Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 191 (2007) 3–23

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Frameworks Based on Templates for Rigorous Model-driven Development

Nuno Amálio[1] Fiona Polack[2] Susan Stepney[3]

Department of Computer Science, University of York
York, YO10 5DD, UK

Abstract

The engineering of systems that are acceptably correct is a hard problem. On the one hand, semi-formal modelling approaches that are used in practical, large-scale system development, such as the UML, are not amenable to formal analysis and consistency checking. On the other hand, formal modelling and analysis requires a level of competence and expertise that is not common in commercial development communities, and formal approaches are not well integrated with the rest of the development process. This paper advocates an approach to building engineering environments (or frameworks) for rigorous model-driven development (MDD) that is based on combining semi-formal notations with formal modelling languages. To support the approach, there is a formal language of templates, which captures patterns of formal development and enables an approach to proof with templates. This allows the construction of catalogues of patterns (represented as templates) and meta-theorems for frameworks. The paper presents and illustrates a framework for sequential systems that combines UML and the formal language Z.

Keywords: UML, MDD, Z, patterns, templates.

1 Introduction

Model-driven development (MDD) [17] tries to raise the level of abstraction by electing models, rather than code, as primary artifacts of software development. Models describe the domain and required behaviour of a system, being useful in the stages of construction and maintenance of software. The analysis of models uncovers flaws and brings up fundamental issues related with the requirements and design of the system. It is in the early stages of development, when code does not yet exist, that model analysis is most rewarding, exposing problems that cost much more to fix if not discovered until later.

[1] Email:namalio@cs.york.ac.uk

[2] Email:fiona@cs.york.ac.uk

[3] Email:susan@cs.york.ac.uk

doi:10.1016/j.entcs.2007.09.002

Mainstream software engineering uses *semi-formal techniques* for MDD.[4] These are based on diagrammatic notations, which are used to describe different aspects of systems. It is their graphical nature and their pragmatic approach to development that makes them popular. In fact, semi-formal notations are intuitive and provide easy to read sketches of different aspects of systems. Furthermore, some semi-formal notations, such as UML and entity-relationship diagrams, are *de facto* modelling idioms among software engineers. However, it is in the details that we find their weaknesses. Semi-Formal notations lack a formal semantics, thus: models are likely to be ambiguous, inconsistent and not amenable to mechanical semantic analysis. The semantics issue is aggravated by the fact that semi-formal notations have many semantic interpretations: the choice of semantics becomes a matter of convenience, developers use one semantics or the other depending on the kind of problem at hand. Moreover, not all properties of systems can be expressed with diagrams; usually, developers resort to textual notations to describe detailed system constraints.

Formal techniques, on the other hand, are used mostly in niche domains that require rigorous development (e.g. safety-critical systems). Formal modelling languages (e.g. Z, B, CSP, Alloy) are based on mathematics and formal logic and they embody years of research and best practice in formal development. The resulting models are precise, unambiguous, and amenable to formal analysis. However, formal modelling and analysis requires a level of competence and expertise that is not common in commercial development communities, and formal approaches are not well integrated with the rest of the development process.

There have been numerous attempts to introduce formal techniques in mainstream model development [2]. Some approaches propose a translation from diagrams into a formal modelling language, but give no support to the analysis itself (e.g. [13,14,18]); to explore analysis users are required to be experts in the formal language. Other approaches spend effort in developing yet another special-purpose formal language, rather than building on existing mature work (e.g. [15,7,8]). Most approaches do not take into account the multiple semantics of semi-formal notations (see [2] for examples); however, developers may want alternative semantics for certain high-level modelling concepts, even within the same development, so, to cope with this, a flexible and practical approach to define semantics of semi-formal notations is required.

To tackle these problems, in [2] we draw on the ideas of *pattern*-based development [10] and *problem*-driven methods [11] to advocate an approach based on frameworks for *rigorous, but practical* MDD. Our MDD frameworks are environments for engineers to construct, analyse and refine models of software systems that are designed to address the needs of a specific problem domain. They use diagrams and formal modelling languages, with the diagrams acting as a graphical interface for the formality that lies beneath. The aim is to hide the formality completely, but this is not always possible.

To support the construction of frameworks, we develop the Formal Template Language (FTL) [2,6], a language of templates enabling proof with template repre-

[4] They are called semi-formal because their notations have a formal syntax, but no formal semantics.

sentations. FTL allows the representation of patterns of formal development (e.g. a model structure) and to reason at the pattern-level using meta-proof (e.g. calculating a pre-condition or proving an initialisation theorem at the pattern level) to establish *meta-theorems*. Templates and meta-theorems are assembled in the frameworks catalogue, so that every pattern instance is generated by instantiating a template, and, if applicable, the proof of consistency for the pattern instance is simplified (often to the trivial case) by instantiating a meta-theorem. [2] develops a framework for sequential systems that combines the modelling languages UML and Z — the $UML+Z$ framework.

This paper gives an overview of the general approach to build generative frameworks advocated in [2] and the $UML+Z$ framework in particular, also developed in [2]. It illustrates a generative model development with the $UML+Z$ framework, where the Z model is generated from templates of the $UML+Z$ catalogue. The paper shows how published results related to the research reported here can be used in the wider context of the $UML+Z$ framework. In particular, it shows how the language FTL developed in [2,6] can be used to build a catalogue of templates and meta-theorems for $UML+Z$, how the catalogue can be used to generate a Z model with the object-oriented (OO) style developed in [2,3] by applying $UML+Z$ to a simpler version of the case study developed in [4], and how the resulting $UML+Z$ model can be formally analysed by using the snapshot analysis technique developed in [2,5].

The following starts by giving a brief overview of FTL and its meta-proof approach. Then, it gives an overview of the method proposed to build frameworks in general and the $UML+Z$ framework in particular. Next, it illustrates the $UML+Z$ framework by applying it to a simple problem. Finally, it discusses the results of the research presented here, compares it with related work, and makes the conclusions.

2 FTL and Meta-proof

Templates capture the form (or shape) of sentences of some language, generate, upon instantiation, sentences of that language whose form is as prescribed by the template, and can be used to describe those commonly occurring structures that make a pattern. Our Formal Template Language (FTL)[6,2] is a language to express templates that enables proof with template representations.

For example, FTL can be used to capture the state of a Z promoted ADT [20]:

$$\begin{array}{l} \lll P \ggg == [\ \lll ids \ggg : \mathbb{P}\ \lll ID \ggg;\ \lll st \ggg : \lll ID \ggg \mathrel{+\!\!\!\!\!\rightarrow} \lll S \ggg \mid \\ \qquad \mathrm{dom}\ \lll st \ggg = \lll ids \ggg \land \lll I \ggg\ ] \end{array}$$

This introduces five placeholders, P, ids, ID, st, S and I, which are to be replaced by some text values when instantiated. This template can be instantiated to yield:

$$\begin{array}{l} Bank == [\ accs : \mathbb{P}\ ACCID;\ accSt : ACCID \mathrel{+\!\!\!\!\!\rightarrow} Acc \mid \\ \qquad \mathrm{dom}\, accSt = accs \land \mathrm{true}\ ] \end{array}$$

Any formal sentence or sentences of some formal language (here Z) can be represented as templates expressed in FTL. Is it possible to reason (or do proof) with these template representations? If it were possible, that would have substantial practical value. It would mean that *reuse* could be brought to the level of proofs: *meta-theorems* for certain templates would be proved once, but could be applicable every time those templates are instantiated. This approach of proof with templates is called *meta-proof*. First, the practical value of meta-proof is motivated with an example.

