Software Engineering and Service-Oriented Systems

– Analysing Service-Oriented Systems with COWS –

Francesco Tiezzi

IMT - Institutions, Markets, Technologies

Institute for Advanced Studies Lucca

Lucca, Italy - September, 2012

In co-operation with SENSORIA members, in particular Stefania Gnesi, Alessandro Lapadula, Franco Mazzanti, and Rosario Pugliese,
and also Alessandro Fantechi and Nobuko Yoshida

Analysis techniques for COWS specifications

1. A bisimulation-based observational semantics [ICALP'09]
2. A type system for checking confidentiality properties [FSEN'07]
3. A logical verification methodology [FASE'08]

Analysis techniques: an observational semantics

Behavioural equivalences: key concepts

- An important ingredient of a process calculus is a notion of behavioural equivalences between its terms
- Behavioural equivalences, and the related proof techniques, are a tool providing a means to establishing formal correspondences between terms of a process calculus
- Syntactically different terms may behave the same way, hence they ought to be considered behaviourally equivalent
- Behavioural equivalences can take into account diverse *observable* properties of terms (name mobility, asynchrony, . . .)
  - Several different classes of behavioural equivalences have been introduced, each one being characterised by a specific notion of *observable behaviour*
  - The semantics induced by such equivalences are indeed called *observational semantics*

Behavioural equivalences: key concepts

- An important ingredient of a process calculus is a notion of behavioural equivalences between its terms
- Behavioural equivalences, and the related proof techniques, are a tool providing a means to establishing formal correspondences between terms of a process calculus
- Syntactically different terms may behave the same way, hence they ought to be considered behaviourally equivalent
- Behavioural equivalences can take into account diverse *observable* properties of terms (name mobility, asynchrony, . . .)
  - Several different classes of behavioural equivalences have been introduced, each one being characterised by a specific notion of *observable behaviour*
  - The semantics induced by such equivalences are indeed called *observational semantics*

Behavioural equivalences: key concepts

- An important ingredient of a process calculus is a notion of behavioural equivalences between its terms
- Behavioural equivalences, and the related proof techniques, are a tool providing a means to establishing formal correspondences between terms of a process calculus
- Syntactically different terms may behave the same way, hence they ought to be considered behaviourally equivalent
- Behavioural equivalences can take into account diverse *observable* properties of terms (name mobility, asynchrony, . . .)
  - Several different classes of behavioural equivalences have been introduced, each one being characterised by a specific notion of *observable behaviour*
  - The semantics induced by such equivalences are indeed called *observational semantics*

Behavioural equivalences: key concepts

- Powerful and widespread used techniques are based on the notion of *bisimulation*

- Intuitively, a bisimulation is a relation that permits associating two terms if one simulates the behaviour (i.e. the actions that can be performed) of the other and vice-versa

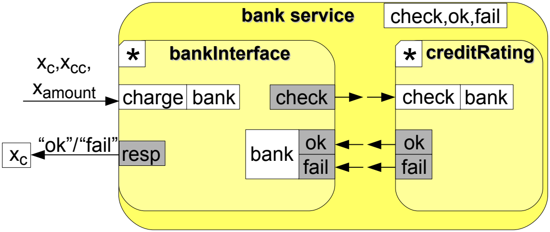
- In doing this, the behaviour of intermediate states that the terms traverse as they evolve have taken into account
  - The action capabilities of the intermediate states does matter: e.g. to observe different deadlock behaviours

An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications

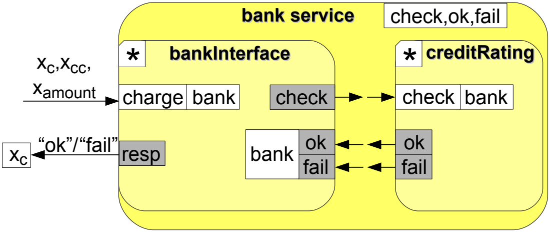
An observational semantics for COWS

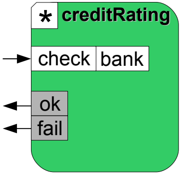
An *observational semantics* permits checking interchangeability of services and conformance against service specifications



An observational semantics for COWS

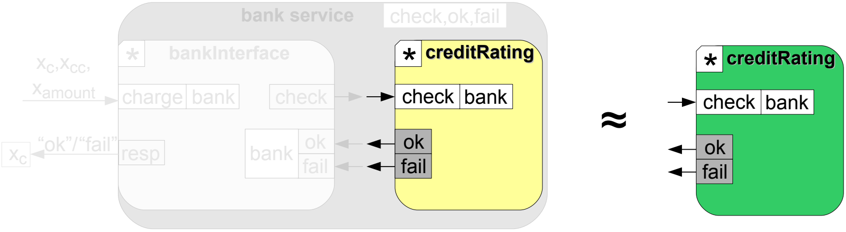
An *observational semantics* permits checking interchangeability of services and conformance against service specifications





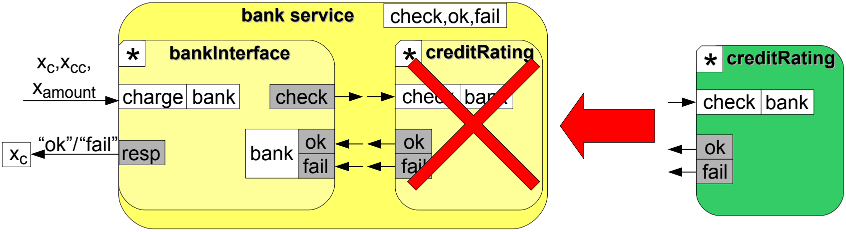
An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications



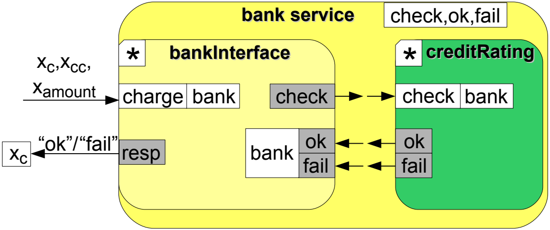
An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications



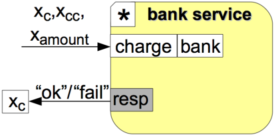
An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications

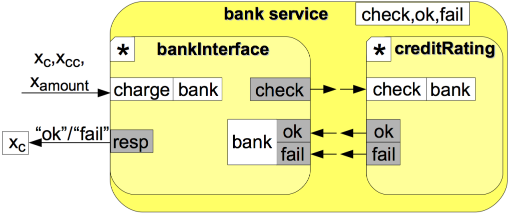


An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications



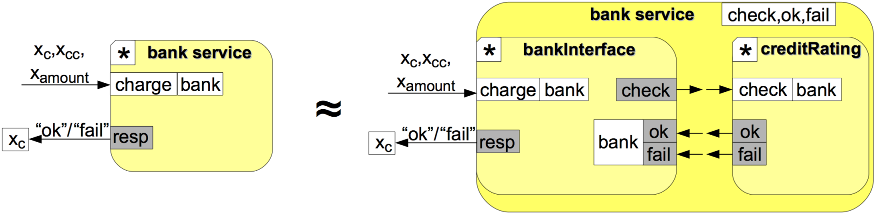
high-level spec.



low-level spec.

An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications



high-level spec.

low-level spec.

An observational semantics for COWS

An *observational semantics* permits checking interchangeability of services and conformance against service specifications

- We have defined:
  - natural notions of strong and weak *open barbed bisimilarities*
  - manageable characterisations in terms of *labelled bisimilarities*
- These semantics show that:
  - COWS's priority mechanisms partially recover the capability to observe receive actions
  - primitives for termination impose specific conditions on the bisimilarities

A natural notion of bisimulation

Observable (barb)

Predicate $s \downarrow_{\mathrm{n}}$ holds true if there exist s', $\bar{n}$ and $\bar{v}$ s.t. $s \xrightarrow{\mathrm{n} \lhd [\bar{n}]\, \bar{v}} s'$, i.e. only the output capabilities are considered as observable

E.g. $[\bar{x}]\,(\mathrm{n}?\bar{x} \mid \mathrm{n}!\bar{v}) \downarrow_{\mathrm{n}}$, while $[\bar{x}]\,(\mathrm{n}?\bar{x}) \not\downarrow_{\mathrm{n}}$

Barbed bisimilarity $\simeq$

is the largest symmetric, barb preserving, computation and context closed relation over COWS terms

- Barbed bisimilarity suffers from universal quantification over all possible language contexts
  - this makes the reasoning on terms very hard
- We have provided a purely co-inductive notion of bisimulation
  - only requires considering transitions of the labelled transition system defining the semantics of the terms under analysis

A natural notion of bisimulation

Observable (barb)

Predicate $s \downarrow_{\mathrm{n}}$ holds true if there exist s', $\bar{n}$ and $\bar{v}$ s.t. $s \xrightarrow{\mathrm{n} \lhd [\bar{n}]\, \bar{v}} s'$, i.e. only the output capabilities are considered as observable

E.g. $[\bar{x}]\,(\mathrm{n}?\bar{x} \mid \mathrm{n}!\bar{v}) \downarrow_{\mathrm{n}}$, while $[\bar{x}]\,(\mathrm{n}?\bar{x}) \not\downarrow_{\mathrm{n}}$

Barbed bisimilarity $\simeq$

is the largest symmetric, barb preserving, computation and context closed relation over COWS terms

- Barbed bisimilarity suffers from universal quantification over all possible language contexts
  - this makes the reasoning on terms very hard
- We have provided a purely co-inductive notion of bisimulation
  - only requires considering transitions of the labelled transition system defining the semantics of the terms under analysis

A natural notion of bisimulation

Observable (barb)

Predicate $s \downarrow_{\mathrm{n}}$ holds true if there exist s', $\bar{n}$ and $\bar{v}$ s.t. $s \xrightarrow{\mathrm{n} \triangleleft [\bar{n}]\, \bar{v}} s'$, i.e. only the output capabilities are considered as observable

E.g. $[\bar{x}]\,(\mathrm{n}?\bar{x} \mid \mathrm{n}!\bar{v}) \downarrow_{\mathrm{n}}$, while $[\bar{x}]\,(\mathrm{n}?\bar{x}) \not\downarrow_{\mathrm{n}}$

Barbed bisimilarity $\simeq$

is the largest symmetric, barb preserving, computation and context closed relation over COWS terms

- Barbed bisimilarity suffers from universal quantification over all possible language contexts
  - this makes the reasoning on terms very hard
- We have provided a purely co-inductive notion of bisimulation
  - only requires considering transitions of the labelled transition system defining the semantics of the terms under analysis

A natural notion of bisimulation

Observable (barb)

Predicate $s \downarrow_{\mathrm{n}}$ holds true if there exist s', $\bar{n}$ and $\bar{v}$ s.t. $s \xrightarrow{\mathrm{n} \lhd [\bar{n}]\, \bar{v}} s'$, i.e. only the output capabilities are considered as observable

E.g. $[\bar{x}]\,(\mathrm{n}?\bar{x} \mid \mathrm{n}!\bar{v}) \downarrow_{\mathrm{n}}$, while $[\bar{x}]\,(\mathrm{n}?\bar{x}) \not\downarrow_{\mathrm{n}}$

Barbed bisimilarity $\simeq$

is the largest symmetric, barb preserving, computation and context closed relation over COWS terms

- Barbed bisimilarity suffers from universal quantification over all possible language contexts
  - this makes the reasoning on terms very hard
- We have provided a purely co-inductive notion of bisimulation
  - only requires considering transitions of the labelled transition system defining the semantics of the terms under analysis

A co-inductive notion of bisimulation

Labelled bisimilarity $\sim$

A names-indexed family of symmetric binary relations $\{\mathcal{R}_\mathcal{N}\}_\mathcal{N}$ is a *labelled bisimulation* if $s_1 \mathcal{R}_\mathcal{N} s_2$ then $\mathrm{halt}(s_1)\, \mathcal{R}_\mathcal{N}\, \mathrm{halt}(s_2)$ and if $s_1 \xrightarrow{\alpha} s_1'$, where $\mathrm{bu}(\alpha)$ are fresh, then:

