A Model-Based Approach to Formal Assurance Cases

A Model-Based Approach to Formal Assurance Cases

By

Nicholas Annable, B.Eng Mechatronics

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements for the Degree

Master of Applied Science

McMaster University

Master of Applied Science (March, 2020) McMaster University

(Software Engineering) Hamilton, Ontario

Title:	A Model-Based Approach to Formal Assurance Cases
Author:	Nicholas Annable, B.Eng Mechatronics (McMaster University)
Supervisors:	Alan Wassyng, Mark Lawford
Number of Pages:	vii, 92

Abstract

The rapidly increasing complexity of safety-critical embedded systems has been the cause of difficulty in assuring the safety of safety-critical embedded systems and managing their documentation. More specifically, current approaches to safety assurance are struggling to keep up with the complex relationships between the ever growing number of components and the sheer amount of code underlying safety-critical embedded systems such as road vehicles. We believe that an approach to safety assurance able to cope with this complexity must: i) have sound mathematical foundations on which safety assurance can be built; and ii) provide a formal framework with precisely defined semantics in which the assurance can be represented. In doing this, assurance can be made less ad-hoc, more precise and more repeatable. Sound mathematical foundations also facilitate the creation of tools that automate many aspects of assurance, which will be invaluable in coping with the complexity of modern-day and future embedded systems. The model-based framework that achieves this is Workflow^{+}. This framework is rigorous, developed on proven notations from model-based methodologies, comprehensively integrates assurance within the development activities, and provides the basis for more formal assurance cases.

Acknowledgments

Thank you to my supervisors Dr. Alan Wassyng and Dr. Mark Lawford for their support and guidance along the way.

A big thank you to Zinovy Diskin. This work would not have been possible without him. I am extremely grateful for the countless hours he spent working with me on and teaching me about topics related to my research. He challenged me to get things done right, no matter how long it took or how tight the deadline was, and was always there working alongside me. We often worked tirelessly to get things done, and in that time I learned a lot about what is possible with enough dedication and resolve, for which I am grateful.

Thank you to my examiners Richard Paige and Franya Franek for showing interest in my research and taking the time to examine my work.

Thank you to Joseph D'Ambrosio, Galen Ressler, Sigrid Wagner, Sahar Kokaly, Lucian Patcas and Ramesh S for providing invaluable advice and feedback on my work from an industry perspective.

Finally, thank you to my family and friends for their unwavering support and for tolerating the many nights and weekends I spent working on my research.

Contents

List of Figures

Declaration of Academic Achievement

My contribution has been mainly to the development of the Workflow$^+$framework. This has included creating and documenting detailed examples as part of producing deliverables for a research project with an industrial partner, and to be used in technical reports. I have worked closely with our industrial partner to understand their needs and ensure Workflow$^+$ addresses those needs. I have also contributed by developing explanations and detailed figures that form the basis of a recent publication on Workflow$^+$ and also aid in introducing Workflow$^+$ to practitioners and researchers who are new to this form of modelling.

Chapter 1

Introduction

1.1 Current Approaches

A common approach to certifying safety-critical systems is to produce an assurance case arguing the safety of some particular system of interest that a regulator can then review [1]. Popular techniques are to use Goal Structuring Notation (GSN) or Claims Arguments Evidence (CAE) diagrams to structure the argument for a systems safety. The main objective of these techniques is to provide a systematic, structured way to use evidence to argue the safety of a system. The idea behind this is to bring scientific rigour into the realm of system safety, thus improving the safety of systems by requiring objective arguments supported by evidence that can be assessed and critiqued by experts before safety-critical systems are approved for use.

The arguments in assurance cases can generally be divided into two groups: process-focused and product-focused arguments [2]. Process-focused arguments focus on demonstrating that certain processes have been properly executed by qualified people while building the system of interest. The idea is

that building a system by following processes and procedures known to produce safe systems can give confidence in and be used to argue the safety of a system. Product-focused arguments focus on demonstrating that the system of interest has certain properties desirable for safety. The idea is that the satisfaction of all properties P desirable for safety gives confidence in and can be used to argue the safety of a system. It is common to see both approaches used in assurance cases.

1.2 Why do we Need a Formal Approach?

Unfortunately, current approaches to building assurance cases fall short of their goal to provide a way to bring rigour into the world of system safety. The formal foundations of current methods for reasoning in assurance cases are largely unclear, leading to questions about the effectiveness of such approaches, as explained in [2]. This leaves regulators with the task of assessing the soundness of methods of reasoning on a case-by-case basis. Ideally, there should be a fixed set of established principles (both product-focused and process-focused) which can be used to form arguments, allowing regulators to focus on ensuring these principles were respected rather than identifying the underlying principles themselves.

Another drawback of current approaches is that there is a lack of well-defined meaning for the building blocks of such assurance cases. This becomes clear when looking at different assurance cases built by different teams. Each team will have assigned their own ad-hoc meaning to each diagrammatic element used in the assurance case, potentially resulting in drastically different structures of argumentation. Additionally, in looking at assurance cases made

using current approaches it can be seen that this lack of well-defined meaning leads to a great deal of information being left implicit; as teams create and become accustomed to their own ad-hoc methodologies, information that seems obvious to them (but is not to someone from the outside) tends to be omitted. This lack of well-defined meaning further increases the burden on regulators reviewing assurance cases, again shifting the focus from the content of the assurance case to understanding its structure in the first place. This is discussed in more detail in [3].

Together, these issues undermine a regulators ability to review assurance cases by requiring them to either decipher assurance cases on their own and introduce subjectivity to the reviewing process or work with developers to understand the assurance case, introducing confirmation bias to the reviewing process. Moreover, when each assurance case is structured differently than the last, it is difficult to gain expertise in evaluating assurance cases, and identifying flaws with safety implications is much more difficult. We believe this is largely the result of a lack of formal foundations and that a more formal approach could address these issues.

Additionally, a lack of consistency between assurance cases makes it difficult to produce tools to support their development. We believe that introducing formal foundations will improve consistency between assurance cases and thus enable effective tooling to be built.

1.3 How to Make Assurance More Formal

It is not possible to formally demonstrate the safety of a system. When it comes to process-focused approaches, the nondeterministic nature of human

processing makes it impossible to formally demonstrate that a process was properly executed. The most that can be done is to review the documentation of execution and the outputs of processes in the context of its input to make a judgement as to whether or not the process was properly executed. This results in a rather indirect and thus informal proof that the processes were in fact well executed. Even if it were possible to formally demonstrate that processes were properly executed, the reasoning as to why those processes will lead to a safe system when properly executed is based on experience which cannot be formalised. These issues prevent safety from being formally demonstrated by any process-focused arguments.

Formal demonstration of safety is not any easier for product-focused approaches. The infinite number of interactions between a system and its environment make it impossible to formally demonstrate that safety-related properties will always hold. Even if this were formally possible, the exact reasons as to why the satisfaction of these properties gives confidence in the safety of a system are based on experience and expert opinion that cannot be formalised. Similarly to the process-focused situation, these issues prevent safety from being formally demonstrated by any product-focused arguments.

Although neither product- or process-focused approaches can formally prove safety, the major issues with assurance cases (discussed in 1.2) can be addressed by:

1) Providing sound mathematical foundations on which arguments for safety assurance can be built

2) Providing a formal framework with precisely defined semantics in which the arguments from a) can be precisely represented

We believe that in doing this, we can directly improve the understandability of and consistency between assurance cases, allowing regulators to gain expertise in making objective assessments on the safety of the product being considered. We also believe that providing a framework with precisely defined semantics and introducing regularity into the structure of assurance cases will allow for more powerful tools to be built.

1.4 Introducing Workflow$^+$

We propose the Workflow$^+$ framework as a mathematical foundation that enables a rigorous model-driven approach to the development of assurance cases for embedded systems. Model-based development of complex embedded systems has proven to be effective, in part by enabling more detailed analysis and automated management of design artefacts through model management (a study on the use of MBD in industry can be found in [4]). We believe it only makes sense to utilize these model-based approaches for safety assurance for the same reasons. For the reasons mentioned in 1.2 we believe that the improved traceability, the well-defined approach, the improved detail and potential for automation inherent in the model-based Workflow$^+$ approach will pave the way for more rigorous safety assurance. Additionally, the Workflow$^+$ approach creates an opportunity for the safety of systems to be planned for ahead of development, as well as an opportunity to better integrate safety and development.

The Workflow$^+$ framework allows for the modelling of assurance cases by providing mechanisms to model all aspects of Safety Engineering Processes (SEPs) and assurance cases, including processes, data, verification and vali-

dation(V&V), detailed traceability and argumentation. In doing this, we can capture both process-focused and product-focused aspects of safety assurance and present them in a fully traceable and unified way. The "plus" in Workflow$^+$ refers to a special feature of Workflow$^+$ that allows for complex data flow and process flow to be captured in one unified model, rather than having them disjoint.

1.5 Structure of the Thesis

The remaining sections of the thesis are structured as follows: First, we describe Workflow$^+$ in an informal and intuitive way, along with relatable examples to demonstrate its core mechanisms and how they are used in Section 3. Next, we describe the Workflow$^+$ approach to safety assurance more formally in Section 4.1. With the approach described both formally and informally, we proceed by providing a more relevant example of how Workflow$^+$ can be used to model processes used in the automotive industry by creating Workflow$^+$ metamodels of the ISO 26262 standard ([5]) in Section 4.3. Finally, the advantages of the Workflow$^+$ approach are discussed in Section 5.

1.6 Contribution

The work presented in this thesis is heavily based on the work done in a collaborative research project between the McMaster Centre for Software Certification (McSCert) and an industrial partner. While the idea of Workflow$^+$ and its mathematical foundations are not my own, I played a critical role in its development by working with the McSCert team to shape Workflow$^+$ into

its current form, which we believe is very promising (See Section 6).

Broadly speaking, the development of Workflow$^+$ aimed to create a framework that is a marriage of formal mathematical modelling and software engineering principles. My main role was to live in between these two areas, working with experts on both sides to ensure that the Workflow$^+$ framework met the needs of a model-based framework for safety cases. I worked closely with experts on both sides to figure out how to shape the Workflow$^+$ framework to capture the safety-related software engineering principles.

Additionally, I was heavily involved in the development of the material used to deliver the outcomes of our work on the Workflow$^+$ framework. To be more specific, I co-authored several technical reports and a paper on Workflow$^+$ submitted for publication ([3], [6]) and was heavily involved in:

1. Producing deliverables for a research project with an industrial partner
2. Conceiving of and producing examples to illustrate Workflow$^+$ to stakeholders
3. Working with our industrial partner to understand their needs and ensure they are met

Finally, this thesis is a summary of much of the work done by McSCert in developing Workflow$^+$ to date, and is intended to serve as an introduction to the Workflow$^+$ framework that can be provided to those who are interested in understanding the framework. Thus, this thesis itself serves as another contribution to the research project.

Chapter 2

Related Work

GSN/CAE approaches to safety assurance (see [7], [8]) were inspirational sources due to their focus on decomposition and the hierarchical structuring of assurance cases, along with their use of development artefacts (i.e. evidence) as support for arguments of a systems safety, both of which are cornerstones of the Workflow$^+$ approach to assurance.

There is a large body of work related to assurance case development. [9] and [10] provide good descriptions of this body of work, and list many related references. [11] provides a very detailed analysis of the types of evidence that are used to support arguments in assurance cases.

Researchers have been aware of the issues we discussed regarding assurance cases for quite some time (see [2], published in 2010). There has been work towards overcoming these issues, for example through the use of assurance case templates as discussed in [12] and through exploration of applying formal methods to assurance cases [13].

[14] discusses how most tools for GSN/CAE assurance case development target manual development of safety cases, and motivates the use of tooling

built upon formal foundations to facilitate automation of the development of GSN/CAE assurance cases.

The Structured Assurance Case Metamodel (SACM) ([15]) is a standard that specifies a metamodel for assurance cases that is more expressive than GSN/CAE approaches and facilitates model-based assurance [16]. A recent example of work towards using SACM for model-based assurance is [17]. Other work related to model-based management of assurance cases includes [18] (which also discusses modelling of ISO 26262 [5]), and [19].

The Software and Systems Process Engineering Meta-Model (SPEM) Specification provides a framework for the standardized specification, development and management of software/systems engineering processes [20]. Work towards using SPEM as a basis for the automatic generation of process definitions based on the context of a process's application can be found in [21]. This idea of model-based automatic process generation has the potential to be applied to the situational refinement of process definitions in Workflow$^+$.

In contrast to the works described above, the approach presented in this thesis does not use GSN/CAE-style argumentation as a basis for assurance. The theoretical foundations of the Workflow$^+$ approach to assurance are laid out in [3] (cited many times throughout this thesis), with [22] as a precursor to these ideas.

Chapter 3

Workflow$^+$ Framework

3.1 Requirements of Workflow$^+$

A model-based approach to formal assurance cases of embedded systems requires a modelling framework that is able to capture all information relevant to safety and, using this information, develop a convincing argument for safety. More specifically, to be effective in supporting safety assurance the modelling framework must capture the following to an arbitrary level of granularity:

i) safety and design related data about the system, its operating environment (includes users), and assumptions

ii) process specifications, definitions and control flow

iii) people-related information regarding processes, such as qualifications, role, etc.

iv) input/output relationships between data and processes

v) data-to-data and data-to-process constraints

vi) reviews of data and processes to determine that they are valid

vii) argumentation over these entities explaining their contribution to safety

viii) the mechanism that facilitates building arguments over this information must allow for explicit and detailed traceability to any of the information mentioned above to ensure that all necessary information and context is referenced.

Currently, no single modelling framework is well equipped to capture all information needed for safety assurance of embedded systems. Block diagrams are well equipped to model the complex dynamic structures such as processes or control flows followed during the development of embedded systems, but are unable to adequately represent the accompanying complex data structures. On the other hand, model transformations are very well equipped to represent these complex data structures, but are extremely difficult to compose even in simple cases[23], let alone in complex (but common) situations such as control loops.

As first discussed in [22], it is possible to fill in this gap by modelling both workflow and data structure together in UML (Unified Modelling Language) class diagrams. This is exactly what was done in developing the Workflow$^+$ framework (see Figure 3.1). Class diagrams inherently allow for complex data structures with complex diagrammatic constraints (e.g. using the Object Constraint Language (OCL)) to be represented, and their extensibility allows processes and control flows to be modelled and related to data with precise traceability. Workflow$^+$ is a fully formalizable framework, but at the time of writing its formalization is still a work in progress and is left for future work. However, it is already far more rigorous, precise and comprehensive than existing

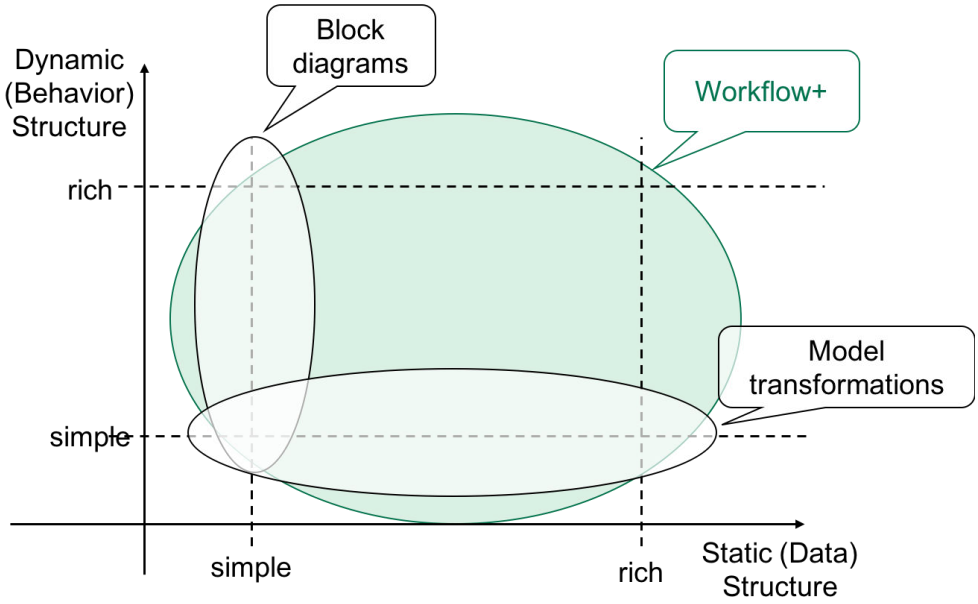

Figure 3.1: Workflow$^+$, block diagrams and model transformations

methods for developing (safety) assurance cases.

3.2 Workflow$^+$ Framework Overview

We begin by describing the model-based engineering terms used to describe the Workflow$^+$ framework. We consider a model to be a representation of some real-world data created for some particular purpose. We consider metamodels to be abstractions that describe the structure these models must have to be useful for their purpose. It is common for this relationship between models and metamodels to be described as an instance-metamodel relationship, where an instance is a model that follows all of the rules in the metamodel. For example, suppose that we want to model real-world cars and their owners for the purpose of record keeping (see Figure 3.2). We first create a (UML) metamodel (left) to describe the structure of models that are useful for record keeping, which

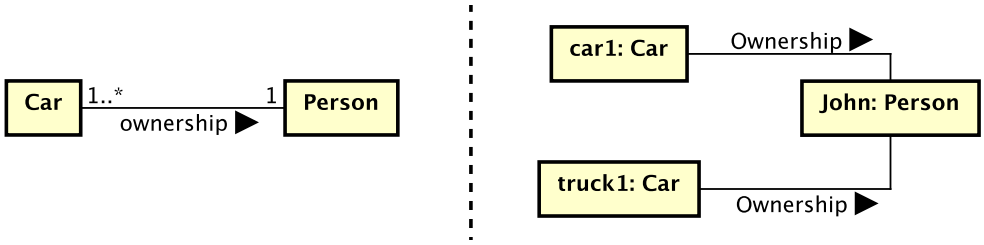

Figure 3.2: An example of a metamodel (left) and an instance (right)

specifies that each model of a car can be associated with one model of a person, and each model of a person can be associated with more than one model of a car. That is, in the records each person can own more than one car, but each car can only be owned by one person. With this metamodel, we can instantiate it to model real-world cars and their owners. One possible instance (right) shows an example of a model representing a person John and the ownership of his car and truck. To reiterate, the metamodel (left) specifies the required structure of models representing real-world people and the cars they own. Instances of this metamodel are models of real-world people and the cars they own that honour the structure defined in the metamodel. We talk about metamodels being *instantiated* to realize an instance of the metamodel. We hope that this example is simple enough to avoid notation-related confusion. The notation used is explained in more detail in the following sections.

The Workflow$^+$ framework provides a formal way to model workflows with both complex dataflow and control flow, such as those carried out by engineers during the development of safety-critical embedded systems. (The "+" in Workflow$^+$ signifies the fact that in our workflow models, the output of the workflow, sometimes referred to as a *work product*, is refined at different stages of the process.) This enables precise definitions of workflows that are to be followed, along with precise documentation of the execution of these workflows.

The Workflow$^+$ framework provides the modelling mechanisms to achieve this through the use of Workflow$^+$ *metamodels* and *instances*. Workflow$^+$ metamodels serve as definitions of workflows (including the data used). When the workflow described in a Workflow$^+$ metamodel is executed, it is instantiated to model its real-world execution (again, including the related data). Currently, Workflow$^+$ metamodels are created using profiled UML class diagrams designed to capture process definitions, data definitions, control flow involving those processes and data, data-to-data and data-to-process traceability, as well as constraints over processes and data. When the workflow defined by a Workflow$^+$ metamodel (defined using a class diagram) is executed, the defining class diagram is instantiated as an object diagram to document that execution. Finally, assurance-related arguments can be included, almost mechanically, over the content of the Workflow$^+$ metamodel. It can be shown that arguments on a metamodel also apply to its instances (more on this later).

These concepts are illustrated in Figure 3.3. A black-box Workflow$^+$ metamodel, which defines some complex workflow is shown to be executed in the real-world (note that traceability is included in process and data definitions). That execution is documented by an instance of the Workflow$^+$ metamodel defining the workflow. There is argumentation that is built over the Workflow$^+$ metamodel that also applies to instances of that metamodel. Note the differences between the lines in Figure 3.3: the solid line represents an instance-metamodel relationship, the dashed black line represents a mapping from argumentation to the metamodel, and the blue dashed line represents a derived mapping from argumentation over the metamodel to an instance of that metamodel (recall that argumentation over metamodels also applies to instances).

