Frameworks based on Templates for model-driven development

Nuno Amálio

Department of Computer Science, University of York
Background (I)

- Model-driven development (MDD):
  - more emphasis on models rather than code
  - models to describe and analyse SW systems
  - aim: get requirements right early!
Model-driven development (MDD):
- more emphasis on models rather than code
- models to describe and analyse SW systems
- aim: get requirements right early!

Formal vs semi-formal modelling techniques
Background (II)

- Formal techniques:
  - not widely used
  - models are precise, unambiguous and analysable
  - but require high degree of expertise; not universal
  - sometimes not very practical
Background (II)

- Formal techniques:
  - not widely used
  - models are precise, unambiguous and analysable
  - but require high degree of expertise; not universal
  - sometimes not very practical

- Semi-formal techniques:
  - the ones most SW engineers use
  - intuitive; provide good sketches of systems
  - defacto idioms among engineers
  - models are ambiguous, imprecise, not analysable
Background (III)

Combination formal/semi-formal to introduce formal techniques in SW engineering practice
Combination formal/semi-formal to introduce formal techniques in SW engineering practice

This has substantial practical value:
- formal models by drawing diagrams
- mechanical verification of diagram-based models
- improve communication
Background (III)

- Combination formal/semi-formal to introduce formal techniques in SW engineering practice
- This has substantial practical value:
  - formal models by drawing diagrams
  - mechanical verification of diagram-based models
  - improve communication
- Current approaches in this area:
Background (III)

- Combination formal/semi-formal to introduce formal techniques in SW engineering practice
- This has substantial practical value:
  - formal models by drawing diagrams
  - mechanical verification of diagram-based models
  - improve communication
- Current approaches in this area:
  - just translate from diagrams to formal model
Background (III)

- Combination formal/semi-formal to introduce **formal techniques** in **SW engineering practice**

This has substantial practical value:
- **formal models** by drawing diagrams
- **mechanical verification** of diagram-based models
- improve communication

Current approaches in this area:
- just translate from diagrams to formal model
- define **special purpose formal-language**
Combination formal/semi-formal to introduce formal techniques in SW engineering practice

This has substantial practical value:
- formal models by drawing diagrams
- mechanical verification of diagram-based models
- improve communication

Current approaches in this area:
- just translate from diagrams to formal model
- define special purpose formal-language
- don’t deal with multi-semantics of diagrams
Background (III)

- Combination formal/semi-formal to introduce formal techniques in SW engineering practice.
- This has substantial practical value:
  - formal models by drawing diagrams
  - mechanical verification of diagram-based models
  - improve communication
- Current approaches in this area:
  - just translate from diagrams to formal model
  - define special purpose formal-language
  - don’t deal with multi-semantics of diagrams
  - don’t provide practical approach to V&V
Background (III)

- Combination formal/semi-formal to introduce formal techniques in SW engineering practice
- This has substantial practical value:
  - formal models by drawing diagrams
  - mechanical verification of diagram-based models
  - improve communication
- Current approaches in this area:
  - just translate from diagrams to formal model
  - define special purpose formal-language
  - don’t deal with multi-semantics of diagrams
  - don’t provide practical approach to V&V
  - very few explore refinement
PhD Thesis

Approach to build frameworks for rigorous MDD:
- build, analyse and refine models of SW systems
- each framework targets one problem domain
- combine diagrammatic and formal languages
- template language and meta-proof approach
PhD Thesis

- Approach to build frameworks for rigorous MDD:
  - build, analyse and refine models of SW systems
  - each framework targets one problem domain
  - combine diagrammatic and formal languages
  - template language and meta-proof approach
- Underlying ideas:
  - effort in development can be factored to meta-level
  - pattern-based development
  - customised rather than generic solutions
  - explore problem; gain reuse; save effort
PhD Thesis

- Approach to build frameworks for rigorous MDD:
  - build, analyse and refine models of SW systems
  - each framework targets one problem domain
  - combine diagrammatic and formal languages
  - template language and meta-proof approach

- Underlying ideas:
  - effort in development can be factored to meta-level
  - pattern-based development
  - customised rather than generic solutions
  - explore problem; gain reuse; save effort

- Thesis builds framework for seq systems: UML + Z
Outline

1. Templates and Meta-Proof
2. UML+Z framework using case study
   - Modelling
   - Analysis
   - Refinement
3. Conclusions
Templates

- Templates represent sentences of some language that follow a particular form (or pattern)
- and yield actual language sentences when instantiated
Templates

- Templates represent sentences of some language that follow a particular form (or pattern) and yield actual language sentences when instantiated.

