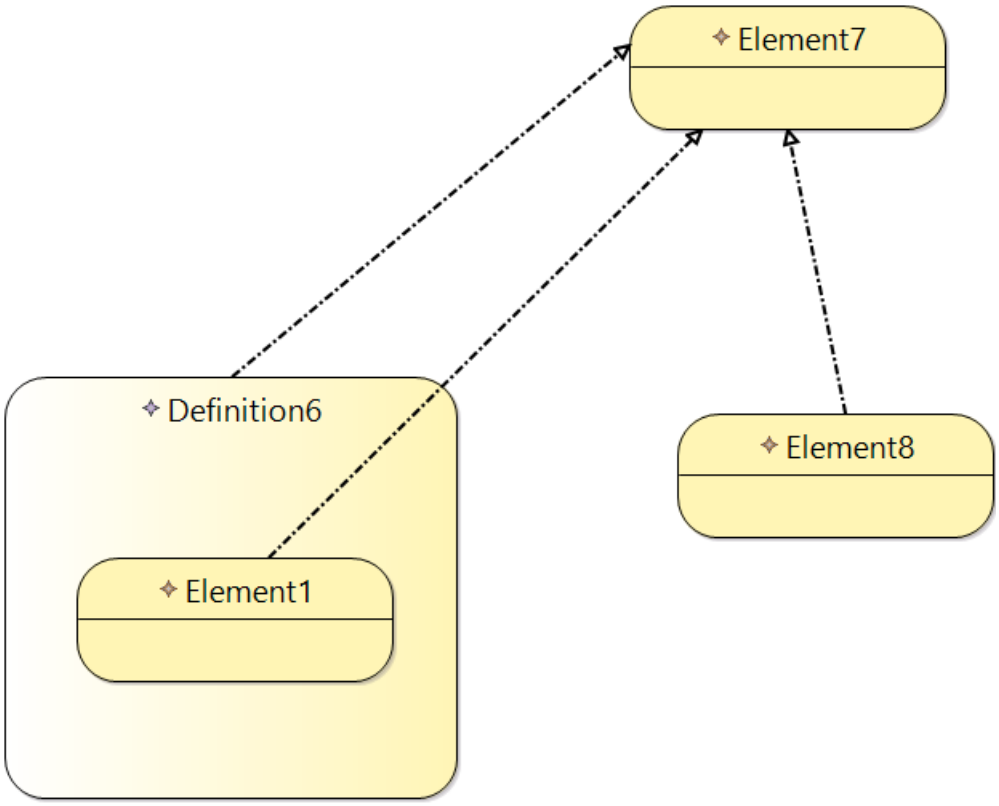

Figure 5.11: The Inheritance class syntax

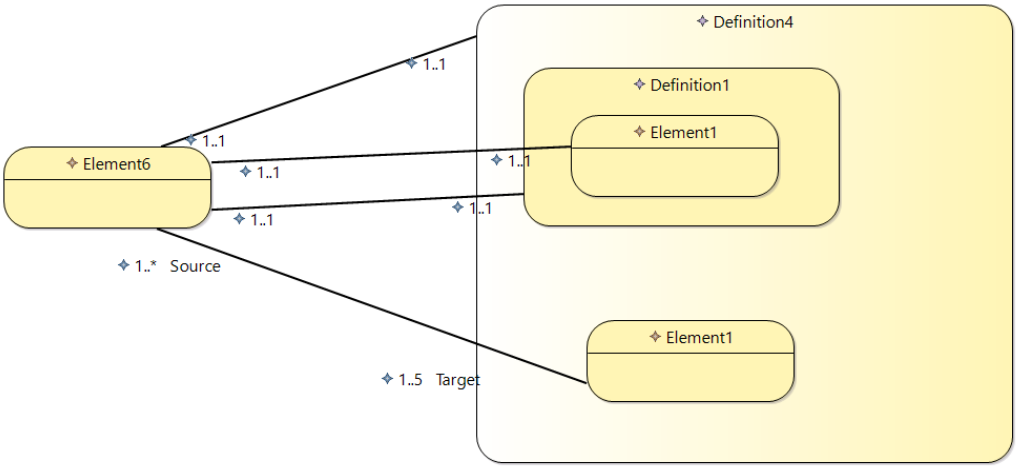

Figure 5.12: The Association class syntax

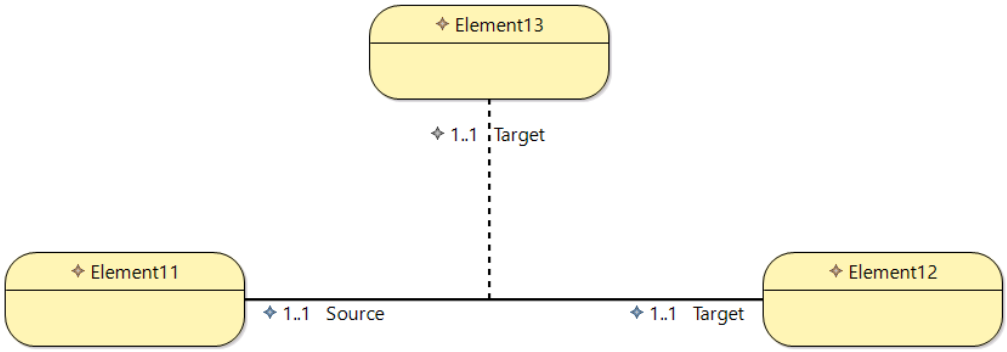

Figure 5.13: The Reify Association class syntax

5.13. Being able to connect edges to edges is not something that is directly supported in Sirius or Ecore and must be defined in order to be able to specify the syntax and semantics of such an action.

Finally there is the Constraint class. The syntax of this class if fairly straightforward. It has its own color and shape; red and a circle with a dashed line border. This shape is unique to the Constraint class and should therefore be extremely easy for a user to be able to understand. The only difference within the Constraint class is if it is a semantic or syntactic constraint, in which case the color of the circle will change; red for syntactic and pink for

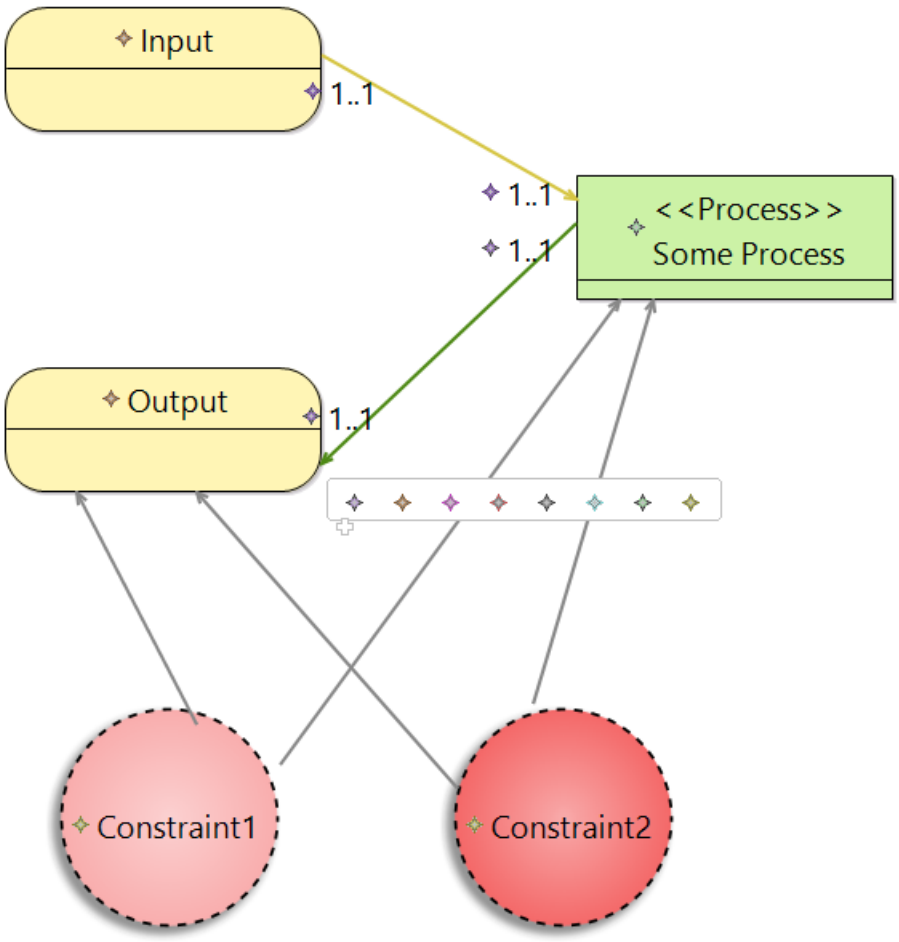

Figure 5.14: How the Constraint class looks

semantic. This condition is set by the user and can be toggled back and forth by setting the boolean attribute 'Syntactic" to either true or false.