There are significant advantages of this approach that will become more

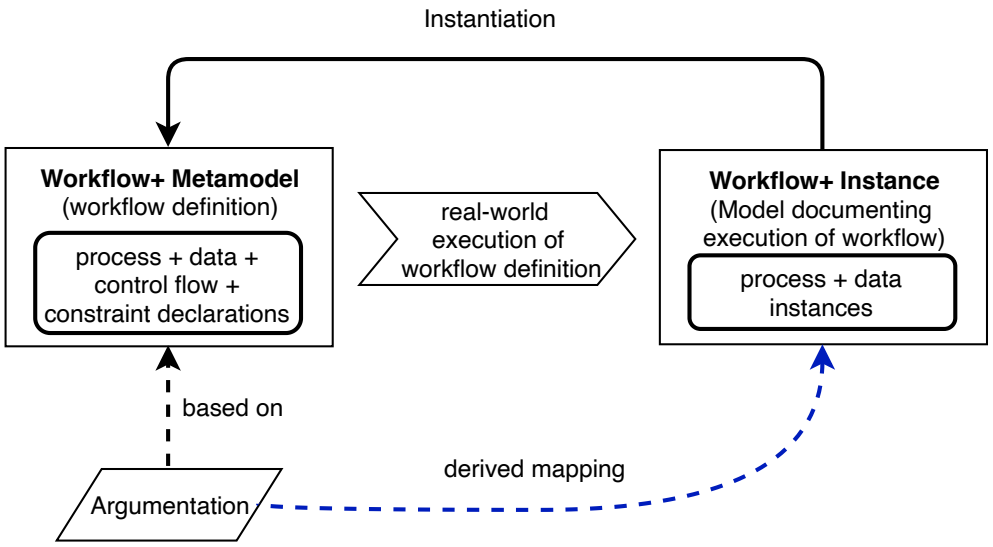

Figure 3.3: Workflow$^+$ overview: metamodels, their instances and argumentation

apparent in later sections, and were strong motivators of this methodology. Three of the most important are: i) assurance becomes less ad hoc and much more rigorous; ii) assurance is not artificially separated from development and so traceability is comprehensive and natural; and iii) the modelling paradigm lends itself naturally to generating various views of the models, and one of those views can be GSN-like, so readers who appreciate the intuitive aspects of GSN still get that view of the assurance, but it is based on this rigorous more predictable approach. Finally, Workflow$^+$ metamodels can be instantiated to document execution of the process defined by the metamodel.

The following sections provide more detail on these core mechanisms of the Workflow$^+$ framework with the help of a running example of how Workflow$^+$ can be used to model and create an assurance case for the process a bakery follows when making bread. The example was chosen to illustrate Workflow$^+$ using an example that most people will understand without getting embroiled in system/software engineering specifics. The example still has real-world com-

plexity that has to be taken into account, and we believe it has many parallels with embedded-systems development.

3.3 Workflow$^+$ Metamodels and Their Instances

Workflow$^+$ metamodels are profiled class diagrams defining workflows consisting of process definitions, data definitions, control flow, data-to-data and data-to-process traceability, and constraints over processes and data.

Data definitions specify how instances of data (i.e. models of real-world data) can be structured. Classes stereotyped as *Process* have been introduced to specify processes whose instances document their real-world executions (i.e. when a process defined by a Process class is executed, instances of that class document its real-world execution). *Process* classes take data classes and/or associations as input and output classes and/or associations. It is this ability to model processes that use and produce associations that fundamentally distinguishes Workflow$^+$ from traditional block diagrams, and the ability to add detailed control flow to complex data structures that distinguishes it from basic model transformations.

We use green boxes stereotyped *Process* to denote process classes, yellow boxes to denote data classes, green arrows to denote dataflow to/from processes, black lines to denote regular UML associations, and red text to denote OCL-style constraints. Instances of objects are denoted by name : type. Instances of processes are denoted by exeX : type, where X is a unique identifier for some particular execution of a process used to distinguish between different executions of the same process. Figure 3.4 illustrates this notation. The metamodel defines a process usefulProcess that, when executed, takes some data of

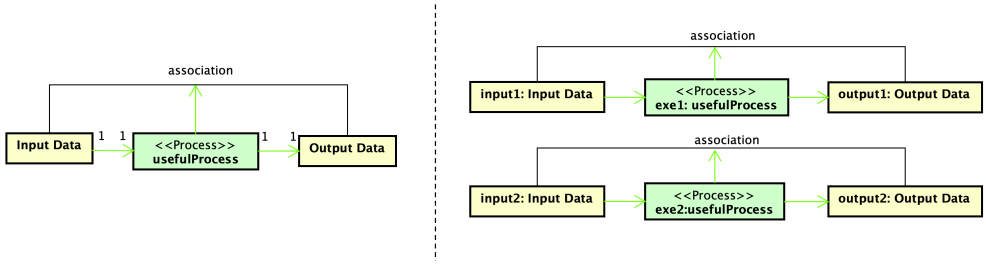

Figure 3.4: An illustration of a simple Workflow$^+$ process and two of its instances

type InputData and produces data of type OutputData, along with an association between InputData and OutputData. Two example instances documenting two hypothetical executions of this process are shown.

In general, there are two types of processes that can be modelled using Workflow$^+$: Process and Query. A Process class is used to model any processes that must be carried out by humans, whereas a Query class represents a process that is simple enough to be mechanized and automated. A simple example of a Query is using input variables and a look up table to determine an output value. No Queries appear in the running example, but they are used in the automotive Workflow$^+$ models appearing in Section 4.

When a Workflow$^+$ metamodel is executed, we instantiate the metamodel as an object diagram to document its execution. That is, when the workflow defined by the Workflow$^+$ metamodel is executed, the metamodel is instantiated as an object diagram where objects model real-world data and process executions.

There is extensive literature on this concept of instantiation, for example see [24]. In order for an instance to be considered valid, it must conform to all constraints in its metamodel.

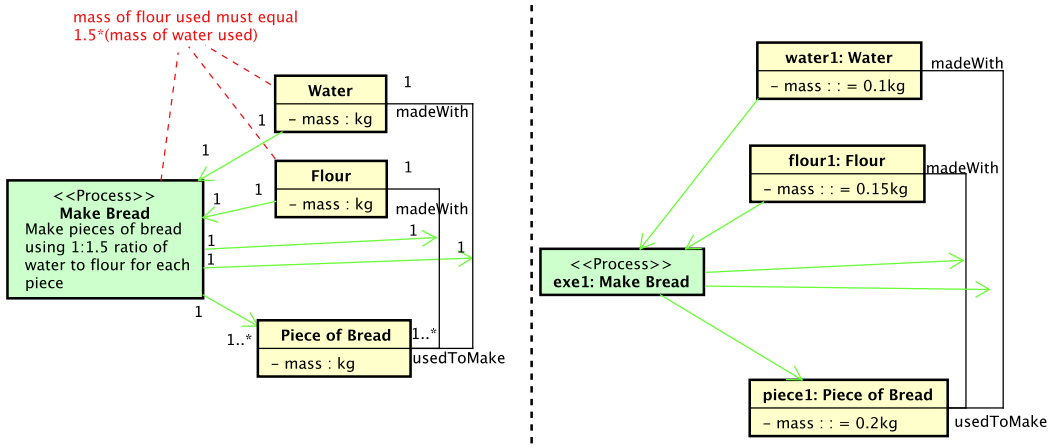

Figure 3.5: Bakery: a simple Workflow$^+$ metamodel defining the process of making bread and an instance documenting its execution

3.3.1 Running Example: Workflow$^+$ Metamodels and Their Instances

Figure 3.5 shows a very simple example of a Workflow$^+$ metamodel specifying the workflow used to make a very bland piece of bread (left) and a sample instance documenting its execution (right). In simple terms, Figure 3.5 defines a workflow where flour and water are processed to make a piece of bread and shows the documentation of a possible execution of that process. We explain the metamodel and its instance in parallel.

In the metamodel, there is one (green) process class named MakeBread defined, which is specified as *Make pieces of bread using 1:1.5 ratio of water to flour for each piece*. This process has two (yellow) data classes defined as its inputs (Water and Flour) and one defined as its output (PieceofBread), each with an attribute *mass*. When instantiated, these data classes model real-world data; for example, Water is instantiated by the object water1 : Water, which is a model representing 0.1kg of real-world water. When real-world

water and flour are used to execute MakeBread, we can model this process execution by instantiating the MakeBread process (exe1 : MakeBread) and creating input dataflow associations from the models of that real-world water and flour used (water1 : Water and flour1 : Flour respectively) to exe1 : MakeBread, and an output dataflow association from exe1 : MakeBread to a model of the real-world piece of bread produced (piece1 : PieceofBread).

The associations between data classes in the metamodel specify the relationships between those class's instantiations. For example, when instantiated the association between Water and MakeBread relates instances of PieceofBread to the instance of Water it was made with. Instances of these associations are generated during the execution of a process (hence the green arrow from MakeBread to these associations in the metamodel and in the instance).

Constraints over the metamodel are used to restrict its possible instances. Thus, when a process's definition is executed and documented by instantiating that process's metamodel, the constraints restrict how that process can be executed. We make use of two types of constraints: multiplicities and OCL-style constraints.

Multiplicities, placed at association ends, specify how many instances of classes can be associated with one another. For example, the multiplicities on the arrow from Water to MakeBread specify that exactly one instance of water (the 1 closer to Water) is input to one execution of MakeBread (the 1 closer to MakeBread). If, for example, an instance of this process was created where an instance of water is associated with two instances of MakeBread, this multiplicity constraint would be violated and the instance would be invalid. In this metamodel, most multiplicities are 1 (i.e. only a one-to-one instance relationship is allowed), with the exception of those attached to PieceofBread. The

one-to-many (1..*) constraint on the end of the association from MakeBread to PieceofBread specifies that each instance modelling the execution of MakeBread can be associated with 1..* instances of PieceofBread; that is, each real-world execution of the MakeBread process can produce one or more pieces of bread. The 1..* on the associations from Water and Flour to PieceofBread indicate that each instance of Water and Flour can be associated with 1..* instances of PieceofBread; that is, the same water and flour can be used to make more than one piece of bread (but would obviously need to be divided at some point within the execution of MakeBread).

OCL-style constraints are used to express more complex constraints over the metamodel. We refer to these as "OCL-style" because we use the concepts underlying OCL constraints, but write them in natural language to avoid the heavy syntax (and lack of readability) of OCL constraints. In this example, there is only one OCL-style constraint (shown in red text). The dashed red lines leading from this association link the constraint to the classes that it applies to. This constraint states *mass of flour used must equal 1.5*(mass of water used)*, which requires that the values of *mass* of instances of Water and Flour must be in the correct proportions.

With the components of the metamodel and its instances described, we summarize the two. The metamodel defines the process of making bread by specifying the process MakeBread, its input classes Water and Flour, its output class PieceofBread, associations between its input and output classes, and constraints over these classes/associations. The instance documents a sample execution of this process, where 0.1kg of water and 0.15kg of flour (modelled by water1 : Water, and flour1 : Flour respectively) were used to execute the MakeBread process. This execution produced a 0.2kg piece of bread (modelled

by piece1 : PieceofBread), whose model was then associated with the instances of Water and Flour used. Note that it is not a mistake that the mass of the bread is not equal to the sum of the masses of flour and water used - water weight is lost during the baking process. This instance is considered to be a valid instance because all constraints in the metamodel are satisfied.

This is just a sample instance; there are many other possible instances that will also satisfy all constraints in the metamodel. The purpose of the metamodel is to include constraints such that all of its possible instances are acceptable for the purpose of the metamodel.

3.4 Workflow$^+$ Metamodel Decomposition

Workflow$^+$ metamodels are defined using a compositional and formalizable framework, and can be decomposed into more detailed metamodels by refining data and process definitions. The concept of metamodel refinement is discussed in [25]. At the time of writing the compositionality of the Workflow$^+$ framework has yet to be formally demonstrated, and is left for future work.

Figure 3.6 illustrates a Workflow$^+$ metamodel's decomposition. The more abstract metamodel Workflow + Metamodel is less detailed, and provides less detail on the specifics of what its processes, data, etc. entail. In other words, it provides less detail on how to execute the workflow it defines. When it is refined into Workflow + Metamodel Refinement′, more detail is added to the definitions of processes, data, etc., providing more detail (or, from another perspective, more limitations) on how the workflow can be executed.

In general, processes and data in metamodels are decomposed alongside one another; after all, more detailed process or data descriptions do not add

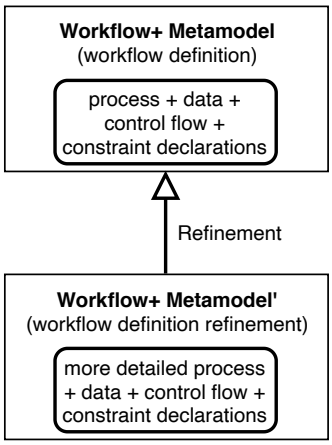

Figure 3.6: An illustration of Workflow$^+$ metamodel decomposition

much unless their counterparts are also more detailed. In theory, there is no limit on how much detail can be added in each decomposition step, or on how many times a metamodel can be decomposed. The level of detail included is ultimately up to the designer of a Workflow$^+$ metamodel, who can chose to include (or omit) as much detail as makes sense depending on its use. An added benefit of Workflow$^+$ metamodel refinement is that the compositionality of Workflow$^+$ metamodels allows different parts of a metamodel to be decomposed independently. This allows for parts of a process that are well-understood (e.g. mechanizable processes) to be decomposed to great levels of detail, while still leaving black-box processes abstractly defined (e.g. a process requiring creativity and human execution).

In our work with industrial partners, we have found that utilizing the concept of modular decomposition can be of great help when explaining and presenting metamodels. By presenting metamodels from an abstract perspective at first and modularly introducing more detail, we have been able to effectively communicate our models to industrial partners, even if they did not

have experience in metamodelling.

3.4.1 Running Example: Workflow$^+$ Metamodel Decomposition

Suppose that a bakery would like to start producing bread using the process defined in Figure 3.5. However, MakeBread as defined in Figure 3.5 is not detailed enough and thus not very useful to someone attempting to make bread, so the bakery decomposes the process to better define how their bread will be made. Figure 3.7 shows the decomposition of the metamodel in Figure 3.5.

In this decomposed metamodel, the classes Water and Flour have had their *mass* attribute set to 5kg and 7.5kg respectively. That is, instances of each are forced to have the masses specified in the metamodel. Because the actual values in instances of Figure 3.7 must comply with the constraint on Water, Flour and MakeBread in Figure 3.5, the constraint is no longer necessary. The process MakeBread has been decomposed into three subprocesses: Mix, Knead and Bake. When executed, the first of these processes, Mix, takes 5kg of water and 7.5kg of flour and produces a 12.5kg mixture of water and flour. This mixture is used for an execution of the process Knead, which kneads the mixture and results in a batch of dough consisting of 50 pieces of 0.25kg dough. Finally, this batch of dough is used by an execution of Bake, which loads the dough onto pans and bakes it for a specific amount of time at a specific temperature, producing a batch of 50 pieces of bread.

Output arrows to associations have been omitted to simplify the diagram. These associations, for example between Mixture, Flour and Water, can be assumed to always be created between the inputs and outputs of a process, in

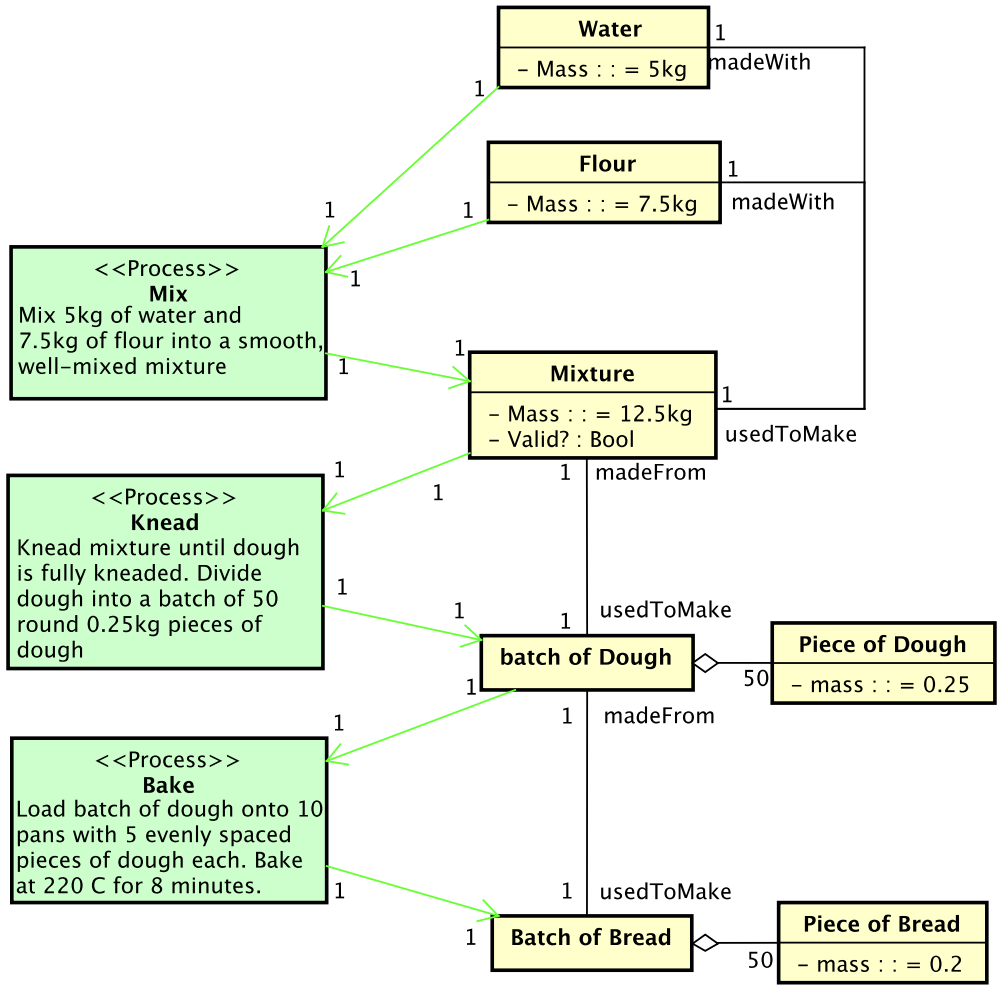

Figure 3.7: Bakery: a refinement of the process defined in Figure 3.5

this case by Mix.

The refined process of making bread in Figure 3.7 can be refined again and again until the desired level of detail is reached (e.g. different decompositions for different sizes of bread), but we stop the decomposition here. This decomposed metamodel can be instantiated to document its execution in a way that complies with the metamodel described in Figure 3.7.

3.5 Using Aspects for Cross-Cutting Concerns

It is not uncommon to see identically named *aspects* of a process occurring in multiple places within a larger, more complex process. For example, a review process needs to be carried out after a requirements process, after a design process, after a coding process, etc. The review may be a review of the process, it may be a review of the output of the process, or it may even be a review of the people who carried out the process. Rather than include multiple versions of these reviewing processes in the model, we propose to document these cross-cutting concerns similar to the way developed by the aspect-oriented programming community [26].

Workflow$^+$ metamodels can be enriched with aspect metamodels which can be developed separately and weaved into the main Workflow$^+$ metamodel. We are not aware of any aspect-oriented frameworks for metamodelling, but we will make use of the main idea of aspect weaving [27] and leave the development and formalization of such a framework for future work. More precisely, a cross-cutting concern is specified by an advice metamodel whose entry point specifies where it is to be weaved into the main metamodel. The entry point(s) of an aspect metamodel can be specified to apply to any subset of processes in the metamodel being weaved into, ranging from a single process to all of them.

It should be noted that the individual instances of the aspect metamodel can (will) be different. This fits the general idea of instances of metamodels, and is especially necessary for aspect metamodels since it is highly unlikely that the individual instances should be identical. For example, the high level descriptions of the essential components of a requirements review and design review can be specified in the metamodel and may be worded identically, but

the details of how they are implemented will surely be different.

A more formal description of aspect weaving in Workflow$^+$ can be found in [3].

3.5.1 Running Example: Using Aspects for Cross-Cutting Concerns

Suppose that the bakery would like to implement some quality-control measures to ensure their bread is of the highest quality. More specifically, they would like to keep track of the people involved in each process and add reviews by experienced bakers to each stage in their process. Both of these are typical cross-cutting concerns for a process, where very similar (if not identical) additions are added to many stages in a process.

Figure 3.8 shows an example of a simple advice metamodel that implements the needs of the bakery's bread making process. In this advice metamodel, a review process is specified where an experienced baker looks at the bakers involved along with the input and output data of a process, and makes a determination as to whether or not the output is valid (i.e. acceptable) in the context of the input data. The red constraint T on the value of *Valid?* requires that the review must determine that outputData is valid, otherwise this constraint will be violated and a flag will be raised to notify the bakers. The entry point for this advice metamodel specifies that it applies to processes of any type (i.e. it applies to all processes).