1. if $\alpha = \mathrm{n} \rhd [\bar{x}]\, \bar{w}$ then one of the following holds:
   (a) $\forall\, \bar{v}$ s.t. $\mathcal{M}(\bar{x}, \bar{v}) = \sigma$ and $noc(s_2, \mathrm{n}, \bar{w}\cdot\sigma, |\,\bar{x}\,|)$:
   $\exists\, s_2' \;:\; s_2 \xrightarrow{\mathrm{n}\rhd[\bar{x}]\,\bar{w}} s_2'$ and $s_1'\cdot\sigma\, \mathcal{R}_\mathcal{N}\, s_2'\cdot\sigma$
   (b) $|\,\bar{x}\,| = |\,\bar{w}\,|$ and $\forall\, \bar{v}$ s.t. $\mathcal{M}(\bar{x}, \bar{v}) = \sigma$ and $noc(s_2, \mathrm{n}, \bar{w}\cdot\sigma, |\,\bar{x}\,|)$:
   $\exists\, s_2' \,:\, s_2 \xrightarrow{\emptyset} s_2'$ and $s_1'\cdot\sigma\, \mathcal{R}_\mathcal{N}\, (s_2' \mid \mathrm{n}!\bar{v})$ or $s_1'\cdot\sigma\, \mathcal{R}_\mathcal{N}\, (s_2' \mid \{\!|\mathrm{n}!\bar{v}|\!\})$
2. if $\alpha = \mathrm{n}\,\emptyset\,\ell\,\bar{v}$ where $\ell = |\,\bar{v}\,|$ then one of the following holds:
   (a) $\exists\, s_2' \;:\; s_2 \xrightarrow{\mathrm{n}\,\emptyset\,\ell\,\bar{v}} s_2'$ and $s_1'\, \mathcal{R}_\mathcal{N}\, s_2'$ (b) $\exists\, s_2' \;:\; s_2 \xrightarrow{\emptyset} s_2'$ and $s_1'\, \mathcal{R}_\mathcal{N}\, s_2'$
3. if $\alpha = \mathrm{n} \lhd [\bar{n}]\, \bar{v}$ where $\mathrm{n} \notin \mathcal{N}$ then $\exists\, s_2' \;:\; s_2 \xrightarrow{\mathrm{n}\lhd[\bar{n}]\,\bar{v}} s_2'$ and $s_1'\, \mathcal{R}_{\mathcal{N}\cup\bar{n}}\, s_2'$
4. if $\alpha = \emptyset$, $\alpha = \dagger$ or $\alpha = \mathrm{n}\,\emptyset\,\ell\,\bar{v}$, where $\ell \neq |\,\bar{v}\,|$, then $\exists\, s_2' \;:\; s_2 \xrightarrow{\alpha} s_2'$ and $s_1'\, \mathcal{R}_\mathcal{N}\, s_2'$

Two closed terms s_1 and s_2 are $\mathcal{N}$*-bisimilar*, written $s_1 \sim^\mathcal{N} s_2$, if $s_1 \mathcal{R}_\mathcal{N} s_2$ for some $\mathcal{R}_\mathcal{N}$ in a labelled bisimulation. They are *labelled bisimilar*, written $s_1 \sim s_2$, if they are $\emptyset$-bisimilar. $\sim^\mathcal{N}$ is called $\mathcal{N}$-bisimilarity, while $\sim$ is called *labelled bisimilarity*

A co-inductive notion of bisimulation

Labelled bisimilarity $\sim$

A names-indexed family of symmetric binary relations $\{\mathcal{R}_{\mathcal{N}}\}_{\mathcal{N}}$ is a *labelled bisimulation* if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ then $\mathrm{halt}(s_1)\, \mathcal{R}_{\mathcal{N}}\, \mathrm{halt}(s_2)$ and if $s_1 \xrightarrow{\alpha} s_1'$, where $\mathrm{bu}(\alpha)$ are fresh, then:

1. if s_1 performs a *receive* then one of the following holds:
   - (a) s_2 performs the same receive and the continuations stand in the same relation for any matching tuple of values that can be effectively received
   - (b) if the argument of the receive contains only variables or is the empty tuple, s_2 performs an internal action leading to a term that, composed with the consumed invoke, stands in the same relation
2. if $\alpha = \mathrm{n}\, \emptyset\, \ell\, \bar{v}$ where $\ell = \mid \bar{v} \mid$ then one of the following holds:
   (a) $\exists\, s_2' \ :\ s_2 \xrightarrow{\mathrm{n}\, \emptyset\, \ell\, \bar{v}} s_2'$ and $s_1' \,\mathcal{R}_{\mathcal{N}}\, s_2'$ (b) $\exists\, s_2' \ :\ s_2 \xrightarrow{\emptyset} s_2'$ and $s_1' \,\mathcal{R}_{\mathcal{N}}\, s_2'$
3. if $\alpha = \mathrm{n} \lhd [\bar{n}]\, \bar{v}$ where $\mathrm{n} \notin \mathcal{N}$ then $\exists\, s_2' \ :\ s_2 \xrightarrow{\mathrm{n} \lhd [\bar{n}]\, \bar{v}} s_2'$ and $s_1' \,\mathcal{R}_{\mathcal{N} \cup \bar{n}}\, s_2'$
4. if $\alpha = \emptyset$, $\alpha = \dagger$ or $\alpha = \mathrm{n}\, \emptyset\, \ell\, \bar{v}$, where $\ell \neq \mid \bar{v} \mid$, then $\exists\, s_2' \ :\ s_2 \xrightarrow{\alpha} s_2'$ and $s_1' \,\mathcal{R}_{\mathcal{N}}\, s_2'$

Two closed terms s_1 and s_2 are $\mathcal{N}$*-bisimilar*, written $s_1 \sim^{\mathcal{N}} s_2$, if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ for some $\mathcal{R}_{\mathcal{N}}$ in a labelled bisimulation. They are *labelled bisimilar*, written $s_1 \sim s_2$, if they are $\emptyset$-bisimilar. $\sim^{\mathcal{N}}$ is called $\mathcal{N}$-bisimilarity, while $\sim$ is called *labelled bisimilarity*

A co-inductive notion of bisimulation

Labelled bisimilarity $\sim$

A names-indexed family of symmetric binary relations $\{\mathcal{R}_{\mathcal{N}}\}_{\mathcal{N}}$ is a *labelled bisimulation* if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ then $\mathrm{halt}(s_1)\, \mathcal{R}_{\mathcal{N}}\, \mathrm{halt}(s_2)$ and if $s_1 \xrightarrow{\alpha} s_1'$, where $\mathrm{bu}(\alpha)$ are fresh, then:

1. if s_1 performs a *receive* then one of the following holds:
   (a) s_2 performs the same receive and the continuations stand in the same relation for any matching tuple of values that can be effectively received
   (b) if the argument of the receive contains only variables or is the empty tuple, s_2 performs an internal action leading to a term that, composed with the consumed invoke, stands in the same relation
2. if s_1 performs a *communication* involving an unobservable receive then: s_2 performs (a) the same action or (b) an internal action and . . .
3. if $\alpha = \mathrm{n} \lhd [\bar{n}]\, \bar{v}$ where $\mathrm{n} \notin \mathcal{N}$ then $\exists\, s_2' \;:\; s_2 \xrightarrow{\mathrm{n} \lhd [\bar{n}]\, \bar{v}} s_2'$ and $s_1' \,\mathcal{R}_{\mathcal{N} \cup \bar{n}}\, s_2'$
4. if $\alpha = \emptyset$, $\alpha = \dagger$ or $\alpha = \mathrm{n}\, \emptyset\, \ell\, \bar{v}$, where $\ell \neq |\, \bar{v}\, |$, then $\exists\, s_2' \;:\; s_2 \xrightarrow{\alpha} s_2'$ and $s_1' \,\mathcal{R}_{\mathcal{N}}\, s_2'$

Two closed terms s_1 and s_2 are $\mathcal{N}$*-bisimilar*, written $s_1 \sim^{\mathcal{N}} s_2$, if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ for some $\mathcal{R}_{\mathcal{N}}$ in a labelled bisimulation. They are *labelled bisimilar*, written $s_1 \sim s_2$, if they are $\emptyset$-bisimilar. $\sim^{\mathcal{N}}$ is called $\mathcal{N}$-bisimilarity, while $\sim$ is called *labelled bisimilarity*

A co-inductive notion of bisimulation

Labelled bisimilarity $\sim$

A names-indexed family of symmetric binary relations $\{\mathcal{R}_{\mathcal{N}}\}_{\mathcal{N}}$ is a *labelled bisimulation* if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ then $\mathrm{halt}(s_1)\,\mathcal{R}_{\mathcal{N}}\,\mathrm{halt}(s_2)$ and if $s_1 \xrightarrow{\alpha} s_1'$, where $\mathrm{bu}(\alpha)$ are fresh, then:

1. if s_1 performs a *receive* then one of the following holds:
   - (a) s_2 performs the same receive and the continuations stand in the same relation for any matching tuple of values that can be effectively received
   - (b) if the argument of the receive contains only variables or is the empty tuple, s_2 performs an internal action leading to a term that, composed with the consumed invoke, stands in the same relation
2. if s_1 performs a *communication* involving an unobservable receive then: s_2 performs (a) the same action or (b) an internal action and ...
3. if s_1 performs an *invoke* then s_2 performs the same invoke and ...
4. if $\alpha = \emptyset$, $\alpha = \dagger$ or $\alpha = \mathrm{n}\,\emptyset\,\ell\,\bar{v}$, where $\ell \neq |\,\bar{v}\,|$, then $\exists\, s_2' \;:\; s_2 \xrightarrow{\alpha} s_2'$ and $s_1' \,\mathcal{R}_{\mathcal{N}}\, s_2'$

Two closed terms s_1 and s_2 are $\mathcal{N}$*-bisimilar*, written $s_1 \sim^{\mathcal{N}} s_2$, if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ for some $\mathcal{R}_{\mathcal{N}}$ in a labelled bisimulation. They are *labelled bisimilar*, written $s_1 \sim s_2$, if they are $\emptyset$-bisimilar. $\sim^{\mathcal{N}}$ is called $\mathcal{N}$-bisimilarity, while $\sim$ is called *labelled bisimilarity*

A co-inductive notion of bisimulation

Labelled bisimilarity $\sim$

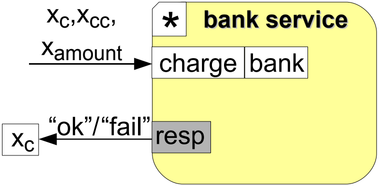
A names-indexed family of symmetric binary relations $\{\mathcal{R}_{\mathcal{N}}\}_{\mathcal{N}}$ is a *labelled bisimulation* if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ then $\mathrm{halt}(s_1)\, \mathcal{R}_{\mathcal{N}}\, \mathrm{halt}(s_2)$ and if $s_1 \xrightarrow{\alpha} s_1'$, where $\mathrm{bu}(\alpha)$ are fresh, then:

1. if s_1 performs a *receive* then one of the following holds:
   - (a) s_2 performs the same receive and the continuations stand in the same relation for any matching tuple of values that can be effectively received
   - (b) if the argument of the receive contains only variables or is the empty tuple, s_2 performs an internal action leading to a term that, composed with the consumed invoke, stands in the same relation
2. if s_1 performs a *communication* involving an unobservable receive then: s_2 performs (a) the same action or (b) an internal action and ...
3. if s_1 performs an *invoke* then s_2 performs the same invoke and ...
4. if s_1 performs either an *internal action*, a *kill* or a *communication* involving an observable receive then s_2 performs the same action and ...