In Z, the introduction of a description of the state space of an abstract data type (ADT), such as *Bank* above, into a specification, entails a demonstration that the description is consistent: at least one state satisfying the description does exist. This involves defining the initial state of the ADT (the so-called initialisation) and proving that the initial state does exist (the initialisation theorem). The initialisation of *Bank* assumes that in the initial state there are no accounts:

$$BankInit == [\ Bank\ '\mid accs' = \varnothing \land accSt' = \varnothing\ ]$$

To demonstrate the consistency of *Bank*, one is required to discharge the conjecture $\vdash ?\ \ \exists\ BankInit \bullet true$, which is automatically discharged in the Z/Eves theorem prover [16].

This proved theorem applies to the *Bank* ADT only. The question is: does it apply to all promoted ADTs that are similar in form to *Bank*? And if it does, can this result be proved once and for all, so that developers don't have to do it again and again?

The empty initialisation of a promoted ADT and the associated conjecture is represented with templates :

$$\begin{array}{l} \lessdot P \gtrdot Init == [\ \lessdot P \gtrdot\ ' \mid \lessdot ids \gtrdot' = \varnothing \land \lessdot st \gtrdot' = \varnothing\ ] \\ \vdash ?\ \ \exists\ \lessdot P \gtrdot Init \bullet true \end{array}$$

We can reason with these templates by analysing their *well-formed* instantiations. In those cases, P, id and st hold names, ID and S are sets, and I is a predicate. By expanding the template schemas using the laws of the schema calculus, and apply the one-point rule (see proof above), we get the formula,

$$\begin{array}{l} \mathrm{dom}\ \varnothing = \varnothing \land \varnothing \in \mathbb{P} \lessdot ID \gtrdot \land \varnothing \in \lessdot ID \gtrdot \mathrel{+\!\!\!\rightarrow} \lessdot S \gtrdot \\ \qquad \land \lessdot I \gtrdot' [\lessdot ids \gtrdot' := \varnothing, \lessdot st \gtrdot' := \varnothing] \end{array}$$

which reduces to, $\lessdot I \gtrdot' [\lessdot ids \gtrdot' := \varnothing, \lessdot st \gtrdot' := \varnothing]$.[5] If $\lessdot I \gtrdot$ is instantiated with *true*, then the formula reduces to *true*. This establishes two meta-theorems, where the latter is a specialisation (or a corollary) of the former, that are applicable to all promoted ADTs instantiated from these templates. The specialised meta-theorem gives the nice property of *true by construction*: whenever these templates

[5] $P[x := val]$ denotes the Z expression that results from substituting the free variable x in P by the value val. We resort to this special notation because in Z the usual logical substitution operator symbol, /, is used to denote variable renaming in a schema reference.

are instantiated, such that $\lessdot I \gtrdot$ is instantiated with *true*, then the initialisation conjecture is simply *true*. Even when $\lessdot I \gtrdot$ is not instantiated with *true*, the formula to prove is simpler than the initial one.

The argument outlined in this meta-proof is rigorous and valid, but it is not formal. To follow a formal approach towards meta-proof, a formal semantics has been given to FTL [6]. This allows the definition of proof rules for Z template expressions, which are proved by appeal to the semantics of FTL [6].

3 The *UML* + *Z* framework

In [2], we develop a framework for modelling sequential systems in the object-oriented paradigm, which is based on the modelling languages UML and Z. The *UML* + *Z* framework (Figure 1) uses UML class, state and object diagrams, which are represented in a Z semantic domain.

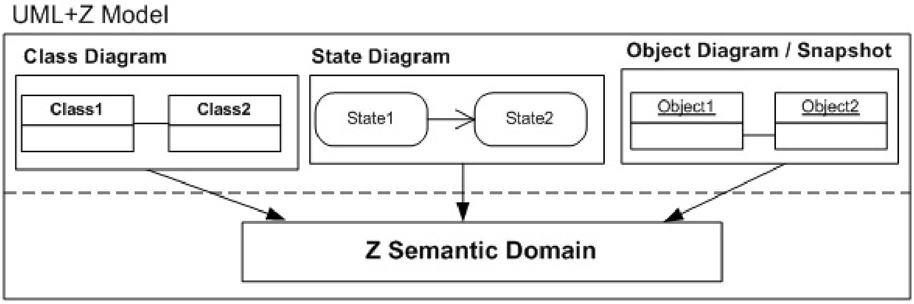

Fig. 1. Models in the *UML* + *Z* framework.

In the following, we discuss the three main components of MDD frameworks, using the *UML* + *Z* framework.

3.1 Modelling

The modelling components of frameworks are defined following the denotational approach of semantic definitions [19]. This includes a set of diagram types, which constitute the *syntactic domain* (every diagram of the framework's models is an instance of these diagram types), a catalogue of FTL templates, which capture the structure of the *semantic domain* (every formal sentence of a model is generated by instantiating a template from the catalogue), and the *semantic mapping*, which maps diagrams to template instantiations.

In the *UML* + *Z* framework [2], there is a Z semantic domain to express OO models [3] so that every diagram of a *UML*+*Z* model is represented in this semantic domain. There is also a catalogue of templates and meta-theorems related to model consistency, which capture the structure of the Z semantic domain; every Z sentence of a *UML* + *Z* model is generated by instantiating one of the templates of the catalogue. The formal definition of the semantic mapping of *UML* + *Z* is left for future work; currently it is performed by hand.

3.2 Analysis

The analysis component defines an approach to analyse the framework's models. [2,5] defines an approach to analyse $UML + Z$ models based on formal proof and Catalysis [9] snapshots: *snapshot-based* validation.

This technique is based on drawing snapshots (object diagrams). A snapshot describe one state of the modelled system. Snapshots can be used in pairs to describe the effects of an operation upon the state of the system: one snapshot describes the before state, the other the after state. The analysis consists of representing snapshots in the Z semantic domain, and then proving, in the Z world, that the snapshot or snapshot pair is satisfied by the model of the system. In the examples used here, these proofs are performed using the Z/Eves theorem prover [16].

3.3 Refinement

The refinement component defines a strategy to refine the framework's models, and includes a catalogue of model transformations (refactorings). The transformations of the catalogue are expressed in FTL.

The refinement component of $UML + Z$ is left for future work. The aim is to define a strategy to refine $UML + Z$ models, based on the theory of refinement for Z [20], and some example model transformations. The idea is to use FTL to capture refactorings and to explore meta-proof to reduce the proof overhead associated with these refactorings. The process is similar to the one followed in modelling: (a) refactorings and associated correctness conjectures are captured with templates; (b) meta-proof is applied upon these representations to simplify (and in some cases fully prove) correctness conjectures. This will allow model transformations to be carried out by instantiating templates: the associated correctness proofs can then be simplified to smaller proofs after applying the associated meta-theorems.

