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!

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!
- 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

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

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

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

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 lang
  - that follow a particular form (or pattern)
  - and yield actual lang sentences when instantiated

Templates

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

Common in CS and Mathematics:

$$(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$$

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 lang
  - that follow a particular form (or pattern)
  - and yield actual lang sentences when instantiated

$$(2 \geq 2 \wedge 1 < 2) \wedge 3 \leq 3 \equiv 2 \geq 2 \wedge (1 < 2 \wedge 3 \leq 3)$$

Templates

- Templates represent sentences of some lang
  - that follow a particular form (or pattern)
  - and yield actual lang sentences when instantiated
- Templates appear frequently in CS literature

Templates

- Templates represent sentences of some lang
  - that follow a particular form (or pattern)
  - and yield actual lang sentences when instantiated
- Templates appear frequently in CS literature

Name

declaration

predicates

Templates

- Templates represent sentences of some lang
  - that follow a particular form (or pattern)
  - and yield actual lang sentences when instantiated
- Templates appear frequently in CS literature

Name

declaration

predicates

Bank

$accounts : \mathbb{P}\ ACCID$

$accountSt : ACCID \rightarrowtail Account$

$\mathrm{dom}\ accountSt = accounts$

Templates

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

Name

declaration

predicates

Bank

$accounts : \mathbb{P}\ ACCID$

$accountSt : ACCID \pfun Account$

$\mathrm{dom}\ accountSt = accounts$

Templates

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

$\langle\!\langle$ *Name* $\rangle\!\rangle$

$[\![\langle\!\langle$ *declaration* $\rangle\!\rangle]\!]$

$[\![\langle\!\langle$ *predicate* $\rangle\!\rangle]\!]$

Bank

$accounts : \mathbb{P}\ ACCID$

$accountSt : ACCID \pfun Account$

$\mathrm{dom}\ accountSt = accounts$

Meta-Proof

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

Meta-Proof

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

Bank

$$accounts : \mathbb{P}\ ACCID$$
$$accountSt : ACCID \pfun Account$$

$$\mathrm{dom}\, accountSt = accounts$$

Meta-Proof

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

Bank

$$accounts : \mathbb{P}\ ACCID$$
$$accountSt : ACCID \rightarrowtail Account$$

$$\mathrm{dom}\, accountSt = accounts$$

BankInit

$$Bank'$$

$$accounts' = \varnothing$$
$$accountSt' = \varnothing$$

Meta-Proof

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

Bank

$$\begin{array}{l} accounts : \mathbb{P}\ ACCID \\ accountSt : ACCID \rightarrowtail Account \\ \hline \mathrm{dom}\, accountSt = accounts \end{array}$$

BankInit

$$\begin{array}{l} Bank' \\ \hline accounts' = \varnothing \\ accountSt' = \varnothing \end{array}$$

$\vdash ?\ \exists\, BankInit \bullet \mathrm{true}$

Meta-Proof

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

Bank

$$\begin{array}{l} accounts : \mathbb{P}\ ACCID \\ accountSt : ACCID \rightarrowtail Account \\ \hline \mathrm{dom}\, accountSt = accounts \end{array}$$

BankInit

$$\begin{array}{l} Bank\ ' \\ \hline accounts' = \varnothing \\ accountSt' = \varnothing \end{array}$$

$\vdash ?\ \exists\, BankInit \bullet \mathrm{true}$

- This conjecture is true: *Bank* is consistent.
- Can we generalise this result?

Meta-Proof (II)

- *Bank* instance of common structure: promoted ADT

Bank

$$accounts : \mathbb{P}\ ACCID$$
$$accountSt : ACCID \rightarrowtail Account$$

$$\mathrm{dom}\, accountSt = accounts$$

Meta-Proof (II)

- *Bank* instance of common structure: promoted ADT

$$
\begin{array}{|l}
\hline
\langle\!\langle\ P\ \rangle\!\rangle \\
\langle\!\langle\ ids\ \rangle\!\rangle : \mathbb{P}\ \langle\!\langle\ ID\ \rangle\!\rangle \\
\langle\!\langle\ st\ \rangle\!\rangle : \langle\!\langle\ ID\ \rangle\!\rangle \pfun \langle\!\langle\ S\ \rangle\!\rangle \\
\hline
\mathrm{dom}\ \langle\!\langle\ st\ \rangle\!\rangle = \langle\!\langle\ ids\ \rangle\!\rangle \\
\hline
\end{array}
$$