Two closed terms s_1 and s_2 are $\mathcal{N}$*-bisimilar*, written $s_1 \sim^{\mathcal{N}} s_2$, if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ for some $\mathcal{R}_{\mathcal{N}}$ in a labelled bisimulation. They are *labelled bisimilar*, written $s_1 \sim s_2$, if they are $\emptyset$-bisimilar. $\sim^{\mathcal{N}}$ is called $\mathcal{N}$-bisimilarity, while $\sim$ is called *labelled bisimilarity*

Observational semantics at work: bank service

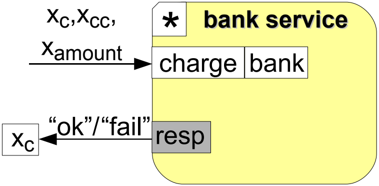
We can compare the high-level specification



$$* \, [x_c, x_{cc}, x_{amount}] \\ \mathrm{bank} \bullet \mathrm{charge}?\langle x_c, x_{cc}, x_{amount}\rangle . \\ x_c \bullet \mathrm{resp}!\langle chk(x_{cc}, x_{amount})\rangle$$

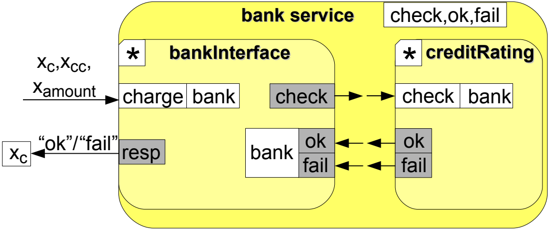
Observational semantics at work: bank service

We can compare the high-level specification



$$* [x_c, x_{cc}, x_{amount}]\ \text{bank} \bullet \text{charge}?\langle x_c, x_{cc}, x_{amount}\rangle .\ x_c \bullet \text{resp}!\langle chk(x_{cc}, x_{amount})\rangle$$

with the low-level specification

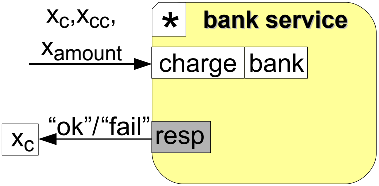


$$[\text{check}, \text{ok}, \text{fail}]\ (\, * \text{bankInterface} \mid * \text{creditRating}\,)$$

$$\begin{aligned}
\text{bankInterface} \triangleq\ & [x_c, x_{cc}, x_{amount}] \\
& \text{bank} \bullet \text{charge}?\langle x_c, x_{cc}, x_{amount}\rangle . \\
& (\,\text{bank} \bullet \text{check}!\langle x_{cc}, x_{amount}\rangle \\
& \mid \text{bank} \bullet \text{ok}?\langle x_{cc}\rangle .\ x_c \bullet \text{resp}!\langle \text{“ok”}\rangle \\
& \quad + \text{bank} \bullet \text{fail}?\langle x_{cc}\rangle .\ x_c \bullet \text{resp}!\langle \text{“fail”}\rangle)
\end{aligned}$$

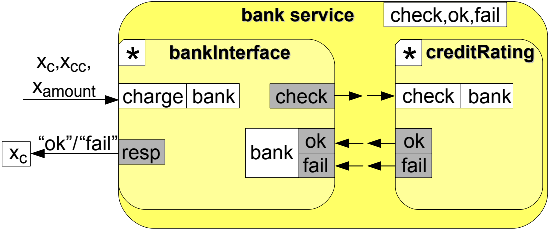
Observational semantics at work: bank service

We can compare the high-level specification



$$*\,[x_c, x_{cc}, x_{amount}]\ \mathrm{bank} \bullet \mathrm{charge}?\langle x_c, x_{cc}, x_{amount}\rangle.\ x_c \bullet \mathrm{resp}!\langle chk(x_{cc}, x_{amount})\rangle$$

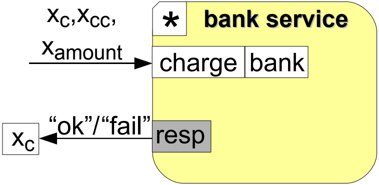
with the low-level specification



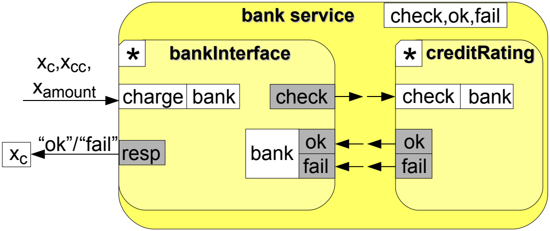
$$[\mathrm{check}, \mathrm{ok}, \mathrm{fail}]\ (\,*\,\mathrm{bankInterface} \mid *\,\mathrm{creditRating}\,)$$

$$\begin{aligned}
&\mathrm{creditRating} \triangleq \\
&[x_{cc}, x_a] \\
&\mathrm{bank} \bullet \mathrm{check}?\langle x_{cc}, x_a\rangle. \\
&[p, o]\,(\,p \bullet o!\langle chk(x_{cc}, x_{amount})\rangle \\
&\quad \mid p \bullet o?\langle \text{"ok"}\rangle.\,\mathrm{bank} \bullet \mathrm{ok}!\langle x_{cc}\rangle \\
&\quad\ + p \bullet o?\langle \text{"fail"}\rangle.\,\mathrm{bank} \bullet \mathrm{fail}!\langle x_{cc}\rangle\,)
\end{aligned}$$

Observational semantics at work: bank service



≈



Examples: observation of receive actions

Asynchronous π-calculus: the input absorption law

$$\tau + a(b).\,\bar{a}b \sim \tau$$

COWS without priority: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \sim \emptyset$$

where $\emptyset \triangleq [p', o']\,(p' \bullet o'!\langle\rangle \mid p' \bullet o'?\langle\rangle)$

COWS: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \not\sim \emptyset$$

since $\mathbb{C} \triangleq [y, z]\, p \bullet o?\langle y, z\rangle.\, p'' \bullet o''!\langle\rangle \mid p \bullet o!\langle v', v\rangle \mid [\![\cdot]\!]$ can distinguish them

However

$$[x, v]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \sim \emptyset$$

Examples: observation of receive actions

Asynchronous π-calculus: the input absorption law

$$\tau + a(b).\,\bar{a}b \sim \tau$$

COWS without priority: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \sim \emptyset$$

where $\emptyset \triangleq [p', o']\,(p' \bullet o'!\langle\rangle \mid p' \bullet o'?\langle\rangle)$

COWS: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \not\sim \emptyset$$

since $\mathbb{C} \triangleq [y, z]\, p \bullet o?\langle y, z\rangle.\, p'' \bullet o''!\langle\rangle \mid p \bullet o!\langle v', v\rangle \mid [\![\cdot]\!]$ can distinguish them

However

$$[x, y]\,(\,\emptyset + p \bullet o?\langle x, y\rangle.\, p \bullet o!\langle x, y\rangle\,) \sim \emptyset$$

Examples: observation of receive actions

Asynchronous π-calculus: the input absorption law

$$\tau + a(b).\,\bar{a}b \;\sim\; \tau$$

COWS without priority: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \;\sim\; \emptyset$$

where $\emptyset \triangleq [p', o']\,(p' \bullet o'!\langle\rangle \mid p' \bullet o'?\langle\rangle)$

COWS: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \;\not\sim\; \emptyset$$

since $\mathbb{C} \triangleq [y, z]\, p \bullet o?\langle y, z\rangle.\, p'' \bullet o''!\langle\rangle \mid p \bullet o!\langle v', v\rangle \mid [\![\cdot]\!]$ can distinguish them

However

$$[x, y]\,(\,\emptyset + p \bullet o?\langle x, y\rangle.\, p \bullet o!\langle x, y\rangle\,) \;\sim\; \emptyset$$

Examples: observation of receive actions

Asynchronous π-calculus: the input absorption law

$$\tau + a(b).\,\bar{a}b \;\sim\; \tau$$

COWS without priority: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \;\sim\; \emptyset$$

where $\emptyset \triangleq [p', o']\,(p' \bullet o'!\langle\rangle \mid p' \bullet o'?\langle\rangle)$

COWS: the receive absorption law

$$[x]\,(\,\emptyset + p \bullet o?\langle x, v\rangle.\, p \bullet o!\langle x, v\rangle\,) \;\not\sim\; \emptyset$$

since $\mathbb{C} \triangleq [y, z]\, p \bullet o?\langle y, z\rangle.\, p'' \bullet o''!\langle\rangle \mid p \bullet o!\langle v', v\rangle \mid [\![\cdot]\!]$ can distinguish them

However

$$[x, y]\,(\,\emptyset + p \bullet o?\langle x, y\rangle.\, p \bullet o!\langle x, y\rangle\,) \;\sim\; \emptyset$$

Analysis techniques: a type system

A type system for confidentiality properties

- Type systems could be a scalable way to provide evidence that a large number of SOC applications enjoy some given properties

Confidentiality properties

Critical data (e.g. credit card information) are shared only by authorized partners

- Our type system permits
  - expressing and forcing policies regulating the exchange of data among interacting services
  - ensuring that, in that respect, services do not manifest unexpected behaviours

Syntax of typed COWS

$s ::=$		(services)
	$\mathbf{kill}(k)$	(kill)
$\mid$	$u \bullet u' ! \langle \{\epsilon_1\}_{r_1}, \ldots, \{\epsilon_n\}_{r_n} \rangle$	(invoke)
$\mid$	$\sum_{i=0}^{r} p_i \bullet o_i ? \bar{w}_i . s_i$	(choice)
$\mid$	$s \mid s$	(parallel)
$\mid$	$\{\mid s \mid\}$	(protection)
$\mid$	$[e]\ s$	(delimitation)
$\mid$	$* s$	(replication)

(notations)
k: *(killer) labels*
ϵ: *expressions*
x: *variables*
v: *values*
n, p, o: *names*
u: *vars* | *names*
w: *vars* | *values*
e: *labels* | *vars* | *names*

Programmers can settle the partners usable to exchange any given datum, thus avoiding the datum be accessed by unwanted services

- Data are annotated with *regions*: $u \bullet u' ! \langle \{\epsilon_1\}_{r_1}, \ldots, \{\epsilon_n\}_{r_n} \rangle$
- Regions $r_1 \ldots r_n$ specify the policies regulating the exchange of the data resulting from evaluation of $\epsilon_1 \ldots \epsilon_n$
- A region r can be either a finite subset of partners and variables or the distinct element $\top$ (denoting the universe of partners)

Syntax of typed COWS

$s ::=$		(services)
	$\mathbf{kill}(k)$	(kill)
$\mid$	$u \bullet u' ! \langle \{\epsilon_1\}_{r_1}, \ldots, \{\epsilon_n\}_{r_n} \rangle$	(invoke)
$\mid$	$\sum_{i=0}^{r} p_i \bullet o_i ? \bar{w}_i . s_i$	(choice)
$\mid$	$s \mid s$	(parallel)
$\mid$	$\{\!\mid s \mid\!\}$	(protection)
$\mid$	$[e]\, s$	(delimitation)
$\mid$	$* \, s$	(replication)

(notations)
k: *(killer) labels*
ϵ: *expressions*
x: *variables*
v: *values*
n, p, o: *names*
u: *vars* | *names*
w: *vars* | *values*
e: *labels* | *vars* | *names*

Programmers can settle the partners usable to exchange any given datum, thus avoiding the datum be accessed by unwanted services

- Data are annotated with *regions*: $u \bullet u' ! \langle \{\epsilon_1\}_{r_1}, \ldots, \{\epsilon_n\}_{r_n} \rangle$
- Regions $r_1 \ldots r_n$ specify the policies regulating the exchange of the data resulting from evaluation of $\epsilon_1 \ldots \epsilon_n$
- A region r can be either a finite subset of partners and variables or the distinct element $\top$ (denoting the universe of partners)

Static and dynamic semantics

Static semantics

A static type system infers region annotations for variable declarations and returns well-typed terms

Dynamic semantics

The operational semantics exploits region annotations to authorize or block the exchange of data

Static semantic

- The *static* type inference system has two main tasks
  - performs some coherence checks
    e.g. the partner used by an invoke must belong to the regions of all data occurring in the argument of the activity
  - derives the minimal region annotations for variable declarations that ensure consistency of services initial configuration
    - $[\{x\}^r]\, s$ means that the datum that dynamically will replace x will be used at most by the partners in r

- Typing judgements are written $\Gamma \vdash s \succ \Gamma' \vdash s'$, where the type environment Γ is a finite function from variables to regions

- s is *well-typed* if $\emptyset \vdash s' \succ \emptyset \vdash s$, for some s'
  i.e. s is the (typed) service obtained by decorating s' with the regions describing the use of each variable of s' in its scope

Static semantic

- The *static* type inference system has two main tasks
  - performs some coherence checks
    e.g. the partner used by an invoke must belong to the regions of all data occurring in the argument of the activity
  - derives the minimal region annotations for variable declarations that ensure consistency of services initial configuration
    - $[\{x\}^r]\, s$ means that the datum that dynamically will replace x will be used at most by the partners in r