4 Illustration

This section illustrates the $UML + Z$ framework with a use case of a trivial system to track orders that are placed on products. This use case is a simpler version of the one modelled in [4]. For reasons of space, only illustrative parts of the model are given. Further details about a model for this case study can be found in [4]. First, we build a model of the system, and then we analyse it with snapshots.

4.1 Modelling

A $UML + Z$ model is divided into a UML part and a Z one. The UML part of the ordering system comprises one class diagram and one statechart.

Figure 2 presents the class diagram of the system. There are two classes, Order and Product, each representing a set of objects (the orders and products of the system). Each Order object has a *quantity* attribute, referring to the ordered quantity of a product, and each Product object has a *stock* attribute, recording how much of the product is available in stock. Order and Product are related through an association, which says that each order refers to exactly one product, and that each product may be referred by many orders.

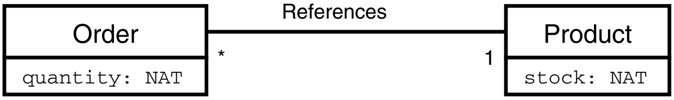

Fig. 2. The UML class diagram of the trivial ordering system.

The objects of Order have distinct states that can be identified. This is described in the UML state diagram (or statechart) of Figure 3. When an Order object is created its state is *pending*. When invoiced, an Order changes from the state *pending* to the state *invoiced*.

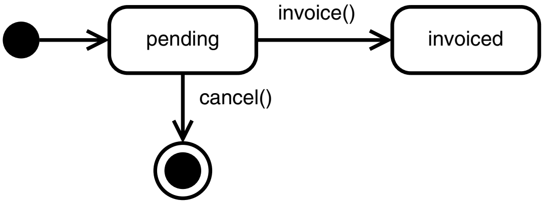

Fig. 3. The UML state diagram for the class *Order*.

The following presents a partial Z model of the ordering system for each view of the OO Z style (structural, intensional, extensional and relational; see [3,2] for further details). The Z model is generated from templates of the *UML* + *Z* catalogue, by instantiating them with information coming from the diagrams and extra information coming from the user. Appendix B presents the templates used to generate the Z presented here. The templates use Z generics from the *UML* + *Z* toolkit (the ones used here are given in Appendix A).

The Z generated from templates is referred to in the following text with the terminology:

- *fully generated* — the Z is fully generated from templates with instantiation information coming from diagrams. If we had a tool, the Z would be automatically generated.
- *partially generated* — the instantiation depends on information not coming from UML diagrams and that needs to be explicitly added by the user (usually constraints not expressible diagrammatically).

4.1.1 Structural View

The definitions from this view are *fully generated*. We need to define the set of all possible object atoms of each class of the model. So, first, we define the set of all classes (represented as atoms) of a model, by instantiating template T1:

$$CLASS ::= OrderCl \mid ProductCl$$

There is a set of all object atoms, OBJ, which is defined in the toolkit (Appendix A). The set of all possible object atoms of a class (a subset of OBJ) is given by the function $\mathbb{O}$, which is defined by instantiating template T2:

$$\begin{array}{|l}
\mathbb{O}\ : CLASS \rightarrow \mathbb{P}_1\ OBJ \\
\hline
\text{disjoint } CLASS \lhd \mathbb{O}
\end{array}$$

Using this function, the set of objects of, say, Order can be obtained with the expression $\mathbb{O}\ OrderCl$.

4.1.2 Intensional View

This view defines the state space and initialisation of the objects of each class. The intension of Order is *fully generated* from template T3, for classes with a state diagram: [6]

There is a Z type representing the possible states of Order, as defined in the state diagram:

$$OrderST ::= pending \mid invoiced$$

The state space includes the fields *quantity* (coming from the UML class diagram), and *state* (holds the current state of an object as defined in the state diagram). The initialisation sets the *state* field to the initial state of the state diagram, and *quantity* to some value received from the environment:

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Order\ \hphantom{xxxxxxxxxxxx}}} \\
state : OrderST \\
quantity : \mathbb{N} \\
\hline
true \\
\hline
\end{array}
\qquad\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ OrderInit\ \hphantom{xxxxxxxxx}}} \\
Order\ ' \\
quantity? : \mathbb{N} \\
\hline
state' = pending \\
quantity' = quantity? \\
\hline
\end{array}$$

The consistency of this definition is checked by proving two conjectures (also generated from the template). The first, the *well-formedness conjecture*,

$$\vdash ?\ \forall\, quantity? : \mathbb{N} \bullet quantity? \in \mathbb{N}$$

is proved automatically in Z/Eves. The second, the initialisation conjecture,

$$\vdash ?\ \exists\, OrderInit \bullet true$$

is *true by construction* by appeal to meta-theorem init-int-ni of the template.

[6] The intension of Product would be instantiated from the template for classes without state diagram, see [2] for details.

4.1.3 Extensional View

This view defines the set of existing objects of a class. The state extension and initialisation of Order is *fully generated* from template T4:

$$\begin{array}{l}
\mathbb{S}\mathit{Order} == \mathbb{S}\mathrm{CL}[\mathbb{O}\ \mathit{OrderCl}, \mathit{Order}][\mathit{sOrder}/\mathit{os}, \mathit{stOrder}/\mathit{oSt}] \\
\mathbb{S}\mathit{OrderInit} == [\ \mathbb{S}\mathit{Order}\ ' \mid \mathit{sOrder}' = \varnothing \land \mathit{stOrder}' = \varnothing\ ]
\end{array}$$

This definition uses the $\mathbb{S}$CL generic from the ZOO toolkit (Appendix A). The initialisation conjecture is *true by construction* by meta-theorem init-ext of the template.

We now define the operation to create new Order objects (also *fully generated*). First, there is the definition of the new promotion frame for Order, which is generated from template T5:

$$\begin{array}{|l}
\multicolumn{1}{l}{\Phi\mathbb{S}\mathit{OrderNew}} \\
\hline
\Delta\mathbb{S}\mathit{Order} \\
\mathit{Order}\ ' \\
\mathit{oOrder}! : \mathbb{O}\ \mathit{OrderCl} \\
\hline
\mathit{oOrder}! \in \mathbb{O}\ \mathit{OrderCl} \setminus \mathit{sOrder} \\
\mathit{sOrder}' = \mathit{sOrder} \cup \{\mathit{oOrder}!\} \\
\mathit{stOrder}' = \mathit{stOrder} \cup \{\mathit{oOrder}! \mapsto \theta\,\mathit{Order}'\} \\
\hline
\end{array}$$

The operation to create new Order objects is generated from template T6:

$$\mathbb{S}_{\Delta}\mathit{OrderNew} == \exists\ \mathit{Order}\ ' \bullet \Phi\mathbb{S}\mathit{OrderNew} \land \mathit{OrderInit}$$

The meta-theorems of the template are used to calculate the precondition of this operation, and to prove that the operation is consistent. The precondition is obtained by instantiating meta-theorem pre-ext-nop of the template:

$$\mathrm{pre}\,\mathbb{S}_{\Delta}\mathit{OrderNew} = [\ \mathbb{S}\mathit{Order};\ \mathit{quantity}? : \mathbb{N} \mid \mathbb{O}\ \mathit{OrderCl} \setminus \mathit{sOrder} \neq \varnothing\ ]$$

The consistency conjecture is always true by meta-theorem epre-ext-nop.