Figure 3.9 shows how we denote the entry points of the advice metamodel, where purple circles are entry points for the reviewing process which produces the *Valid?* attribute of the process being reviewed. This same advice meta-

model is woven into each step in the bakery's process.

Figure 3.10 shows an example of instance of Mix with the full advice metamodel woven into it. Note that purple arrows are used for inputs/outputs of reviewing processes to differentiate them from regular process inputs/outputs and that the outputs to associations were omitted to ease reading of the diagram. In this example, the experienced baker sees that one instance of flour1 : Flour and water1 : Water were used by Mix to produce mixture1 : Mixture. The experienced baker uses their expertise to make a judgement as to whether or not the real-world mixture modelled by mixture1 : Mixture is valid given the inputs used. In this example it turns out that mixture1 : Mixture is in fact valid. We could make this review more specific by defining the steps of the review, for example by requiring reviews of texture, consistency, etc., but for the purposes of this example we leave the reviewing solely up to the judgement of the experienced baker.

3.6 Adding Argumentation to the Model

Argumentation in Workflow$^+$ is centred around data-driven inference directly over metamodels. Workflow$^+$ metamodels are built to require all instances to be a) syntactically correct (i.e. well-formed) via OCL-style constraints and multiplicities and b) semantically correct (i.e. verified/validated) via constraints on results of reviewing, simulation, testing, etc. In our running example, validation is achieved solely by reviewing, but as stated could also be achieved by testing, simulation, etc. These syntactic and semantic constraints can be augmented with natural-language descriptions allowing them to be interpreted as arguments as to why they ensure executions will be correct.

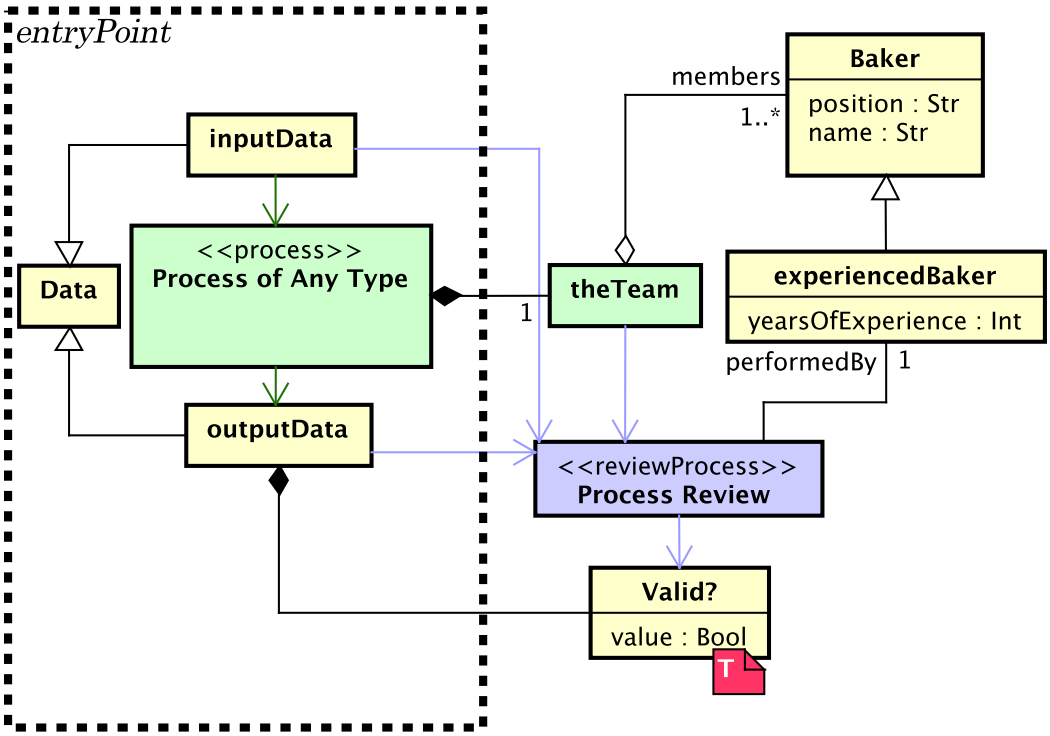

Figure 3.8: Bakery: Advice metamodel

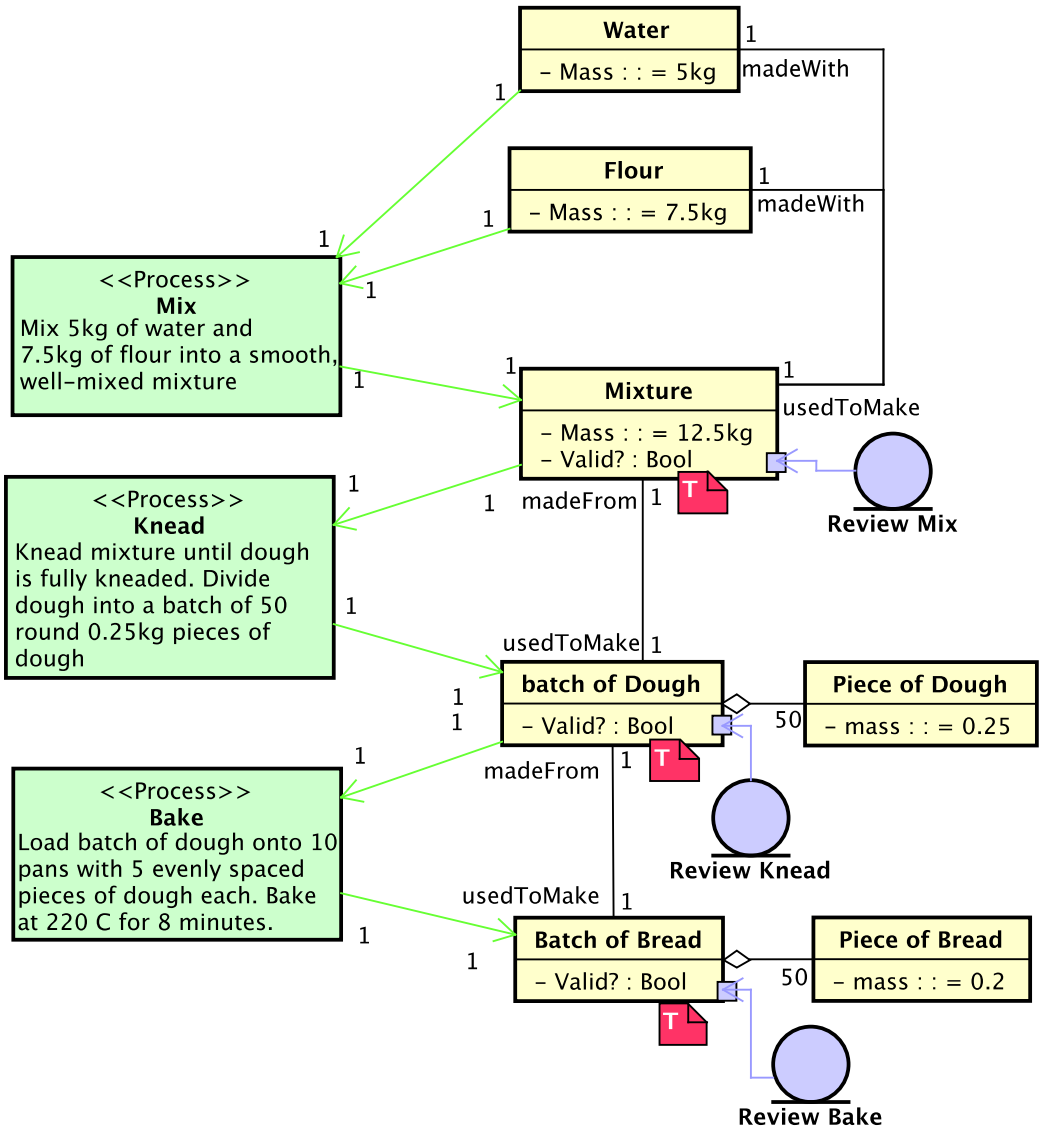

Figure 3.9: Bakery: entry points for advice metamodel

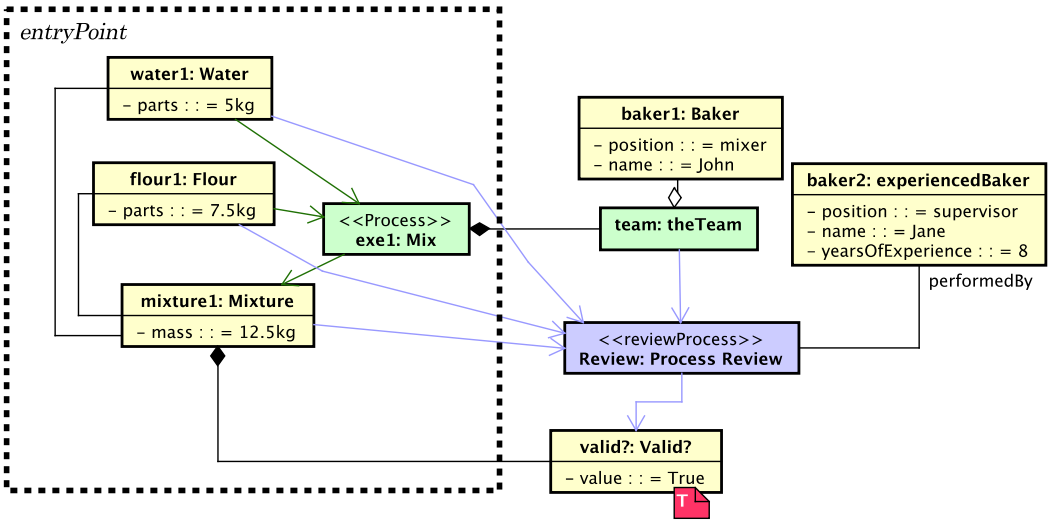

Figure 3.10: Bakery: weaving in the review metamodel for the Mix process

These syntactic and semantic constraints (and their associated natural-language arguments) can be composed into higher-level derived constraints (and arguments). These higher-level constraints (and arguments) can then be composed repeatedly, resulting in a constraint derivation tree. That is, low-level semantic and syntactic constraints can be used to derive higher-level arguments until the highest-level argument in the derivation tree is reached. This is illustrated in Figure 3.11, and a more detailed definition of this inference can be found in [3].

In every case of constraint derivation, the underlying justification (similar to strategies in GSN assurance cases) is "The underlying constraints are all that is necessary to claim the derived constraint". In future work, mechanisms to supplement these constraint derivations with external justifications (e.g. expert opinion, case studies, etc.) will be introduced. The idea is that in specific domains, collections of established derivation patterns with trusted justifications will be available for the designers of workflows to draw on.

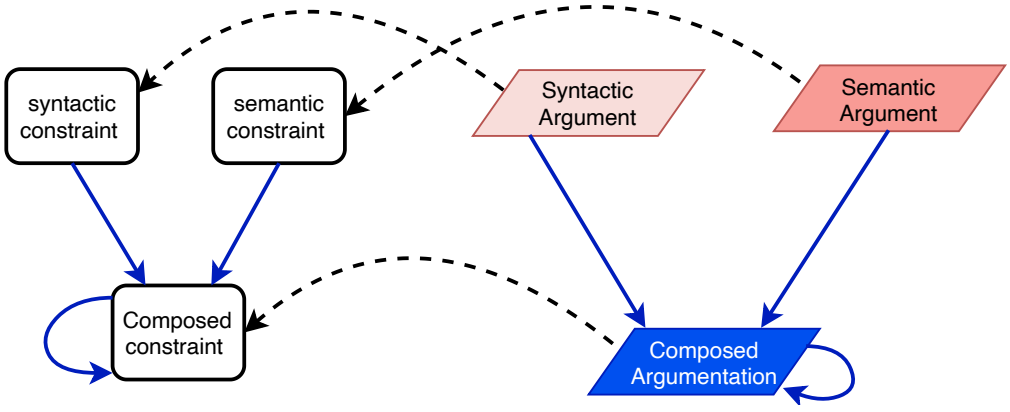

Figure 3.11: An illustration of the mapping of constraint composition and the mapping of natural-language arguments to constraints

Whether or not these arguments do in fact ensure that executions will be correct is beyond the scope of the arguments themselves; all they do is provide an explanation as to why the creators of the workflow believe the that the constraints will result in correct executions. Additionally, even if these constraints do result in the executions desired by the workflow's creators, whether or not these executions are appropriate for the task at hand is another question altogether. Workflow$^+$ metamodels are subject to (and actually invite) criticism from those interested in ensuring there are no flaws in the workflow definition. This issue of determining whether or not the process definition is adequate is discussed in more detail in 3.7.

A view of this argumentation akin to a GSN assurance case can be developed quite mechanically.

With these syntactic and semantic constraints in place in the metamodel, the soundness of this constraint derivation allows us to be sure that any valid execution of the process will satisfy all low-level constraints, and thus all high-level constraints will hold for any valid execution of the process. Again, see [3] for more details.

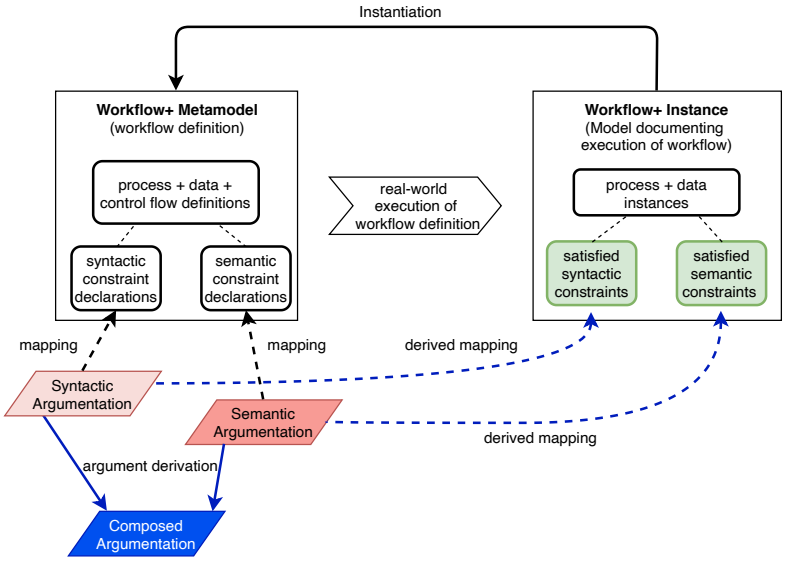

Figure 3.12: An overview of Workflow$^+$ with argumentation (see Figure 3.3)

Figure 3.12 illustrates the idea of argumentation over a Workflow$^+$ metamodel. Syntactic and semantic constraints over process, data and control flow definitions in the metamodel are annotated with syntactic and semantic argumentation, respectively, and used to derive higher-level argumentation. When an instance of this metamodel is created that satisfies all constraints, we can be sure that all argumentation built over the constraints definitions also applies to an instance in which those constraints are satisfied.

3.6.1 Running Example: Adding Assurance to the Model

Suppose now that the bakery needs to do more than satisfy themselves about the quality of their bread. They also need to guarantee quality to their customers! Thus, the bakery wants to document a convincing argument for each loaf of bread, demonstrating that it is of acceptable quality. The abstract process definition in Figure 3.5 annotated with syntactic and semantic arguments is shown in Figure 3.13.

The syntactic argument summarizes the syntactic constraints for the entire process definition. That is, it is a natural-language description of what is ensured through syntactic constraints over the process definition. While we could annotate each individual constraint, this becomes cumbersome so we summarize them as one argument. In this case, the syntactic constraints ensure that the *mass* attributes of instances of Water and Flour used are in the correct proportions(red OCL-style constraint) and that the ingredients used are in fact documented (the multiplicities requiring an association from instances of Water and Flour to PieceofBread). These constraints are captured by the light pink argument attached to the entire process definition. If these constraints are indeed satisfied by an instance, we can be sure that, according to what was documented, the correct amount of flour and water were used and everything documented is properly structured. This gives us some confidence in the correctness of the instances, but it does not provide any confidence that the process was actually executed properly, or documented properly. For example, it is certainly possible that the correct amount of water and flour were used in the wrong way, or that the amounts documented do not reflect what was actually used; detecting this is left for semantic checks. The intention of syntactic arguments is to provide a "sanity check" on what is documented, and to enable the automatic detection of (relatively) simple errors when documenting executions. These automatic checks can provide feedback on violated constraints in the process of constructing the instance to bring attention to issues before the entire instance is built, and ensure that documentation is well-formed before more time-consuming (and expensive) semantic checks (i.e. reviews, testing, etc.) are carried out.

Semantic arguments explain the semantic constraints in the process defi-

nition. In this case there is only one semantic check, achieved by reviewing the output of MakeBread. This review uses the same advice metamodel shown previously. In the review, an experienced baker checks the validity of each piece of bread output by assessing its quality (i.e. it is not burned, it is crispy and is soft but firm). The pink constraint T requires that the value of *Valid?* output by this process must be True for an instance to be considered correct. In doing this, we are syntactically encoding a requirement for semantic validity of an instance. That is, if for any reason the value of *Valid?* is anything but True, the instance will be considered invalid. The argument related to this validity check is captured by the dark pink argument. It is in this review that the correct execution of the process is assessed. Based on the documentation of the execution, and with access to the real-world piece(s) of bread being assessed, an experienced baker determines whether or not the process was properly executed and that the output is valid in the context of its input. That is, by looking at documentation of what was used and what was produced, an experienced baker can use their expertise to identify signs that something went wrong during the process execution. The signs to look out for can be included in the specification of the review process, but are omitted for simplicity.

We can compose these syntactic and semantic arguments to derive the argument *bread is of high quality*, which applies to the entire process specification. In the future when support for justification of derivations is included in Workflow$^+$ a justification here could be "We have had success making this bread for a very long time". This justification is obviously informal, and highlights the informal nature of assurance. The experience of the bakery cannot be formalized, but is still a very important part of their assurance case. Nev-

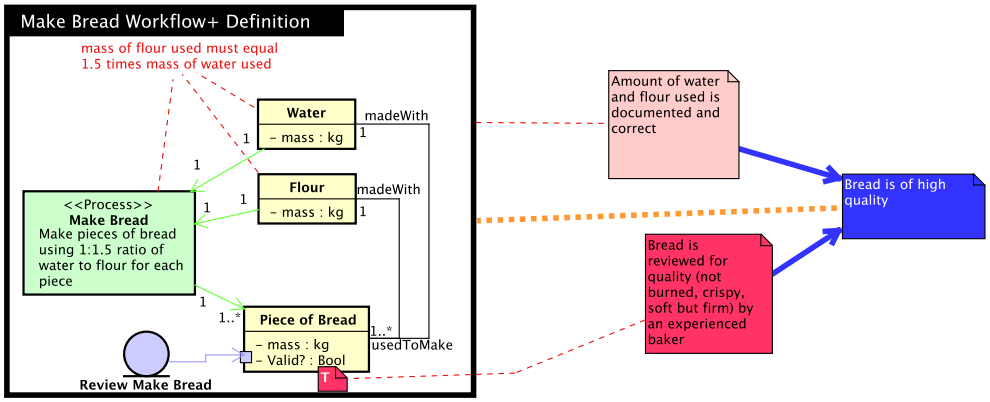

Figure 3.13: Bakery: Make Bread process from Figure 3.5 with argumentation

ertheless, Workflow$^+$ provides the bakery a framework in which justifications can be included in a formal way.

To summarize, the syntactic and semantic constraints on the metamodel (i.e. process definition) level require checks on instances of the metamodel. Syntactic correctness of an instance can be checked automatically through instance conformance. Semantic correctness cannot be determined automatically, so we weave in a review process outputting the value of the attribute *Valid?*, and put the constraint T on the output of that review process. In doing this, we require that any valid execution must run the review process, and that the review determines if the output is valid (i.e. execution is correct). If instances of this metamodel are valid, i.e. all syntactic constraints are satisfied and all reviews determine that outputs are valid, then we can be sure that the arguments related to these constants hold.

Just as the process definition in Figure 3.13 is not very informative for someone trying to make bread, the constraints and corresponding arguments do not do much to convince us that the bread will be of acceptable quality. The bakery now looks to the more instructive metamodel in Figure 3.7 to

provide a more transparent and convincing argument for the quality of the bread produced.

When the MakeBread process was decomposed, what the bakery did was add more detail on the steps followed to produce PieceofBread from Flour and Water. While doing this, they also added more detailed constraints over the (more detailed) workflow to ensure that its executions will be correct. From these more detailed constraints, we can create more detailed argumentation over the (more detailed) workflow.

To create argumentation for this workflow with multiple processes, the approach remains the same: we must annotate each process with syntactic and semantic argumentation and derive higher-level argumentation. The argumentation for each process within MakeBread is explained individually below.