Meta-Proof (II)

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

$$
\begin{array}{|l}
\hline
\langle\!\langle\ P\ \rangle\!\rangle \\
\langle\!\langle\ ids\ \rangle\!\rangle : \mathbb{P}\ \langle\!\langle\ ID\ \rangle\!\rangle \\
\langle\!\langle\ st\ \rangle\!\rangle : \langle\!\langle\ ID\ \rangle\!\rangle \rightarrowtail\!\!\!\!\!+ \langle\!\langle\ S\ \rangle\!\rangle \\
\hline
\mathrm{dom}\ \langle\!\langle\ st\ \rangle\!\rangle = \langle\!\langle\ ids\ \rangle\!\rangle \\
\hline
\end{array}
$$

Meta-Proof (II)

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

$$
\begin{array}{l}
\langle\!\langle P \rangle\!\rangle \\
\hline
\langle\!\langle ids \rangle\!\rangle : \mathbb{P}\ \langle\!\langle ID \rangle\!\rangle \\
\langle\!\langle st \rangle\!\rangle : \langle\!\langle ID \rangle\!\rangle \rightarrow\!\!\!\!+ \langle\!\langle S \rangle\!\rangle \\
\hline
\mathrm{dom}\ \langle\!\langle st \rangle\!\rangle = \langle\!\langle ids \rangle\!\rangle
\end{array}
\qquad
\begin{array}{l}
\langle\!\langle P \rangle\!\rangle Init \\
\hline
\langle\!\langle P \rangle\!\rangle' \\
\hline
\langle\!\langle ids \rangle\!\rangle' = \varnothing \\
\langle\!\langle st \rangle\!\rangle' = \varnothing
\end{array}
$$

$\vdash ?\ \ \exists \langle\!\langle P \rangle\!\rangle Init \bullet \mathrm{true}$

Meta-Proof (II)

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

$$\begin{array}{l} \langle\!\langle P \rangle\!\rangle \\ \hline \langle\!\langle ids \rangle\!\rangle : \mathbb{P}\ \langle\!\langle ID \rangle\!\rangle \\ \langle\!\langle st \rangle\!\rangle : \langle\!\langle ID \rangle\!\rangle \mathrel{+\!\!\!\rightarrow} \langle\!\langle S \rangle\!\rangle \\ \hline \mathrm{dom}\ \langle\!\langle st \rangle\!\rangle = \langle\!\langle ids \rangle\!\rangle \end{array}$$

$$\begin{array}{l} \langle\!\langle P \rangle\!\rangle Init \\ \hline \langle\!\langle P \rangle\!\rangle' \\ \hline \langle\!\langle ids \rangle\!\rangle' = \varnothing \\ \langle\!\langle st \rangle\!\rangle' = \varnothing \end{array}$$

$$\vdash ?\ \exists \langle\!\langle P \rangle\!\rangle Init \bullet \mathrm{true}$$

- 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

$$\ll x \gg : \ll t \gg \qquad \{x \mapsto a, t \mapsto \mathbb{N}\} \qquad a : \mathbb{N}$$