4.1.4 Relational View

This view defines the Z representation of UML associations. The state space, initialisation and association link schema are *fully generated* from template T7. The state space definition and initialisation for the association References are:

$$\begin{array}{l}
\mathbb{A}\mathit{References} == [\ \mathit{rReferences} : \mathbb{O}\ \mathit{OrderCl} \leftrightarrow \mathbb{O}\ \mathit{ProductCl}\ ] \\
\mathbb{A}\mathit{ReferencesInit} == [\ \mathbb{A}\mathit{References}\ ' \mid \mathit{rReferences}' = \varnothing]
\end{array}$$

The initialisation conjecture is *true by construction* by appeal to meta-theorem assoc-init.

The link schema links the objects referred to in an association to existing objects (objects of class extensions) and states the association multiplicity constraint. This is used to build the system structure in the global view (below). The required constraints of this schema are expressed using the mult generic of the $UML + Z$ toolkit (see Appendix A):

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Link}\mathbb{A}\textit{References}} \\ \hline
\mathbb{A}\textit{References};\ \mathbb{S}\textit{Order};\ \mathbb{S}\textit{Product} \\ \hline
\mathsf{mult}(\textit{rReferences}, \textit{sOrder}, \textit{sProduct}, \textit{mo}, \varnothing, \varnothing) \\ \hline
\end{array}$$

This says that the association References is a many-to-one (mo) association. The last two arguments to mult (here they take the value $\varnothing$) are used when the association has user-defined sets of multiplicity constraints (e.g. $1 \mathrel{..} 5$).

When a new order is entered into the system, there is a tuple, made up of the new order and the ordered product, that needs to be added to the association relation. This is described by an operation that adds a new tuple to the association, which is *fully generated* from template T8:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathbb{A}_{\Delta}\textit{ReferencesAdd}} \\ \hline
\Delta\ \mathbb{A}\textit{References} \\
\textit{oOrder}? : \mathbb{O}\ \textit{OrderCl} \\
\textit{oProduct}? : \mathbb{O}\ \textit{ProductCl} \\ \hline
\textit{rReferences}' = \textit{rReferences} \cup \{\textit{oOrder}? \mapsto \textit{oProduct}?\} \\ \hline
\end{array}$$

The precondition of this conjecture is given by meta-theorem assoc-pre-atop of the template; it gives a true predicate (it is a total operation). The consistency conjecture is *true by construction* (meta-theorem assoc-pre-atop).

4.1.5 Global View

This view defines the system as a whole. The state space and initialisation of the system is generated from template T9. In this case, as there are no global constraints, this is *fully generated*. There is a schema representing all the system constraints, as there are no global constraints this includes just the association link schema:

$$\textit{SysConst} == \textit{Link}\mathbb{A}\textit{References}$$

The system state space is defined by including all classes and associations, and the system constraint schema:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{System}} \\ \hline
\mathbb{S}\textit{Order};\ \mathbb{S}\textit{Product};\ \mathbb{A}\textit{References} \\ \hline
\textit{SysConst} \\ \hline
\end{array}$$

The system initialisation is the initialisation of the classes and associations of the system:

$$SysInit == System' \land \mathbb{S}OrderInit \land \mathbb{S}ProductInit \land \mathbb{A}ReferencesInit$$

The system initialisation conjecture,

$$\vdash ?\ \exists\, SysInit \bullet true$$

is *true by construction* by meta-theorem sys-init-ni of template T9.

We now define the new order system operation, which is *partially generated*. First, we defined the frame of the operation by instantiating template T10:

$$\Psi NewOrder == \Delta System \land \Xi\mathbb{S}Product$$

The actual system operation creates a new object of the class Order, and adds a link (or tuple) with the object to an association. This kind of system operations is captured by template T11; the instantiation of this template for this system operation gives:

$$\begin{aligned}&SysNewOrder == \Psi NewOrder \land \mathbb{S}_{\Delta}OrderNew\\ &\qquad \land \mathbb{A}_{\Delta}ReferencesAdd[oOrder!/oOrder?]\end{aligned}$$

The meta-theorems of the template are used to calculate the operation's precondition and to prove the operation's consistency conjecture. Using meta-theorem pre-sop-newadd and Z/Eves, the precondition of the operation is:

$$\begin{aligned}&[System;\ quantity? : \mathbb{N};\ oProduct? : \mathbb{O}\ ProductCl \mid\\ &\qquad \mathbb{O}\ OrderCl \setminus sOrder \neq \varnothing \land oProduct? \in sProduct\ ]\end{aligned}$$

The consistency conjecture (also generated from the template),

$$\vdash ?\ \exists\, \mathrm{pre}\, SysNewOrder \bullet true$$

reduces by meta-theorem epre-sop-newadd-ncs to something that is easily provable in Z/Eves.

The model presented here has been generated from two diagrams by instantiating templates, and its consistency proved by instantiating template conjectures and associated meta-theorems, which are used to discharge the proofs. It is now time to check if the model satisfies the system requirements.

4.2 Analysis

The model of the system is now analysed using snapshots. First, single snapshots are used to analyse the state space, and then snapshot pairs are used to analyse system operations.

4.2.1 State space

State space analysis is illustrated with a requirement of the ordering system that says that each order must reference one product only. To check that the model satisfies this requirement, we draw a snapshot of a *negative* case, that is, it describes a state that should not be accepted by the model of the system. Figure 4 presents a snapshot with an Order object referring to two Product objects.

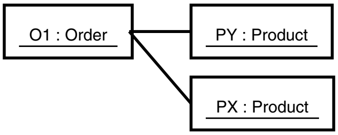

Fig. 4. Snapshot of an Order associated with two products.

To check whether the model accepts this snapshot or not, first the snapshot is represented in Z, and then a conjecture is proved. The representation of a snapshot is *fully generated* by instantiating some template of the full catalogue (see [2]). A partial representation of the snapshot in Z is:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\mathit{StSnap1}} \\ \hline
\mathit{System} \\ \hline
\mathit{sOrder} = \{oO1\} \land \mathit{stOrder} = \{oO1 \mapsto O1\} \\
\mathit{sProduct} = \{oPX, oPY\} \land \mathit{stProduct} = \{oPX \mapsto PX, oPY \mapsto PY\} \\
\mathit{rReferences} = \{oO1 \mapsto oPX, oO1 \mapsto oPY\} \\ \hline
\end{array}
$$

(Here, *System* is the Z schema that defines the system as whole, see above.)

As the snapshot should not be accepted by the system, we prove the conjecture (the negation of the positive case):

$$\vdash ?\ \neg\, (\exists\ \mathit{StSnap1} \bullet \mathit{true})$$

And this conjecture is true (it has been proved in Z/Eves [16]), meaning that the state described by the snapshot is not accepted by the model of the system.