- Typing judgements are written $\Gamma \vdash s \succ \Gamma' \vdash s'$, where the type environment Γ is a finite function from variables to regions

- s is *well-typed* if $\emptyset \vdash s' \succ \emptyset \vdash s$, for some s'
  i.e. s is the (typed) service obtained by decorating s' with the regions describing the use of each variable of s' in its scope

Static semantics : significant typing rules

- Rule for (monadic) invoke activity:

$$\frac{u \in r}{\Gamma \vdash u \bullet u'!\{e(\bar{y})\}_r \succ (\Gamma + \{x : r\}_{x \in \bar{y}}) \vdash u \bullet u'!\{e(\bar{y})\}_r}$$

  - it checks if the invoked partner u belongs to the region of the datum
  - if it succeeds, the type environment Γ is extended by associating a proper region to each variable used in the argument expression e

- Rule for variable delimitation:

$$\frac{\Gamma \uplus \{x : \emptyset\} \vdash s \succ \Gamma' \uplus \{x : r\} \vdash s' \qquad x \notin \mathrm{reg}(\Gamma')}{\Gamma \vdash [x]\, s \succ \Gamma' \vdash [\{x\}^{r-\{x\}}]\, s'}$$

  - it annotates the delimitation with the region associated to it by the type environment
  - premiss $x \notin \mathrm{reg}(\Gamma')$ and annotation $r - \{x\}$ prevent initially closed services to become open at the end of the inference

Static semantics : significant typing rules

- Rule for (monadic) invoke activity:

$$\frac{u \in r}{\Gamma \vdash u \bullet u'!\{e(\bar{y})\}_r \succ (\Gamma + \{x : r\}_{x \in \bar{y}}) \vdash u \bullet u'!\{e(\bar{y})\}_r}$$

  - it checks if the invoked partner u belongs to the region of the datum
  - if it succeeds, the type environment Γ is extended by associating a proper region to each variable used in the argument expression e

- Rule for variable delimitation:

$$\frac{\Gamma \uplus \{x : \emptyset\} \vdash s \succ \Gamma' \uplus \{x : r\} \vdash s' \qquad x \notin \mathrm{reg}(\Gamma')}{\Gamma \vdash [x]\, s \succ \Gamma' \vdash [\{x\}^{r-\{x\}}]\, s'}$$

  - it annotates the delimitation with the region associated to it by the type environment
  - premiss $x \notin \mathrm{reg}(\Gamma')$ and annotation $r - \{x\}$ prevent initially closed services to become open at the end of the inference

Static semantics : the other rules

$$\Gamma \vdash \mathbf{0} \succ \Gamma \vdash \mathbf{0} \qquad\qquad \Gamma \vdash \mathbf{kill}(k) \succ \Gamma \vdash \mathbf{kill}(k)$$

$$\frac{\Gamma + \{x : \{p\}\}_{x \in var(\bar{w})} \vdash s \succ \Gamma' \vdash s'}{\Gamma \vdash p \bullet o?\bar{w}.s \succ \Gamma' \vdash p \bullet o?\bar{w}.s'}$$

$$\frac{\Gamma \vdash g_1 \succ \Gamma_1 \vdash g_1' \qquad \Gamma \vdash g_2 \succ \Gamma_2 \vdash g_2'}{\Gamma \vdash g_1 + g_2 \succ \Gamma_1 + \Gamma_2 \vdash g_1' + g_2'}$$

$$\frac{\Gamma \vdash s_1 \succ \Gamma_1 \vdash s_1' \qquad \Gamma \vdash s_2 \succ \Gamma_2 \vdash s_2'}{\Gamma \vdash s_1 \mid s_2 \succ \Gamma_1 + \Gamma_2 \vdash s_1' \mid s_2'}$$

Static semantics : the other rules

$$\frac{\Gamma \vdash s \succ \Gamma' \vdash s' \qquad n \notin \mathrm{reg}(\Gamma')}{\Gamma \vdash [n]\, s \succ \Gamma' \vdash [n]\, s'}$$

$$\frac{\Gamma \vdash s \succ \Gamma' \vdash s'}{\Gamma \vdash [k]\, s \succ \Gamma' \vdash [k]\, s'}$$

$$\frac{\Gamma \vdash s \succ \Gamma' \vdash s'}{\Gamma \vdash \{\!|s|\!\} \succ \Gamma' \vdash \{\!|s'|\!\}}$$

$$\frac{\Gamma \vdash s \succ \Gamma' \vdash s'}{\Gamma \vdash *s \succ \Gamma' \vdash *s'}$$

Dynamic semantics

- The language operational semantics only performs efficiently implementable checks to authorize or block communication
  - types are just sets of (partner) names
  - the region annotation (policy) of output data must contain the region annotation of the corresponding input variables

- The most significant modified rule:

$$\frac{s \xrightarrow{\mathrm{n}\,\sigma \uplus \{x \mapsto \{v\}_r\}\,\ell\,\bar{v}} s' \qquad r' \cdot \sigma \subseteq r}{[\{x\}^{r'}]\, s \xrightarrow{\mathrm{n}\,\sigma\,\ell\,\bar{v}} s' \cdot \{x \mapsto \{v\}_r\}}$$

Dynamic semantics

- The language operational semantics only performs efficiently implementable checks to authorize or block communication
  - types are just sets of (partner) names
  - the region annotation (policy) of output data must contain the region annotation of the corresponding input variables

- The most significant modified rule:

$$\frac{s \xrightarrow{\mathrm{n}\,\sigma \uplus \{x \mapsto \{v\}_r\}\,\ell\,\bar{v}} s' \qquad r' \cdot \sigma \subseteq r}{[\{x\}^{r'}]\, s \xrightarrow{\mathrm{n}\,\sigma\,\ell\,\bar{v}} s' \cdot \{x \mapsto \{v\}_r\}}$$

Results

Major results

Subject reduction & *type safety* results imply that services always comply with the constraints (expressed by the type) of each datum

- *Subject reduction* states that well-typedness is preserved along computations
- *Type safety* states that well-typed services do respect region annotations

Results

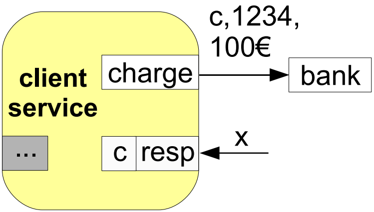
Major results

Subject reduction & *type safety* results imply that services always comply with the constraints (expressed by the type) of each datum

Soundness

A service s is *sound* if, for any datum v occurring in s associated to region r and for all possible evolutions of s, it holds that v can only be exchanged using partners in r

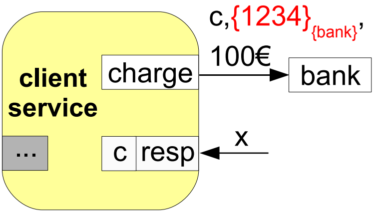
The bank service with security policies



client $\triangleq$
bank • charge!⟨c, 1234, 100€⟩
| [x] (c • resp?⟨x⟩.s | s′)

Client policy : *only* bank is authorized to access credit card data

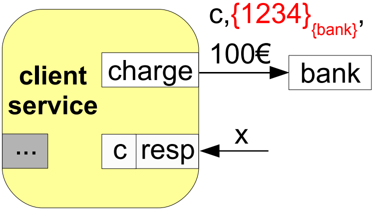
The bank service with security policies



Tclient $\triangleq$
bank • charge!⟨c, {1234}$_{\{bank\}}$, 100€⟩
| [x] (c • resp?⟨x⟩.s | s′)

Client policy : *only* bank is authorized to access credit card data

The bank service with security policies

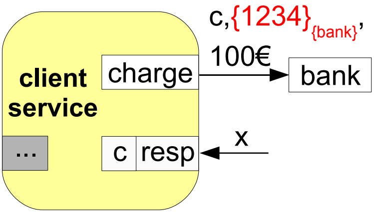


Tclient $\triangleq$
bank • charge!⟨c, $\{1234\}_{\{bank\}}$, 100€⟩
| [x] (c • resp?⟨x⟩.s | s′)

Client policy : *only* bank is authorized to access credit card data

The type system infers the region annotations for the bank service, e.g. . . .

The bank service with security policies

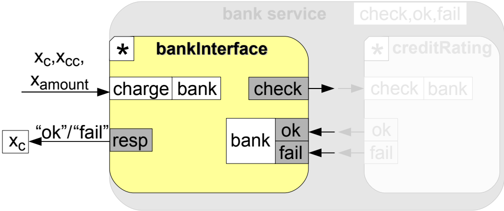


Tclient $\triangleq$

$\text{bank} \bullet \text{charge}!\langle \text{c}, \{1234\}_{\{\text{bank}\}}, 100€ \rangle$
$\mid [x] \, (\, \text{c} \bullet \text{resp}?\langle x \rangle.s \mid s' \,)$

Client policy : *only* bank is authorized to access credit card data

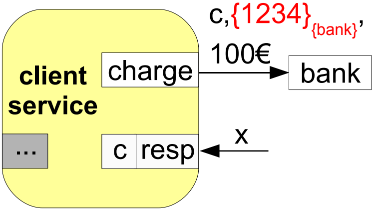
The type system infers the region annotations for the bank service, e.g. . . .



bankInterface $\triangleq$

$[x_c, x_{cc}, x_{amount}]$
$\text{bank} \bullet \text{charge}?\langle x_c, x_{cc}, x_{amount} \rangle.$
$(\, \text{bank} \bullet \text{check}!\langle x_{cc}, x_{amount} \rangle$
$\mid \text{bank} \bullet \text{ok}?\langle x_{cc} \rangle.\, x_c \bullet \text{resp}!\langle \text{“ok”} \rangle$
$+ \,\text{bank} \bullet \text{fail}?\langle x_{cc} \rangle.\, x_c \bullet \text{resp}!\langle \text{“fail”} \rangle \,)$

The bank service with security policies

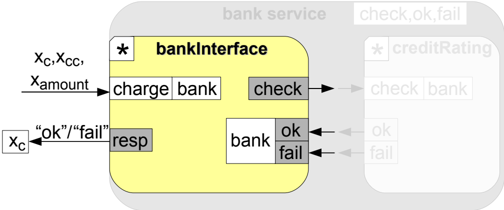


$\text{Tclient} \triangleq$

$\text{bank} \bullet \text{charge}!\langle c, \{1234\}_{\{bank\}}, 100€\rangle$
$\mid [x]\,(\, c \bullet \text{resp}?\langle x\rangle.s \mid s'\,)$

Client policy : *only* bank is authorized to access credit card data

The type system infers the region annotations for the bank service, e.g. . . .



$\text{TbankInterface} \triangleq$

$[\{x_c\}^{\{bank\}}, \{x_{cc}\}^{\{bank\}}, \{x_{amount}\}^{\{bank\}}]$
$\text{bank} \bullet \text{charge}?\langle x_c, x_{cc}, x_{amount}\rangle.$
$(\,\text{bank} \bullet \text{check}!\langle x_{cc}, x_{amount}\rangle$
$\mid \text{bank} \bullet \text{ok}?\langle x_{cc}\rangle.\, x_c \bullet \text{resp}!\langle \text{“ok”}\rangle$
$+\, \text{bank} \bullet \text{fail}?\langle x_{cc}\rangle.\, x_c \bullet \text{resp}!\langle \text{“fail”}\rangle\,)$

The bank service with security policies

$$\text{Tclient} \triangleq \text{bank} \bullet \text{charge}!\langle \text{c}, \{1234\}_{\{\text{bank}\}}, 100€\rangle \mid [x]\,(\,\text{c} \bullet \text{resp}?\langle x\rangle.s \mid s'\,)$$

Client policy: *only* bank is authorized to access credit card data

$$\begin{aligned}\text{TbankInterface} \triangleq\ & [\{x_c\}^{\{\text{bank}\}}, \{x_{cc}\}^{\{\text{bank}\}}, \{x_{amount}\}^{\{\text{bank}\}}] \\ & \text{bank} \bullet \text{charge}?\langle x_c, x_{cc}, x_{amount}\rangle. \\ & (\,\text{bank} \bullet \text{check}!\langle x_{cc}, x_{amount}\rangle \\ & \ \mid \text{bank} \bullet \text{ok}?\langle x_{cc}\rangle.\, x_c \bullet \text{resp}!\langle \text{“ok”} \rangle \\ & \quad + \text{bank} \bullet \text{fail}?\langle x_{cc}\rangle.\, x_c \bullet \text{resp}!\langle \text{“fail”} \rangle\,)\end{aligned}$$

By using the statically inferred annotations, . . .