Another attribute of the Constraint class is the description; what is the constraint? Moving forward this attribute is to hold an in-line OCL editor so that users can define their constraints formally and without the ambiguity of natural language. This will eventually also lead to the formulation of the formal assurance case that is to be generated from the WF+ metamodel that is being defined by the user.

5.3 Conventions

As the timeline for this work was limited there are some constraints on the editor that have not yet been able to be put into place. As previously shown in figure 5.9, there is the ability to draw models that contain semantic conflicts using the composition class. As a result, there are some conventions that have established to inform users so that they can make well founded models despite the lack of constraints on the syntax. While the list is relatively short at the moment, we also rely heavily on the users previous experience with modeling to apply similar conventions that are present in other modeling languages.

Convention 1: Ensure that any hierarchy built using containers or the composition edge do not have conflicting ownership.

Convention 2: Ensure that associations edges are only connected between nodes of the same type. That is Process to Process and Data to Data.

Convention3: Ensure that classes can only inherit from one other class, and do not conflict with the composition hierarchy.

Convention 4: While all the classes can have their color changed as the user desires, ensure consistency with color coding between classes of the same type.

Convention5: Input/Output edges to/from container items apply to all of the contained items as well.

Convention6: Associations between two items apply only to those items and not to any contained elements.

Chapter 6

Evaluation

In order to evaluate the usability of the tool at this current stage we aimed to satisfy the primary requirement of this tool; can we successfully build WF$^+$ metamodels. This requirement was broken down to the two requirements that were guiding the design decisions for the specifications of the editor; input/output relations between process and data, and usability. In order to test this we did both unit testing and integration testing. First we had to test to make sure that each individual node class could be created the way that we wanted them to be represented (see 5.2). After testing all of the individual node classes we had to check that we could connect them together using the Reference classes. Finally, after checking that we were able to integrate all of the classes together to a satisfactory degree, we decided to recreate the Hazard Analysis and Risk Assessment (HARA) standard, as set by ISO 26262 [18], in the editor for WF$^+$. This would allow us to test the usability of the tool, as well as further stress the unit and integration testing to make sure that we would be able to create a model that could be understood.

6.1 Creating a Hazard Analysis and Risk Assessment Metamodel

As this model is very large, we split it up into smaller sub pictures to analyze the effectiveness of the WF$^+$ editor as a tool for creating WF$^+$ metamodels. To start with, we have figure 6.2 which shows a metamodel that depicts the input to the HARA as defined within ISO 26262. We show in the meatmodel that a requirements specification data class composed of requirements, which can be further specified into functional and non-functional requirements, as well as constraints, all items that should be found within a requirement specification. This requirement specification is then associated with an item. According to ISO-26262, there needs to be an item boundary defined as part of the input to the HARA. However what this boundary actually contains/aggregates is left ambiguous and open to interpretation. Therefore when it came to our interpretation for this model, figure 6.3, we used the figure from ISO 26262 Volume 10 [19], which shows how items interact with other components of a vehicle.

On top of the figure from volume 10, we also used the definitions of item and vehicle function that are defined in ISO 26262 Volume 1 [20], along with their references. Taking these definitions, the model, and putting it together with the description of what the boundary of an item is, as defined in 5.4.2 of ISO 26262 Volume 3 [18] we ended up with the hierarchy that we have as the input to the HARA shown within figure 6.2, which we believe is a very accurate representation of the input. Thus, the requirement specification and item boundary have been successfully drawn and shown to aggregate to the work product that is the 'Item Definition', which is an input to the actual

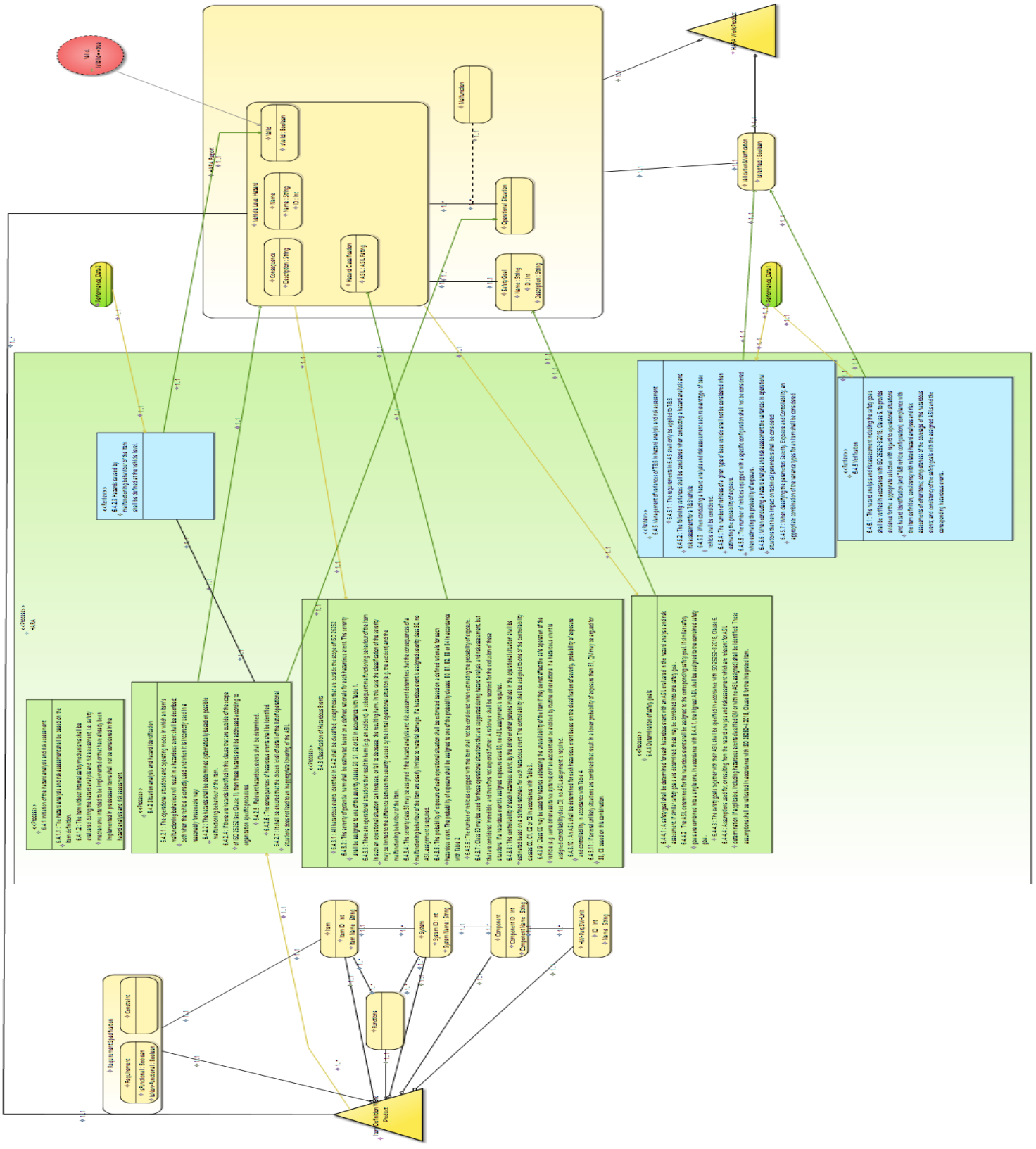

Figure 6.1: Overall view of the HARA standard set by ISO 26262, defined using the WF$^+$ methodology.