4.2.2 Operations

We now analyse the model of the system operation to add orders to the system. Figure 5 describes two snapshot pairs. In the first, before the operation there is a product with no orders, and after the operation there is a new order in the system, which references the product. In the second snapshot pair, the before state is the state with the product being referenced by one order, and the after state is the product being referenced by two orders. These two state transitions should be accepted by the system.

Snapshots are represented in Z as shown above (instantiated from templates). The validity of a snapshot-pair is checked by proving two conjectures. The first demonstrates that the before state snapshot and inputs to the operation describe a

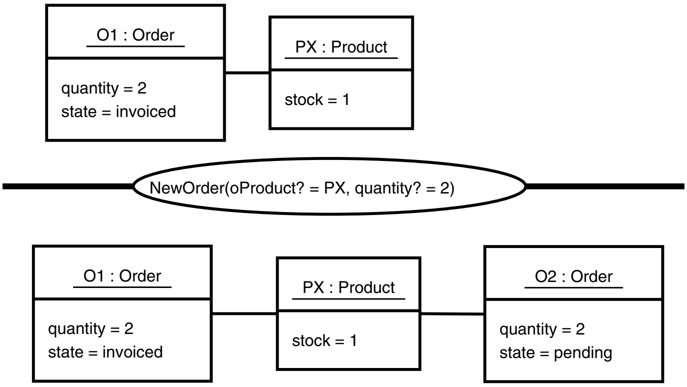

Fig. 5. Snapshot for operation *new order*, one order made on a product.

valid state of the system that satisfies the operation's precondition:

$$\vdash? \; \exists \operatorname{pre} SysNewOrder \bullet BOpSnap1 \land IOpSnap1$$

The second conjecture demonstrates that the snapshot pair describes a valid state transition as specified by the system operation:

$$\vdash? \; \exists SysNewOrder \bullet BOpSnap1 \land IOpSnap1 \land AOpSnap1\,'$$

All the required conjectures for the two snapshot pairs given above are provable in Z/Eves: the operation new order satisfies the state transitions described by the snapshot. See [5,4] for other examples of analysis with snapshots.

5 Discussion

The MDD frameworks advocated here aim to hide the formal language from the user. In *UML* + *Z* this could not be fully achieved. At least one expert is required to write Z operation specifications and invariants that are not expressible in terms of UML diagrams. Nevertheless, the *UML* + *Z* framework allows non-Z experts to engage in the modelling and analysis effort, by drawing class, state and object diagrams.

The semantics problem of semi-formal notations is addressed by considering one semantics that is suitable to the problem domain targeted by the framework. If a framework requires variants of a modelling concept, then these can be defined as UML stereotypes, each given a unique semantics.

The snapshot-based validation still requires creativity in the construction of a set of suitable snapshots "test cases". However, this is done in the diagram language, not the formal language. The analysis is then done formally.

The use of template patterns and frameworks tailored to problem domains helps to foster knowledge reuse. MDD frameworks encapsulate knowledge and experience in the form of patterns (captured with templates); this grows as the framework is applied to more problems. The same body of work can be reused and adapted to meet the needs of other problems (either within the same framework, or for new frameworks exploring new problem domains).

Our pattern-based approach, based on FTL and meta-proof, contributes to make formal methods more practical. FTL allows the representation of structural patterns, so that they can be reused (or adapted to a context) by instantiation. Meta-Proof allows reasoning at the level of patterns to establish meta-theorems so that the same reasoning effort can be reused. In $UML + Z$, all Z is generated by template instantiation, and the proof effort associated with consistency-checking is reduced by applying meta-theorems.

Tool support would bring MDD frameworks into full bloom. Currently, templates are instantiated by hand and users need to switch from the UML tools to Z tools. However, we envisage a tool that could automate most of the process, minimising the exposure of developers to Z tools.

6 Related Work

The work presented here borrows ideas from several sources. The idea of development based on *patterns* comes from the work on design patterns [10], which, in turn, is inspired by the work of Christopher Alexander in Architecture [1].

The idea of *problem-driven* methods and frameworks that are tailored to problem domains is inspired in the work of Michael Jackson [11,12], who advocates that different problems demand different methods and the use of different concepts and notations.

A work that is close to ours is Catalysis [9], a modelling method based on the UML. Like Catalysis, we also use the idea of model frameworks and templates to make models reusable assets, the ideas of model refinement with UML diagrams, and the idea of defining semantics of UML constructs adapted to the context in which they are used.

Our approach differs from these works in that it is fully formal. Our frameworks are designed to integrate formal and semi-formal modelling languages for the purpose of rigorous development. $UML + Z$ for instance is designed for the combined use of UML and Z. Another key feature is that our approach is based on FTL, a formal language to express patterns, which is used to build a catalogue of templates for a a framework. Such a catalogue has been illustrated above in the context of $UML + Z$.

7 Conclusions

This paper advocates an approach to build frameworks for rigorous MDD. To support this approach, the paper presented a language of templates (FTL) that enables proof with templates (meta-proof). FTL is used to construct catalogues of patterns (expressed as templates) and meta-theorems for frameworks; so that pattern instances are generated by instantiating templates and some properties of patterns can be proved at the pattern-level, so that proof at the instance level is either simplified or not required (*true by construction*). The paper also presents and illustrates a framework for sequential systems that combines the formal specification languages *UML* and *Z*. In the illustration, a formal model has been built by instantiating templates from a catalogue, the consistency of the model has been demonstrated by using meta-theorems of the catalogue, and the model has been analysed based on snapshots and formal proof (the analysis strategy of the $UML + Z$ framework).

Acknowledgement

This work was funded by the Portuguese Foundation for Science and Technology, grant 6904/2001. Some of the work presented here was supervised by Prof. Augusto Sampaio, during Amálio's academic visit to the Federal University of Pernambuco, Brazil, which was was funded by the industrial sponsors of the department of computer science of the University of York. All funding and supervision is gratefully acknowledged.

References

[1] Alexander, C., S. Ishikawa and M. Silverstein, "A pattern language: towns, buildings, construction," Oxford University Press, 1977.

[2] Amálio, N., "Generative frameworks for rigorous model-driven development," Ph.D. thesis, Department of Computer Science, University of York (2006), to appear.

[3] Amálio, N., F. Polack and S. Stepney, *An object-oriented structuring for Z based on views*, in: H. Treharne et al., editors, *ZB 2005: International Conference of B and Z users*, LNCS **3455** (2005), pp. 262–278.

[4] Amálio, N., F. Polack and S. Stepney, *UML+Z: UML augmented with Z*, in: M. Frappier and H. Habrias, editors, *Software Specification Methods: an overview using a case study*, Hermes Science, 2006 .

[5] Amálio, N., S. Stepney and F. Polack, *Formal proof from UML models*, in: J. Davies et al., editors, *ICFEM 2004: Int. Conference on Formal Engineering Methods*, LNCS **3308** (2004), pp. 418–433.

[6] Amálio, N., S. Stepney and F. Polack, *A formal template language enabling meta-proof*, in: *FM 2006: Formal Methods*, LNCS **4085** (2006), pp. 252–267.

[7] Clark, T., A. Evans and S. Kent, *The metamodelling language calculus: foundation semantics for UML*, in: H. Hussmann, editor, *Fundamental Approaches to Software Engineering (FASE 2001)*, LNCS **2029** (2001), pp. 17–31.