For the Mix process, the syntactic constraints require that exactly the correct ingredients (5kg of water and 7.5kg of flour) must be combined to produce a 12.5kg mixture. These constraints give us confidence that executions will be correct because correct instances of Water and Flour must be used, and the mass of the instance of Mixture will be what we were expecting. From these constraints, we create the syntactic argument *mixture is correct weight, amount of flour in mixture is 1.5 times amount of water*. The semantic constraint on Mixture ensures that its instances are reviewed for quality (i.e. ingredients are evenly distributed throughout the mixture), from which we create the semantic argument *Mixture is reviewed for quality (ingredients evenly distributed throughout)*. This semantic argument gives us confidence that executions will be correct because an experienced baker must determine that the resulting instance of Mixture is acceptable. From these arguments, we can derive the argument *mixture is made well*. See Figure 3.14.

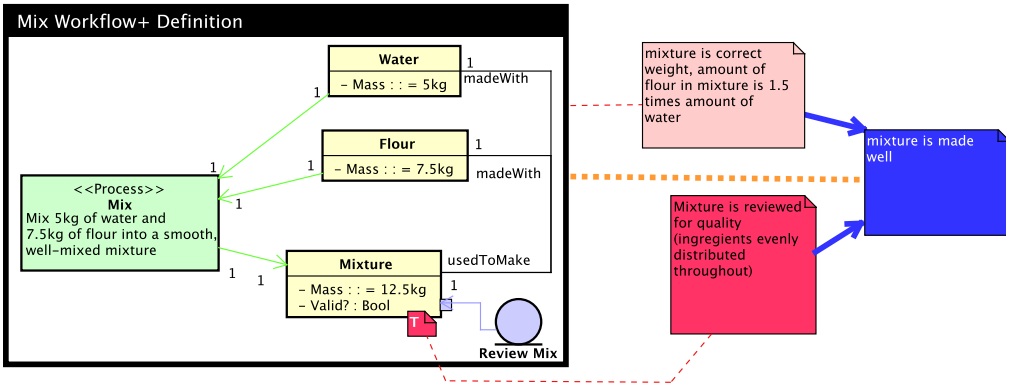

Figure 3.14: Bakery: Mix process from Figure 3.7 with argumentation

For the Knead process, the syntactic constraints require that the mixture used is documented and that each piece of dough produced has the correct mass. These constraints give us confidence that executions will be correct because the instance of Mixture used must be documented and each instance of PieceofDough must have the mass expected. From these constraints, we create the syntactic argument *mixture used is documented, each piece of dough has correct mass*. The semantic constraint on BatchofDough ensures that its instances are reviewed for quality (i.e. the correct shape and well-kneaded), from which we create the semantic argument *Batch of dough is reviewed for quality (correct shape, not under or over kneaded)*. This semantic argument gives us confidence that executions will be correct because an experienced baker must determine that the resulting instance of BatchofDough is acceptable. From these arguments, we can derive the argument *dough is made well*. See Figure 3.15.

For the Bake process, the syntactic constraints require that the batch of dough used is documented, that each piece of dough used has the correct mass, and that each piece of bread produced has the correct mass. These constraints give us confidence that executions will be correct because we will know which

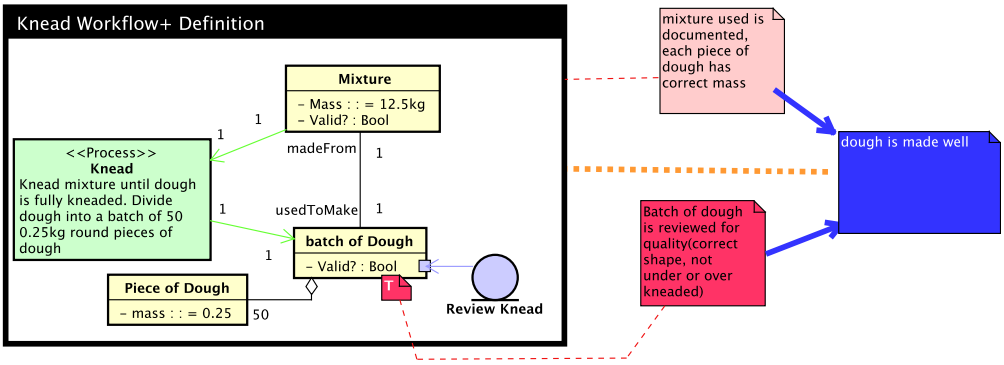

Figure 3.15: Bakery: Knead process from Figure 3.7 with argumentation

batch of dough is used (knowing that each piece of dough used was of acceptable quality, i.e the correct weight and shape, is essential for a correct execution of Bake), and that the mass of each instance of PieceofBread produced is correct (we can infer if bread is well-baked based on the mass; bread that is too heavy could be a sign of high water content in undercooked bread, and bread that is too light could be a sign of low water content in overcooked bread). From these constraints, we create the syntactic argument *batch of dough used is documented, each piece of bread has correct mass.* The semantic constraint on BatchofBread ensures that its instances will be reviewed for quality (i.e. bread is not burned, crispy, soft but firm), from which we create the semantic argument *Batch of bread is reviewed for quality (not burned, crispy, soft but firm).* This semantic argument gives us confidence that executions will be correct because an experienced baker must determine that the resulting instance of BatchofBread is acceptable. From these arguments, we can derive the argument *bread is made well.* See Figure 3.16

Now that each process has its arguments specified, what remains is to compose these arguments to complete the argument derivation tree. Composition of arguments is done by following the composition of processes. In this simple

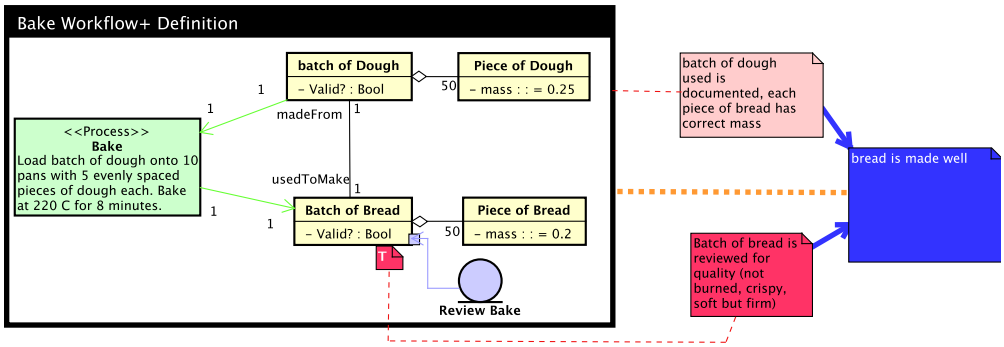

Figure 3.16: Bakery: Bake process from Figure 3.7 with argumentation

case with sequential composition of processes, the constraint derivation is also done in the same sequential order.

From the (blue) argument combining the syntactic and semantic arguments of Mix, we know that all conforming instances will use water and flour to produce a mixture of acceptable quality. Similarly, from the argument of Knead, we know that all conforming instances will use a mixture to produce a batch of dough of acceptable quality. Finally, from the argument of Bake, we know that all conforming instances will use a batch of dough to produce a batch of bread of acceptable quality. Composing these arguments, we can say that if a) the mixture is made well, b) the dough is made well, c) the bread is made well, then that batch of bread is of acceptable quality. This is captured by the derived argument *bread is of high quality*. See Figure 3.17.

In other words, if the execution every step along the way is syntactically and semantically correct, then the final product that is ultimately created using the inputs to the process is correct. While this may sound like it is an entirely process-focused argument, the inclusion of reviewing processes and data-related constraints (e.g. on *mass* attributes) allows us to add product-focused components to the argumentation.

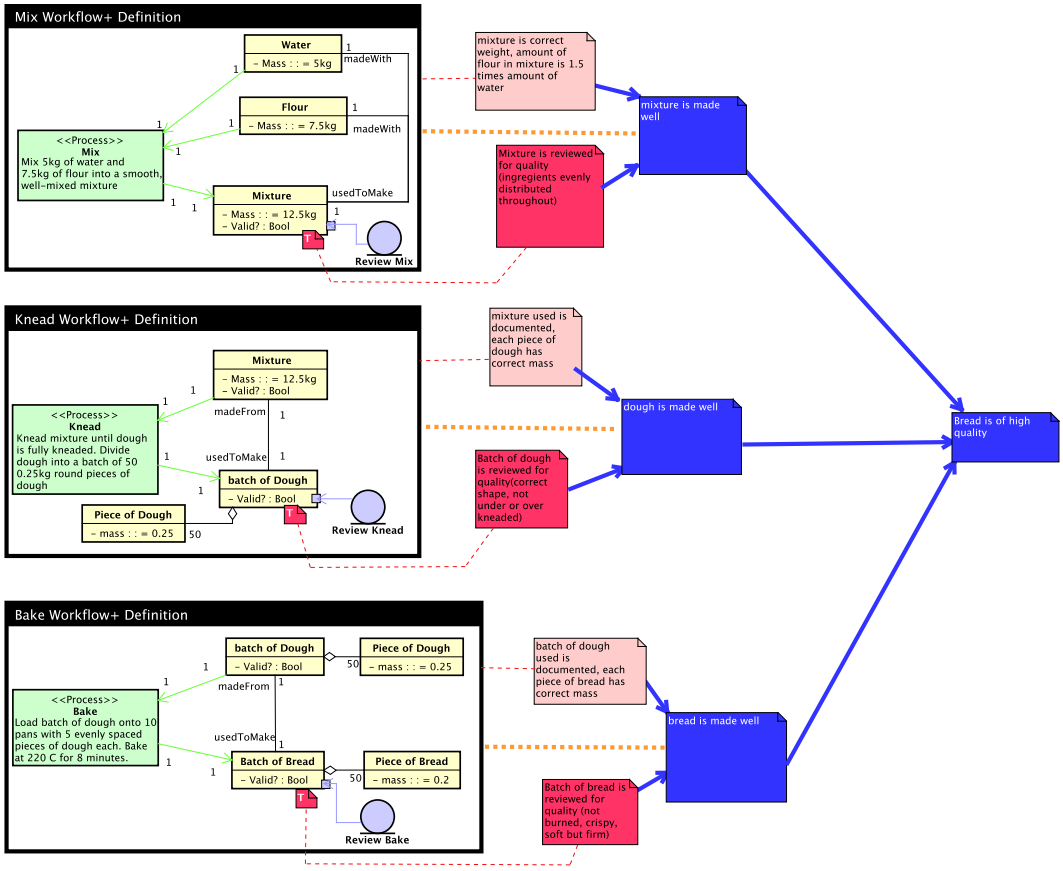

Figure 3.17: Bakery: Mix, Knead and Bake processes with composed argumentation

Thus, we have created an argument structure over the abstract and decomposed workflows that show arguments as to why the bakery's bread is of acceptable quality. To reiterate, these argument structures do not prove that the bread is of acceptable quality, it merely explains the intentions of the process's creators and why they think it will produce quality bread. The workflows and their argumentation are subject to criticism and feedback from those who think it may be flawed. For example, one could argue that 50 pieces of bread per batch is too many because they believe small batches result in higher quality bread. The precisely defined semantics of the Workflow$^+$ process definition, the detailed traceability and well-defined argument pattern allows for external critics to understand and analyze the workflow effectively and provide meaningful feedback.

3.7 Conformance to Normative Metamodel

So far, we have discussed how Workflow$^+$ metamodels can be built, instantiated and have arguments built over them. What we have not addressed is how we can be sure that these Workflow$^+$ process definitions are appropriate for their purpose. The complexity of real-world environments and the processes used to develop safety-critical systems makes it impossible to prove without doubt that a safety engineering process (including both product- and process-focused aspects) will produce a safe system for some particular application in a particular environment. In general, the approach used is to call upon domain experts with years of experience in building and observing safety-critical systems in the real world to create guidelines by which safety-critical systems in their domain should be built, and analyze these systems and how

they were developed to make sure that the safety risk they pose is adequately low (a societal decision dependent on many factors including the application domain) – before they are put into use. These guidelines and analysis do not ensure that safety-critical systems will operate safely with absolute certainty, but do represent an honest attempt to use all available knowledge to make sure these systems operate safely. If this attempt turns out to fail and harm is caused, the cause can be determined and addressed in the development guidelines and avoided in the future. This issue is particularly prevalent when using emerging technologies with little to no use in the real world to draw on when making guidelines for development.

To be more specific, in the world of safety-critical embedded systems, it is common for a regulator (i.e. domain experts empowered by legislation) to require that a safety engineering process must conform to some standards (i.e. normative documents). These standards should be conservative in that they must restrict how safety-critical systems can be developed by not allowing radical changes from design patterns/technologies that are already well understood [2].

With that said, we can now explain how to gain confidence in argumentation over Workflow$^+$ metamodels. In the running example, we demonstrated how a process definition can have argumentation built over it. We propose to use Workflow$^+$ metamodels to encode the requirements of normative documents to form a normative Workflow$^+$ metamodel. This normative Workflow$^+$ metamodel should be sufficiently restrictive to prevent radical changes to well-known design patterns, but still be general enough to allow manufactures to use their own established procedures/processes to produce their systems. It would also provide argumentation related to the constraints in the normative

Workflow$^+$ metamodel to precisely express their intent. Manufacturers would then take this normative Workflow$^+$ metamodel and specialize it however they see fit. In their specialization, they must also include argumentation that refines (i.e. implements) the argumentation in the normative Workflow$^+$ metamodel. Following this, the regulators who issued the normative Workflow$^+$ metamodel can assess the manufacturer's specialization and make a judgement as to whether or not the manufacturer has actually designed a workflow that meets their requirements. Finally, with an approved workflow, any instance of a system created using the manufacturer's approved workflow will satisfy all constraints and arguments by virtue of instance-metamodel conformance and can be considered acceptably safe according to the norms in that domain and at that time.

This idea is illustrated in Figure 3.18. In this figure, we show that there is a normative metamodel built by regulators to provide guidance to manufacturer's in the specific domain it applies to. Over this workflow, the regulator includes argumentation that needs to be supported by argumentation over the workflow's refinements. When a manufacturer implements the workflow they want to use by refining the normative metamodel, they also include argumentation over their workflow and use the inference mechanism from Section 3.6 to show how the constraints over their refined workflow implement the argumentation required by the normative metamodel. Again, when the workflow created by a manufacturer is executed, we can be sure that instances also satisfy all argumentation in the metamodel. Thus, if the normative Workflow$^+$ metamodel specifies what must be done during the development of a system, and if the manufacturer implements the normative Workflow$^+$ metamodel in a way that satisfies the regulator, and an instance of the manufacturer's workflow

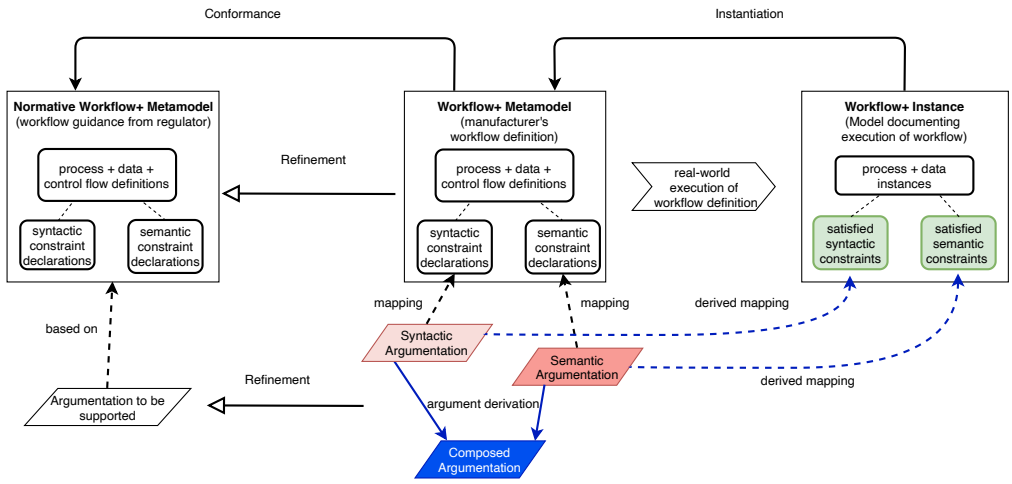

Figure 3.18: An overview of Workflow$^+$ with a normative workflow

satisfies all constraints, then we can be sure that the execution documented by that instance (and the system it produced) are acceptable – based on the content of the normative metamodel.

For example, consider that during the development of a certain type of system, regulators deem that simulations validating the behaviour of part of a system are required. The regulator could include a black-box process requiring simulation and include an associated argument specifying what the simulation must achieve. Then, a manufacturer could implement that black-box simulation process based on their own internal approach (which is likely to differ from other manufacturers) and add argumentation as to why their approach satisfies the intent of the normative metamodel. A regulator could then look at the manufacturer's workflow to assess their approach to simulation, try to identify issues or gaps in argumentation and determine if it does indeed satisfy their intent.

It is important to note that this approach does not exclude the possibility for instances to require external validation and/or verification. It is entirely

possible for independent validation and verification processes to be included in the workflow. For example, a normative Workflow$^+$ metamodel could require that certain processes/data must be reviewed by a third party rather than the manufacturer themselves. If these reviews are not executed, the instance would be invalid, resulting in violated constraints and unsatisfied arguments.

3.7.1 Running Example: Conformance to Normative Metamodel

In Figure 3.17 in Section 3.6.1 we showed the bakery's argumentation as to why it's process will result in quality bread, but the question as to whether or not this process will in fact result in quality bread remains unanswered. To answer this question, we will use the concept of conformance to a normative metamodel.

We will now frame the metamodel and argumentation specified in Figure 3.5 as a normative metamodel, then use it to argue the correctness of the bakery's process based on conformance to that normative metamodel. Figure 3.19 shows this metamodel with slightly different argumentation than Figure 3.16; the metamodel now only has one white argument attached to it, which states *bread is of high quality*. This argument is now white to denote that it is not fully supported, and that the argument and workflow it applies to require decomposition. We also modified Figure 3.17 such that it decomposes both the process and argumentation in Figure 3.19, shown in Figure 3.20.

The first part of demonstrating conformance to the normative metamodel is syntactic: does the decomposition of the normative metamodel conform to all constraints? The second part of conformance to the normative metamodel

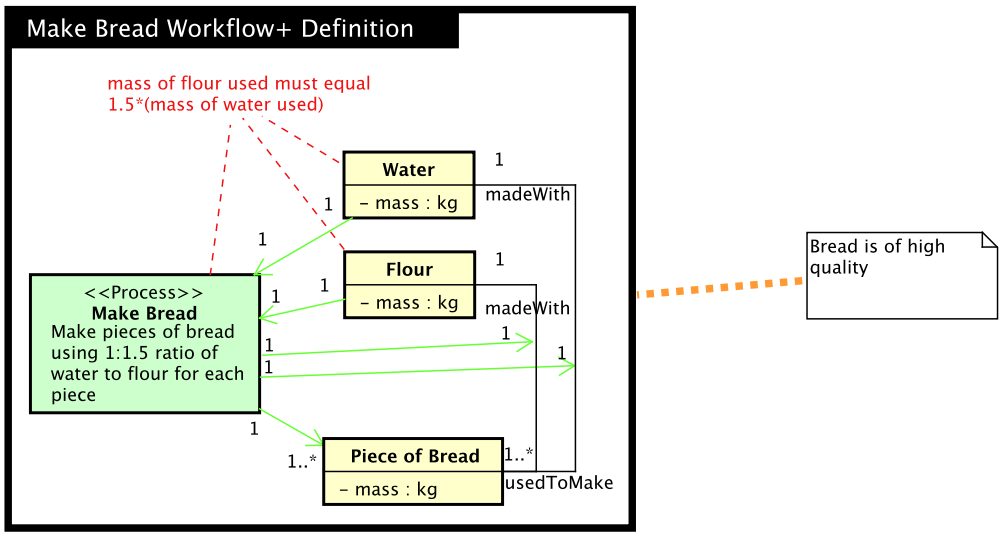

Figure 3.19: Bakery: normative Workflow$^+$ metamodel

is semantic: does the decomposition implement the normative metamodel in a way that its creators would agree with?

To answer the syntactic question related to conformance, we carry out the mechanical (and automatable) task of checking that all syntactic constraints in the normative metamodel are satisfied by its refinement. In this example, we see that all syntactic constraints are satisfied (e.g. Water and Flour are to be used in a ratio of 1.5:1, all multiplicities are satisfied). We also must check that the refined metamodel implements the correct argumentation. In this example, we can see that the highest level of argumentation in the normative metamodel and its refinement are the same (note the white arguments).