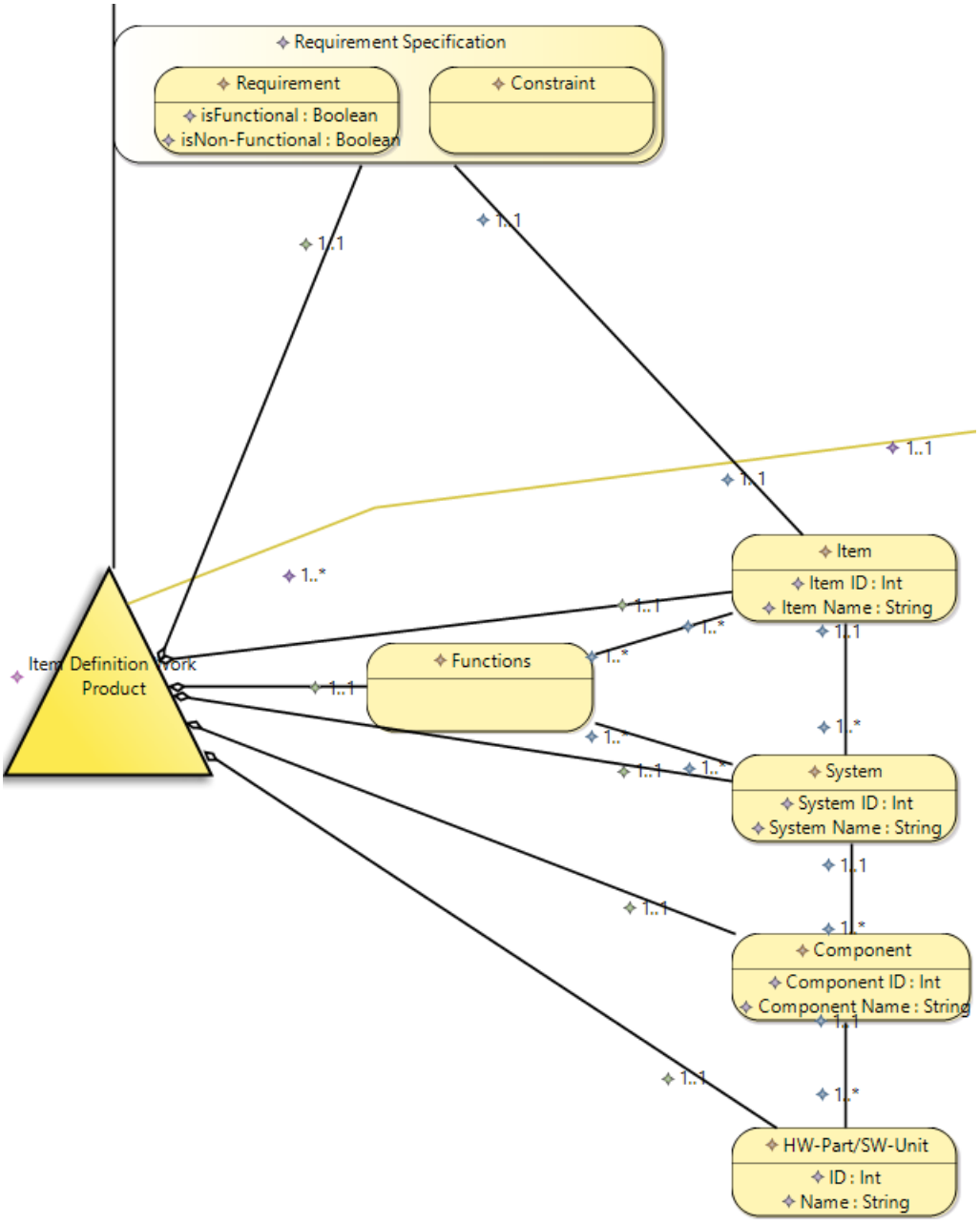

Figure 6.2: A closer look at the input to the HARA.

processes of the HARA.

Item: system (3.163) or combination of systems (3.163), to which ISO 26262 is applied, that implements a function or part of a function at the vehicle level

Vehicle Function: behaviour of the vehicle, intended by the implementation of one or more items (3.84), that is observable by the customer

Next we follow the input line from the Item Definition Work Product to the top half of the composed process that is the HARA, shown in figure 6.4. However, while creating the representation for the processes that are found in a HARA, we realized that 6.4.1 not an actual executable process so much as it is a statement that the HARA has formally begun. We left it in the metamodel but we are unsure as to how it would actually interact with the input or output data that is defined. Thus we have the input data skip and head straight towards 6.4.2, which does specify what the outputs should be. Interestingly, it also seems to define a constraint on the output, the identified hazards. Process 6.4.2.3 specifies that hazards caused by the malfunctions shall be defined at the vehicle level. To us, this sounded more like a review than an actual executable process; there should be a review of the generated outputs to ensure that they are all defined at the vehicle level so make sure that the process has been executed correctly. As a result we extracted that atomic process and treat it as a review of 6.4.2 to ensure that it is valid. Furthermore, we show that this review has performance data which would be where users could specify the requirements they may wants for the resources that are reviewing the process and its output. The output of 6.4.2 is shown in figure 6.6. This output of 6.4.2 is then used as input to 6.4.3.

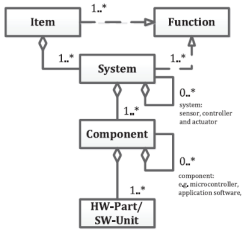

Figure 6.3: Item, system, element, component, hardware part and software unit hierarchy as defined in [19], section 4.2, pg 4

The pattern of output data of one process becoming the input data to another continues to the final process shown in figure 6.5, which shows the final executable process along with two review nodes. These review nodes are defined in ISO 26262, but according to our interpretations, rather than being processes they seem to behave more as reviews to verify the process that have come prior. Therefore we have them defined as reviews within the WF^{+} metamodel that share performance data as we assumed that it would be the same team that would perform both reviews, though this is heavily dependent on whatever the engineer/team/company wants.

Finally we have the output data that is shown in figure 6.6. The 'Valid' data class is the output of the review from figure 6.4, which also has a constraint on what its boolean attribute need to be in order for it to be valid. The 'Malfunction' data class shows how for every association between an operational situation and a correlated hazard there must be a malfunction via a reification of the association. We also show that every safety goal must be associated to one or more vehicle level hazards. Finally The 'HARA Report' and the 'Validation&Verification' classes are then aggregated to the 'HARA Work Product'.

One thing to note about this metamodel is that there is a lack of granularity as to which process is actually responsible for creating which output data. This