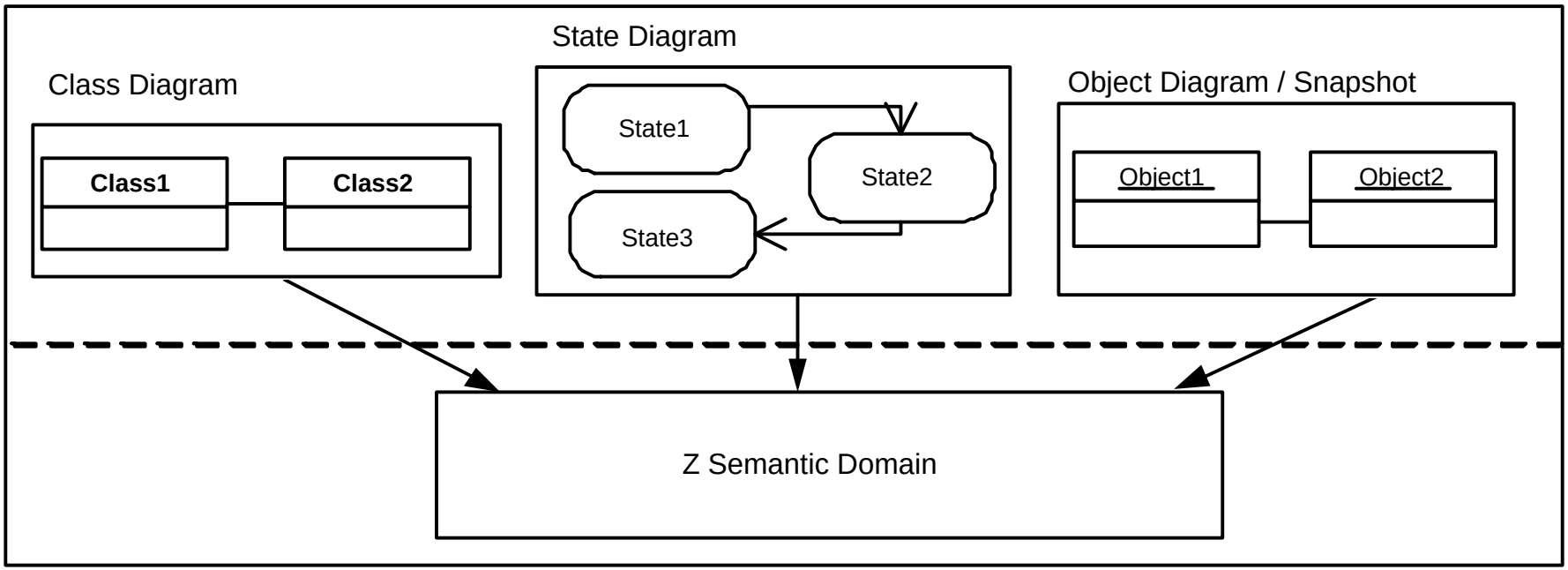
$UML + Z$: Overview (I)

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

$UML + Z$: Overview (I)

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

UML+Z Model



$UML + Z$: 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

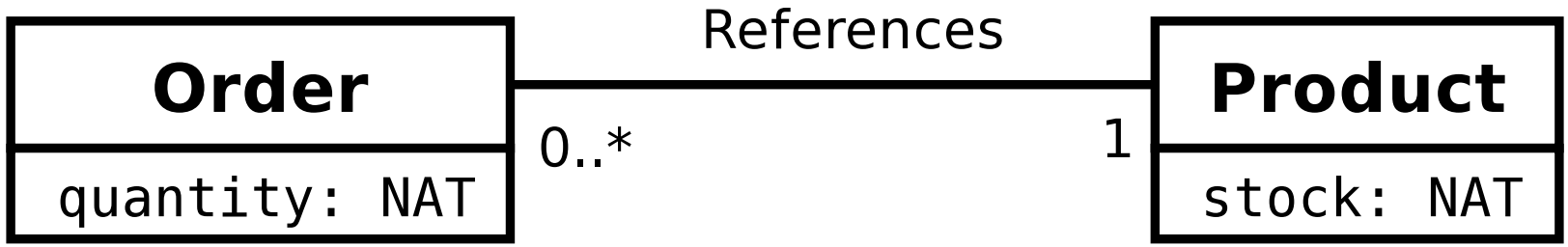
$UML + Z$: Overview (I)