To answer the semantic question related to conformance, some sort of regulatory body must check the decomposition of the process and its argumentation to determine if it does indeed implement the intentions of the normative metamodel. For the sake of example, assume that this normative workflow belongs to an organization that must approve a processes before it can be used to

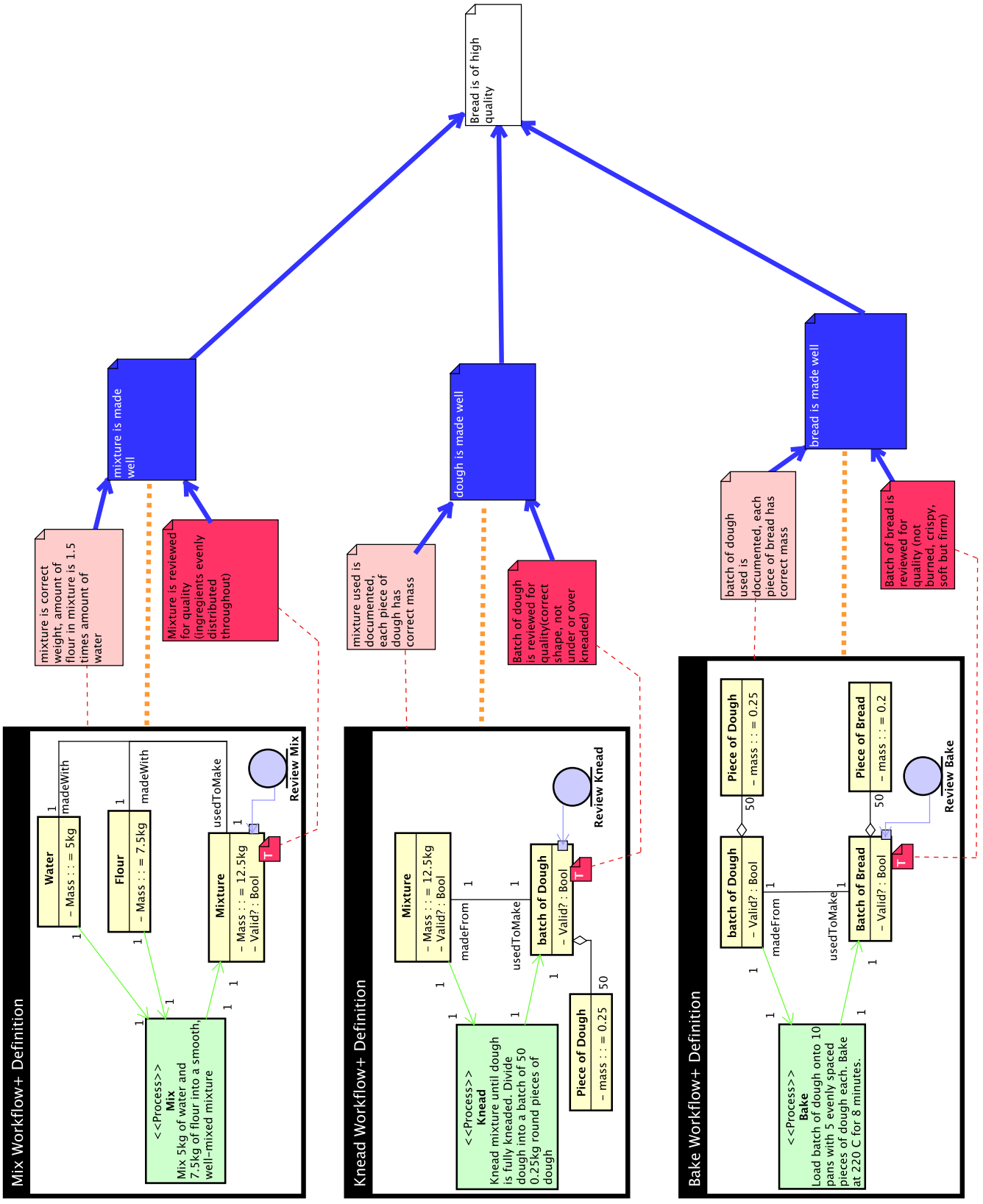

Figure 3.20: Bakery: A Workflow$^+$ metamodel conforming to the normative metamodel in Figure 3.19

make bread using their recipe. This organization would look at the bakery's implementation of their normative workflow along with its argumentation and make a judgement as to whether or not the implementation does indeed implement their intentions, and determine if it will in fact produce quality bread according to their expertise in their recipe.

Chapter 4

Workflow$^+$ Approach To Assurance

4.1 The Generic Workflow$^+$ Approach

The Workflow$^+$ approach to assurance can be seen as an integration of process- and product-focused approaches. The approach relies on process-focused foundations, but using Workflow$^+$ modelling techniques allows product-focused aspects to be accurately encoded in a process-focused approach. For example, if a product-focused approach requires that a system satisfies some property P, we can implement that product-focused requirement by requiring the execution of a process that ensures the property P is satisfied. In this way Workflow$^+$ can be used as a unifying foundation for the modelling and analysis of both systems themselves (the product-focused side) and the safety engineering processes (SEPs) followed to produce these systems (the process-focused side), including all verification and validation (V&V) required.

We propose to use the following definition of safety as the core of assurance

cases built using Workflow$^+$:

Definition 1 (Main) *System X is considered to be* acceptably safe (for a given application A in a given environment E) *if the manufacturer's safety engineering process, SEP, satisfies the following two conditions:*

$$\textit{SEP is acceptably-well defined for a class of systems including } X \tag{4.1}$$

$$\textit{SEP's definition is acceptably-well executed for } X \tag{4.2}$$

Demonstrating that (4.1) is satisfied entails proving that some set of accepted standards, best practices, etc., (as determined by experts in the field) are respected by the SEP. In the field of safety-critical embedded systems, a well-defined SEP must

a) Demonstrate that the requirements specification will result in a safe system

b) Demonstrate that the system satisfies its requirements within tolerance

c) Demonstrate that the system does not implement behaviour not specified within the requirements, unless those behaviours have been shown to not interfere with the specified requirements

These points are often demonstrated using a combination of process- and product-focused approaches. In a Workflow$^+$ setting, a well-defined SEP must include the necessary process, data and constraint definitions that if properly followed will result in a system that satisfies the three conditions above.

Condition (4.2) is then achieved by demonstrating that this SEP was acceptably well executed using artefacts from development. That is, if the SEP definition includes everything needed to ensure safety, and the SEP is properly executed, then the system is considered acceptably safe for its application in a given environment. To reiterate, a well-defined SEP must include both product- and process-focused considerations for the safety of a system.

To make the conditions in Definition 1 technical and verifiable, we use three Workflow$^+$ models: $\mathsf{Wf}_{\mathsf{SEP}}$ that models the SEP, $\mathsf{Wf}^{\mathsf{exe}}_{\mathsf{SEP}}(X)$ that models the SEPs execution for the system X, and $\mathsf{Wf}_{\mathsf{Norm}}$ that models a body of normative documents (standards, best practices, expert knowledge, on-site manuals, etc.) which prescribe how the SEP should be defined. In this setting, we can redefine the conditions in Definition 1 in the following way:

Definition 2 (Main, more formal) *System X is considered to be* acceptably safe *if the following two conformance conditions are satisfied:*

$$\mathsf{Wf}^{\mathsf{exe}}_{\mathsf{SEP}}(X) \Vdash^{\mathsf{V\&V}}_{\mathsf{inst}} \mathsf{Wf}_{\mathsf{SEP}} \Vdash^{\mathsf{V\&V}}_{\mathsf{ref}} \mathsf{Wf}_{\mathsf{Norm}} \tag{4.3}$$

where $\mathsf{Wf}_{\mathsf{Norm}}$ and $\mathsf{Wf}_{\mathsf{SEP}}$ represent Workflow$^+$ models, $\mathsf{Wf}^{\mathsf{exe}}_{\mathsf{SEP}}$ represents an execution of $\mathsf{Wf}_{\mathsf{SEP}}$. $\Vdash_{\mathsf{inst}}$ represents an instance-of conformance mapping, and $\Vdash_{\mathsf{ref}}$ represents a conformance mapping between a workflow and its refinement. The significance of V&Vis explained in the following paragraphs.

$\mathsf{Wf}_{\mathsf{Norm}}$ is considered to be given (e.g., by regulators and/or appropriate standards) and models a high-level workflow along with argumentation as to why this workflow assures safety. Then, a manufacturer's $\mathsf{Wf}_{\mathsf{SEP}}$ must a) refine and conform to the workflow in $\mathsf{Wf}_{\mathsf{Norm}}$ and b) produce argumentation over

their refined workflow conforming to the argumentation in $\mathsf{Wf}_{\mathsf{Norm}}$. Finally, $\mathsf{Wf}^{\mathsf{exe}}_{\mathsf{SEP}}$ must conform to $\mathsf{Wf}_{\mathsf{SEP}}$. Through the soundness of logical inference, we can be sure that the arguments in $\mathsf{Wf}_{\mathsf{SEP}}$ hold for its executions (see 3.6).

V&V of each conformance mapping has two parts: semantic and syntactic. Syntactic conformance requires that all structure and constraints in the model being conformed to are satisfied. For $\Vdash_{\mathsf{inst}}$, this entails ensuring that $\mathsf{Wf}^{\mathsf{exe}}_{\mathsf{SEP}}$ satisfies all constraints (typing mappings, multiplicity constraints and more complex OCL constraints) in $\mathsf{Wf}_{\mathsf{SEP}}$. For $\Vdash_{\mathsf{ref}}$, this entails ensuring that $\mathsf{Wf}_{\mathsf{SEP}}$ does not violate any structural constraints in $\mathsf{Wf}_{\mathsf{Norm}}$. Checking syntactic conformance in this sense is a well-known model management problem, and with appropriate tool support can be done automatically.

Semantic conformance requires all syntactically valid information to be meaningful and correct. For $\Vdash_{\mathsf{inst}}$, this requires manual reviewing of $\mathsf{Wf}^{\mathsf{exe}}_{\mathsf{SEP}}$ by qualified individuals to ensure all data is valid and processes were properly executed, which is captured by integrating V&V (e.g. reviewing, testing, simulation, etc.) processes to be executed into $\mathsf{Wf}_{\mathsf{SEP}}$. This reviewing can be done by the manufacturer, regulator or both. For $\Vdash_{\mathsf{ref}}$, this requires the regulator to ensure that $\mathsf{Wf}_{\mathsf{SEP}}$ (including both the workflow and argumentation) adequately implements the workflow and argumentation in $\mathsf{Wf}_{\mathsf{Norm}}$.

Once all of these conformance conditions are satisfied, we can be sure that a system satisfies all product-focused conditions and was produced according to all process-focused constraints encoded in $\mathsf{Wf}_{\mathsf{Norm}}$. If we have confidence that $\mathsf{Wf}_{\mathsf{Norm}}$ properly encodes what is required for a system to be considered safe, we know that $\mathsf{Wf}_{\mathsf{SEP}}$ properly implements the intent of $\mathsf{Wf}_{\mathsf{Norm}}$, and $\mathsf{Wf}^{\mathsf{exe}}_{\mathsf{SEP}}$ is a correct execution of $\mathsf{Wf}_{\mathsf{SEP}}$, we can then be confident that the system itself is safe. In the field of safety-critical embedded systems, if $\mathsf{Wf}_{\mathsf{Norm}}$ adequately

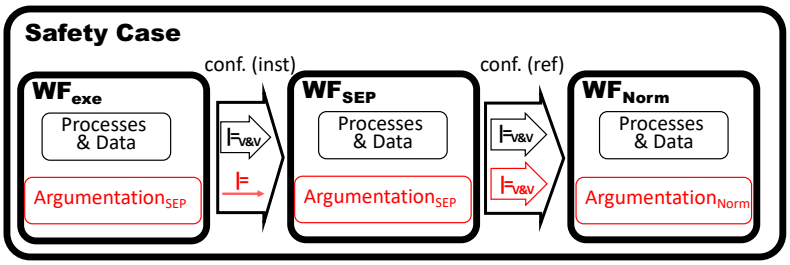

Figure 4.1: Workflow$^+$ conformance chain

implements the three points mentioned alongside Definition (1), then we can be confident that the embedded system is acceptably safe. Altogether, $\mathsf{Wf}_{\mathsf{Norm}}$, $\mathsf{Wf}_{\mathsf{SEP}}$, $\mathsf{Wf}^{\mathsf{exe}}_{\mathsf{SEP}}$ and all conformance mappings constitute an assurance case (see Figure 4.1).

To summarize, satisfaction of (4.1) and (4.2) in Definition 1 can now be demonstrated by the well-defined and formalizable tasks of demonstrating the conformance mappings $\mathsf{Wf}_{\mathsf{SEP}} \Vdash_{\mathsf{ref}} \mathsf{Wf}_{\mathsf{Norm}}$ and $\mathsf{Wf}^{\mathsf{exe}}_{\mathsf{SEP}} \Vdash_{\mathsf{inst}} \mathsf{Wf}_{\mathsf{SEP}}$, respectively.

It is worth mentioning again that while Workflow$^+$ provides a formal framework within which safety can be argued, it does not provide a way to formally prove safety. The qualitative and quantitative infinity of possible interactions of a system with its environment makes a fully formal proof of safety impossible, and any efforts to ensure the safety of a system that can be completed in finite time have the potential to be incomplete. Instead, our approach, similar to most other approaches currently used, relies heavily on the expertise of those creating $\mathsf{Wf}_{\mathsf{Norm}}$ and evaluating conformance of $\mathsf{Wf}_{\mathsf{SEP}}$ and $\mathsf{Wf}^{\mathsf{exe}}_{\mathsf{SEP}}$. Through years of experience developing and watching systems operating in the field, confidence in the efficacy of approaches to safety can be evaluated and used to update best practices that should be followed to ensure safety. What Workflow$^+$ does provide is a modelling framework capable of formally expressing these best practices and checking that they are adequately followed

in a semi-automatable way. For those issues which cannot be demonstrated automatically, the detailed traceability and granularity of Workflow$^+$ models allows experts to conduct reviews with total access to all necessary information. We believe that a model-based approach to arguing safety facilitates the construction of both theory and tools that will cope effectively with the complexity of the systems of the future, and that Workflow$^+$ is well suited to play this role.

4.2 Workflow$^+$ Models As Assurance Case Templates

When dealing with safety-critical systems, is it necessary to plan for the safety of the system ahead of its development. To help in this planning, it has been proposed to use assurance case templates to specify nearly-complete assurance cases for a particular type of system before development begins [12]. The idea is that a template should include sufficiently prescriptive limitations on systems it applies to (as determined collectively by experts in the field), but still allow enough flexibility so as to not unduly interfere with the creative design of the systems. That is, assurance case templates should specify higher-level safety strategy and the overall structure of the corresponding assurance case to guide the development of lower-level safety strategies. A significant addition to assurance case templates was the specification of *acceptance criteria* in evidence nodes within the template. This guides development of the system and the assurance case that is instantiated from the template can then be checked for conformance with these acceptance criteria.

Workflow$^+$ is well suited to implementing the idea behind assurance case templates. In fact, the inclusion of $\mathsf{Wf}_{\mathsf{Norm}}$ in the Workflow$^+$ approach to safety does exactly this by requiring the definition of a high-level workflow with argumentation created by domain experts that guides the specification of the $\mathsf{Wf}_{\mathsf{SEP}}$ that will actually be executed. Since Workflow$^+$ is formally defined, it is possible to formally express what is required by an assurance case template. When compared to assurance case templates created using current approaches to safety assurance, we believe that the Workflow$^+$ approach can solve common problems in the refinement of assurance case templates for execution by offering well-defined, accurate process specifications with detailed traceability and by providing suitable decomposition (i.e. implementation) mechanisms.

As we shall see, data items in $\mathsf{Wf}_{\mathsf{SEP}}$ most often are associated with evidence in the (to be) developed system, and the notion of acceptance criteria is supported by both relevant constraints as well as textual specification. In fact, the derived assurance steps that are introduced based on constraints can lead the way to providing acceptance criteria. For example, in a model regarding hazard analysis, we specify both process and data associated with the hazard analysis. In particular, in automotive functional analysis we will likely specify that safety gaols are developed to mitigate hazards. The associated constraints will dictate that each hazard is mitigated by at least one safety goal and that each safety goal mitigates at least one hazard. We then derive a syntactic check that can be automatically verified, that these associations exist. There are also semantic checks to see if each safety goal associated with a hazard does, in reality, mitigate that hazard. The review can list criteria that reviewers must check for in order to "prove" the validity of that review. For example, reviewers may be told to check that: i) the safety goal is described in

sufficient detail to be able to determine its scope and effect; ii) the safety goal describes reasoning as to why and how the safety goal mitigates the hazard; iii) any prior history of that safety goal being used to mitigate that hazard is described; and iv) sufficient references to back up the claim that the safety goal mitigates that hazard are included. In this example, i) ..iv) are really acceptance criteria for the evidence to be provided, i.e. the review.

The modular nature of Workflow$^+$ also allows for multiple normative workflows (i.e. templates) be be built hierarchically for different use cases. For example, a broad standard such as ISO 26262 [5] could be translated into a normative workflow that applies to all electrical and/or electronic systems within road vehicles. Then, this generic template could be refined to be more applicable to specific types of systems, such as adaptive cruise control and automatic braking. Then, it could be refined yet again to apply to specific vehicle platforms. This is analogous to optional paths in GSN-style templates (see [28]). The benefits of this hierarchical Workflow$^+$ approach include:

- *ease of audit/validation* – as many templates share as much structure as possible;
- *repeatability* – as each execution will follow many of the same processes; and
- *potentially increased productivity* – as widely-applicable tools for model transformation can be used to carry out refinements and instantiate templates.

4.3 An Example from the Automotive Domain

To illustrate a real-world Workflow$^+$ model and its role as an assurance case template, a normative workflow derived from ISO 26262 [5] (referred to as "the standard") is provided. It is important to note that because of the complexity of the standard, full Workflow$^+$ models of it are too complex to present in this thesis. This normative workflow is only meant to demonstrate how standards can be encoded as Workflow$^+$ metamodels, and does not completely capture all information contained within the portions of the standard modelled. To provide context and ease understanding, we will introduce the normative workflow of the standard by starting at an abstract level and decomposing until a sufficient level of detail is reached.

4.3.1 Abstract Model

We begin by creating a model of the workflow defined by the standard at a very abstract level. Using Figure 4.2 (figure 1 from ISO 26262-3 [29]), we can model this abstract workflow as shown in Figure 4.3. In this workflow, we can see that at a high level the standard consists of 5 processes, each producing one work product: Concept Phase; Product Development at the System Level; Product Development at the Hardware Level; Product Development at the Software Level; and Production, Operation, Service and Decommissioning. The remaining boxes in Figure 4.2 numbered 1, 2, 8, 9 and 10 do not represent processes, but rather supporting information that is to be integrated into the processes modelled where needed. We assume that each work product is be used by all subsequent processes, that each work product will have traceability with all other work products, and that the entire process begins with some

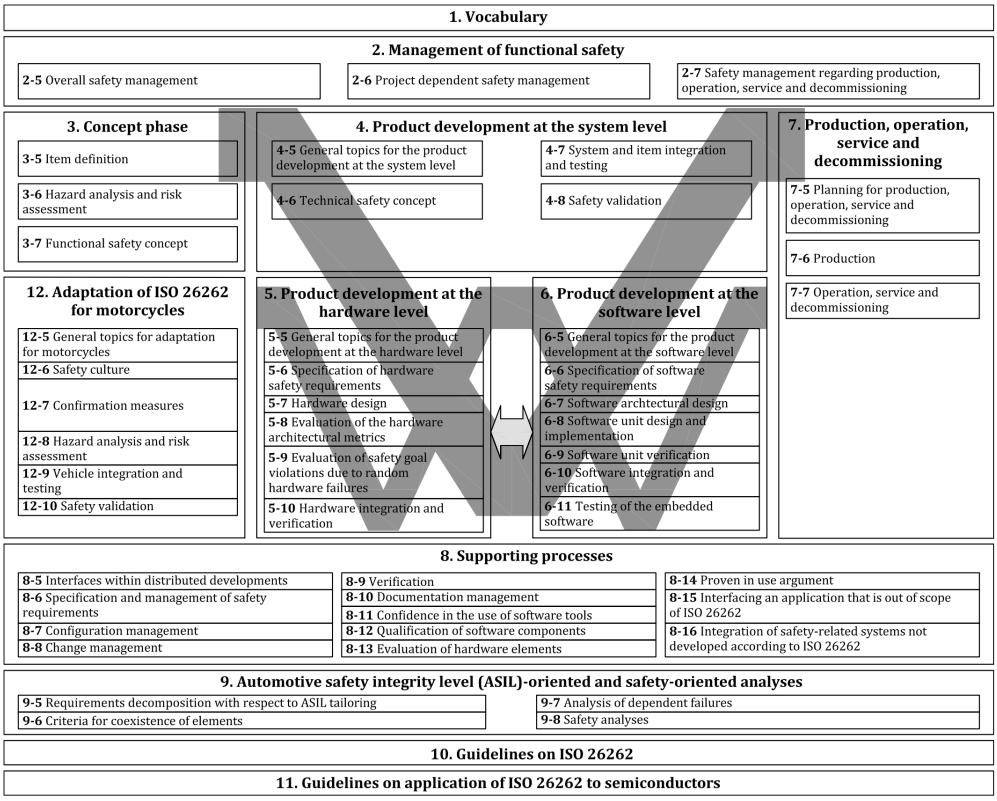

Figure 4.2: V-model as shown in [5]

informal requirements for the system under consideration. Note that there is an double-headed arrow between Product Development at the Hardware Level and Product Development at the Software Level in Figure 4.2 indicating that these processes are carried out iteratively, which is captured by including a loop from $\mathsf{WorkProducts}(4)$ back to $3 - \mathsf{ProductDevelopmentattheHardwareLevel}$. Argumentation for this level of abstraction are not provided by the standard. Thus, we have a Workflow$^+$ metamodel capturing the process dictated by the standard at an abstract level. Note that multicoloured arrows are used only to improve readability, and multiplicities are omitted to avoid clutter (for those interested, they would all be 1).