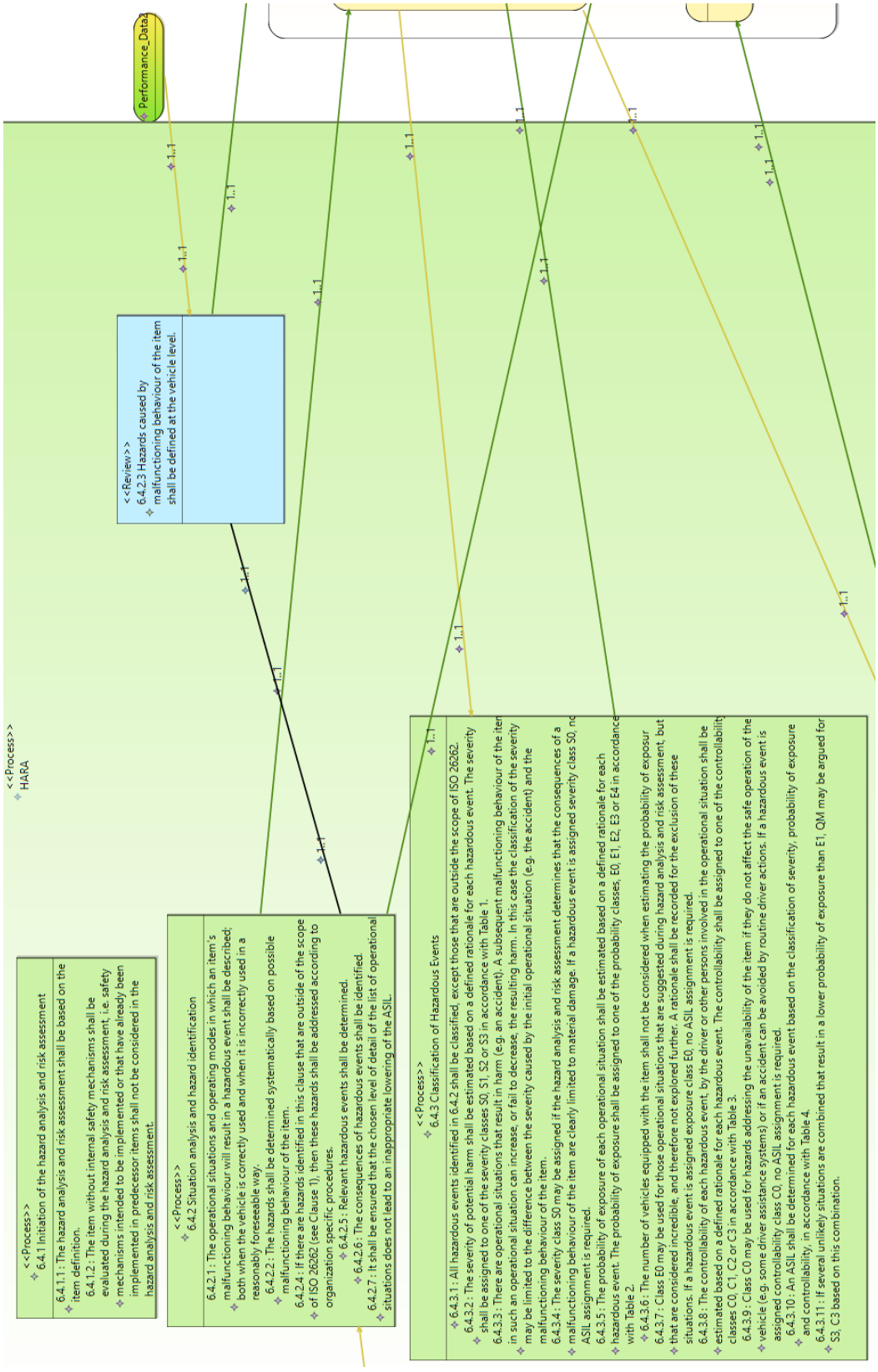

Figure 6.4: A closer look at the top half of the processes of the HARA.

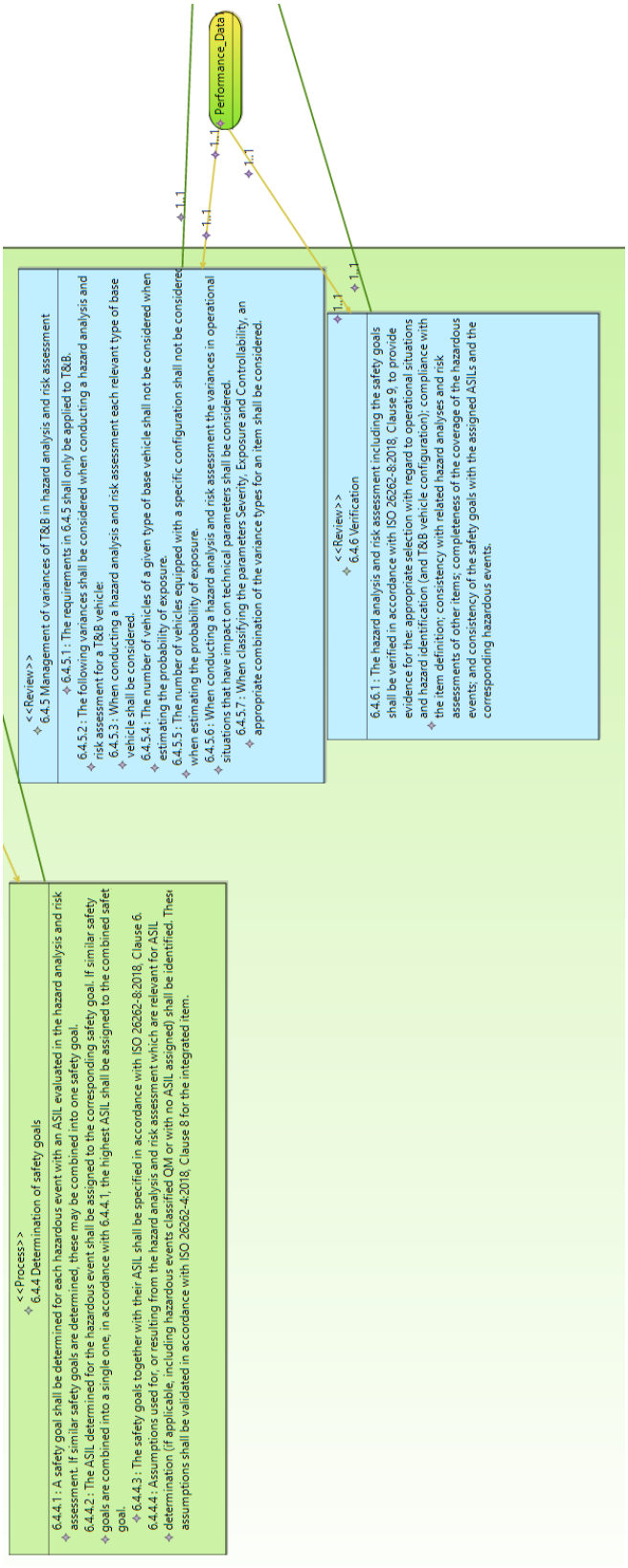

Figure 6.5: A closer look at the bottom half of the processes of the HARA.

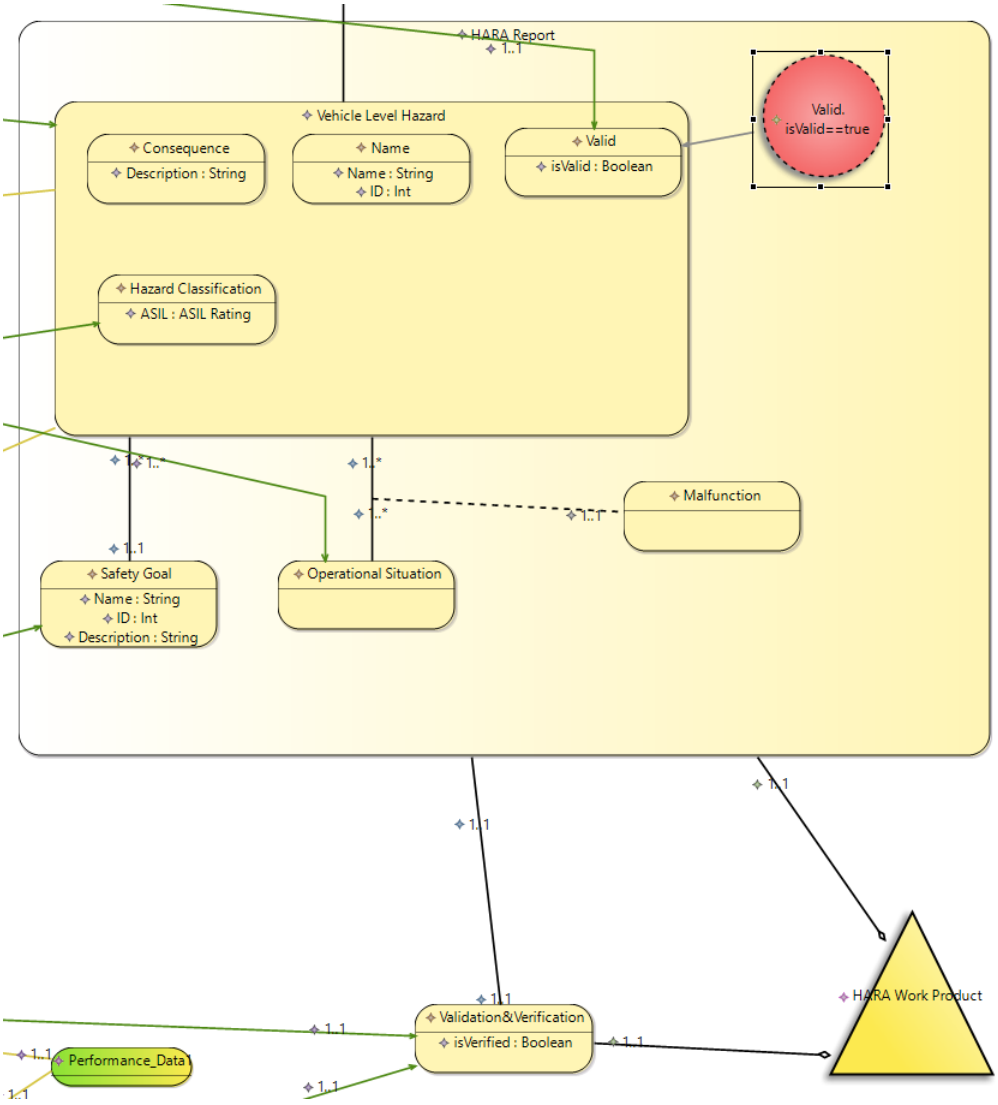

Figure 6.6: A closer look at the output of the HARA.