- 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

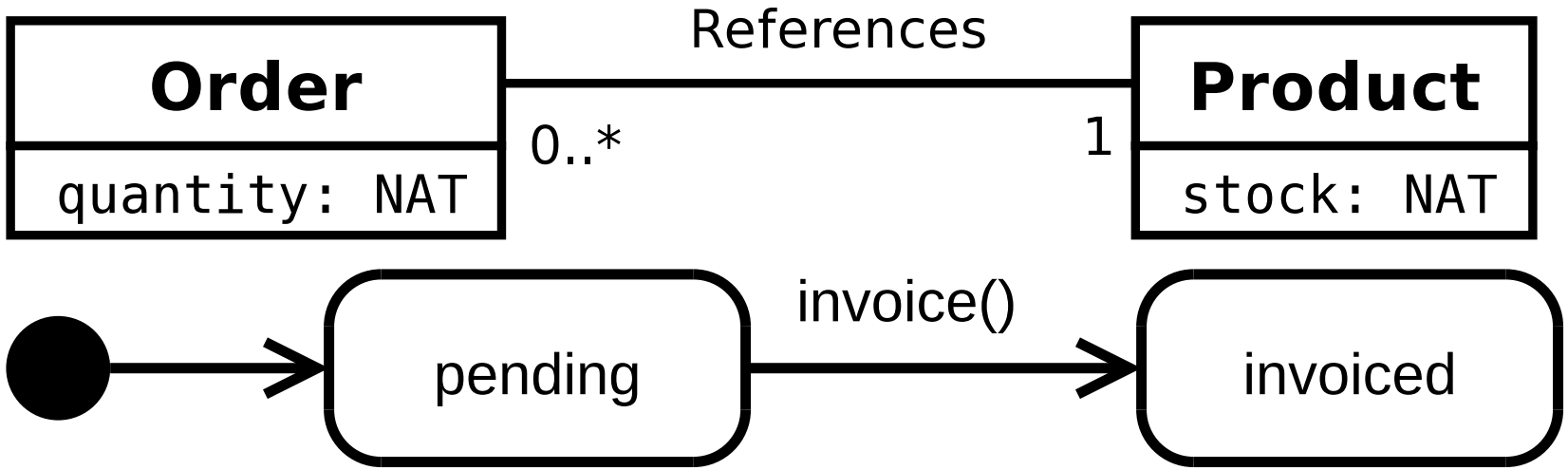
Modelling

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

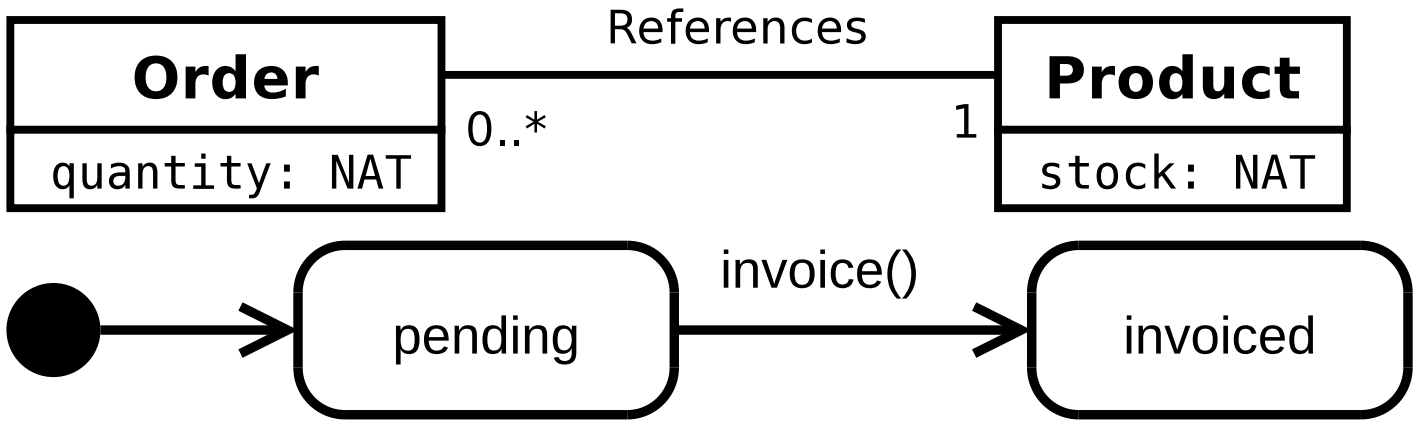
Modelling (II)



Modelling (II)