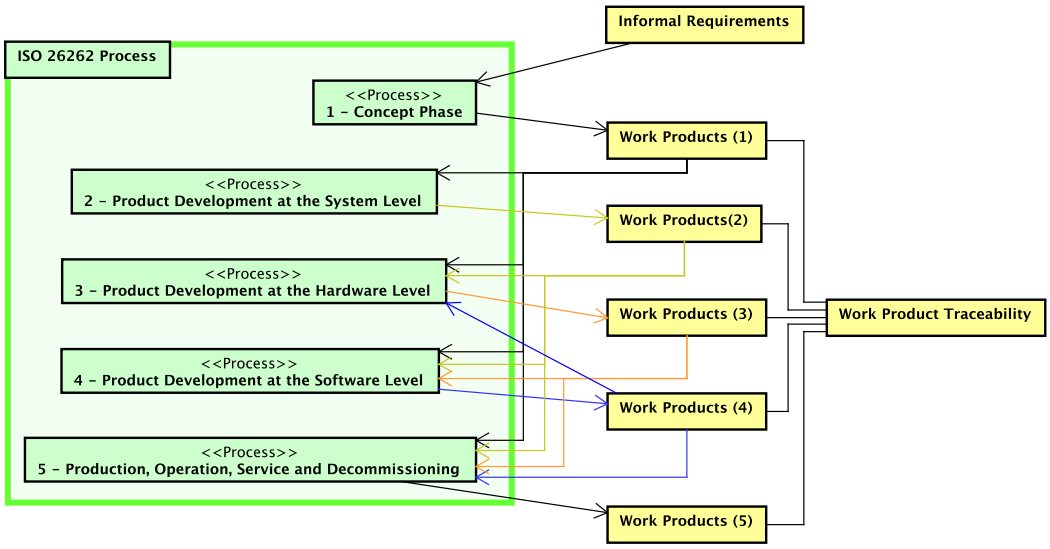

Figure 4.3: High-level Workflow$^+$ metamodel of the process described in ISO 26262 [5]

4.3.2 First Refinement: Concept Phase

Now, we decompose the process $1 - \mathsf{ConceptPhase}$ and its associated data. Figure 4.4 shows a close up of the Concept Phase portion of Figure 4.2, where we can see that the Concept Phase consists of 3 subprocesses: Item Definition, Hazard Analysis and Risk Assessment (HARA), and Functional Safety Concept. Looking further into ISO 26262-3 [29], Table A.1 conveniently lists the work product(s) of each process, which in combination constitute $\mathsf{WorkProducts}(1)$. This is captured in the Workflow$^+$ metamodel in Figure 4.5. In this metamodel, informal requirements for the system in consideration are used to create the item definition. Following this, HARA is is carried out using the item definition. Finally the Functional Safety Concept is carried out using the Item Definition and HARA Report.

It is at this level of detail that the concrete objectives of the processes within

the standard become apparent. Table A.1 in ISO 26262-3 [29] describes these objectives. As is suggested by ISO 26262-10 [30] section 5.3.1, we interpret these objectives as the high-level arguments that must be demonstrated by the work products in the concept phase.

In a Workflow$^+$ setting, we can formulate these as high-level arguments that refinements of the workflows producing these work products must implement (see Section /refnormativesection). This is also shown in Figure 4.5, but with slightly different notation than was used in Section 3.7. In the interest of compact and readable diagrams, we mapped these arguments only to the data produced by a workflow, rather than to the entire workflow itself. Formally, this is to be interpreted as applying to the entire workflow producing the output data mapped to an argument, including input data, output data, constraints and process definitions. For example, the argument attached to ItemDefinition indicates that it applies to the entire Workflow$^+$ definition of the ItemDefinition process, including InformalRequirements, ItemDefinition, the associations between these elements, and the constraints over this metamodel.

To reiterate, these arguments are included to express what the argumentation over each workflow must achieve, but do not specify how it is to be achieved. The derived arguments, however, remain blue as they are derived from the arguments as shown, regardless of how the white arguments are implemented.

Note that while the standard does not provide an objective for the Concept Phase as a whole, we were still able to derive a higher-level argument for the entire Concept Phase. This argument could be included in the higher-level metamodel in Figure 4.3.

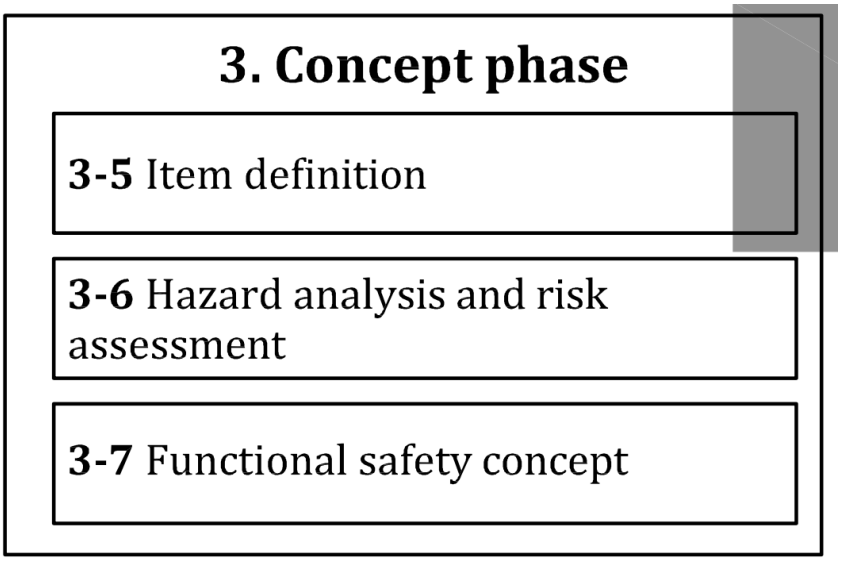

Figure 4.4: Concept phase as shown in ISO 26262-3 [5]

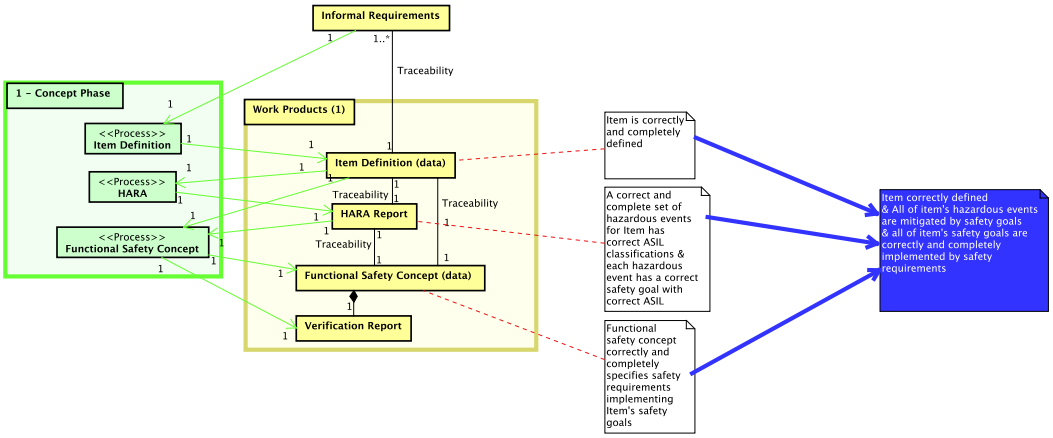

Figure 4.5: Workflow$^+$ metamodel of the concept phase as described in ISO 26262-3 [29]

4.3.3 Second Refinement: HARA

Next we proceed by decomposing the HARA process in Figure 4.6. We begin by creating an abstract outline of the clauses described in ISO 26262-3 [29] section 6.4, where there are 3 subsections that describe the processes to be carried out by HARA: Situation analysis and hazard identification, Classification of hazardous events, and Determination of safety goals. When each of these processes is executed, it produces a part of the HARA report. Note that because the standard does not give the output of these clauses a name, for convenience we have used the name of the subsection as the name of its corresponding process and output.

Without considering details, it is clear that executions of these processes, in order of execution, must produce a correct and complete set of Hazardous Events, produce a correct ASIL classification for each Hazardous Event, and produce a correct Safety Goal for each Hazardous Event. According to the multiplicity constraints in the Workflow$^+$ metamodel, an execution of the process Situation Analysis and Hazard Identification produces one instance of Situation Analysis and Hazard Ide , which must include a complete set of Hazardous Events for the Item Definition used as input. This is then to used by many executions of Classification of Hazardous Event, which each produce a classification for one Hazardous Event (i.e. one instance of Classification of Hazardous Event (data) is produced per execution). Finally, each execution of Determine Safety Goal must take one Hazardous Event in Situation Analysis and Hazard Identification (data) along with its ASIL classification from Classification of Hazardous Event (data) and produce a safety goal (i.e. one instance of Determination of Safety Goal (data) per execution).

We can formulate the objectives of each process as arguments that refine-

ments of the workflows producing the data shown must implement. In Figure 4.6 we include these objectives as arguments mapped to the data produced by their respective processes (recall that these arguments are to apply to the entire workflow producing the data with a mapping to an argument). Again, these arguments are white to indicate that they require implementation. These arguments can then be composed to show how they are an implementation of the argument on HARA as a whole.

These three HARA processes can now be further decomposed. It is at this next level of decomposition that things get more interesting - the standard provides a great deal of detail on how the processes within HARA are to be executed, the data produced and how they are interrelated in ISO 26262-3 [29] section 6.4. We will treat the following as the final decomposition step and add syntactic and semantic arguments to the process definitions. In the interest of compact and readable diagrams, different notation will be used to show how arguments relate to constraints, but the meaning is the same as explained in 3.6. Again, note that our model of ISO 26262-3 [29] section 6 is not necessarily complete; we are only illustrating how it can be modelled using Workflow$^+$, not providing a complete model.

4.3.4 Third Refinement (1) - Situation Analysis and Hazard Identification

Figure 4.7 shows the decomposition of Situation Analysis and Hazard Identification. Through analysis of the clauses in ISO 26262-3 [29] section 6.4.2, we have modelled the data that is to be documented and their relationships. First, we notice that the operational situations of items must be defined. For

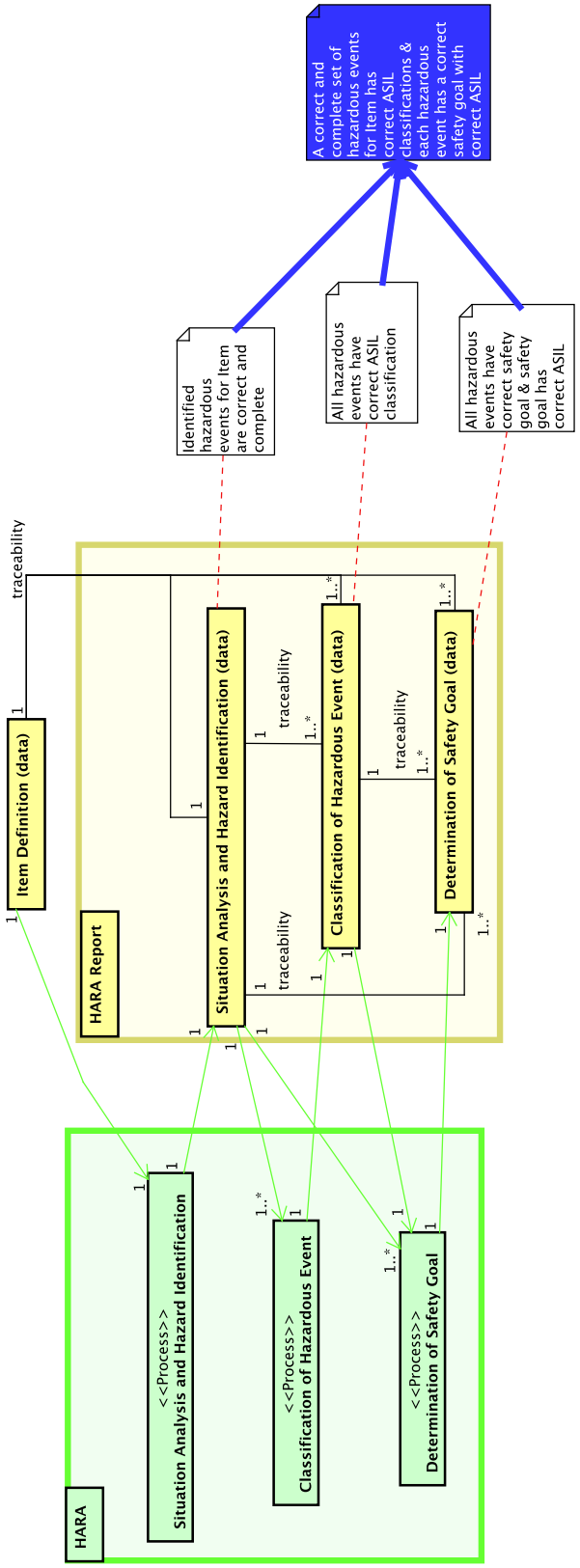

Figure 4.6: Workflow$^+$ metamodel of HARA as described in ISO 26262-3 [29]

convenience in adding constraints, we say that an item has a group of Intended Operational Situations (Operational Situation Set) that consists of all Intended Operational Situations. Next, we see that an Item is analysed to determine the set of vehicle-level hazards that it's malfunctioning behaviour can potentially cause. Again, we model this using a group of hazards (Hazard Set) that consists of all Vehicle Level Hazards identified for an Item. These Vehicle Level Hazards and Intended Operational Situations of an item are then analysed to determine when they, in combination, can lead to a Hazardous Event, producing a group of Hazardous Events related to an Item.

Over this refined data, we can add constraints whose associated arguments support the (white) argument of Situation Analysis and Hazard Identification as shown in Figure 4.8. First, we place syntactic constraints on Operational Situation Set and Hazard Set that ensure they are in fact present in an instance. Next, we place semantic constraints on the same data requiring their instances to be reviewed for correctness and completeness. Next, we place similar syntactic and semantic constraints on Hazardous Event Set requiring it to be present and reviewed for correctness and completeness in an instance. Finally, with all of these syntactic and semantic constraints and their associated arguments, we can derive higher level arguments which, as shown in Figure 4.8, do in fact implement the argument on Situation Analysis and Hazard Identification.

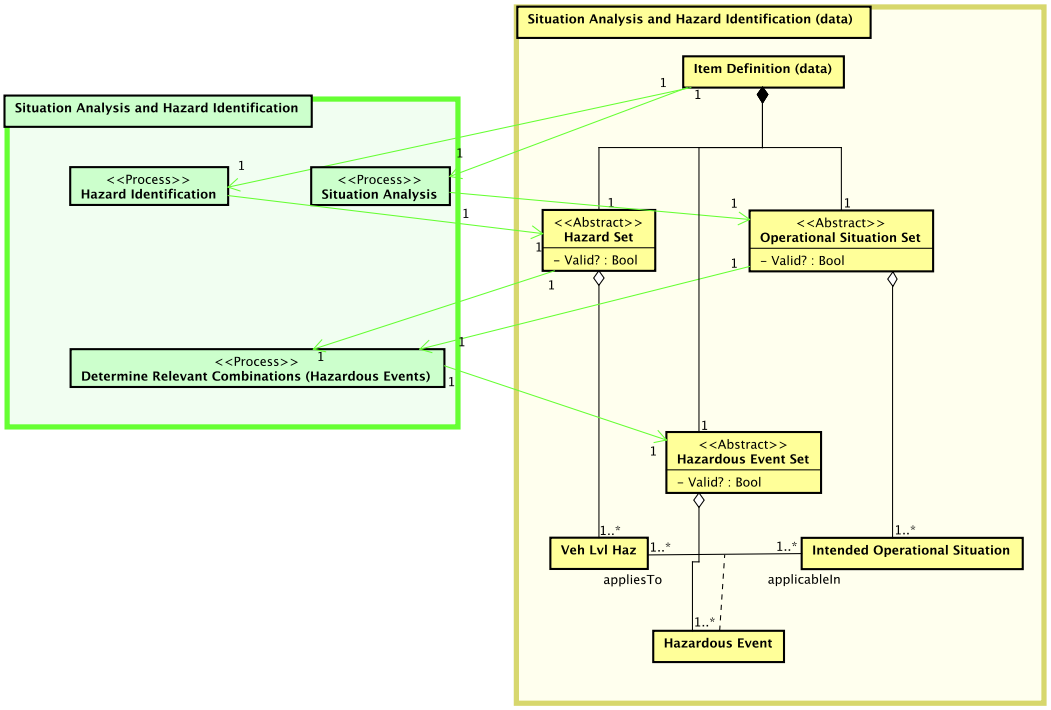

Figure 4.7: Workflow$^+$ metamodel of situation analysis and hazard identification as described in ISO 26262-3 [29]

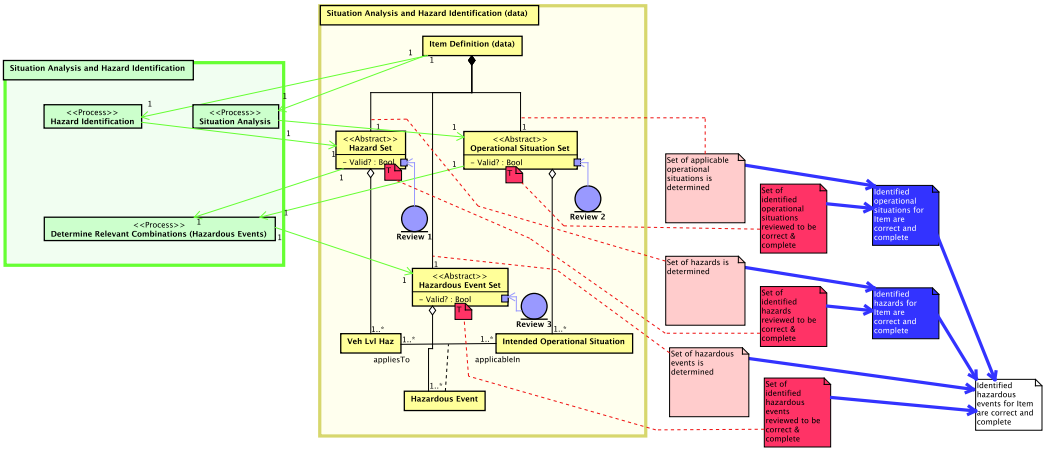

Figure 4.8: Workflow$^+$ metamodel of situation analysis and hazard identification with argumentation as as described in ISO 26262-3 [29]

4.3.5 Third Refinement (2) - Classification of Hazardous Event

Figure 4.9 shows the decomposition of Classification of Hazardous Event. Each execution of this process takes a hazardous event identified by an execution of Situation Analysis and Hazard Identification as input, and first determines the Severity and Controllability of the Hazardous Event, along with the Exposure of the operational situation involved in that Hazardous Event. Finally, the Hazardous Event has an ASIL computed based on these attributes. Because computing the ASIL amounts to using the look up table in ISO 26262-3 [29]Table 4, the ASIL determination can be implemented as an automated Query that takes in a Hazardous event along with its Severity, Exposure and Controllability. Note that according to the multiplicities, Classification of Hazardous Event must be executed once for each hazardous event.

Note that here, it can be seen why it is necessary to be able to model processes that output relationships between input and output data. The data from Situation Analysis and Hazard Identification is not simply used as input data, it is built upon, and for safety assurance we need to be able to model this accurately in order to assure that the process was correctly followed.