[8] Clark, T., A. Evans and S. Kent, *Engineering modelling languages: A precise meta-modelling approach*, in: R.-D.Kutsche and H. Weber, editors, *Fundamental Approaches to Software Engineering (FASE 2002)*, LNCS **2306** (2002), pp. 159–173.

[9] D'Sousa, D. and A. C. Wills, "Object Components and Frameworks with UML: the Catalysis approach," Addison-Wesley, 1998.

[10] Gamma, E., R. Helm, R. Johnson and J. Vlissides, "Design Patterns: Elements of Resusable Object-Oriented Software," Professional Computing, Addison-Wesley, 1995.

[11] Jackson, M., *Formal methods and traditional engineering*, The Journal of Systems and Software **40** (1998), pp. 191–194.

[12] Jackson, M., "Problem Frames: Analyzing and structuring software development problems," Addison-Wesley, 2001.

[13] Mander, K. C. and F. Polack, *Rigorous specification using structured systems analysis and Z*, Information and Software Technology **37** (1995), pp. 285–291.

[14] Nguyen, H. P., "Dérivation De Spécifications Formelles B à partir de Spécifications Semi-Formelles," Ph.D. thesis, Laboratoire CEDRIC, Conservatoire National des Arts et Métiers, Evry, France (1998).

[15] Övergaard, G., *Formal specification of object-oriented meta-modelling*, in: T. Maibaum, editor, *Fundamental Approaches to Software Engineering (FASE 2000)*, LNCS **1783** (2000), pp. 193–207.

[16] Saaltink, M., *The Z/EVES system*, in: *ZUM'97: The Z Formal Specification Notation*, LNCS **1212** (1997).

[17] Schmidt, D. C., *Model-driven engineering*, IEEE Computer **39** (2006), pp. 25–31.

[18] Snook, C. and M. Butler, *UML-B: Formal modeling and design aided by UML*, ACM Trans. Softw. Eng. Methodol. **15** (2006), pp. 92–122.

[19] Tennent, R. D., *The denotational semantics of programming languages*, Commun. ACM **19** (1976), pp. 437–453.

[20] Woodcock, J. and J. Davies, "Using Z: Specification, Refinement, and Proof," International series in computer science, Prentice-Hall, 1996.

A Condensed Z Generics Toolkit

This appendix contains the Z generics that are used in the illustration in section 4. For the full toolkit see [2].

$$[OBJ]$$

$$\begin{array}{|l}
OBJ \neq \varnothing
\end{array}$$

$$\begin{array}{|l}
\mathbb{S}\mathrm{CL}\,[OS, OST] \\
\hline
os : \mathbb{P}\, OS \\
oSt : OS \pfun OST \\
\hline
\operatorname{dom} oSt = os \\
\hline
\end{array}$$

$$\begin{array}{l}
MultTy ::= mm \mid mo \mid om \mid mzo \mid zom \mid oo \mid zozo \mid zoo \mid ozo \mid ms \mid sm \mid ss \\
\qquad \mid so \mid os \mid szo \mid zos
\end{array}$$

$$\begin{array}{|l}
[X, Y] \\
\hline
\mathsf{mult}\_ : \mathbb{P}((X \leftrightarrow Y) \times \mathbb{P}\, X \times \mathbb{P}\, Y \times MultTy \times \mathbb{F}\,\mathbb{N} \times \mathbb{F}\,\mathbb{N}) \\
\hline
\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\,\mathbb{N} \bullet \\
\qquad (\mathsf{mult}(r, sx, sy, mm, s_1, s_2)) \Leftrightarrow r \in sx \leftrightarrow sy \\
\forall\, r : X \leftrightarrow Y;\ sx : \mathbb{P}\, X;\ sy : \mathbb{P}\, Y;\ s_1, s_2 : \mathbb{F}\,\mathbb{N} \bullet \\
\qquad (\mathsf{mult}(r, sx, sy, mo, s_1, s_2)) \Leftrightarrow r \in sx \rightarrow sy \\
\vdots \\
\hline
\end{array}$$

B Condensed Catalogue of Templates

This appendix contains the templates used in the illustration in section 4. For the full template catalogue see [2].

Template T1 (Set of class atoms)

$$CLASS ::= \llbracket \leqslant Cl \geqslant Cl \rrbracket_{(|,``")}$$

Template T2 (Set of objects of a class)

$$\begin{array}{|l}
\mathbb{O} \ : CLASS \to \mathbb{P}_1\ OBJ \\
\hline
\mathrm{disjoint}(CLASS \lhd \mathbb{O}\ )
\end{array}$$

Template T3 (Intensional state space and initialisation with state diagram)

$$\leqslant Cl \geqslant ST ::= \leqslant initSt \geqslant \llbracket \ \mid \leqslant oSt \geqslant \ \rrbracket$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\leqslant Cl \geqslant} \\
\hline
\llbracket \ \leqslant at \geqslant : \leqslant atT \geqslant \ \rrbracket \\
state : \leqslant Cl \geqslant ST \\
\hline
\leqslant CLI \geqslant \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\leqslant Cl \geqslant Init} \\
\hline
\leqslant Cl \geqslant\ ' \\
\llbracket \ \leqslant ii \geqslant ? : \leqslant iiT \geqslant \ \rrbracket \\
\hline
\llbracket \ \leqslant at \geqslant' = \leqslant iv \geqslant \ \rrbracket \\
state' = \leqslant initSt \geqslant \\
\hline
\end{array}$$

$$\vdash ?\ \forall \llbracket \ \leqslant ii \geqslant ? : \leqslant iiT \geqslant \ \rrbracket \bullet \llbracket \ \leqslant iv \geqslant \in \leqslant atT \geqslant \ \rrbracket$$

$$\vdash ?\ \exists \leqslant Cl \geqslant Init \bullet true$$

Meta-Theorems.

$$\frac{\Gamma\ \vdash \exists\ \leqslant Cl \geqslant Init \bullet true}{\begin{array}{l}\Gamma\ \vdash \exists\ \llbracket \ \leqslant ii \geqslant ? : \leqslant iiT \geqslant \ \rrbracket \bullet \\ \qquad \leqslant CLI \geqslant' \ [\ \llbracket \ \leqslant at \geqslant' := \leqslant iv \geqslant \ \rrbracket, state' := \leqslant initSt \geqslant \ ]\end{array}} \ [\ \textsf{int-init}\ ]$$

$$\frac{\Gamma;\ \leqslant CLI \geqslant \vdash \exists\ \leqslant Cl \geqslant Init \bullet true}{true} \ \begin{array}{l}[\textsf{init-int-ni}] \\ [\llbracket \ \leqslant atT \geqslant \neq \varnothing \ \rrbracket]\end{array}$$

Template T4 (Extensional state space and initialisation)

$$\mathbb{S}\lessdot Cl\gtrdot == \mathbb{S}\mathrm{CL}[\mathbb{O}\lessdot Cl\gtrdot Cl, \lessdot Cl\gtrdot][s\lessdot Cl\gtrdot/os, st\lessdot Cl\gtrdot/oSt]$$

$$\mathbb{S}\lessdot Cl\gtrdot Init == [\ \mathbb{S}\lessdot Cl\gtrdot\ '\mid s\lessdot Cl\gtrdot' = \varnothing \land st\lessdot Cl\gtrdot' = \varnothing\ ]$$