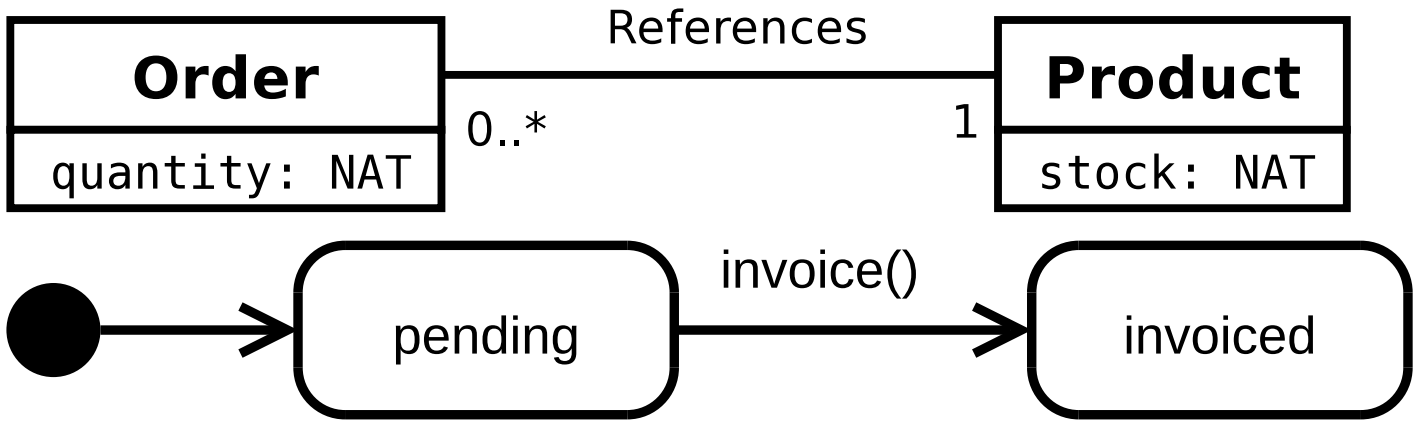


Modelling (II)



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

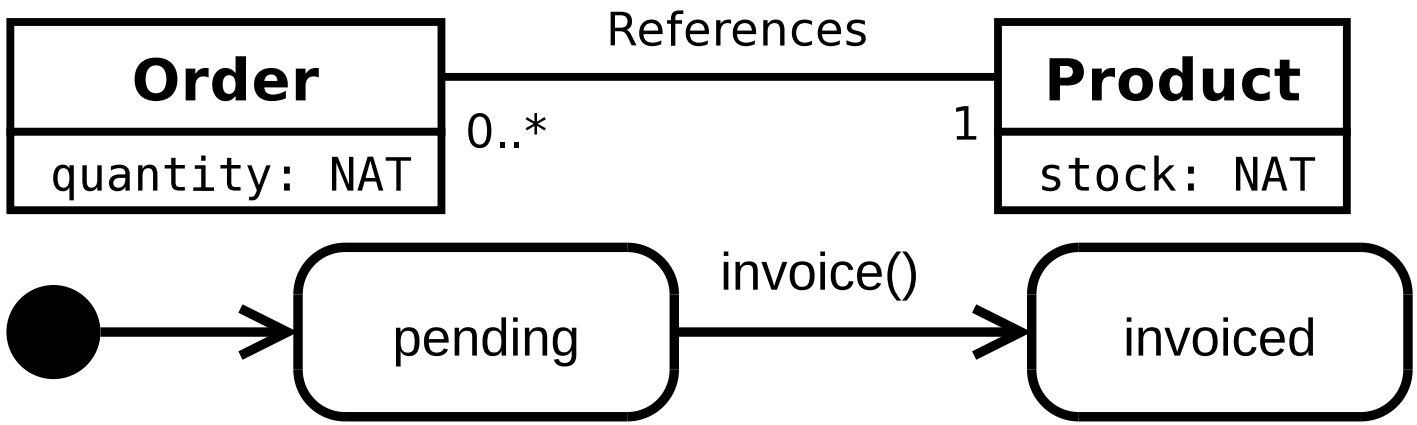
Modelling (II)



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

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

Modelling (II)