Common in CS and Mathematics:

\[(P \land Q) \land R \equiv P \land (Q \land R)\]

\(P, Q\) and \(R\) stand for predicates, but do not belong to the language of predicates (meta-linguistic variables).
Templates

- Templates represent sentences of some language that follow a particular form (or pattern)
- and yield actual language sentences when instantiated

\[(2 \geq 2 \land 1 < 2) \land 3 \leq 3 \equiv 2 \geq 2 \land (1 < 2 \land 3 \leq 3)\]
Templates

- Templates represent sentences of some language
  - that follow a particular form (or pattern)
  - and yield actual language sentences when instantiated

- Templates appear frequently in CS literature
Templates

Templates represent sentences of some language that follow a particular form (or pattern) and yield actual language sentences when instantiated.

Templates appear frequently in CS literature.

<table>
<thead>
<tr>
<th>Name</th>
<th>declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>predicates</td>
</tr>
</tbody>
</table>
Templates

- Templates represent sentences of some language that follow a particular form (or pattern) and yield actual language sentences when instantiated.

- Templates appear frequently in CS literature.

```
Name declaration

predicates

Bank

accounts : \( \mathbb{P} \) ACCID

accountSt : ACCID \rightarrow Account

dom accountSt = accounts
```
Templates

- Templates represent sentences of some language that follow a particular form (or pattern) and yield actual language sentences when instantiated.
- Templates appear frequently in CS literature.
- Language to express templates precisely.

<table>
<thead>
<tr>
<th>Name</th>
<th>Bank</th>
</tr>
</thead>
<tbody>
<tr>
<td>declaration</td>
<td>accounts : ( \mathbb{P} ) ACCID</td>
</tr>
<tr>
<td>predicates</td>
<td>accountSt : ACCID ( \rightarrow ) Account</td>
</tr>
<tr>
<td></td>
<td>dom accountSt = accounts</td>
</tr>
</tbody>
</table>
Templates

- **Templates** represent sentences of some language that follow a particular form (or pattern) and yield actual language sentences when instantiated.

- Templates appear frequently in CS literature.

- Language to express templates precisely.

```
< Name >
[ < declaration > ]
[ < predicate > ]
```

```
Bank
accounts : \( \mathbb{P} \) ACCID
accountSt : ACCID \rightarrow Account

\text{dom } \text{accountSt} = \text{accounts}
```
Meta-Proof

Can we do proof with templates? Is it useful?
Can we do **proof with templates?** Is it useful?

```
Bank

accounts : \(\mathcal{P} \text{ ACCID}\)
accountSt : ACCID \(\rightarrow\) Account

\text{dom accountSt} = accounts
```
Meta-Proof

Can we do proof with templates? Is it useful?

\[
\begin{align*}
\text{Bank} & \quad \text{BankInit} \\
\text{accounts} : \mathcal{P} \text{ACCID} & \quad \text{accounts}' = \emptyset \\
\text{accountSt} : \text{ACCID} \rightarrow \text{Account} & \quad \text{accountSt}' = \emptyset \\
\text{dom accountSt} = \text{accounts} & \\
\end{align*}
\]
Meta-Proof

Can we do proof with templates? Is it useful?

\[
\begin{align*}
\text{Bank} & : \exists \text{ACCID} \\
\text{accountSt} & : \text{ACCID} \rightarrow \text{Account} \\
\text{dom accountSt} & = \text{accounts}
\end{align*}
\]

\[
\begin{align*}
\text{BankInit} & : \exists \text{BankInit} \rightarrow \text{true} \\
\text{Bank}' & \\
\text{accounts}' & = \emptyset \\
\text{accountSt}' & = \emptyset
\end{align*}
\]
Can we do proof with templates? Is it useful?

\[ \text{Bank} \]
\[ \text{accounts} : \mathbb{P} \text{ ACCID} \]
\[ \text{accountSt} : \text{ACCID} \rightarrow \text{Account} \]
\[ \text{dom} \text{ accountSt} = \text{accounts} \]

\[ \text{BankInit} \]
\[ \text{Bank}' \]
\[ \text{accounts}' = \emptyset \]
\[ \text{accountSt}' = \emptyset \]

\[ \vdash \exists \text{BankInit} \models \text{true} \]