was intentionally done to show some of the limitations of the tool, which is the heavy reliance on the engineer that is designing the metamodel. If they have a clearer understand of the workflow they want to define, in this case a HARA, they would be able to be more specific about how the process and data classes are connected. Despite this lack of granularity in the metamodel shown in this thesis however, it is still granular enough so that it is relatively easy to understand the overall relationship between the input data, the output data, and the processes that connect them. However, one of the main point of WF$^+$ is introducing an increase to the granularity with which engineers can define workflows and the input/output relationships between data/work products, and the processes that consume and produce them. In figure 6.7, we can see a much more detailed model of the HARA process.

By separating out all of the processes into individual classes, it then becomes possible to determine specifically which sub-process is responsible for the production of which specific data node. This increase in process granularity can be seen in figures 6.8 and 6.9.

As a result of this increase in the granularity for the processes, it forces an increase in granularity for the output data as well, as shown in figure 6.10. This allows of a more complete model of the engineering process as defined in ISO 26262. It also shows more specifically which process is responsible for which output data. This increase in granularity introduces a massive increase in traceability between produced data, the processes that produce them, and the input data that is required as well, thus satisfying the claim that WF$^+$ can increase the traceability in workflow management.

As there are more processes to which input and output arrows can be attached, there is now more to the model overall. As this was foreseen however

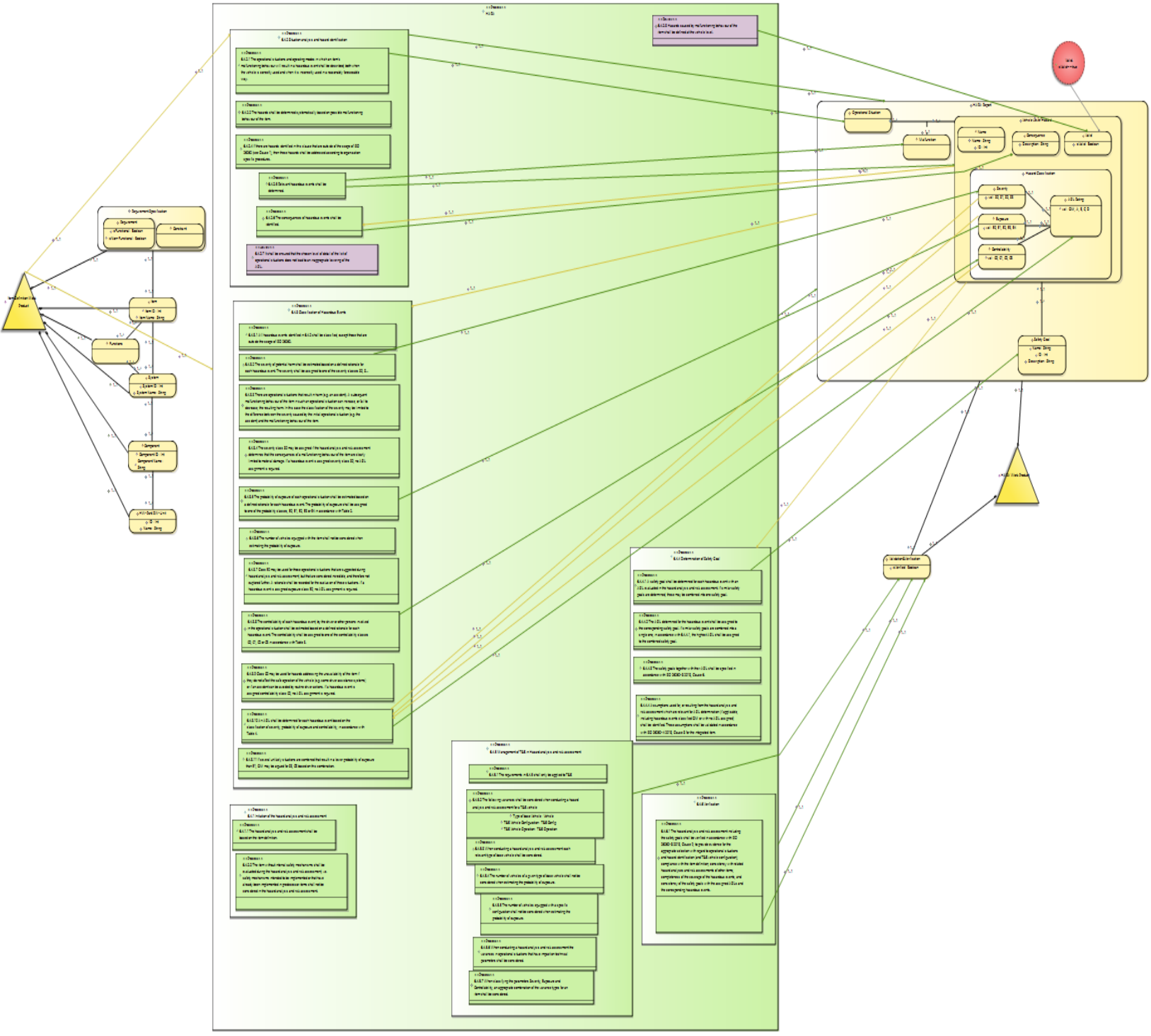

Figure 6.7: Overall view of the HARA standard set by ISO 26262, defined using the WF$^+$ methodology showcasing the increase in granularity.