$\ll Cl \gg ST ::= \ll initSt \gg [\![\ \mid \ll oSt \gg]\!]$

$OrderST ::= pending \mid invoiced$

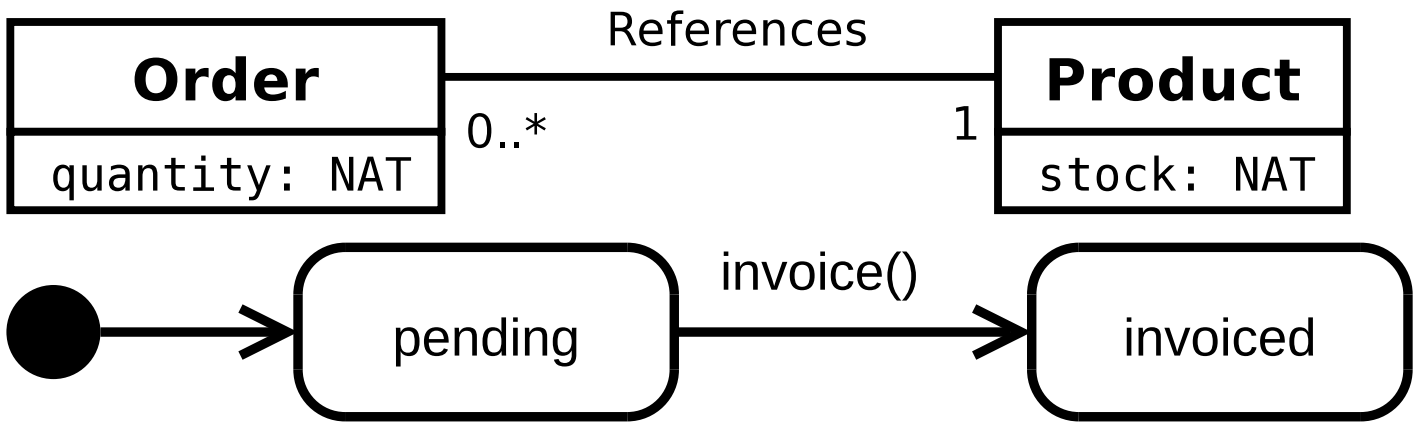
$\ll Cl \gg$

$state : \ll Cl \gg St$

$[\![\ll at \gg : \ll atT \gg]\!]$

$\ll ICL \gg$

Modelling (II)



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

$OrderST ::= pending \mid invoiced$

$\langle\!\langle Cl \rangle\!\rangle$

$state : \langle\!\langle Cl \rangle\!\rangle St$

$[\![\langle\!\langle at \rangle\!\rangle : \langle\!\langle atT \rangle\!\rangle]\!]$

$\langle\!\langle ICL \rangle\!\rangle$

Order

$state : OrderSt$

$quantity : \mathbb{N}$

true

Modelling (II)

$\ll Cl \gg ST ::= \ll initSt \gg [\![\ \mid \ll oSt \gg]\!]$

$OrderST ::= pending \mid invoiced$

$\ll Cl \gg$	Order
$state : \ll Cl \gg St$ $[\![\ll at \gg : \ll atT \gg]\!]$	$state : OrderSt$ $quantity : \mathbb{N}$
$\ll ICL \gg$	true

$\vdash ? \ \exists [\![\ll in \gg ? : \ll inT \gg]\!] \bullet$
$\quad \ll ICL \gg [[\![\ll at \gg := \ll ival \gg]\!]] [\![\ \wedge \ll ival \gg \in \ll atT \gg]\!]$

Modelling (II)

$\ll Cl \gg ST ::= \ll initSt \gg [\![\ \mid \ll oSt \gg]\!]$

$OrderST ::= pending \mid invoiced$

$\ll Cl \gg$	Order
$state : \ll Cl \gg St$ $[\![\ll at \gg : \ll atT \gg]\!]$	$state : OrderSt$ $quantity : \mathbb{N}$
$\ll ICL \gg$	true

$$\vdash ? \ \exists [\![\ll in \gg ? : \ll inT \gg]\!] \bullet \ll ICL \gg [[\![\ll at \gg := \ll ival \gg]\!]] [\![\wedge \ll ival \gg \in \ll atT \gg]\!]$$

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

Modelling (II)

$\ll Cl \gg ST ::= \ll initSt \gg [\![\ \mid \ll oSt \gg]\!]$

$OrderST ::= pending \mid invoiced$

$\ll Cl \gg$	Order
$state : \ll Cl \gg St$ $[\![\ll at \gg : \ll atT \gg]\!]$	$state : OrderSt$ $quantity : \mathbb{N}$
$\ll ICL \gg$	true

$\vdash ? \ \exists [\![\ll in \gg ? : \ll inT \gg]\!] \bullet$
$\quad \ll ICL \gg [[\![\ll at \gg := \ll ival \gg]\!]] [\![\ \wedge \ll ival \gg \in \ll atT \gg]\!]$