Over this refined data, we can add constraints whose associated arguments support the (white) argument of Classification of Hazardous Event as shown in Figure 4.10. First, we place syntactic constraints on all data requiring that they are present in an instance. Next, we add semantic constraints over Severity, Controllability and Exposure requiring their instances to be reviewed for correctness. Because Look Up ASIL is an automated query, instead of reviewing its output we can certify the Query to always output correct results

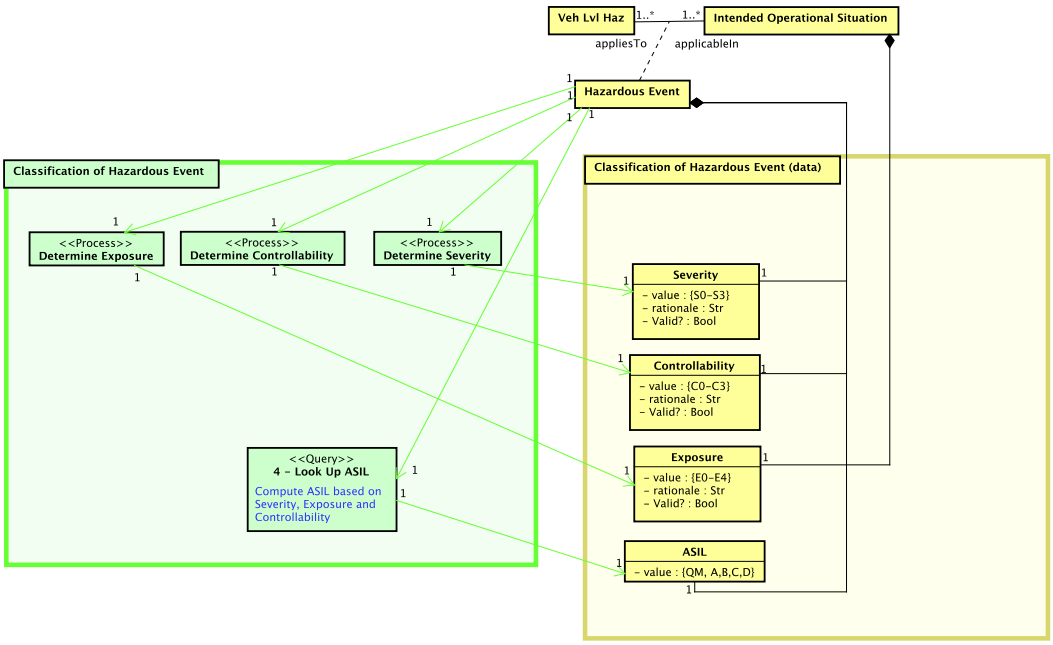

Figure 4.9: Workflow$^+$ metamodel of classification of hazardous events as described in ISO 26262-3 [29]

given correct input, shown by the constraint on Look Up ASIL saying that it is certified. This is in many ways similar to compiler certification, where compilers used for safety-critical applications are, or should be, certified to generate correct code.

4.3.6 Third Refinement (3) - Determination of Safety Goal

Figure 4.11 shows the decomposition of Determination of Safety Goal. Each execution of this process takes a Hazardous Event along with its ASIL and computes a Safety Goal to mitigate that Hazardous Event. Following this, through a query the Safety Goal is traced to the ASIL ratings of the Hazardous Event it mitigates, and assigned an ASIL equal to the highest ASIL of all

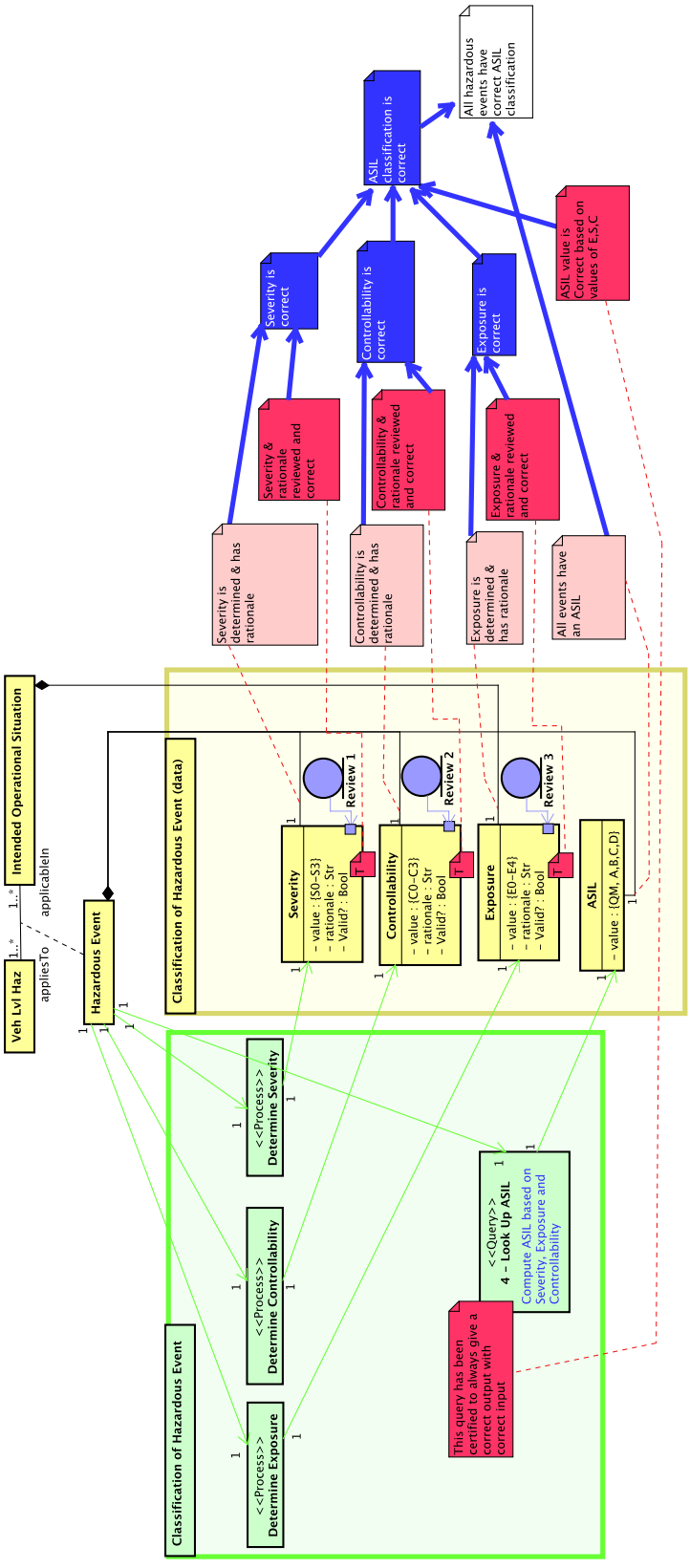

Figure 4.10: Workflow$^+$ metamodel of classification of hazardous events with argumentation as described in ISO 26262-3 [29]

mitigated Hazardous Events if there is more than one.

Over this refined data, we can add constraints whose associated arguments support the (white) argument of Determination of Safety Goals as shown in Figure 4.12 . First, we place a syntactic constraint requiring a Safety Goal to be present in an instance and add a semantic constraint requiring that the safety goal is reviewed for correctness, allowing us to say that the safety goal is correct. Recall the mention of acceptance criteria in Section 4.2 – the acceptance criteria of this review could, for example, involve checking that: i) the safety goal is described in sufficient detail to be able to determine its scope and effect; ii) the safety goal describes reasoning as to why and how the safety goal mitigates the hazard; iii) any prior history of that safety goal being used to mitigate that hazard is described; and iv) sufficient references to back up the claim that the safety goal mitigates that hazard are included. Next, we add a constraint that the Query Assign ASIL must be certified and that a Safety Goal must be associated with an ASIL, allowing us to say that each Safety Goal is assigned the correct ASIL.

4.3.7 Tying it All Together

Thus, we have defined a partial decomposition of the standard, ranging from the most abstract level to the lowest level of HARA. The processes that remain can also be decomposed using information provided by the standard, but for the purpose of this thesis we stop our decomposition here. Figure 4.13 illustrates this decomposition. In it, we show that we began by defining a high-level Workflow$^+$ model of the standard with constraints/argumentation omitted (Figure 4.3). Then, we decomposed the Concept Phase while leaving

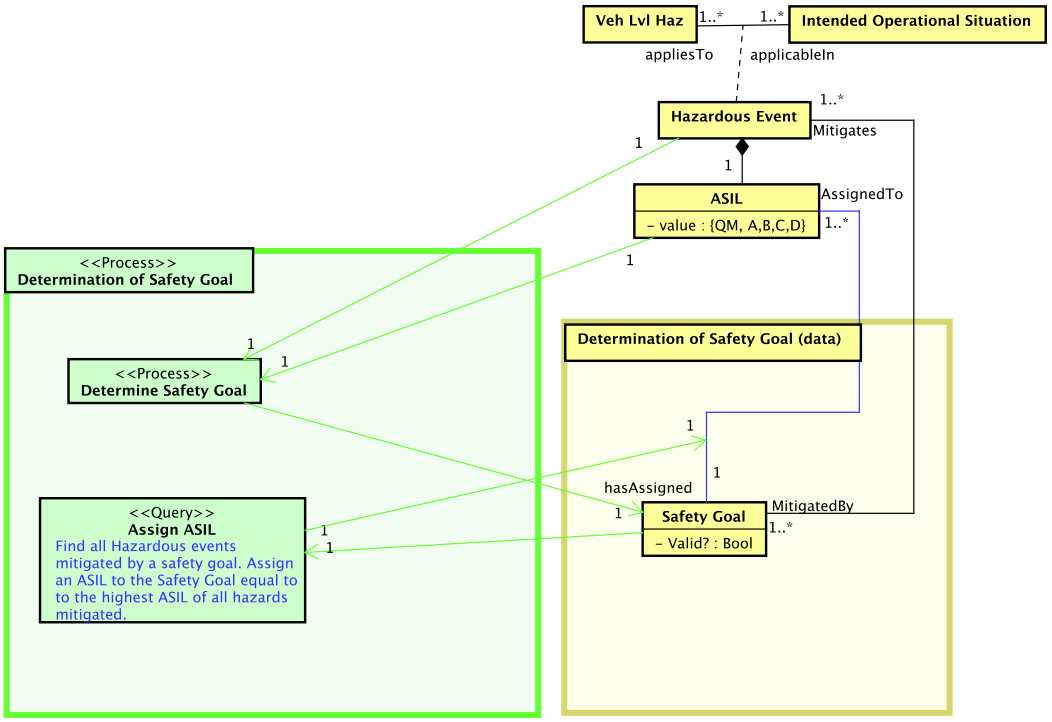

Figure 4.11: Workflow$^+$ metamodel of determination of safety goal as described in ISO 26262-3 [29]

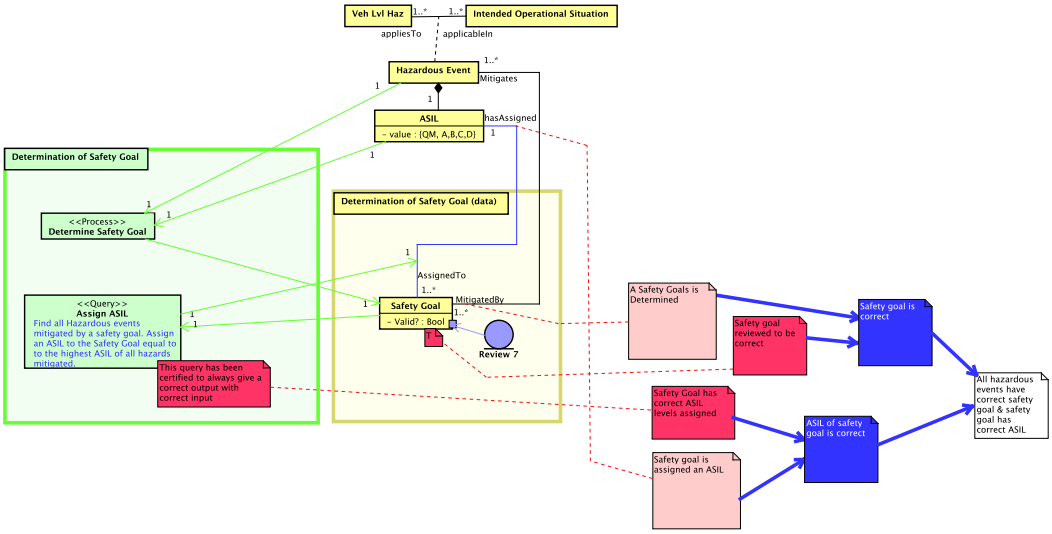

Figure 4.12: Workflow$^+$ metamodel of determination of safety goal with argumentation as described in ISO 26262-3 [29]

the remaining four processes as black boxes (Figure 4.5). Next, we decomposed the HARA portion of the Concept Phase, leaving Item Definition and Functional Safety Concept as black boxes (Figure 4.6). Following this, we refined the three processes within HARA (Figure 4.8, Figure 4.10, Figure 4.12). Of course, there are traceability and input/output relationships between these decomposed processes as shown previously, which are omitted for simplicity.

4.3.8 A More Specific Template

Because the standard is intended to apply to all electronic/electrical systems within road vehicles, a complete Workflow$^+$ metamodel based on the standard would serve as a $\mathsf{Wf}_{\mathsf{Norm}}$ for any electronic/electrical system within a road vehicle. That is, the $\mathsf{Wf}_{\mathsf{Norm}}$ is a template for the design and analysis of all electronic/electrical systems within road vehicles.

This $\mathsf{Wf}_{\mathsf{Norm}}$ could then be decomposed to apply to a specific subset of electronic/electrical systems within vehicles. To illustrate this, we will use an example where the Situation Analysis and Hazard Identification is decomposed to apply specifically to systems within pickup trucks. During the decomposition, the Intended Operational Situations of pickup trucks are determined ahead of time and fixed (i.e. cannot be changed). This is illustrated in Figure 4.14, where the Operational Situation Set for pickup trucks has been changed to grey to indicate that it is fixed, and the Situation Analysis process has been removed as it does not need to be executed in this refinement. These Intended Operational Situations include city driving, highway driving, towing, etc., for example. Now, for assurance, this set of Operational Situations is reviewed to be correct beforehand, so we make the constraints grey to indicate that they

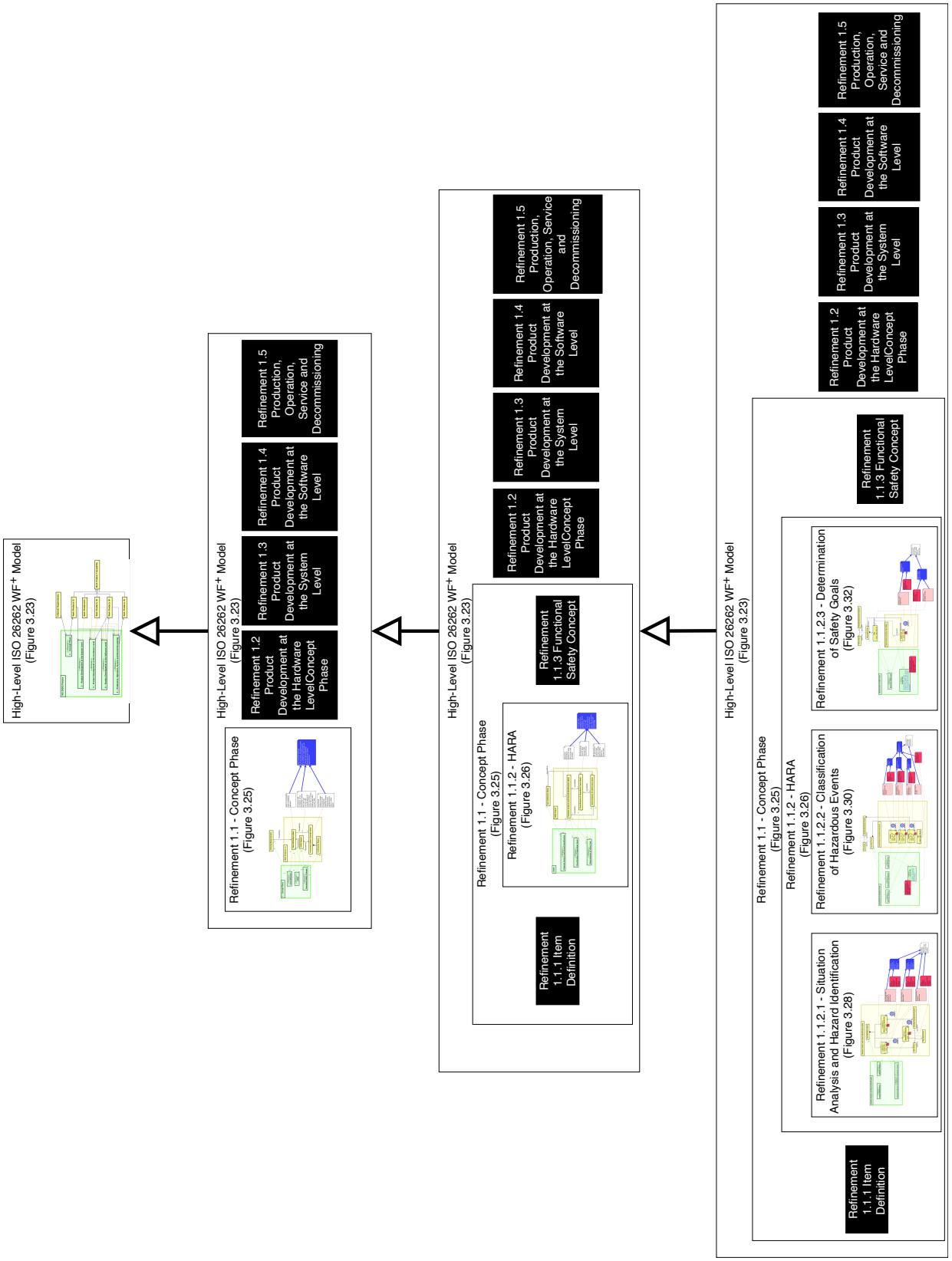

Figure 4.13: Overview of decomposition of the processes within ISO 26262 [5]

are already checked. Note that the rest of the constraints have been omitted from the refined model to improve readability, but they are exactly the same as in the unrefined model.

Now, with this set of Intended Operational Situations fixed, the Exposure classification of these Intended Operational Situations can also be determined ahead of time. Figure 4.14 shows an illustrative example in which the city and highway driving Exposure ratings have been fixed at E4, and the towing Exposure rating fixed at E2.

Thus, we have demonstrated how a normative workflow can be decomposed to apply to more specific types of systems. In doing this, we have effectively reduced the workload in executing the $\mathsf{Wf}_{\mathsf{Norm}}$prescribed by the standard for a pickup truck by only requiring the execution of Situation Analysis once to be used for any electronic/electric system in a pickup truck, rather than having it re-executed every time. Obviously, there is opportunity to decompose tasks that are much more time consuming than determining operational situations, and when done at scale, provides a great opportunity for improved productivity. Additionally, in making this refined template we have also reduced the burden on those reviewing the SEP execution by only requiring the set of operational situations to be reviewed once, rather than after each execution of Situation Analysis.

To anyone with knowledge of how SEPs are executed in practice, it is obvious that the idea of reusing previously determined data is done often, and was not invented by us. What we have done, however, is provided a means of accurately documenting this reuse and integrating it directly into an assurance case. Non-model-based approaches to this same reuse often result in the previously mentioned issues related to productivity and reviewing.