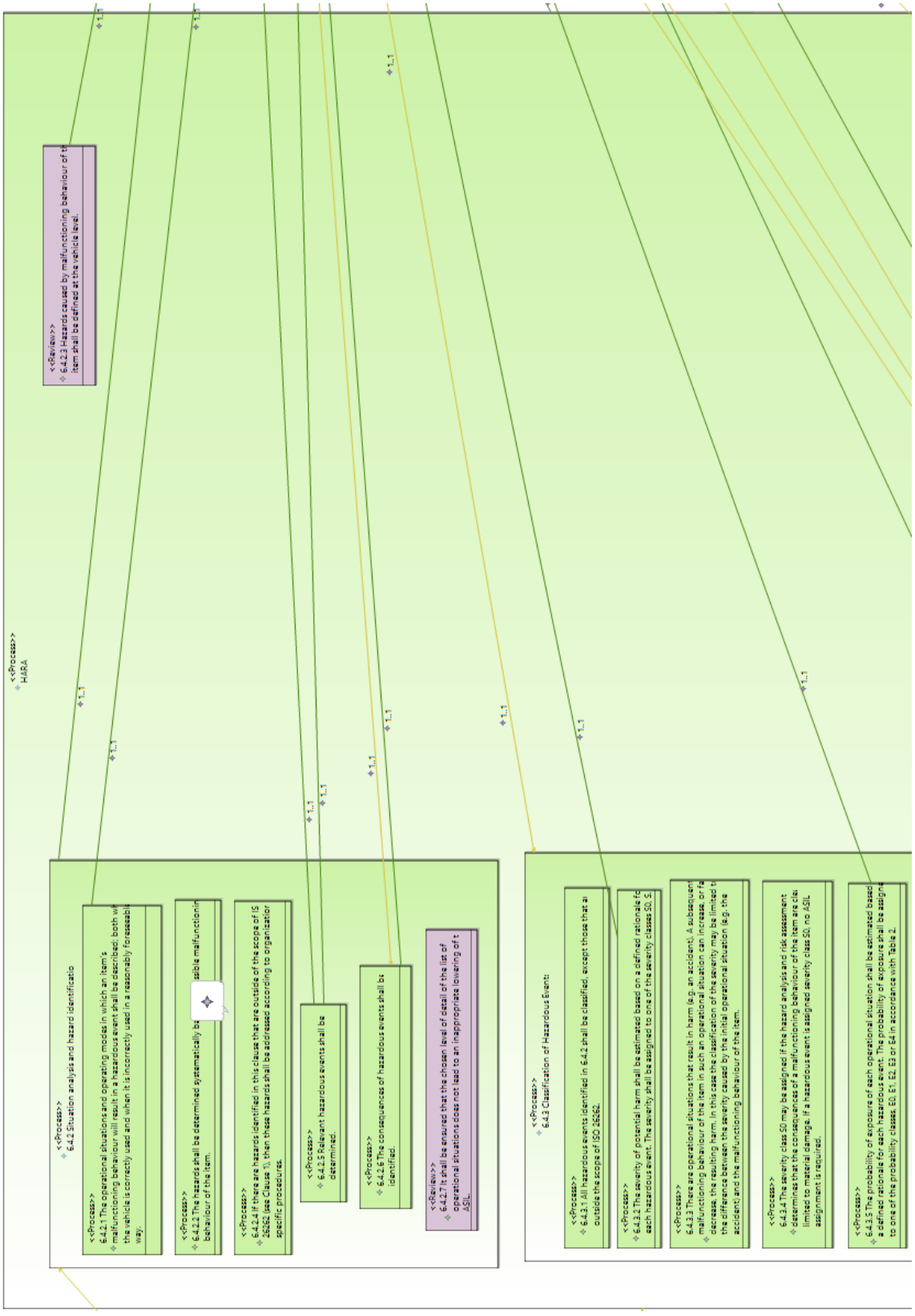

Figure 6.8: A closer look at the top half of the processes of the HARA with a more granular approach

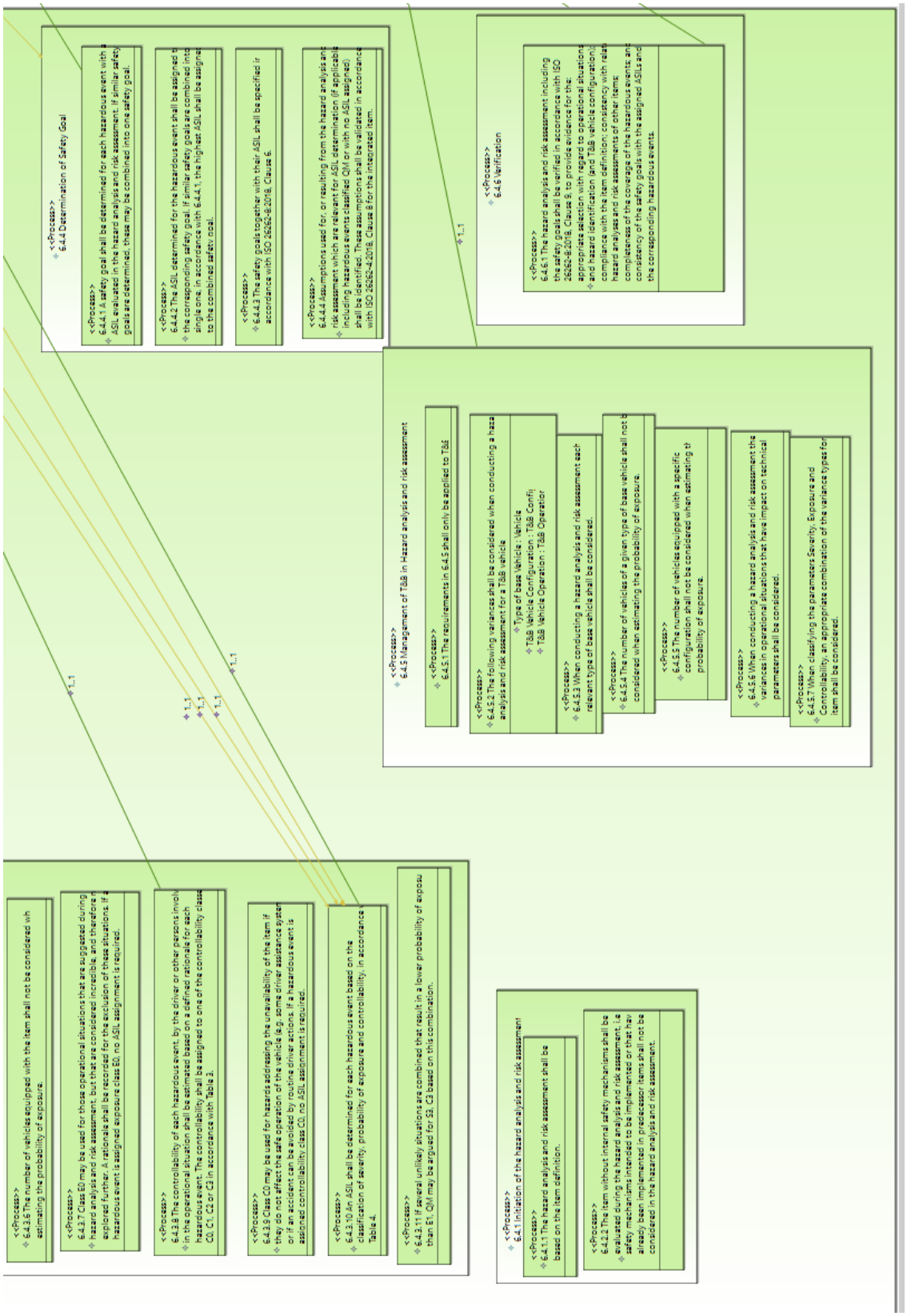

Figure 6.9: A closer look at the bottom half of the processes of the HARA with a more granular approach