Meta-Theorems.

$$\frac{\Gamma \vdash \exists\,\mathbb{S}\lessdot Cl\gtrdot Init \bullet true}{true}\ \text{[init-ext]}$$

Template T5 (New promotion frame)

$$\begin{array}{|l}
\hline
\multicolumn{1}{l}{\Phi\mathbb{S}\lessdot Cl\gtrdot N} \\
\Delta\mathbb{S}\lessdot Cl\gtrdot \\
\lessdot Cl\gtrdot\ ' \\
o\lessdot Cl\gtrdot! : \mathbb{O}\lessdot Cl\gtrdot Cl \\
\hline
o\lessdot Cl\gtrdot! \in \mathbb{O}\lessdot Cl\gtrdot Cl \setminus s\lessdot Cl\gtrdot \\
s\lessdot Cl\gtrdot' = s\lessdot Cl\gtrdot \cup \{o\lessdot Cl\gtrdot!\} \\
st\lessdot Cl\gtrdot' = st\lessdot Cl\gtrdot \cup \{o\lessdot Cl\gtrdot! \mapsto \theta\lessdot Cl\gtrdot\ '\} \\
\hline
\end{array}$$

Template T6 (New class operation)

$$\mathbb{S}_\Delta\lessdot Cl\gtrdot New == \exists\lessdot Cl\gtrdot\ ' \bullet \Phi\mathbb{S}\lessdot Cl\gtrdot N \land \lessdot Cl\gtrdot Init$$

Meta-Theorems.

$$\frac{\vdash \mathrm{pre}\,\mathbb{S}_\Delta\lessdot Cl\gtrdot New}{[\ \mathbb{S}\lessdot Cl\gtrdot[\![\ ;\ \lessdot ii\gtrdot? : \lessdot iiT\gtrdot\ ]\!] \mid \mathbb{O}\lessdot Cl\gtrdot Cl \setminus s\lessdot Cl\gtrdot \neq \varnothing\ ]}\ \text{[pre-ext-nop]}$$

$$\frac{\vdash \exists\,\mathrm{pre}\,\mathbb{S}_\Delta\lessdot Cl\gtrdot New}{true}\ \text{[epre-ext-nop]}$$

Template T7 (Association state space and initialisation)

$$\mathbb{A}\lessdot As\gtrdot == [\ r\lessdot As\gtrdot : \mathbb{O}\lessdot Cl_A\gtrdot Cl \leftrightarrow \mathbb{O}\lessdot Cl_B\gtrdot Cl\ ]$$

$$\mathbb{A}\lessdot As\gtrdot Init == [\ \mathbb{A}\lessdot As\gtrdot\ ' \mid r\lessdot As\gtrdot' = \varnothing\ ]$$

$$\begin{array}{|l}
\hline
\multicolumn{1}{l}{Link\mathbb{A}\lessdot As\gtrdot} \\
\mathbb{A}\lessdot As\gtrdot;\ \mathbb{S}\lessdot Cl_A\gtrdot;\ \mathbb{S}\lessdot Cl_B\gtrdot \\
\hline
\mathsf{mult}(r\lessdot As\gtrdot, s\lessdot Cl_A\gtrdot, s\lessdot Cl_B\gtrdot,\ \lessdot multE\gtrdot, \lessdot MS_1\gtrdot, \lessdot MS_2\gtrdot) \\
\hline
\end{array}$$

Meta-Theorems.

$$\frac{\Gamma \vdash \exists\, \mathbb{A}\leqslant As\geqslant Init \bullet \mathit{true}}{\mathit{true}}\ \text{[assoc-init]}$$

Template T8 (Association, add tuple operation)

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathbb{A}_{\Delta}\leqslant As\geqslant Add} \\ \hline
\Delta\mathbb{A}\leqslant As\geqslant \\
o\leqslant Cl_A\geqslant ? : \mathbb{O} \leqslant Cl_A\geqslant Cl \\
o\leqslant Cl_B\geqslant ? : \mathbb{O} \leqslant Cl_B\geqslant Cl \\ \hline
r\leqslant As\geqslant' = r\leqslant As\geqslant \cup \{o\leqslant Cl_A\geqslant ? \mapsto o\leqslant Cl_B\geqslant ?\} \\ \hline
\end{array}$$

Meta-Theorems.

$$\frac{\mathrm{pre}\, \mathbb{A}_{\Delta}\leqslant As\geqslant Add \bullet \mathit{true}}{[\ \mathbb{A}\leqslant As\geqslant;\ o\leqslant Cl_A\geqslant ? : \mathbb{O} \leqslant Cl_A\geqslant Cl;\ o\leqslant Cl_B\geqslant ? : \mathbb{O} \leqslant Cl_B\geqslant Cl\ ]}\ \text{[assoc-pre-atop]}$$

$$\frac{\Gamma \vdash \exists\ \ \mathrm{pre}\, \mathbb{A}_{\Delta}\leqslant As\geqslant Add \bullet \mathit{true}}{\mathit{true}}\ \text{[assoc-epre-atop]}$$

Template T9 (System state space, initialisation and constraints)

$\llbracket$

$$\begin{array}{|l}
\multicolumn{1}{l}{Const\leqslant SConst\geqslant} \\ \hline
\llbracket\ \mathbb{S}\leqslant ConstCl\geqslant;\ \ \rrbracket\ \llbracket\ \mathbb{A}\leqslant ConstAs\geqslant\ \rrbracket \\ \hline
\leqslant Const\geqslant \\ \hline
\end{array}$$

$\rrbracket$

$$SysConst == \llbracket\ Link\mathbb{A}\leqslant As\geqslant\ \rrbracket \wedge \llbracket\ Const\leqslant SConst\geqslant\ \rrbracket$$

$$\begin{array}{|l}
\multicolumn{1}{l}{System} \\ \hline
\llbracket\ \mathbb{S}\leqslant Cl\geqslant\ \rrbracket\ \llbracket\ ;\ \ \mathbb{A}\leqslant As\geqslant\ \rrbracket \\ \hline
SysConst \\ \hline
\end{array}$$

$$SysInit == System\ ' \wedge\ \llbracket\ \mathbb{S}\leqslant Cl\geqslant Init\ \rrbracket\ \wedge \llbracket\ \mathbb{A}\leqslant As\geqslant Init\ \rrbracket$$

$$\vdash ?\ \ \exists\ \ SysInit \bullet \mathit{true}$$

Meta-Theorems

$$\frac{\Gamma \ \vdash \exists \ \mathit{SysInit} \bullet \mathit{true}}{\Gamma \ \vdash \llbracket \ \lessdot \mathit{Const} \gtrdot' \ \rrbracket [\ \llbracket \ s \lessdot \mathit{Cl} \gtrdot' := \varnothing, st \lessdot \mathit{Cl} \gtrdot' := \varnothing \ \rrbracket \llbracket \ , r \lessdot \mathit{As} \gtrdot' := \varnothing \ \rrbracket \ ]} \ \text{[sys-init]}$$

$$\frac{\Gamma; \ \llbracket \ \lessdot \mathit{Const} \gtrdot \ \rrbracket \vdash \exists \ \ \mathit{SysInit} \bullet \mathit{true}}{\mathit{true}} \ \text{[sys-init-ni]}$$