The bank service with security policies

$$\mathsf{Tclient} \triangleq \mathsf{bank} \bullet \mathsf{charge}!\langle \mathsf{c}, \{1234\}_{\{\mathsf{bank}\}}, 100€ \rangle \mid [\mathsf{x}]\,(\,\mathsf{c} \bullet \mathsf{resp}?\langle \mathsf{x} \rangle.\mathsf{s} \mid \mathsf{s}'\,)$$

Client policy: *only* bank is authorized to access credit card data

$$\begin{aligned}\mathsf{TbankInterface} \triangleq\ & [\{\mathsf{x_c}\}^{\{\mathsf{bank}\}}, \{\mathsf{x_{cc}}\}^{\{\mathsf{bank}\}}, \{\mathsf{x_{amount}}\}^{\{\mathsf{bank}\}}]\\ & \mathsf{bank} \bullet \mathsf{charge}?\langle \mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}} \rangle.\\ & (\,\mathsf{bank} \bullet \mathsf{check}!\langle \mathsf{x_{cc}}, \mathsf{x_{amount}} \rangle\\ & \ \mid \mathsf{bank} \bullet \mathsf{ok}?\langle \mathsf{x_{cc}} \rangle.\, \mathsf{x_c} \bullet \mathsf{resp}!\langle \text{“ok”} \rangle\\ & \quad + \mathsf{bank} \bullet \mathsf{fail}?\langle \mathsf{x_{cc}} \rangle.\, \mathsf{x_c} \bullet \mathsf{resp}!\langle \text{“fail”} \rangle\,)\end{aligned}$$

By using the statically inferred annotations, the operational semantics guarantees that the content of $\mathsf{x_{cc}}$ cannot become available to other services

$$\mathsf{Tclient} \mid [\mathsf{check}, \mathsf{ok}, \mathsf{fail}]\,(\,*\,\mathsf{TbankInterface} \mid *\,\mathsf{creditRating}\,) \longrightarrow \ \ldots$$

Indeed, $\mathsf{region}(\{\mathsf{x_{cc}}\}^{\{\mathsf{bank}\}}) \subseteq \mathsf{region}(\{1234\}_{\{\mathsf{bank}\}})$

A malicious bank service

$$\begin{array}{rcl} \mathsf{Tclient} & \triangleq & \mathsf{bank} \bullet \mathsf{charge}!\langle \mathsf{c}, \{1234\}_{\{\mathsf{bank}\}}, 100€\rangle \\ & & \mid [\mathsf{x}]\,(\,\mathsf{c} \bullet \mathsf{resp}?\langle \mathsf{x}\rangle.\mathsf{s} \mid \mathsf{s}'\,) \end{array}$$

Client policy: *only* bank is authorized to access credit card data

$$\begin{array}{rcl} \mathsf{spyBankInterface} & \triangleq & [\mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}}] \\ & & \mathsf{bank} \bullet \mathsf{charge}?\langle \mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}}\rangle. \\ & & (\,\mathsf{spy} \bullet \mathsf{check}!\langle \mathsf{x_{cc}}, \mathsf{x_{amount}}\rangle \\ & & \;\mid \mathsf{bank} \bullet \mathsf{ok}?\langle \mathsf{x_{cc}}\rangle.\,\mathsf{x_c} \bullet \mathsf{resp}!\langle \text{“ok”} \rangle \\ & & \quad + \mathsf{bank} \bullet \mathsf{fail}?\langle \mathsf{x_{cc}}\rangle.\,\mathsf{x_c} \bullet \mathsf{resp}!\langle \text{“fail”} \rangle\,) \end{array}$$

A malicious bank service

$$\begin{aligned}\mathrm{Tclient} \;&\triangleq\; \mathrm{bank}\bullet\mathrm{charge}!\langle \mathrm{c}, \{1234\}_{\{\mathrm{bank}\}}, 100€\rangle \\ &\quad \mid [\mathrm{x}]\,(\,\mathrm{c}\bullet\mathrm{resp}?\langle \mathrm{x}\rangle.\mathrm{s} \mid \mathrm{s}'\,)\end{aligned}$$

Client policy: *only* bank is authorized to access credit card data

$$\begin{aligned}\mathrm{TspyBankInterface} \;&\triangleq\; [\{x_c\}^{\{\mathrm{bank}\}}, \{x_{cc}\}^{\{\mathrm{bank,\,spy}\}}, \{x_{amount}\}^{\{\mathrm{bank,\,spy}\}}] \\ &\quad \mathrm{bank}\bullet\mathrm{charge}?\langle x_c, x_{cc}, x_{amount}\rangle. \\ &\quad (\,\mathrm{spy}\bullet\mathrm{check}!\langle x_{cc}, x_{amount}\rangle \\ &\quad\;\; \mid \mathrm{bank}\bullet\mathrm{ok}?\langle x_{cc}\rangle.\,x_c\bullet\mathrm{resp}!\langle\,\text{“ok”}\,\rangle \\ &\quad\;\;\; + \mathrm{bank}\bullet\mathrm{fail}?\langle x_{cc}\rangle.\,x_c\bullet\mathrm{resp}!\langle\,\text{“fail”}\,\rangle\,)\end{aligned}$$

From the statically inferred annotations, ...

A malicious bank service

$$\begin{aligned}\text{Tclient} \;\triangleq\;& \text{bank} \bullet \text{charge}!\langle \text{c}, \{1234\}_{\{\text{bank}\}}, 100€\rangle \\ & \mid [\text{x}]\,(\,\text{c} \bullet \text{resp}?\langle \text{x}\rangle.\text{s} \mid \text{s}'\,)\end{aligned}$$

Client policy: *only* bank is authorized to access credit card data

$$\begin{aligned}\text{TspyBankInterface} \;\triangleq\;& [\{x_c\}^{\{\text{bank}\}}, \{x_{cc}\}^{\{\text{bank, spy}\}}, \{x_{amount}\}^{\{\text{bank, spy}\}}] \\ & \text{bank} \bullet \text{charge}?\langle x_c, x_{cc}, x_{amount}\rangle . \\ & (\,\text{spy} \bullet \text{check}!\langle x_{cc}, x_{amount}\rangle \\ & \;\mid \text{bank} \bullet \text{ok}?\langle x_{cc}\rangle .\, x_c \bullet \text{resp}!\langle \text{“ok”}\rangle \\ & \;\;+ \text{bank} \bullet \text{fail}?\langle x_{cc}\rangle .\, x_c \bullet \text{resp}!\langle \text{“fail”}\rangle\,)\end{aligned}$$

From the statically inferred annotations, we can see that the contents of x_{cc} and x_{amount} can become available to spy!

A malicious bank service

$$\begin{array}{rcl} \mathsf{Tclient} & \triangleq & \mathsf{bank} \bullet \mathsf{charge}!\langle \mathsf{c}, \{1234\}_{\{\mathsf{bank}\}}, 100€\rangle \\ & & \mid [\mathsf{x}]\,(\,\mathsf{c} \bullet \mathsf{resp}?\langle \mathsf{x}\rangle.\mathsf{s} \mid \mathsf{s}'\,) \end{array}$$

Client policy: *only* bank is authorized to access credit card data

$$\begin{array}{rcl} \mathsf{TspyBankInterface} & \triangleq & [\{\mathsf{x_c}\}^{\{\mathsf{bank}\}}, \{\mathsf{x_{cc}}\}^{\{\mathsf{bank, spy}\}}, \{\mathsf{x_{amount}}\}^{\{\mathsf{bank, spy}\}}] \\ & & \mathsf{bank} \bullet \mathsf{charge}?\langle \mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}}\rangle. \\ & & (\,\mathsf{spy} \bullet \mathsf{check}!\langle \mathsf{x_{cc}}, \mathsf{x_{amount}}\rangle \\ & & \;\mid \mathsf{bank} \bullet \mathsf{ok}?\langle \mathsf{x_{cc}}\rangle.\,\mathsf{x_c} \bullet \mathsf{resp}!\langle\,\text{“ok”}\,\rangle \\ & & \;\;+ \mathsf{bank} \bullet \mathsf{fail}?\langle \mathsf{x_{cc}}\rangle.\,\mathsf{x_c} \bullet \mathsf{resp}!\langle\,\text{“fail”}\,\rangle\,) \end{array}$$

From the statically inferred annotations, we can see that the contents of $\mathsf{x_{cc}}$ and $\mathsf{x_{amount}}$ can become available to spy!

The operational semantics does block the transition

$$\mathsf{Tclient} \mid [\mathsf{check}, \mathsf{ok}, \mathsf{fail}]\,(\,*\,\mathsf{TspyBankInterface} \mid *\,\mathsf{creditRating}\,) \not\longrightarrow$$

Indeed, $\mathsf{region}(\{\mathsf{x_{cc}}\}^{\{\mathsf{bank, spy}\}}) \not\subseteq \mathsf{region}(\{1234\}_{\{\mathsf{bank}\}})$

The bank service: a bank policy

$$\begin{array}{rcl} \mathsf{TclientKey} & \triangleq & \mathsf{bank} \bullet \mathsf{charge}!\langle \mathsf{c}, \{1234\}_{\{\mathsf{bank}\}}, 100€\rangle \\ & & \mid [\mathsf{x}, \mathsf{y_{key}}]\,(\,\mathsf{c} \bullet \mathsf{resp}?\langle \mathsf{x}, \mathsf{y_{key}}\rangle.\mathsf{s} \mid \mathsf{s}'\,) \end{array}$$

The client can also receive a personal secret key to be used for successive operations

The bank service: a bank policy

$$\mathsf{TclientKey} \triangleq \mathsf{bank} \bullet \mathsf{charge}!\langle c, \{1234\}_{\{bank\}}, 100€\rangle \mid [x, y_{key}]\,(\, c \bullet \mathsf{resp}?\langle x, y_{key}\rangle.s \mid s' \,)$$

The client can also receive a personal secret key to be used for successive operations

$$\begin{aligned} \mathsf{bankKeyInterface} \triangleq\ & [x_c, x_{cc}, x_{amount}] \\ & \mathsf{bank} \bullet \mathsf{charge}?\langle x_c, x_{cc}, x_{amount}\rangle. \\ & (\, \mathsf{bank} \bullet \mathsf{check}!\langle x_{cc}, x_{amount}\rangle \\ & \ \mid \mathsf{bank} \bullet \mathsf{ok}?\langle x_{cc}\rangle.\, x_c \bullet \mathsf{resp}!\langle \text{“ok”}, \{key\}_{\{x_c, bank\}}\rangle \\ & \quad + \mathsf{bank} \bullet \mathsf{fail}?\langle x_{cc}\rangle.\, x_c \bullet \mathsf{resp}!\langle \text{“fail”}, \mathsf{null}\rangle\,) \end{aligned}$$

Policy: the bank service wants to guarantees that the key sent to the client is not disclosed to third parties

The bank service: a bank policy

$$\mathsf{TclientKey} \triangleq \mathsf{bank} \bullet \mathsf{charge}!\langle \mathsf{c}, \{1234\}_{\{\mathsf{bank}\}}, 100€\rangle \mid [\mathsf{x}, \mathsf{y}_{\mathsf{key}}]\,(\mathsf{c} \bullet \mathsf{resp}?\langle \mathsf{x}, \mathsf{y}_{\mathsf{key}}\rangle.\mathsf{s} \mid \mathsf{s}')$$