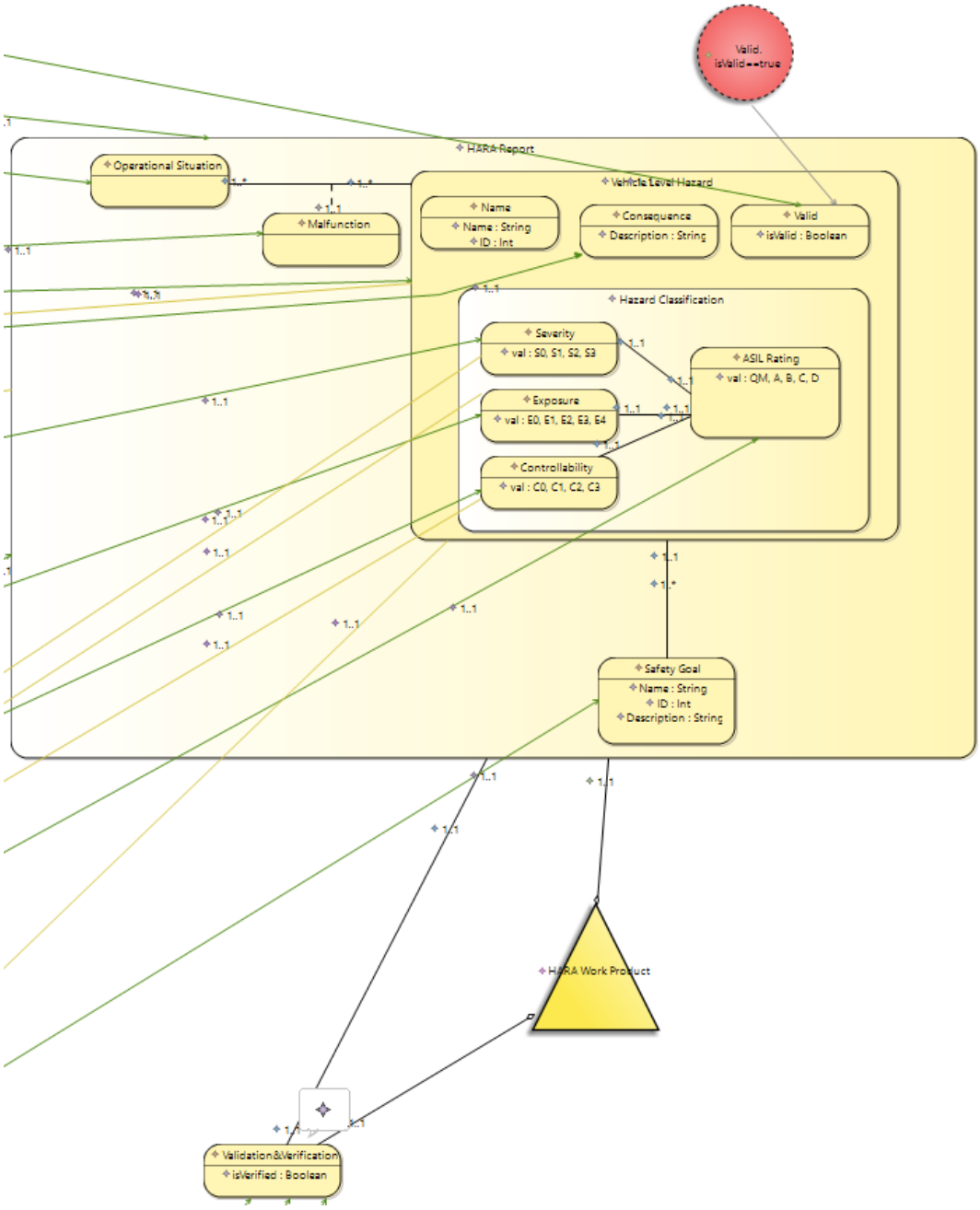

Figure 6.10: A closer look at the output of the HARA with a more granular approach

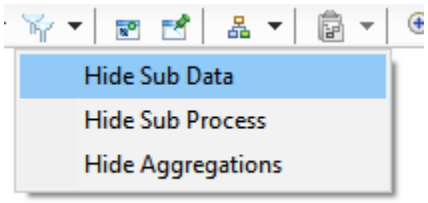

Figure 6.11: The drop down menu that we have leveraged to allow the user to select which elements they would like to hide.

we have leveraged a couple of tricks from Sirius to make the created model a little easier to digest despite the increase in the number of model elements. One of these tricks comes in the form of hide functions that allowed us to hide the sub elements to declutter the model without losing the overall workflow, as shown in figure 6.12. As users can dynamically resize all of the model elements they can then easily resize the containers to make a more compact model. The only downside is that when they want to revisit the complete model they have to again resize all of the containers and manually arrage all of the elements as they want them to be displayed. We are looking at way to improve this user experience with future iterations.

The functionality of hiding sub model elements was deemed as a necessity for the tool in order to help satisfy the requirement of usability for the tool. It becomes very difficult to understand the overall workflow when all of the smaller details are included within the model. However not having the smaller details makes the tool lose the granularity and traceability that is necessary for the future work of generating assurance. Thus the decisions to allow the user to flip between having smaller details and hiding was made in order to satisfy both conditions; ease of use and increase in traceability.

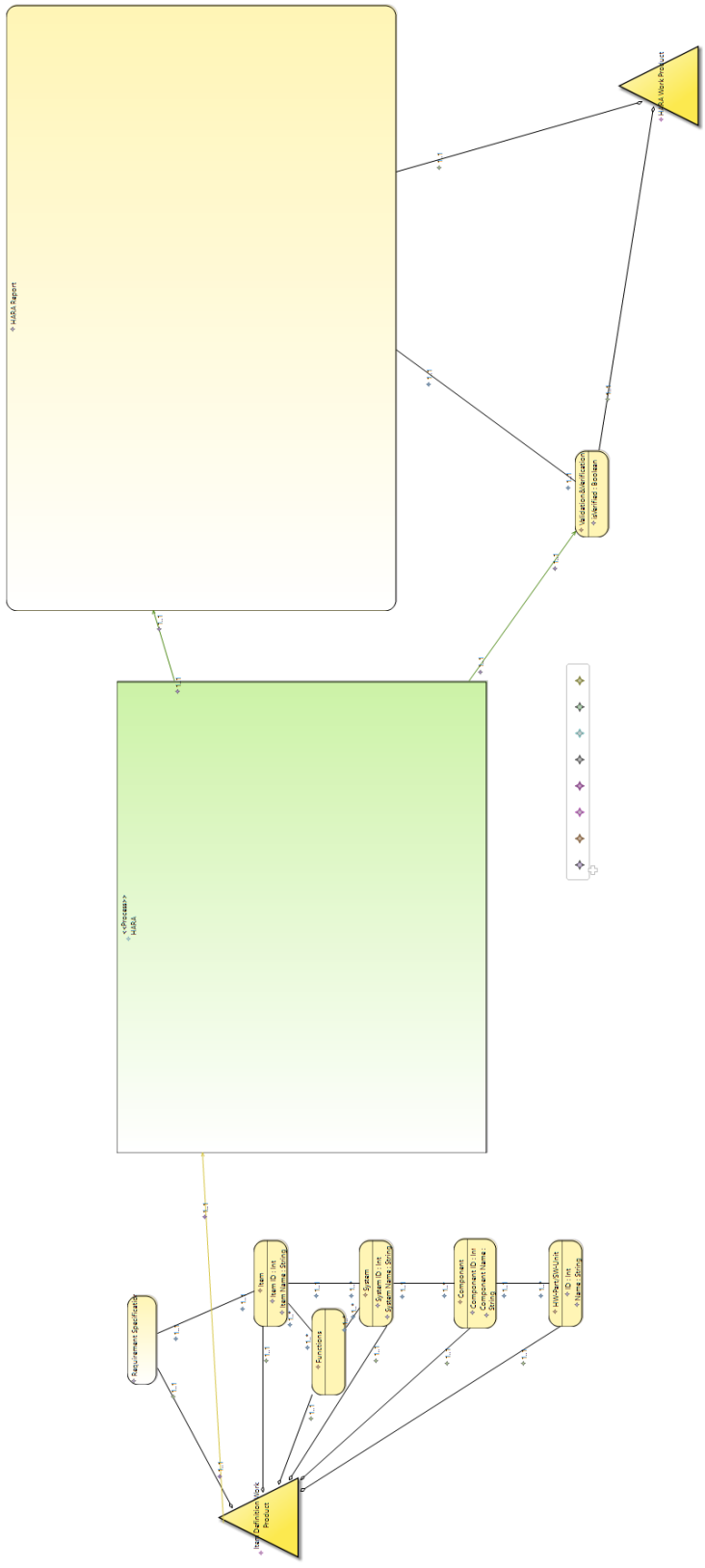

Figure 6.12: The drop down menu that we have leveraged to allow the user to select which elements they would like to hide.

6.2 Results & Future Work

We believe that with this evaluation we have shown that we have mostly met the requirement of being able to draw WF^+ metamodels with our tool. We have shown that we are able to interpret safety engineering workflows that are defined by ISO and represent them faithfully within our framework and editor without compromise. While still rough around the edges we believe that we have shown enough with this metamodel that we have satisfied the requirement of drawing WF^+ metamodels. This iteration however has not implemented all of the requirements that we generated at the start of development. Some of the requirements that we have not implemented yet are:

- Engineer wants to derive assurance from WF^+ metamodel
- Engineer wants to transform WF^+ metamodel to a GSN-type viewpoint
- Engineer wants to connect WF^+ metamodels to each other

Other requirements were also identified for this tool during the development of this iteration, such as being able to interface with other tools that are commonly used in safety engineering (Excel, Simulink). Moving forward it is highly likely the requirements that we started iteration 2 with will continue to evolve as we being iteration 3 and onward.

While changes to requirements may seem vague and difficult to predict, there are several steps that can be immediately identified to further refine this tool. The first step would be to add some more constraints on the syntax to resolve some of the semantic conflicts that have resulted from having a redundant and crowded abstract and concrete syntax. This may take the shape of evaluating the ecore metamodel for the tool again to identify any

possible refinements first, and then apply some OCL/AQL constraints to the editor.