This conjecture is true: \text{Bank} is consistent.
Can we generalise this result?
Meta-Proof (II)

- Bank instance of common structure: promoted ADT

```
Bank

accounts : \[\mathbb{P}\ \text{ACCID}\]
accountSt : ACCID \leftrightarrow Account

\text{dom } accountSt = accounts
```
Meta-Proof (II)

- Bank instance of common structure: promoted ADT

\[
\begin{align*}
\langle P \rangle \\
\langle ids \rangle & : P \langle ID \rangle \\
\langle st \rangle & : \langle ID \rangle \leftrightarrow \langle S \rangle \\
\text{dom} \langle st \rangle & = \langle ids \rangle
\end{align*}
\]
Meta-Proof (II)

- Bank instance of common structure: promoted ADT
- Result proved for Bank true for all instances of the promoted ADT template?

\[
\begin{align*}
\ll P \gg \\
\ll ids \gg : P < ID >
\ll st \gg : < ID > \leftrightarrow < S >
\text{dom } \ll st \gg &= \ll ids \gg
\end{align*}
\]
Meta-Proof (II)

- *Bank* instance of **common structure**: promoted ADT
- Result proved for *Bank* true for all instances of the promoted ADT template?

\[
\begin{align*}
\langle P \rangle \\
\langle ids \rangle : \mathcal{P} \langle ID \rangle \\
\langle st \rangle : \langle ID \rangle \leftrightarrow \langle S \rangle \\
\text{dom} \langle st \rangle = \langle ids \rangle
\end{align*}
\]

\[
\begin{align*}
\langle P \rangle \text{Init} \\
\langle P \rangle' \\
\langle ids \rangle' = \emptyset \\
\langle st \rangle' = \emptyset
\end{align*}
\]

\[\vdash? \ \exists \langle P \rangle \text{Init} \ \bullet \text{true}\]
Meta-Proof (II)

- Bank instance of common structure: promoted ADT
- Result proved for Bank true for all instances of the promoted ADT template?

\[
\begin{align*}
\langle P \rangle \\
\langle ids \rangle : P \langle ID \rangle \\
\langle st \rangle : \langle ID \rangle \leftrightarrow \langle S \rangle \\
dom \langle st \rangle = \langle ids \rangle
\end{align*}
\]

\[
\begin{align*}
\langle P \rangle_{\text{Init}} \\
\langle P \rangle' \\
\langle ids \rangle' = \emptyset \\
\langle st \rangle' = \emptyset
\end{align*}
\]

\[\vdash \exists \langle P \rangle_{\text{Init}} \bullet \text{true}\]

- True for all well-formed instantiations of the template
- Initialisation of PADTs are true by construction
FTL (Formal Template Language)

- Developed a simple and formal template language
  - that enables meta-proof
FTL (Formal Template Language)

- Developed a **simple** and **formal** template language
- that enables **meta-proof**
- FTL is given a **denotational semantics**
FTL (Formal Template Language)

- Developed a simple and formal template language that enables meta-proof
- FTL is given a denotational semantics
- The semantic domain is strings
FTL (Formal Template Language)

- Developed a **simple** and **formal** template language that enables **meta-proof**
- FTL is given a **denotational semantics**
- The semantic domain is **strings**
- All the language does is **text substitution**
FTL (Formal Template Language)

- Developed a **simple** and **formal** template language that enables **meta-proof**
- FTL is given a **denotational semantics**
- The semantic domain is **strings**
- All the language does is **text substitution**
- Semantics is given relative to an **instantiation**
FTL (Formal Template Language)

- Developed a **simple** and **formal** template language that enables **meta-proof**
- FTL is given a **denotational semantics**
- The semantic domain is **strings**
- All the language does is **text substitution**
- Semantics is given relative to an **instantiation**
FTL (Formal Template Language)

- Developed a simple and formal template language that enables meta-proof
- FTL is given a denotational semantics
- The semantic domain is strings
- All the language does is text substitution
- Semantics is given relative to an instantiation

\[\langle x \rangle : \langle t \rangle \{x \mapsto a, t \mapsto \mathbb{N}\} \quad a : \mathbb{N}\]
UML + Z: Overview (I)

- MDD framework for **sequential systems**
  - build, analyse and refine models;
  - based on formal language Z and UML.
MDD framework for **sequential systems**
- build, analyse and refine models;
- based on formal language Z and UML.