Template T10 (System operation frame)

$$\Psi \lessdot \mathit{sOp} \gtrdot == \Delta \mathit{System} \llbracket \ \wedge \Xi \mathbb{S} \lessdot \mathit{fCl} \gtrdot \ \rrbracket \llbracket \ \wedge \Xi \mathbb{A} \lessdot \mathit{fAs} \gtrdot \ \rrbracket$$

Template T11 (System operation new object and add tuple)

$$\begin{array}{l} \mathit{Sys} \lessdot \mathit{sOp} \gtrdot == \Psi \lessdot \mathit{sOp} \gtrdot \wedge \llbracket \ \mathit{OpConst} \lessdot \mathit{opConst} \gtrdot \ \rrbracket \wedge \mathbb{S}_{\Delta} \lessdot \mathit{nCl} \gtrdot \mathit{New} \\ \qquad \mathbb{A}_{\Delta} \lessdot \mathit{aAs} \gtrdot \mathit{Add}[o \lessdot \mathit{nCl} \gtrdot ! / o \lessdot \mathit{nCl} \gtrdot ?] \end{array}$$

$$\vdash ? \ \{ \llbracket \ \lessdot \mathit{Cl} \gtrdot \ \rrbracket \} = \{ \lessdot \mathit{nCl} \gtrdot \llbracket \ , \lessdot \mathit{fCl} \gtrdot \ \rrbracket \} \wedge \{ \llbracket \ \lessdot \mathit{As} \gtrdot \ \rrbracket \} = \{ \lessdot \mathit{aAs} \gtrdot \llbracket \ , \lessdot \mathit{fAs} \gtrdot \ \rrbracket \}$$

$$\vdash ? \ \exists \ \ \mathrm{pre} \ \ \mathit{Sys} \lessdot \mathit{sOp} \gtrdot \bullet \mathit{true}$$

Meta-Theorems

$$\frac{\begin{array}{l} \mathrm{pre}(\Psi \lessdot \mathit{sOp} \gtrdot \wedge \llbracket \ \mathit{OpConst} \lessdot \mathit{opConst} \gtrdot \ \rrbracket \wedge \mathbb{S}_{\Delta} \lessdot \mathit{nCl} \gtrdot \mathit{New} \\ \qquad \wedge \ \mathbb{A}_{\Delta} \lessdot \mathit{aAs} \gtrdot \mathit{Add}[o \lessdot \mathit{nCl} \gtrdot ! / o \lessdot \mathit{nCl} \gtrdot ?]) \end{array}}{\begin{array}{l} \mathit{System} \wedge \mathrm{pre} \, \mathbb{S}_{\Delta} \lessdot \mathit{nCl} \gtrdot \mathit{New} \\ \qquad \wedge \ \exists \, o \lessdot \mathit{nCl} \gtrdot ! : \mathbb{O} \ \lessdot \mathit{nCl} \gtrdot \mathit{Cl}; \ \lessdot \mathit{nCl} \gtrdot \ '; \ \mathbb{A} \lessdot \mathit{aAs} \gtrdot \ ' \bullet \\ \qquad\qquad (\mathit{SysConst}' \wedge \llbracket \ \mathit{OpConst} \lessdot \mathit{opConst} \gtrdot \ \rrbracket)[\\ \qquad\qquad\qquad \llbracket \ \theta \mathbb{S} \lessdot \mathit{fCl} \gtrdot \ ' := \theta \mathbb{S} \lessdot \mathit{fCl} \gtrdot \ \rrbracket \\ \qquad\qquad\qquad \llbracket \ , \theta \mathbb{A} \lessdot \mathit{fAs} \gtrdot \ ' := \theta \mathbb{A} \lessdot \mathit{fAs} \gtrdot \ \rrbracket , \\ \qquad\qquad\qquad s \lessdot \mathit{nCl} \gtrdot' := s \lessdot \mathit{nCl} \gtrdot \cup \{ o \lessdot \mathit{nCl} \gtrdot ! \}, \\ \qquad\qquad\qquad st \lessdot \mathit{nCl} \gtrdot' := st \lessdot \mathit{nCl} \gtrdot \\ \qquad\qquad\qquad\qquad \cup \{ o \lessdot \mathit{nCl} \gtrdot ! \mapsto \theta \lessdot \mathit{nCl} \gtrdot \ ' \} \ ] \\ \qquad\qquad \wedge \ \lessdot \mathit{nCl} \gtrdot \mathit{Init} \\ \qquad\qquad \wedge \ \mathbb{A}_{\Delta} \lessdot \mathit{aAs} \gtrdot \mathit{Add}[o \lessdot \mathit{nCl} \gtrdot ! / o \lessdot \mathit{nCl} \gtrdot ?] \end{array}} \ \text{[pre-sop-newadd]}$$

$$
\begin{array}{l}
\vdash \llbracket \leqslant \mathit{Const} \geqslant \rrbracket \land \llbracket \leqslant \mathit{opConst} \geqslant \rrbracket, \\
\vdash \exists\, \mathrm{pre}(\Psi \leqslant \mathit{sOp} \geqslant \land \llbracket\ \mathit{OpConst} \leqslant \mathit{opConst} \geqslant \rrbracket \\
\qquad \land\ \mathbb{S}_{\Delta} \leqslant \mathit{nCl} \geqslant \mathit{New} \\
\qquad \land\ \mathbb{A}_{\Delta} \leqslant \mathit{aAs} \geqslant \mathit{Add}[o \leqslant \mathit{nCl} \geqslant ! / o \leqslant \mathit{nCl} \geqslant ?]) \bullet \mathit{true} \\
\hline
\mathrm{pre}\, \mathbb{S}_{\Delta} \leqslant \mathit{nCl} \geqslant \mathit{New} \\
\vdash \exists\, o \leqslant \mathit{nCl} \geqslant ! : \mathbb{O} \leqslant \mathit{nCl} \geqslant \mathit{Cl};\ \mathbb{A} \leqslant \mathit{aAs} \geqslant\ ' \bullet \\
\qquad\qquad \mathit{Link}\mathbb{A} \leqslant \mathit{aAs} \geqslant [\llbracket\ \theta\mathbb{S} \leqslant \mathit{fCl} \geqslant\ ' := \theta\mathbb{S} \leqslant \mathit{fCl} \geqslant \rrbracket \\
\qquad\qquad\qquad s \leqslant \mathit{nCl} \geqslant' := s \leqslant \mathit{nCl} \geqslant \cup \{o \leqslant \mathit{nCl} \geqslant !\}\ ] \\
\qquad\qquad \land\ \mathbb{A}_{\Delta} \leqslant \mathit{aAs} \geqslant \mathit{Add}[o \leqslant \mathit{nCl} \geqslant ! / o \leqslant \mathit{nCl} \geqslant ?]
\end{array}
\quad [\ \text{epre-sop-newadd-ncs}\ ]
$$