Another step in further refining this tool would be formalizing the relationships between review nodes and constraint nodes to be able to actually generate assurance cases from a WF^+ meteamodel. This generated assurance case could then be evaluated to check if a WF^+ metamodel does indeed meet the requirement for generating assurance cases. Finally what is absolutely required of this tool is also a method of instantiating the created WF^+ metamodel to an executable model that can be applied to specific products as opposed to a product family. Being able to do so would be able to show the true power of what WF^+ is trying to achieve, which is a repeatable, formal process for generating assurance cases across a product family, rather than having to be redone from scratch for every new product.

Another requirement for this tool is a method for handling incremental assurance changes on a product. That is, if a product already has an assurance case, but a change is made, being able to automate some portions of the impact analysis of the change and aid in updating the assurance case would introduce huge cost savings for companies when they have regular iterations on existing products. While this is not a specific issue as it also requires the support of theory as well, it is something that will need to be researched moving forward.

Finally, within this thesis there was a clear bias towards the automotive industry as the example to demonstrate the tool was created based on ISO-26262, moving forward there is no reason why WF^+ cannot be used in other industries for the development of Assurance Cases.

Chapter 7

Conclusion

WF$^+$ delivers a new solution to the development of Assurance Cases through the use of thorough documentation of the workflows that the engineers must go through in order to develop a product. Through this definition, it becomes much easier to develop the workflow with safety as the primary target. WF$^+$ aims to satisfy this goal by allowing users to define their engineering workflows and eventually derive the safety assurance claims directly from their processes. While WF$^+$ is still in its fledgeling stage, our tool has shown ample ability to define workflows that are in line with ISO 26262 regulations, thus demonstrating the robustness of the syntax. Some basic semantics are also enabled due to the way the metamodel is defined. We have also shown how it is still open to modification without changing its core design, thus demonstrating how it will be robust with respect to requirement changes and updates in the future. Some shortcomings are the fact that it does not have support for other engineering applications at this time (such as DOORS, MEDINI, SIMULINK etc.) though support for these applications and more will be tackled in future iterations. Due to development timelines, the ability to actually derive the

Assurance Case claims from constraints has not been fully implemented yet. This will be done in future work on the tool. In conclusion, WF$^+$ is a tool that is still young in its development lifecycle, but shows immense promise to enable a much more structured approach to modeling engineering workflows, the development of Assurance Cases, and the flexibility to model many different types of workflows that adhere to different types of standards. We have shown that the foundations for a tool that will be relatively easy to maintain are here, as well as the implementation of the concepts of WF$^+$. Moving forward the interfacing with third party software will be addresses, as will the ability to generate a GSN-type of viewpoint. Finally the ability to derive assurance claims will allow for the generation of Assurance Cases just from having a well defined workflow, thus providing an excellent tool for engineers to take advantage of when developing safety-critical products. My contributions to the WF$^+$ research project include:

- The generation of the requirements for the tool
- The creation of the Ecore metamodels for every iteration of the tool
- The specification of the concrete syntax for the editor for every iteration of the tool
- The creation of a tool that allows for the development of WF$^+$ metamodels

These contributions are significant as it provides a true start for WF$^+$ beyond just a theoretical approach to safety engineering and assurance cases. As it is impossible to develop in a model driven environment without tool support, by starting the process of creating a tool for WF$^+$ I have managed to give it a

platform so that it can continue to develop beyond just theory into practical, and hopefully industrial usage.

Bibliography

[1] Nicholas Annable. "A Model-Based Approach to Formal Assurance Cases". In: (2020). URL: http://hdl.handle.net/11375/25343 (cit. on pp. 1, 9, 10).

[2] Tim Kelly and Rob Weaver. "The Goal Structuring Notation – A Safety Argument Notation". In: (2004). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.66.5597&rep=rep1&type=pdf (cit. on p. 5).

[3] Kateryna Netkachiva, Netkachov Netkachov Oleksandr, and Robin Bloomfield. "Tool Support for Assurance Case Building Blocks". In: *Koornneef F., van Gulijk C. (eds) Computer Safety, Reliability, and Security. SAFECOMP 2014. Lecture Notes in Computer Science, vol* 9338 (2014), pp. 63–71. URL: https://doi.org/10.1007/978-3-319-24249-1_6 (cit. on p. 6).

[4] Ewen Denny and Ganesh Pai. "Tool support for assurance case development". In: *Autom Softw Eng* 25 (2018), pp. 435–499. URL: https://doi.org/10.1007/s10515-017-0230-5 (cit. on p. 6).

[5] Mike Maksimov et al. “Two Decades of Assurance Case Tools A Survey”. In: *Computer Safety, Reliability, and Security* 2 (2018), pp. 49–59 (cit. on pp. 6, 7).

[6] Ran Wei et al. “Model based system assurance using the structured assurance case metamodel”. In: *Journal of Systems and Software* 154 (2019), pp. 211–233. ISSN: 0164-1212. URL: https://www.sciencedirect.com/science/article/pii/S0164121219301062 (cit. on p. 8).

[7] Sahar Kokaly et al. “A Model Management Approach for Assurance Case Reuse Due to System Evolution”. In: *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. MODELS ’16. Saint-malo, France: Association for Computing Machinery, 2016, 196–206. ISBN: 9781450343213. URL: https://doi.org/10.1145/2976767.2976792 (cit. on p. 11).

[8] D. Moody. “The “Physics” of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering”. In: *IEEE Transactions on Software Engineering* 35.6 (2009), pp. 756–779 (cit. on p. 12).

[9] Harald Störrle and Andrew Fish. “Towards an Operationalization of the ”Physics of Notations” for the Analysis of Visual Languages”. In: Sept. 2013. ISBN: 978-3-642-41532-6 (cit. on p. 12).

[10] Nicolas Genon, Patrick Heymans, and Daniel Amyot. “Analysing the Cognitive Effectiveness of the BPMN 2.0 Visual Notation”. In: vol. 6563. Oct. 2010, pp. 377–396. ISBN: 978-3-642-19439-9 (cit. on p. 12).

[11] K. Teng, S. A. Thekdi, and J. H. Lambert. “Risk and Safety Program Performance Evaluation and Business Process Modeling”. In: *IEEE Trans-*

actions on Systems, Man, and Cybernetics - Part A: Systems and Humans 42.6 (2012), pp. 1504–1513 (cit. on p. 13).

[12] Object Management Group. "Software and Systems Process Engineering Meta-Model Specification". In: (2008). URL: https://www.omg.org/spec/SPEM/2.0/About-SPEM/ (cit. on p. 13).

[13] *Eclipse Modeling Framework*. URL: https://www.eclipse.org/modeling/emf/ (cit. on p. 14).

[14] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack. "The Epsilon Object Language". In: *Rensink A., Warmer J. (eds) Model Driven Architecture – Foundations and Applications. ECMDA-FA 2006. Lecture Notes in Computer Science, vol* 4066 (2006). URL: https://doi.org/10.1007/11787044_11 (cit. on p. 14).

[15] V. Viyović, M. Maksimović, and B. Perisić. "Sirius: A rapid development of DSM graphical editor". In: *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*. 2014, pp. 233–238 (cit. on p. 15).

[16] Juan de Lara and Esther Guerra. "Deep Meta-modelling with MetaDepth". In: *Objects, Models, Components, Patterns*. Ed. by Jan Vitek. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–20. ISBN: 978-3-642-13953-6 (cit. on p. 16).

[17] Marce Brambilla, Jordi Cabot, and Manuel Wimmer. "Model-Driven Software Engineering in Practice, Second Edition". In: (). URL: https://www.morganclaypool.com/doi/10.2200/S00751ED2V01Y201701SWE004 (cit. on p. 16).

[18] ISO. "ISO 26262: Functional Safety - Concept Phase". In: 3 (2018) (cit. on pp. 79, 80).

[19] ISO. "ISO 26262: Functional Safety - Guidelines on ISO 26262". In: 10 (2018) (cit. on pp. 80, 84).

[20] ISO. "ISO 26262: Functional Safety - Vocabulary". In: 1 (2018) (cit. on p. 80).