**UML + Z: Overview (I)**

UML+Z Model

- Class Diagram
  - Class1
  - Class2
- State Diagram
  - State1
  - State2
  - State3
- Object Diagram / Snapshot
  - Object1
  - Object2

Z Semantic Domain
UML + Z: Overview (I)

- MDD framework for *sequential systems*
  - build, analyse and refine models;
  - based on formal language Z and UML.
- one Z expert needed, but non experts can participate
- catalogue of *templates* and *meta-theorems*
**UML + Z: Overview (I)**

- MDD framework for *sequential systems*
  - build, analyse and refine models;
  - based on formal language Z and UML.
- one Z expert needed, but non experts can participate
- catalogue of templates and meta-theorems

*UML + Z* has 3 components:
- Modelling
- Analysis
- Refinement
Modelling

- Modelling approach and semantics of diagrams
- Templates capture structure of Z OO models
- Meta-proof for model consistency-checking
Modelling (II)

Order

quantity: NAT

References

Product

stock: NAT
Modelling (II)

Order

| quantity: NAT |

Product

| stock: NAT |

References

0..* -> 1

pending

invoice() -> invoiced
Modelling (II)

\[
\langle Cl \rangle ST ::= \langle initSt \rangle [ \langle oSt \rangle ]
\]
<CI> ST ::= <initSt>[ ] | <oSt> ]

OrderST ::= pending | invoiced
Modelling (II)

\[
\text{OrderST} ::= \text{pending} \mid \text{invoiced}
\]

\[
\langle Cl \rangle S_T ::= \langle \text{initSt} \rangle[S \mid \langle oSt \rangle]
\]

\[
\langle Cl \rangle
\]

\[
\text{state} : \langle Cl \rangle S_t
\]

\[
[ \langle \text{at} \rangle : \langle \text{atT} \rangle]
\]

\[
\langle ICL \rangle
\]
$< \text{CI} > \text{ST} ::= < \text{initSt} >[ | < \text{oSt} > ]$

$\text{OrderST} ::= \text{pending} | \text{invoiced}$

$< \text{CI} >$

\begin{itemize}
  \item state : $< \text{CI} > \text{St}$
  \item $[ < \text{at} > : < \text{atT} > ]$
\end{itemize}

$< \text{ICL} >$

$\text{Order}$

\begin{itemize}
  \item state : $\text{OrderSt}$
  \item quantity : $\mathbb{N}$
  \item true
\end{itemize}
Modelling (II)

\[
\langle Cl \rangle ST ::= \langle initSt \rangle [ \mid \langle oSt \rangle ]
\]

\[
OrderST ::= pending \mid invoiced
\]

\[
\begin{align*}
\langle Cl \rangle & \quad \text{state} : \langle Cl \rangle St \\
\langle at \rangle & \quad : \langle atT \rangle \\
\langle ICL \rangle
\end{align*}
\]

\[
\begin{align*}
\text{Order} & \quad \text{state} : OrderSt \\
\text{quantity} & \quad : \mathbb{N} \\
\text{true}
\end{align*}
\]

\[
\exists \ [ \langle in \rangle? : \langle inT \rangle ] \\
\langle ICL \rangle[ [ \langle at \rangle := \langle ival \rangle ] ] [ \wedge \langle ival \rangle \in \langle atT \rangle ]
\]
Modelling (II)

\[< Cl >_{ST} ::= < initSt >\mid < oSt >\]

Order_{ST} ::= pending \mid invoiced

\[< Cl >_{\text{state}} : < Cl >_{St}\]
\[< at > : < atT >\]

\[< ICL >\]

\[\text{Order}_{\text{state}} : \text{OrderSt}\]
\[\text{quantity} : \mathbb{N}\]

\[\text{true}\]

\[\vdash\exists [ [ < \text{in} > ? : < \text{inT} > ] \cdot\]
\[< ICL >[ [ < at > := < \text{ival} > ] ] ] [ \land < \text{ival} > \in < atT > ]\]

\[\vdash\exists \text{quantity}? : \mathbb{N} \cdot \text{quantity}? \in \mathbb{N}\]
Modelling (II)

\[ < Cl > ST ::= \langle \text{initSt} \rangle [ \mid < oSt > ] \]