The client can also receive a personal secret key to be used for successive operations

$$\begin{aligned}\mathsf{bankKeyInterface} \triangleq\ & [\mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}}] \\ & \mathsf{bank} \bullet \mathsf{charge}?\langle \mathsf{x_c}, \mathsf{x_{cc}}, \mathsf{x_{amount}}\rangle. \\ & (\,\mathsf{bank} \bullet \mathsf{check}!\langle \mathsf{x_{cc}}, \mathsf{x_{amount}}\rangle \\ & \ \mid \mathsf{bank} \bullet \mathsf{ok}?\langle \mathsf{x_{cc}}\rangle.\,\mathsf{x_c} \bullet \mathsf{resp}!\langle\, \text{“ok”}, \{\mathsf{key}\}_{\{\mathsf{x_c},\mathsf{bank}\}}\rangle \\ & \quad + \mathsf{bank} \bullet \mathsf{fail}?\langle \mathsf{x_{cc}}\rangle.\,\mathsf{x_c} \bullet \mathsf{resp}!\langle\, \text{“fail”}, \mathsf{null}\rangle\,)\end{aligned}$$

Policy: the bank service wants to guarantees that the key sent to the client is not disclosed to third parties

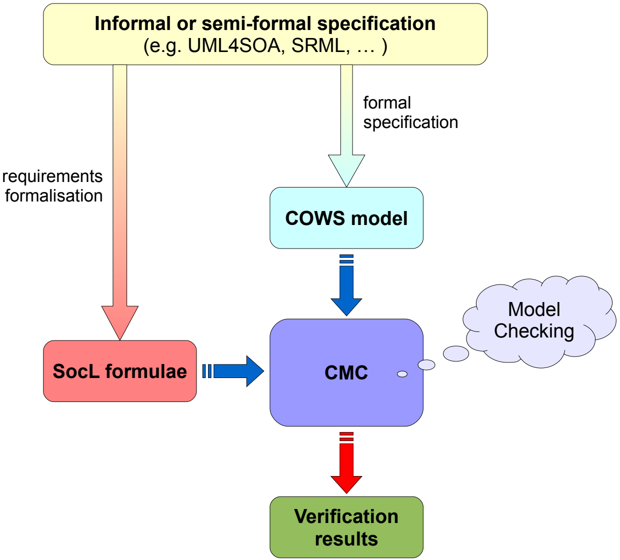
The policy is not fixed at design time, but *depends* on the value of x_c

Analysis techniques: a logical framework

Logics and Model checking

- Process calculi provide behavioral specifications of services
- Logics have been long since proved able to reason about such complex systems as SOC applications
  - provide abstract specifications of these complex systems
  - can be used for describing system properties rather than system behaviors
- Logics and model checkers can be used as tools for verifying that services enjoy desirable properties and do not manifest unexpected behaviors

A logical verification methodology



Requirements formalisation

To formally express service properties we exploit

SocL

an action- and state-based, branching time, temporal logic expressly designed to formalise in a convenient way distinctive aspects of services

action- and state-based logic

⇓

Doubly Labelled Transition Systems (L^2TS) as interpretation domain

⇓

Abstract notion of services

- services are thought of as sw entities which may have an internal state and can interact with each other
- services are characterised by actions and atomic propositions of the form $type/name(interaction, corrTuple)$

Requirements formalisation

To formally express service properties we exploit

SocL

an action- and state-based, branching time, temporal logic expressly designed to formalise in a convenient way distinctive aspects of services

action- and state-based logic

$\Downarrow$

Doubly Labelled Transition Systems (L^2TS) as interpretation domain

$\Downarrow$

Abstract notion of services

- services are thought of as sw entities which may have an internal state and can interact with each other
- services are characterised by actions and atomic propositions of the form $type/name(interaction, corrTuple)$

SocL actions

Actions ($a \in Act$)

have the form $t(i, c)$

- t: type of the action (e.g. *request*, *response*, *fail*, ...)
- i: name of the interaction which the action is part of (e.g. *charge*)
- c: tuple of correlation values and variables identifying the interaction; $\underline{var}$ denotes a binding occurrence of the correlation variable var

Examples

- $request(charge, 1234, 1)$: action starting an (instance of the) interaction *charge* which will be identified through the correlation tuple $\langle 1234, 1 \rangle$

  a corresponding response action can be $response(charge, 1234, 1)$

- $request(charge, 1234, \underline{id})$: request action where the second correlation value is unknown; a (binder for a) correlation variable id is used instead

  a corresponding response action can be $response(charge, 1234, id)$; the (free) occurrence of the correlation variable id indicates the connection with the action where the variable is bound

SocL actions

Actions ($a \in Act$)

have the form $t(i, c)$

- t: type of the action (e.g. *request*, *response*, *fail*, ...)
- i: name of the interaction which the action is part of (e.g. *charge*)
- c: tuple of correlation values and variables identifying the interaction; $\underline{var}$ denotes a binding occurrence of the correlation variable var

Examples

- $request(charge, 1234, 1)$: action starting an (instance of the) interaction *charge* which will be identified through the correlation tuple $\langle 1234, 1 \rangle$

  a corresponding response action can be $response(charge, 1234, 1)$
- $request(charge, 1234, \underline{id})$: request action where the second correlation value is unknown; a (binder for a) correlation variable id is used instead

  a corresponding response action can be $response(charge, 1234, id)$; the (free) occurrence of the correlation variable id indicates the connection with the action where the variable is bound

SocL actions

Actions ($a \in Act$)

have the form $t(i, c)$

- t: type of the action (e.g. *request*, *response*, *fail*, ...)
- i: name of the interaction which the action is part of (e.g. *charge*)
- c: tuple of correlation values and variables identifying the interaction; $\underline{var}$ denotes a binding occurrence of the correlation variable var

Examples

- $request(charge, 1234, 1)$: action starting an (instance of the) interaction *charge* which will be identified through the correlation tuple $\langle 1234, 1 \rangle$

  a corresponding response action can be $response(charge, 1234, 1)$
- $request(charge, 1234, \underline{id})$: request action where the second correlation value is unknown; a (binder for a) correlation variable id is used instead

  a corresponding response action can be $response(charge, 1234, id)$; the (free) occurrence of the correlation variable id indicates the connection with the action where the variable is bound

SocL atomic propositions

Atomic propositions ($\pi \in AP$)

have the form $p(i, c)$

- p: name of the proposition (*accepting_request*, *accepting_cancel*, ...)
- i: name of the interaction (e.g. *charge*)
- c: tuple of correlation values and free variables

Examples

- *accepting_request*(*charge*): proposition indicating that a state can accept requests for the interaction *charge* (regardless of the correlation data)
- *accepting_cancel*(*charge*, 1234, 1): a state permits to cancel those requests for interaction *charge* identified by the correlation tuple $\langle 1234, 1 \rangle$

SocL atomic propositions

Atomic propositions ($\pi \in AP$)

have the form $p(i, c)$

- *p*: name of the proposition (*accepting_request*, *accepting_cancel*, ...)
- *i*: name of the interaction (e.g. *charge*)
- *c*: tuple of correlation values and free variables

Examples

- *accepting_request*(*charge*): proposition indicating that a state can accept requests for the interaction *charge* (regardless of the correlation data)
- *accepting_cancel*(*charge*, 1234, 1): a state permits to cancel those requests for interaction *charge* identified by the correlation tuple $\langle 1234, 1 \rangle$

SocL syntax

State formulae syntax

$$\phi \ ::= \ \mathit{true} \ | \ \pi \ | \ \neg\phi \ | \ \phi \wedge \phi' \ | \ E\Psi \ | \ A\Psi$$

Path formulae syntax

$$\Psi \ ::= \ X_{\gamma}\phi \ | \ \phi \ {}_{\chi}U_{\gamma} \ \phi' \ | \ \phi \ {}_{\chi}W_{\gamma} \ \phi'$$

Action formulae syntax

$$\gamma \ ::= \ \underline{a} \ | \ \chi \qquad\qquad \chi \ ::= \ tt \ | \ a \ | \ \tau \ | \ \neg\chi \ | \ \chi \wedge \chi$$

$\underline{a}$ indicates that the action may contain variables binders

Some derived modalities

$<\gamma>\phi$ stands for	$EX_{\gamma}\,\phi$	$[\gamma]\,\phi$ stands for	$\neg<\gamma>\neg\phi$
$E(\phi\ {}_{\chi}U\ \phi')$ stands for	$\phi' \vee E(\phi_{\chi}U_{\chi\vee\tau}\,\phi')$	$EF\phi$ stands for	$E(\mathit{true}\ {}_{tt}U\phi)$
$AF_{\gamma}\ \mathit{true}$ stands for	$A(\mathit{true}\ {}_{tt}U_{\gamma}\,\mathit{true})$	$AG\,\phi$ stands for	$\neg EF\neg\phi$

SocL syntax

State formulae syntax

$$\phi \;::=\; \textit{true} \;\mid\; \pi \;\mid\; \neg\phi \;\mid\; \phi \wedge \phi' \;\mid\; E\Psi \;\mid\; A\Psi$$

E and *A* are existential and universal (resp.) *path quantifiers*

Action formulae syntax

$$\gamma \;::=\; \underline{a} \;\mid\; \chi \qquad\qquad \chi \;::=\; tt \;\mid\; a \;\mid\; \tau \;\mid\; \neg\chi \;\mid\; \chi \wedge \chi$$

$\underline{a}$ indicates that the action may contain variables binders

Some derived modalities

$< \gamma > \phi$	stands for	$EX_{\gamma}\,\phi$	$[\gamma]\,\phi$	stands for	$\neg < \gamma > \neg\,\phi$
$E(\phi\,{}_{\chi}U\,\phi')$	stands for	$\phi' \vee E(\phi_{\chi} U_{\chi\vee\tau}\,\phi')$	$EF\phi$	stands for	$E(\textit{true}\,{}_{tt}U\phi)$
$AF_{\gamma}\,\textit{true}$	stands for	$A(\textit{true}\,{}_{tt}U_{\gamma}\,\textit{true})$	$AG\,\phi$	stands for	$\neg\,EF\,\neg\,\phi$

SocL syntax

State formulae syntax

$$\phi \quad ::= \quad \mathit{true} \quad | \quad \pi \quad | \quad \neg\phi \quad | \quad \phi \wedge \phi' \quad | \quad E\Psi \quad | \quad A\Psi$$

Path formulae syntax

$$\Psi \quad ::= \quad X_\gamma \phi \quad | \quad \phi\ {}_\chi U_\gamma\ \phi' \quad | \quad \phi\ {}_\chi W_\gamma\ \phi'$$

Action formulae syntax

$$\gamma \quad ::= \quad \underline{a} \quad | \quad \chi \qquad\qquad \chi \quad ::= \quad tt \quad | \quad a \quad | \quad \tau \quad | \quad \neg\chi \quad | \quad \chi \wedge \chi$$

$\underline{a}$ indicates that the action may contain variables binders

Some derived modalities

$< \gamma > \phi$	stands for	$EX_\gamma\ \phi$	$[\gamma]\ \phi$	stands for	$\neg < \gamma > \neg\phi$
$E(\phi\ {}_\chi U\ \phi')$	stands for	$\phi' \vee E(\phi_\chi U_{\chi\vee\tau}\ \phi')$	$EF\phi$	stands for	$E(\mathit{true}\ {}_{tt} U \phi)$
$AF_\gamma\ \mathit{true}$	stands for	$A(\mathit{true}\ {}_{tt} U_\gamma\ \mathit{true})$	$AG\,\phi$	stands for	$\neg\, EF \neg\,\phi$

SocL syntax

State formulae syntax

$$\phi \; ::= \; \mathit{true} \;\mid\; \pi \;\mid\; \neg\phi \;\mid\; \phi \wedge \phi' \;\mid\; E\Psi \;\mid\; A\Psi$$

Path formulae syntax

$$\Psi \; ::= \; X_{\gamma}\phi \;\mid\; \phi\;{}_{\chi}U_{\gamma}\;\phi' \;\mid\; \phi\;{}_{\chi}W_{\gamma}\;\phi'$$

X, U and W are the *next*, *(strong) until* and *weak until* operators

- $X_{\gamma}\phi$ says that in the next state of the path, reached by an action satisfying γ, the formula ϕ holds
- $\phi\;{}_{\chi}U_{\gamma}\;\phi'$ says that ϕ' holds at some future state of the path reached by a last action satisfying γ, while ϕ holds from the current state until that state is reached and all the actions executed in the meanwhile along the path satisfy χ
- $\phi\;{}_{\chi}W_{\gamma}\;\phi'$ holds on a path either if the corresponding strong until operator holds or if for all the states of the path the formula ϕ holds and all the actions of the path satisfy χ