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

$\vdash ?$ 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

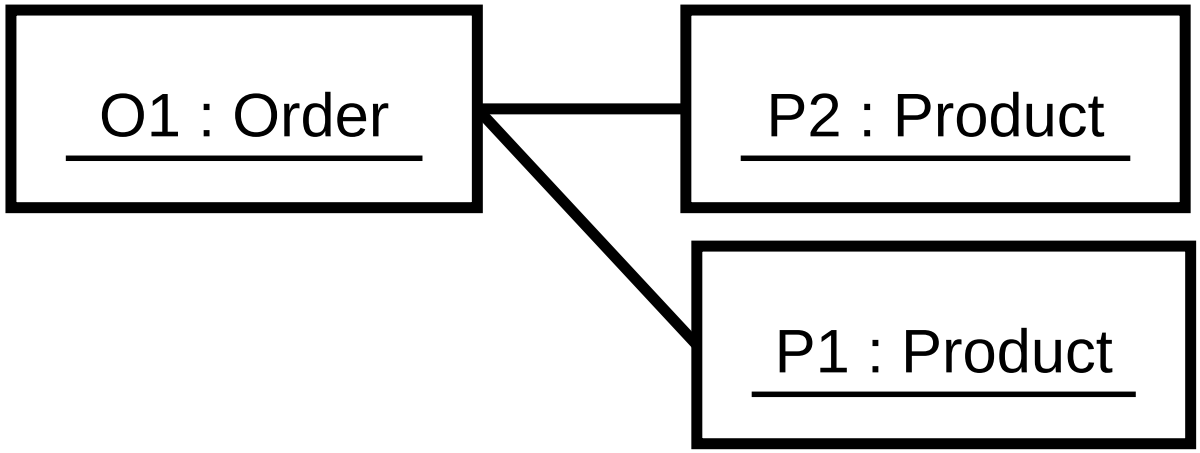
Analysis

- 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

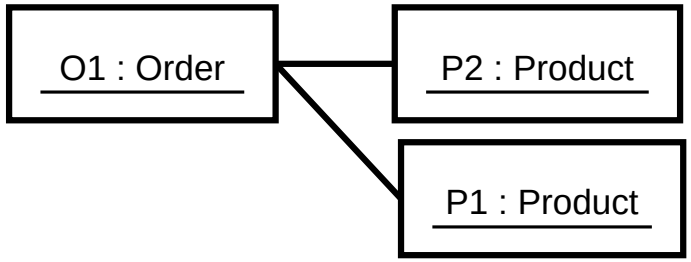
Analysis

- 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)



Analysis (II)



$StSnap1$

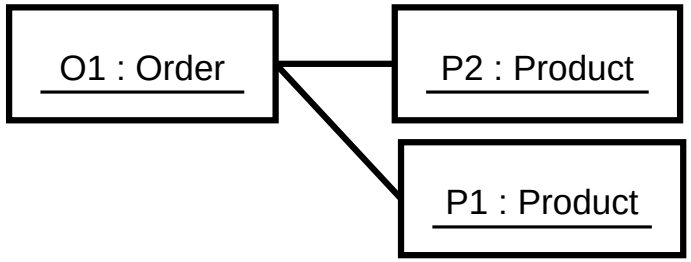
System

$orders = \{oO1\} \wedge orderSt = \{oO1 \mapsto O1\}$

$products = \{oPX, oPY\} \wedge productSt = \{oPX \mapsto PX, oPY \mapsto PY\}$

$references = \{oO1 \mapsto oPX, oO1 \mapsto oPY\}$

Analysis (II)



$StSnap1$

System

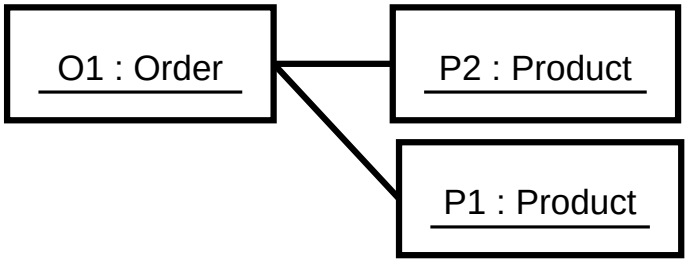
$orders = \{oO1\} \wedge orderSt = \{oO1 \mapsto O1\}$

$products = \{oPX, oPY\} \wedge productSt = \{oPX \mapsto PX, oPY \mapsto PY\}$

$references = \{oO1 \mapsto oPX, oO1 \mapsto oPY\}$

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

Analysis (II)



$StSnap1$

System

$orders = \{oO1\} \wedge orderSt = \{oO1 \mapsto O1\}$

$products = \{oPX, oPY\} \wedge productSt = \{oPX \mapsto PX, oPY \mapsto PY\}$

$references = \{oO1 \mapsto oPX, oO1 \mapsto oPY\}$

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

$\vdash ?$ 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.

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.
- 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