OrderST ::= pending \mid invoiced

\[
\begin{align*}
\langle Cl \rangle & \quad \text{state} : \langle Cl \rangle St \\
& \quad [ \langle at \rangle : \langle atT \rangle ] \\
\langle ICL \rangle & \quad \text{Order} \\
& \quad \text{state} : \langle OrderSt \rangle \\
& \quad \text{quantity} : \mathbb{N} \\
& \quad \text{true}
\end{align*}
\]

\[ \vdash ? \ \exists [ \langle in \rangle? : \langle inT \rangle ] \bullet \]
\[ \langle ICL \rangle [ [ \langle at \rangle := \langle ival \rangle ] ] [ \land \langle ival \rangle \in \langle atT \rangle ] \]

\[ \vdash ? \ \exists \ \text{quantity}? : \mathbb{N} \bullet \ \text{quantity}? \in \mathbb{N} \]
\[ \vdash ? \ \text{true} \]
Analysis

**Strategy** to analyse models of framework
Analysis

- **Strategy** to analyse models of framework
- snapshot-based validation (model testing)
  - based on **proof** and **catalysis** snapshots
Analysis

- **Strategy** to analyse models of framework
- snapshot-based validation (model testing)
  - based on proof and catalysis snapshots
- Snapshots are object diagrams
  - describe system states, or system state transitions
  - represented in Z as model instances
Analysis

- **Strategy** to analyse models of framework
- snapshot-based validation (model testing)
  - based on proof and catalysis snapshots
- Snapshots are object diagrams
  - describe system states, or system state transitions
  - represented in Z as model instances
- From snapshots, we get model-validation conjectures
Analysis (II)

O1 : Order

P2 : Product

P1 : Product
Analysis (II)

\[
\text{System} \\
\text{orders} = \{oO1\} \land orderSt = \{oO1 \leftrightarrow O1\} \\
\text{products} = \{oPX, oPY\} \land productSt = \{oPX \leftrightarrow PX, oPY \leftrightarrow PY\} \\
\text{references} = \{oO1 \leftrightarrow oPX, oO1 \leftrightarrow oPY\}
\]
Analysis (II)

```
orders = {oO1} ∧ orderSt = {oO1 ← O1}
products = {oPX, oPY} ∧ productSt = {oPX ← PX, oPY ← PY}
references = {oO1 ← oPX, oO1 ← oPY}
```

\[ \vdash \neg (\exists StSnap1 \bullet true) \]
Snapshot not accepted by model
Refinement

- **Strategy** to refine $UML + Z$ models.
  - strategy based on **theory of refinement for** $Z$. 
Refinement

- **Strategy** to refine \( UML + Z \) models.
  - strategy based on **theory of refinement for Z**.

- Catalogue of model transformations (**refactorings**)
  - capture refactorings and correctness conjectures;
  - apply meta-proof to simplify proof effort.
Refinement

- **Strategy** to refine *UML + Z* models.
  - strategy based on theory of refinement for *Z*.

- Catalogue of model transformations (*refactorings*)
  - capture refactorings and correctness conjectures;
  - apply meta-proof to simplify proof effort.

- Aim: model refactorings by **instantiating templates**.
  - demonstrate correctness by proving residual (smaller) conjectures.
Strategy to refine $UML + Z$ models.

- strategy based on theory of refinement for $Z$.

Catalogue of model transformations (refactorings)

- capture refactorings and correctness conjectures;
- apply meta-proof to simplify proof effort.

Aim: model refactorings by instantiating templates.

- demonstrate correctness by proving residual (smaller) conjectures.

Still under development
Conclusions

- FTL and meta-proof enhance practicality of FMs
Conclusions

- FTL and meta-proof enhance practicality of FMs
- Approach helps to foster reuse in formal MDD
Conclusions

- FTL and meta-proof enhance **practicality** of FMs
- Approach helps to foster **reuse** in formal MDD
- **non-FM experts** participate in development
Conclusions

- FTL and meta-proof enhance **practicality** of FMs
- Approach helps to foster **reuse** in formal MDD
- **non-FM experts** participate in development
- We would like to hide formal model
  - not possible in *UML* + *Z*, one *Z* expert is required
  - Analysis based on diagrams to involve **customer**