SocL syntax

State formulae syntax

$$\phi \ ::= \ true \ | \ \pi \ | \ \neg\phi \ | \ \phi \wedge \phi' \ | \ E\Psi \ | \ A\Psi$$

Path formulae syntax

$$\Psi \ ::= \ X_\gamma\phi \ | \ \phi \,_\chi U_\gamma\, \phi' \ | \ \phi \,_\chi W_\gamma\, \phi'$$

Action formulae syntax

$$\gamma \ ::= \ \underline{a} \ | \ \chi \qquad\qquad \chi \ ::= \ tt \ | \ a \ | \ \tau \ | \ \neg\chi \ | \ \chi \wedge \chi$$

$\underline{a}$ indicates that the action may contain variables binders

Some derived modalities

$< \gamma > \phi$	stands for	$EX_\gamma\, \phi$	$[\gamma]\, \phi$	stands for	$\neg < \gamma > \neg\phi$
$E(\phi \,_\chi U\, \phi')$	stands for	$\phi' \vee E(\phi_\chi U_{\chi\vee\tau}\, \phi')$	$EF\phi$	stands for	$E(true \,_{tt} U\phi)$
$AF_\gamma\, true$	stands for	$A(true \,_{tt} U_\gamma\, true)$	$AG\,\phi$	stands for	$\neg\, EF \neg\,\phi$

SocL syntax

- $<\gamma>\phi$ states that it is *possible* to perform an action satisfying γ and thereby reaching a state that satisfies formula ϕ
- $[\gamma]\,\phi$ states that no matter how a process performs an action satisfying γ, the state it reaches in doing so will *necessarily* satisfy the formula ϕ
- $EF\phi$ means that there is some path that leads to a state at which ϕ holds; that is, ϕ *eventually* holds on some path
- $AF_\gamma\,\phi$ means that an action satisfying γ will be performed in the future along every path and at the reached states ϕ holds; if ϕ is *true*, we say that an action satisfying γ will *always eventually* be performed
- $AG\,\phi$ states that ϕ holds at every state on every path; that is, ϕ holds *globally*

Some derived modalities

$< \gamma > \phi$	stands for	$EX_\gamma\,\phi$	$[\gamma]\,\phi$	stands for	$\neg < \gamma > \neg\,\phi$
$E(\phi\,{}_\chi U\,\phi')$	stands for	$\phi' \vee E(\phi_\chi U_{\chi\vee\tau}\,\phi')$	$EF\phi$	stands for	$E(true\,{}_{tt}U\phi)$
$AF_\gamma\,true$	stands for	$A(true\,{}_{tt}U_\gamma\,true)$	$AG\,\phi$	stands for	$\neg\,EF\,\neg\,\phi$

SocL description of abstract properties

Availability

the service is always capable to accept a request

$$AG(accepting\_request(i))$$

Reliability

the service guarantees a successful response to each received request

$$AG[request(i, \underline{v})]AF_{response(i,v)}\ true$$

Responsiveness

the service guarantees a response to each received request

$$AG[request(i, \underline{v})]\ AF_{response(i,v) \vee fail(i,v)}\ true$$

...

SocL semantics: action formulae semantics

$\alpha \models \gamma \triangleright \rho$ means: the formula γ is satisfied over the set of closed actions α under substitution ρ

- $\alpha \models \underline{a} \triangleright \rho$ iff $\exists\, b \in \alpha$ such that match($\underline{a}$, b) $= \rho$
- $\alpha \models \chi \triangleright \emptyset$ iff $\alpha \models \chi$

  where the relation $\alpha \models \chi$ is defined as follows

  - $\alpha \models tt$ holds always
  - $\alpha \models a$ iff $a \in \alpha$
  - $\alpha \models \tau$ iff $\alpha = \emptyset$
  - $\alpha \models \neg\chi$ iff not $\alpha \models \chi$
  - $\alpha \models \chi \wedge \chi'$ iff $\alpha \models \chi$ and $\alpha \models \chi'$

SocL semantics

- Let $\langle Q, q_0, Act, R, AP, L\rangle$ be an L^2TS, $q \in Q$ and $\sigma \in path(q)$
- The satisfaction relation of closed SocL formulae, i.e. formulae without unbound variables, is defined as follows

- $q \models true$ holds always
- $q \models \pi$ iff $\pi \in L(q)$
- $q \models \neg\phi$ iff not $q \models \phi$
- $q \models \phi \wedge \phi'$ iff $q \models \phi$ and $q \models \phi'$
- $q \models E\Psi$ iff $\exists \sigma \in path(q) \ : \sigma \models \Psi$
- $q \models A\Psi$ iff $\forall \sigma \in path(q) \ : \sigma \models \Psi$
- $\sigma \models X_\gamma \phi$ iff $\exists \rho : \sigma\{1\} \models \gamma \triangleright \rho$ and $\sigma(2) \models \phi\, \rho$
- ...

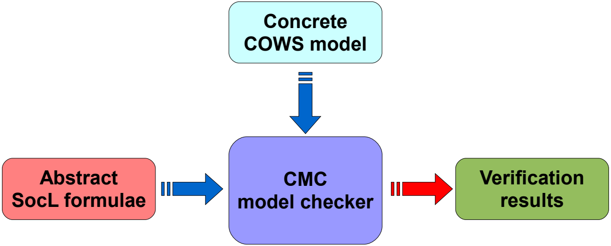
SocL semantics

- Let $\langle Q, q_0, Act, R, AP, L\rangle$ be an L^{2}TS, $q \in Q$ and $\sigma \in path(q)$
- The satisfaction relation of closed SocL formulae, i.e. formulae without unbound variables, is defined as follows

- ...
- $\sigma \models \phi\ {}_{\chi}U_{\gamma}\phi'$ iff $\exists\, j \geq 1$
  $\sigma(j) \models \phi$, and $\exists\, \rho : \sigma\{j\} \models \gamma \rhd \rho$ and $\sigma(j+1) \models \phi'\rho$,
  and $\forall\, 1 \leq i < j :\ \sigma(i) \models \phi$ and $\sigma\{i\} \models \chi$
- $\sigma \models \phi\ {}_{\chi}W_{\gamma}\phi'$ iff either $\sigma \models \phi\ {}_{\chi}U_{\gamma}\phi'$ or $\forall\, i \geq 1 :\ \sigma(i) \models \phi$
  and $\sigma\{i\} \models \chi$

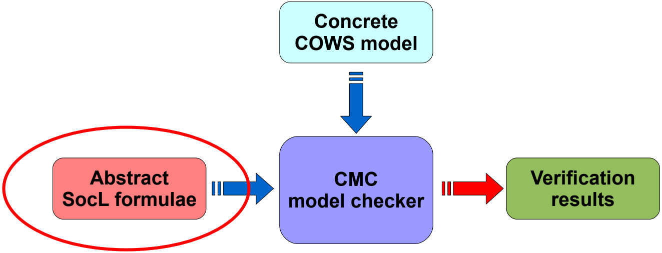
A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms
3. Formulae are tailored to a given specification of a service by means of some abstraction rules that relate actions in the specification with actions of the logic
4. The verification process takes place



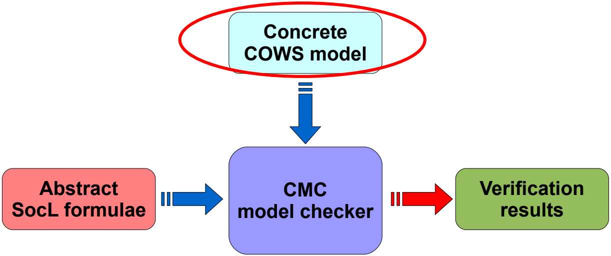
A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms
3. Formulae are tailored to a given specification of a service by means of some abstraction rules that relate actions in the specification with actions of the logic
4. The verification process takes place



A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms
3. Formulae are tailored to a given specification of a service by means of some abstraction rules that relate actions in the specification with actions of the logic
4. The verification process takes place



A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms

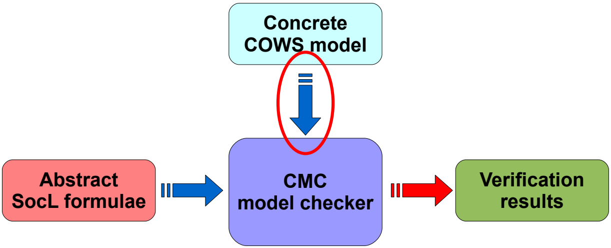
We resort to a linguistic formalism rather than directly using L^2TSs because

- L^2TSs are too low level
- L^2TSs suffer for lack of compositionality,

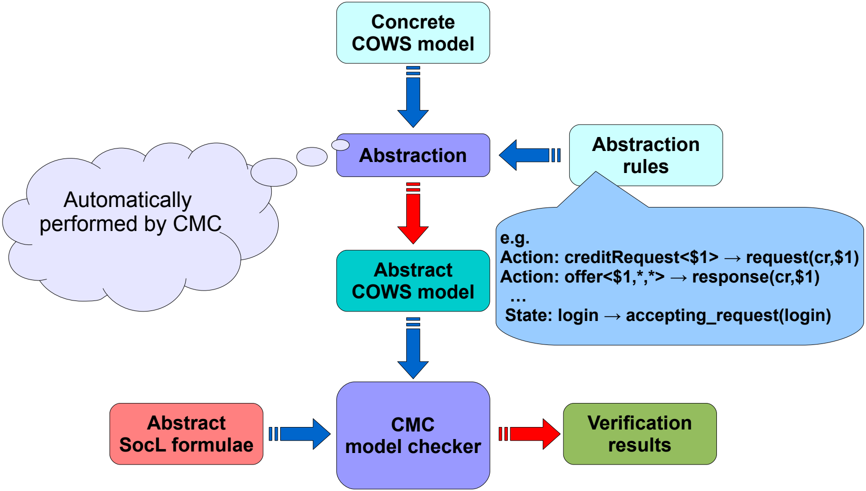
  i.e. they offer no means for constructing the L^2TS of a composed service in terms of the L^2TSs of its components
- linguistic terms are more intuitive and concise notations
- using linguistic terms, services are built in a compositional way
- linguistic terms are syntactically finite, even when the corresponding semantic model (i.e. L^2TSs) is not

A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms
3. Formulae are tailored to a given specification of a service by means of some abstraction rules that relate actions in the specification with actions of the logic
4. The verification process takes place

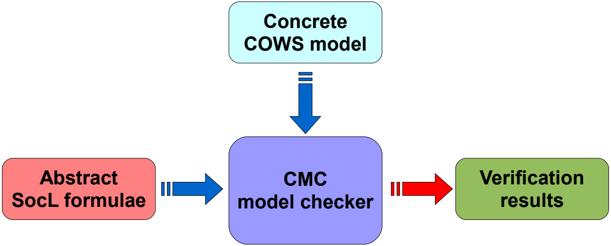


A novel verification methodology of service properties



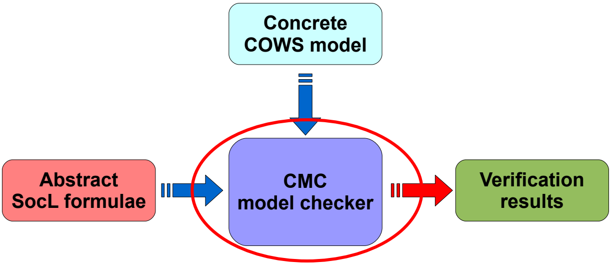
A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms
3. Formulae are tailored to a given specification of a service by means of some abstraction rules that relate actions in the specification with actions of the logic
4. The verification process takes place



A novel verification methodology of service properties

1. Properties are initially formalized as SocL formulae, while preserving their independence from individual service domains and specifications
2. Services behaviour are specified as COWS terms
3. Formulae are tailored to a given specification of a service by means of some abstraction rules that relate actions in the specification with actions of the logic
4. The verification process takes place