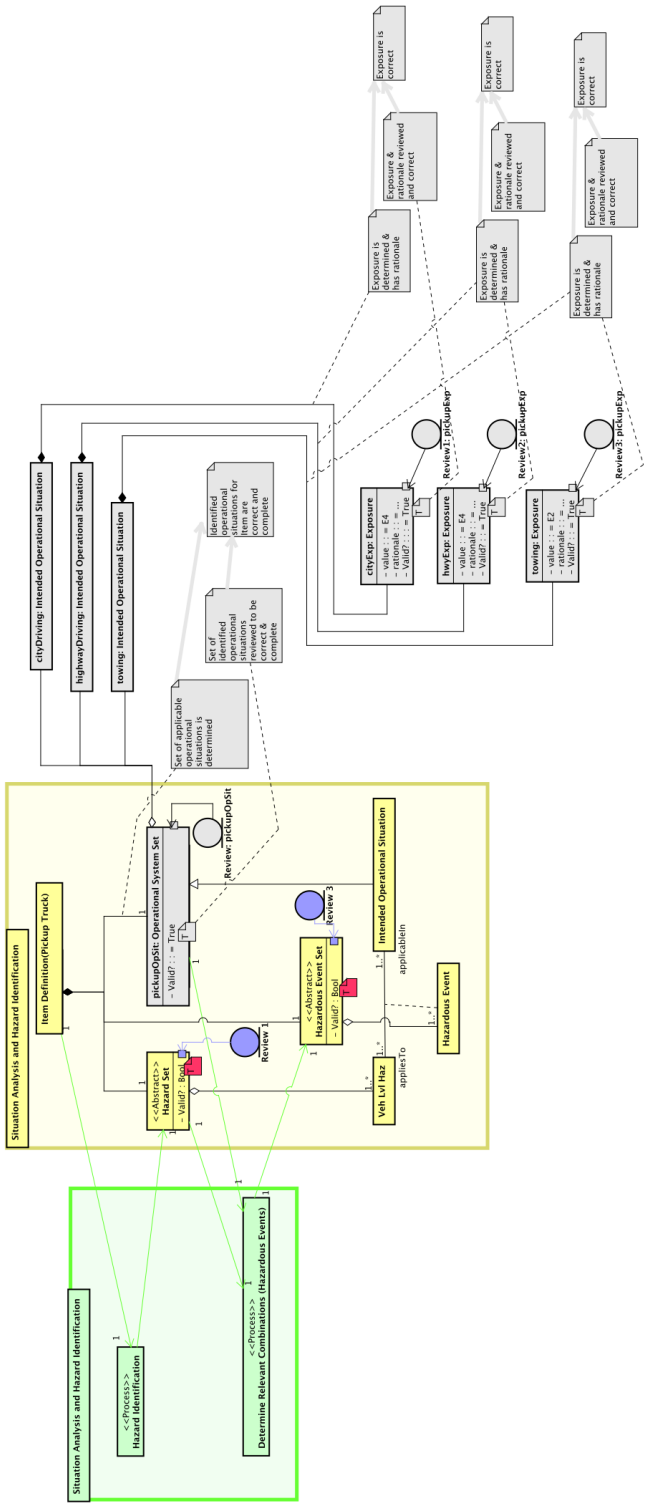

Figure 4.14: An example of a Workflow$^+$ template for Situation analysis and hazard identification (see Figure 4.8)

Chapter 5

Discussion

In this section, we discuss several benefits of the Workflow$^+$ approach to safety assurance. These benefits are divided into two sections, one on the advantages of model-based documentation for safety assurance, and the other on the potential for tool support and automation of Workflow$^+$ based assurance cases.

5.1 Advantages of Model-Based Documentation

Through experience trying to use current approaches to safety assurance in a model-based setting and using Workflow$^+$, we have gained insight into the advantages and disadvantages of model-based documentation using these approaches. We discuss these advantages and disadvantages in the following subsections.

5.1.1 Making Assurance Less Ad-hoc

Current approaches to assurance can be ad-hoc and reliant upon how individual safety experts interpret the safety requirements, the certification standards, and the syntax itself. More specifically, there is no precisely defined methodology for assurance when using GSN; it relies on individual engineer expertise ([3] goes into more detail on this). There are benefits to this, as engineers can optimize the process of constructing an assurance case based on their experience in their respective domains, but this comes at the price of repeatability and learnability. An important benefit of using Workflow$^+$ is the clear methodology that is to be followed when build assurance arguments; this will ideally enable engineers to more readily learn the techniques, and for the steps to be repeatable.

In comparison with GSN, the semantics for each element of Workflow$^+$, as well as the role each element plays in the assurance process, are precisely defined. This should lead to fewer opportunities for misinterpretation. Additionally, Workflow$^+$'s structure makes managing large assurance cases more systematic, repeatable and inexpensive.

5.1.2 Improved Traceability

Detailed traceability is essential for assurance cases of complex systems because it improves understandability and allows for all necessary information to be directly linked to assurance arguments. Current approaches lack mechanisms to include direct traceability and often rely on implicit traceability between arguments (See [3]). This implicit traceability is manageable for small-scale assurance case, but in large-scale industrial safety cases it is often much more

difficult to identify and understand traceability that is left implicit, especially when cross-cutting concerns branch over multiple argument legs. This places an undue burden on independent reviewers to discover this implicit structure on their own, and when compounded with the ad-hoc structure typical of GSN-style safety cases can lead to significant misunderstanding of the intended argument structure. This prevents independent reviewers from being able to review an assurance case with sufficient/required certainty and gets in the way of identifying subtle but potentially dangerous flaws in arguments.

Workflow$^+$ was developed from the ground-up with this in mind and enables detailed data-to-data and data-to-process traceability. All traceability necessary between data and processes underlying arguments is maintained explicitly, eliminating the issues related to implicit data-to-data and data-to-process traceability, and allowing for cross-cutting concerns to be accurately and explicitly represented. When paired with the well-defined structure of Workflow$^+$ assurance cases, Workflow$^+$ provides significantly improved understandability and is better suited for independent reviewing.

5.1.3 Improved Granularity

As was demonstrated, Workflow$^+$ metamodels can be decomposed in a modular manner to an arbitrary level of granularity. In theory, current approaches such as GSN can also be refined to an arbitrary level of granularity, but people tend to avoid doing this. Process refinement and decomposition is a well-understood procedure that can be formally defined and is intuitive. In contrast, GSN assurance case decomposition would be very hard due to the level of implicit information: when one accurately decomposes a GSN diagram, they

run the risk of destroying the implicit structure. Moreover, the navigability and comprehensibility of GSN-style safety cases suffer as higher levels of detail are added due to the presence of cross-cutting concerns and lack of explicit traceability. Workflow$^+$ can provide an advantage over current approaches by providing mechanisms that manage the complexity of fine-grained assurance cases, namely a) maintaining explicit traceability and b) enabling modular process and data decomposition, which improves understandability.

5.2 The Potential for Tool Support and Automation

Through experience trying to use current approaches to safety assurance in a model-based setting and using Workflow$^+$, we have gained insight into the possibilities for tool support and automation using these approaches. In the following subsections, we discuss these advantages and disadvantages.

5.2.1 Impact Analysis

When dealing with highly complex embedded systems, it can be difficult to determine the impact of incremental design changes on the system's assurance case. As systems continue to increase in complexity, even the most experienced engineers will have trouble keeping up with the thousands of connections between design and their respective elements of an assurance case. Continuing to do impact analysis "by hand" will result in increased cost, lost productivity and potentially safety concerns stemming from the inevitable human error when combing through thousands of data points. Current approaches lack the

traceability within/between work products that is necessary for build tooling supporting impact analysis.

The model-based nature of Workflow$^+$ provides the necessary foundations for impact analysis to be automated as much as is possible. The detailed granularity and traceability possible in Workflow$^+$ metamodels allow for tools to be built that can automatically follow traceability links to all related data and their associated arguments (i.e. automatically trace changed data to impacted processes/data/arguments), rather than just to work products that contain large quantities of information. This allows tools that point engineers directly to the processes that need to be re-executed as the result of a change. On top of this, the well-defined semantics of Workflow$^+$ metamodels and assurance cases allows for an explicit ontology of change propagation that enables a well-defined approach to assurance of incremental changes to systems.

5.2.2 Automation

As systems continue to increase in complexity, it is desirable to automate as much of the assurance case development as possible to reduce development costs. Building Workflow$^+$ on well-established model-based development (MBD) principles allows tool developers to leverage a wide range of pre-existing techniques for managing assurance cases, including automated querying to search for assurance cases, and transformations for applying templates.

Additionally, the model-based approach of Workflow$^+$ allows for static syntactic correctness to be checked automatically. As more granularity is added to the Workflow$^+$ metamodel, semantically significant properties can be encoded in the structure of the metamodel through the use of constraints. For example,

if there are certain structural properties of design-related elements desirable for safety, then the corresponding constraints can be placed on the metamodel to allow these properties to be automatically checked. Table 1 in ISO 26262-6 [31] outlines properties of software architectural design that are desirable for avoiding systematic faults. Many of these properties, such as restricted size of interfaces, restricted size of complexity of software components and loose coupling between software components are all good candidates for automatic checking through constraints over detailed models.

An MBD approach also allows for some processing required in a workflow to be automated, as was shown for the process Look Up ASIL in Figure 4.10. It is possible for these automated tasks to be certified to have trustworthy outputs with correct inputs (similar to how compilers can be qualified). Thus, time can be saved by automating the process itself, and also by not requiring the outputs of these automated processes to be reviewed.

5.2.3 Integrating Assurance with Development

The model-based approach of Workflow$^+$ opens up the opportunity for Workflow$^+$ models to be directly integrated with model-based development or V&V tools. This allows assurance to be built directly over data from development, rather than having an assurance case as a separate document with references to development documentation. With direct access to artefacts from development, some aspects of assurance cases to be automatically generated and validated based on the content of those design artefacts. While it is possible to integrate GSN approaches with development [32], integrating Workflow$^+$ with development will allow for more scalable solutions that are more well-suited for impact

analysis. Also, its direct traceability into the environment facilitates dealing with feature interactions that stretch into the environment.

5.3 Multiview Modelling

As is clear by this point, there are many ways of representing different Workflow$^+$ models. To reiterate what was said earlier, models or instances) are representations of reality designed for a specific purpose. In order to fulfil the purpose of a model, it is sometimes necessary to alter the representation or level of abstraction of a model to make it more useful or readily accessible (i.e. readable, understandable, etc.)

In the context of Workflow$^+$ the complexity of workflow definitions makes it particularly important to be able to abstract away unnecessary information depending on the needs of a workflow's users. For example, for a safety engineer executing a specific process from a workflow, it is not necessary for them to be presented any information other than what is directly relevant to the process they are executing. Another example is when presenting safety strategies to stakeholders who are not entirely familiar with the process - a higher-level view of the workflow is appropriate, rather than overloading them with every little detail of a workflow. The modularity and compositionality of Workflow$^+$ enable this by allowing individual parts of a workflow to be composed and decomposed as needed to provide the necessary level of abstraction and scope for a particular purpose.

Chapter 6

Evaluation

The Workflow$^+$ framework was developed during a collaborative research project with a large automotive OEM related to model-based approaches to managing assurance cases. This research project began by trying to find a model-based way to manage GSN-style safety cases, but we quickly came to realize that the potential for automation in this setting fell well short of what we wanted. Thus, Workflow$^+$ was born, and through collaboration with our industrial partners, we were able to develop Workflow$^+$.

The general feedback from our industrial partner was positive, and in their opinion Workflow$^+$ is a very promising way of modelling, documenting, managing and automating their safety assurance processes. Additionally, during an internship with our industrial partner, the author was able to work closely with them to further develop the Workflow$^+$ framework and make sure it was being developed in a way that would meet their needs. One of the projects worked on by the author was to conduct a survey of issues raised during manual reviews of safety-related work products, which found that 46% of issues with safety-related work products were syntactic issues and have the poten-

tial to be automatically detected using the model-based Workflow$^+$ approach, even before a review begins. This was seen as promising as, at the very least, it would improve the productivity of reviewers and allow them to focus on identifying more serious issues within work products.

This experience working with our industrial partner essentially served as an evaluation of Workflow$^+$ and from their feedback we conclude that Workflow$^+$ is indeed a useful contribution in the field of model-based safety assurance and has the potential to be more rigorous and repeatable in industrial projects than is currently achievable.

Chapter 7

Conclusion

This thesis began by discussing the issues surrounding current approaches to safety assurance. We discussed two of the most significant issues that must be overcome to improve the rigour of safety assurance and make them less ad-hoc: a) the absence of mathematical foundations for reasoning; and b) the lack of well-defined semantics for the building blocks of assurance cases. Together, these issues undermine the effectiveness of safety assurance and a regulator's ability to critically assess the safety of a system.

Workflow$^+$ has been presented as an an approach to safety assurance that addresses these issues. Its well-defined semantics and approach to reasoning/argumentation provide a promising way to overcome the issues of current approaches to safety assurance. Additionally, its model-based nature and traceability mechanisms facilitate the creation of powerful tool support based on established MBD principles, enabling automation of the management of safety cases including change impact analysis, and the tight integration of safety with development.

As systems become more complex, the limitations of traditional approaches

to safety assurance are becoming more difficult to manage. Workflow$^+$ is a framework that can improve upon traditional methods and provide solutions to their limitations, ultimately making the problem of safety assurance more manageable.

My contribution has been mainly to the development of Workflow$^+$. This has included creating and documenting detailed examples as part of producing deliverables for a research project with an industrial partner, and to be used in technical reports. I have worked closely with our industrial partner to understand their needs and ensure Workflow$^+$ addresses those needs. I have also contributed by developing explanations and detailed figures that form the basis of a recent publication on Workflow$^+$ and also aid in introducing Workflow$^+$ to practitioners and researchers who are new to this form of modelling.

7.1 Future work

Future work on the Workflow$^+$ framework falls into two categories: formalization and the creation of tooling. The idea of Workflow$^+$ and the role it plays in safety assurance are more or less clear as described in this thesis, but the framework is still in its conceptual phase. Formalization of many aspects of the framework are still works in progress, as is the implementation of tools for Workflow$^+$. As formalization and the creation of tooling are carried out, Workflow$^+$ will take on a more concrete and usable form.

When more fully formalized, and with adequate tool support, concrete design patterns, approaches to assurance in specific domains, automation capabilities and methodologies for assessing Workflow$^+$ assurance cases can be established and put into practice.

Bibliography

[1] Robin E. Bloomfield and Peter G. Bishop. "Safety and Assurance Cases: Past, Present and Possible Future - an Adelard Perspective". In: *Making Systems Safer - Proceedings of the Eighteenth Safety-Critical Systems Symposium, Bristol, UK, February 9-11, 2010*. 2010, pp. 51–67. URL: https://doi.org/10.1007/978-1-84996-086-1_4 (cit. on p. 1).

[2] Alan Wassyng et al. "Software Certification: Is There a Case against Safety Cases?" In: *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems - 16th Monterey Workshop 2010, Redmond, WA, USA, March 31- April 2, 2010, Revised Selected Papers*. 2010, pp. 206–227. URL: https://doi.org/10.1007/978-3-642-21292-5_12 (cit. on pp. 1, 2, 8, 42).

[3] Zinovy Diskin et al. *Assurance via workflow+ modelling and conformance*. McSCert Technical Report. 2019. arXiv: 1912.09912 [cs.SE] (cit. on pp. 3, 7, 9, 26, 30, 31, 77).

[4] John Hutchinson et al. "Empirical Assessment of MDE in Industry". In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. Waikiki, Honolulu, HI, USA: Association for Com-

puting Machinery, 2011, 471–480. ISBN: 9781450304450. URL: https://doi.org/10.1145/1985793.1985858 (cit. on p. 5).

[5] ISO. *ISO 2626: Road Vehicles – Functional Safety*. version 2. 2018 (cit. on pp. 6, 9, 56–59, 61, 73).

[6] Nicholas Annable et al. "Model-driven safety of autonomous vehicles". In: Consortium for Software Engineering Research, 2020 (to appear) (cit. on p. 7).

[7] Tim Kelly. "Arguing Safety – A Systematic Approach to Managing Safety Cases". PhD thesis. University of York, 1998 (cit. on p. 8).

[8] Adelard LLP. *Adelard Safety Case Development Manual*. Tech. rep. http://www.adelard.com/resources/ascad/. 1998 (cit. on p. 8).

[9] David J Rinehart, John C Knight, and Jonathan Rowanhill. *Current Practices in Constructing and Evaluating Assurance Cases With Applications to Aviation*. 2015 (cit. on p. 8).

[10] John Rushby. *Understanding and Evaluating Assurance Cases*. Technical Report SRI-CSL-15-01. SRI International, 2015 (cit. on p. 8).

[11] Sunil Nair et al. "An extended systematic literature review on provision of evidence for safety certification". In: *Information & Software Technology* 56.7 (2014), pp. 689–717. URL: https://doi.org/10.1016/j.infsof.2014.03.001 (cit. on p. 8).

[12] A. Wassyng et al. "Can Product-Specific Assurance Case Templates Be Used as Medical Device Standards?" In: *IEEE Design Test* 32.5 (2015), pp. 45–55. ISSN: 2168-2364 (cit. on pp. 8, 54).

[13] John Rushby. "Formalism in safety cases". In: *Making Systems Safer*. Springer, 2010, pp. 3–17 (cit. on p. 8).

[14] Ewen Denney and Ganesh Pai. "Tool support for assurance case development". In: *Automated Software Engineering* 25.3 (2018), pp. 435–499. ISSN: 1573-7535. URL: https://doi.org/10.1007/s10515-017-0230-5 (cit. on p. 8).

[15] OMG. *Object Management Group Structured assurance case metamodel - SACM*. 2019. URL: https://www.omg.org/spec/SACM/About-SACM/ (cit. on p. 9).

[16] Ran Wei et al. "Model based system assurance using the structured assurance case metamodel". In: *Journal of Systems and Software* 154 (2019), 211–233. ISSN: 0164-1212. URL: http://dx.doi.org/10.1016/j.jss.2019.05.013 (cit. on p. 9).

[17] Yakoub Nemouchi et al. "Isabelle/SACM: Computer-Assisted Assurance Cases with Integrated Formal Methods". In: *Integrated Formal Methods*. Ed. by Wolfgang Ahrendt and Silvia Lizeth Tapia Tarifa. Cham: Springer International Publishing, 2019, pp. 379–398. ISBN: 978-3-030-34968-4 (cit. on p. 9).

[18] Yaping Luo et al. "Extracting Models from ISO 26262 for Reusable Safety Assurance". In: *Safe and Secure Software Reuse - 13th International Conference on Software Reuse, ICSR 2013, Pisa, Italy, June 18-20. Proceedings*. 2013, pp. 192–207. URL: https://doi.org/10.1007/978-3-642-38977-1_13 (cit. on p. 9).

[19] 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, Saint-Malo, France, October 2-7, 2016. 2016, pp. 196–206. URL: http://dl.acm.org/citation.cfm?id=2976792 (cit. on p. 9).

[20] OMG. *Object Management Group Software & Systems Process Engineering Metamodel - SPEM*. 2008. URL: https://www.omg.org/spec/SPEM/About-SPEM/ (cit. on p. 9).

[21] Julio A. Hurtado Alegría et al. "An MDE Approach to Software Process Tailoring". In: *Proceedings of the 2011 International Conference on Software and Systems Process*. ICSSP '11. Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, 43–52. ISBN: 9781450307307. URL: https://doi.org/10.1145/1987875.1987885 (cit. on p. 9).

[22] Zinovy Diskin et al. "Assurance via model transformations and their hierarchical refinement". In: *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018*. 2018, pp. 426–436. URL: https://doi.org/10.1145/3239372.3239413 (cit. on pp. 9, 11).

[23] Zinovy Diskin, Abel Gómez, and Jordi Cabot. "Traceability Mappings as a Fundamental Instrument in Model Transformations". In: *Fundamental Approaches to Software Engineering - 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*. 2017, pp. 247–263. URL: https://doi.org/10.1007/978-3-662-54494-5_14 (cit. on p. 11).

[24] Thomas Kühne. "Matters of (Meta-) Modeling". In: *Software & Systems Modeling* 5.4 (2006), pp. 369–385. ISSN: 1619-1374. URL: https://doi.org/10.1007/s10270-006-0017-9 (cit. on p. 17).

[25] Thomas Kühne. "On model compatibility with referees and contexts". In: *Software & Systems Modeling* 12.3 (2013), pp. 475–488. ISSN: 1619-1374. URL: https://doi.org/10.1007/s10270-012-0241-4 (cit. on p. 21).

[26] Gregor Kiczales et al. "Aspect-oriented programming". In: *ECOOP'97 — Object-Oriented Programming*. Ed. by Mehmet Akşit and Satoshi Matsuoka. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 220–242. ISBN: 978-3-540-69127-3 (cit. on p. 25).

[27] Erik Hilsdale and Jim Hugunin. "Advice Weaving in AspectJ". In: *Proceedings of the 3rd International Conference on Aspect-oriented Software Development*. AOSD '04. Lancaster, UK: ACM, 2004, pp. 26–35. ISBN: 1-58113-842-3. URL: http://doi.acm.org/10.1145/976270.976276 (cit. on p. 25).

[28] T. Chowdhury et al. "Principles for Systematic Development of an Assurance Case Template from ISO 26262". In: *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 2017, pp. 69–72 (cit. on p. 56).

[29] ISO. *ISO 2626: Road Vehicles – Functional Safety – Part 3: Concept phase*. version 2. 2018 (cit. on pp. 57, 59–64, 66–69, 71).

[30] ISO. *ISO 2626: Road Vehicles – Functional Safety – Part 10: Guidelines on ISO 26262*. version 2. 2018 (cit. on p. 60).

[31] ISO. *ISO 2626: Road Vehicles – Functional Safety – Part 6: Product development at the software level*. version 2. 2018 (cit. on p. 81).

[32] R. Hawkins et al. "Weaving an Assurance Case from Design: A Model-Based Approach". In: *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*. 2015, pp. 110–117 (cit. on p. 81).