The model checker CMC

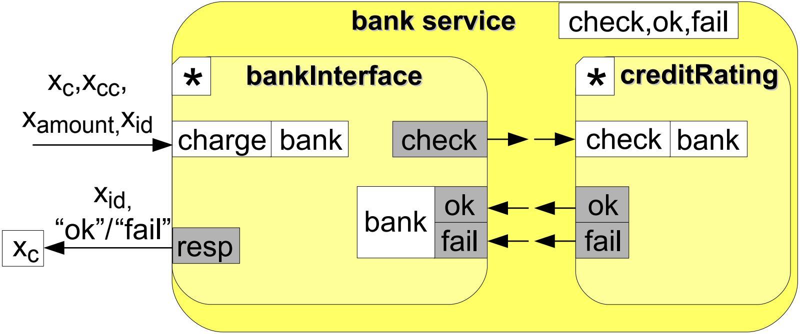
To assist the verification process of SocL formulae over L^2TS

- CMC is an efficient on-the-fly model checker
- The basic idea behind CMC is that, given a state of an L^2TS, the validity of a SocL formula on that state can be established by:
    - checking the satisfiability of the state predicates
    - analyzing the transitions allowed in that state
    - establishing the validity of some subformula in some/all of the next reachable states
- If a SocL formula is not satisfied, a *counterexample* is exhibited

CMC can be used to verify properties of services specified in COWS

CMC can be downloaded or experimented via its web interface at
`http://fmt.isti.cnr.it/cmc`

Model checking the bank service



Model checking the bank service

The instantiation of the generic patterns of formulae over the bank service is obtained by just replacing any occurrence of *i* with *charge*

The bank service is *always available*

$$AG\,(accepting\_request(charge))$$

In every state the service may accept a request for the interaction *charge*

The bank service is *responsive*

$$AG\,[request(charge, \underline{v})]\; AF_{response(charge,v) \vee fail(charge,v)}\; true$$

The response and the failure notification belong to the same interaction *charge* as the accepted request and they are correlated by the variable *v*

The bank service is *reliable*

$$AG\,[request(charge, \underline{v})]\; AF_{response(charge,v)}\; true$$

The service guarantees a successful response to each received request

Model checking the bank service

The instantiation of the generic patterns of formulae over the bank service is obtained by just replacing any occurrence of *i* with *charge*

The bank service is *always available*

$$AG\,(accepting\_request(charge))$$

In every state the service may accept a request for the interaction *charge*

The bank service is *responsive*

$$AG\,[request(charge, \underline{v})]\,AF_{response(charge,v) \vee fail(charge,v)}\ true$$

The response and the failure notification belong to the same interaction *charge* as the accepted request and they are correlated by the variable *v*

The bank service is *reliable*

$$AG\,[request(charge, \underline{v})]\,AF_{response(charge,v)}\ true$$

The service guarantees a successful response to each received request

Model checking the bank service

The instantiation of the generic patterns of formulae over the bank service is obtained by just replacing any occurrence of *i* with *charge*

The bank service is *always available*

$$AG\,(accepting\_request(charge))$$

In every state the service may accept a request for the interaction *charge*

The bank service is *responsive*

$$AG\,[request(charge, \underline{v})]\; AF_{response(charge,v) \vee fail(charge,v)}\; true$$

The response and the failure notification belong to the same interaction *charge* as the accepted request and they are correlated by the variable *v*

The bank service is *reliable*

$$AG\,[request(charge, \underline{v})]\; AF_{response(charge,v)}\; true$$

The service guarantees a successful response to each received request

Model checking the bank service

The instantiation of the generic patterns of formulae over the bank service is obtained by just replacing any occurrence of *i* with *charge*

The bank service is *always available*

$$AG\,(accepting\_request(charge))$$

In every state the service may accept a request for the interaction *charge*

The bank service is *responsive*

$$AG\,[request(charge, \underline{v})]\,AF_{response(charge,v) \vee fail(charge,v)}\ true$$

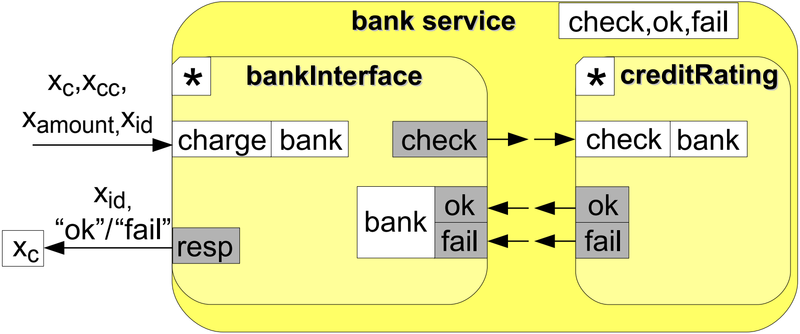
The response and the failure notification belong to the same interaction *charge* as the accepted request and they are correlated by the variable *v*

The bank service is *reliable*

$$AG\,[request(charge, \underline{v})]\,AF_{response(charge,v)}\ true$$

The service guarantees a successful response to each received request

Model checking the bank service



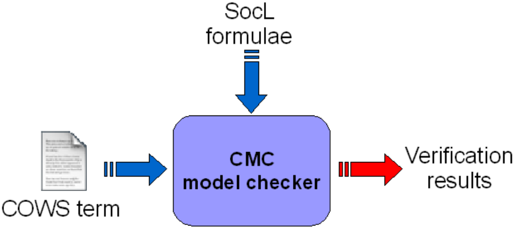
Abstraction rules

Action	charge<*,*,*,$id>	→	request(charge,$id)
Action	resp<$id,"ok">	→	response(charge,$id)
Action	resp<$id,"fail">	→	fail(charge,$id)
State	charge	→	accepting_request(charge)

Tool demonstration . . .

Model checking: a calculus-based approach

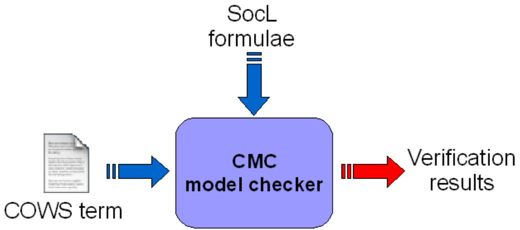
We have seen a calculus-based methodology for model checking COWS specifications



People in charge of verifying systems are required to understand and deal with calculi and logics.
This may not be the case, especially within , where people are usually familiar with higher level UML-based modelling languages.

Model checking: a calculus-based approach

We have seen a calculus-based methodology for model checking COWS specifications



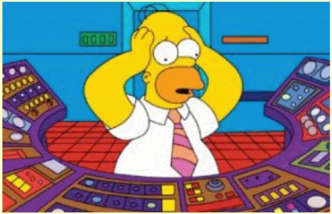
People in charge of verifying systems are required to understand and deal with calculi and logics.

This may not be the case, especially within industrial contexts, where people are usually familiar with higher-level UML-based modelling languages

Model checking: a calculus-based approach

We have seen a calculus-based methodology for model checking COWS specifications

Just an example...

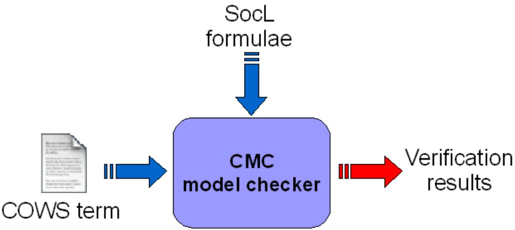


ification
esults

People in cha to understand and deal with calculi and logics.
This may not be the case, especially within industrial contexts, where people are usually familiar with higher-level UML-based modelling languages

Model checking: a calculus-based approach

We have seen a calculus-based methodology for model checking COWS specifications

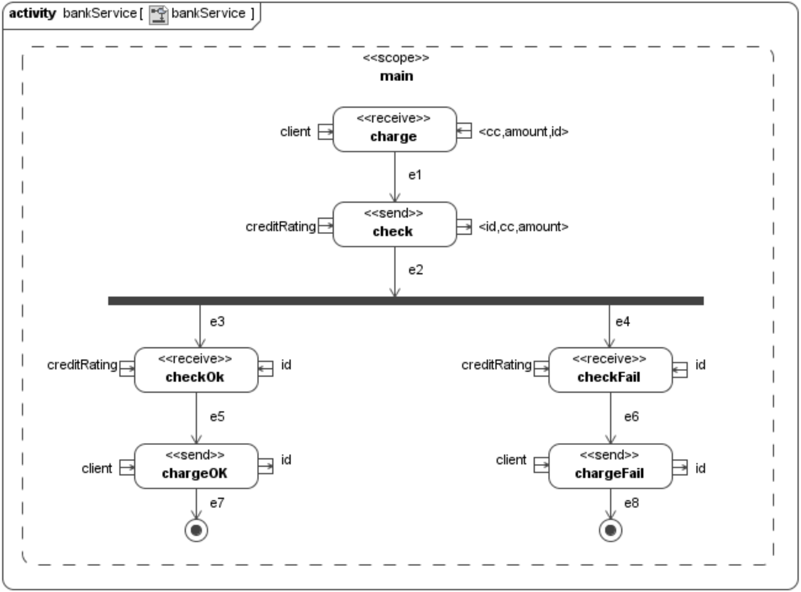


People in charge of verifying systems are required to understand and deal with calculi and logics.
This may not be the case, especially within industrial contexts, where people are usually familiar with higher-level UML-based modelling languages

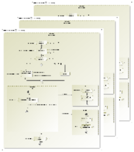
UML4SOA

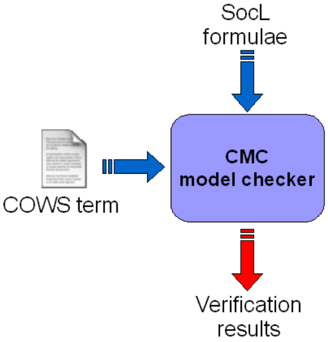
- The most widely used language for modelling sw systems is UML
- UML4SOA is a UML 2.0 profile, inspired by WS-BPEL,
  that has been expressly designed for modeling service-oriented applications
- UML4SOA activity diagrams express the behavioral aspects of services
  - integrate UML with specialized actions for exchanging messages, specialized structured activity nodes and activity edges for representing scopes with event, fault and compensation handlers
- Since UML4SOA specifications are static models, they are not suitable for direct automated analysis

UML4SOA: diagram example

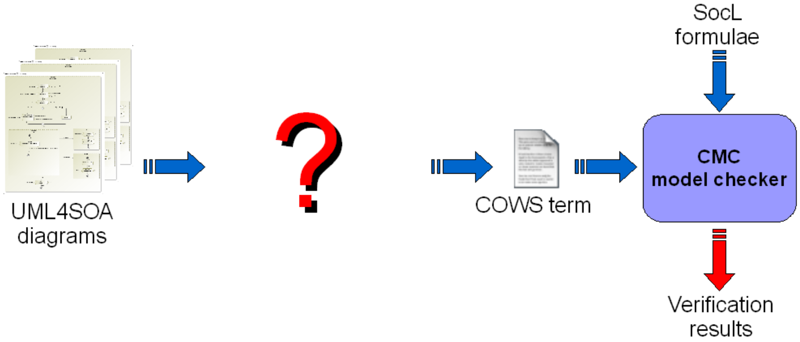


How to reconcile

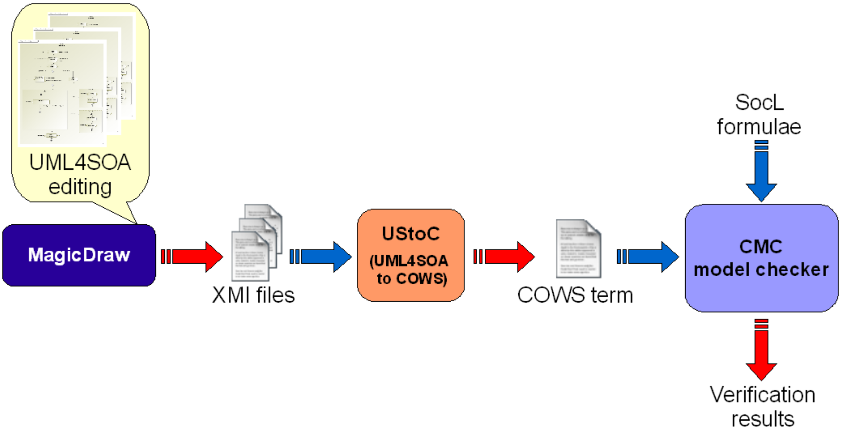




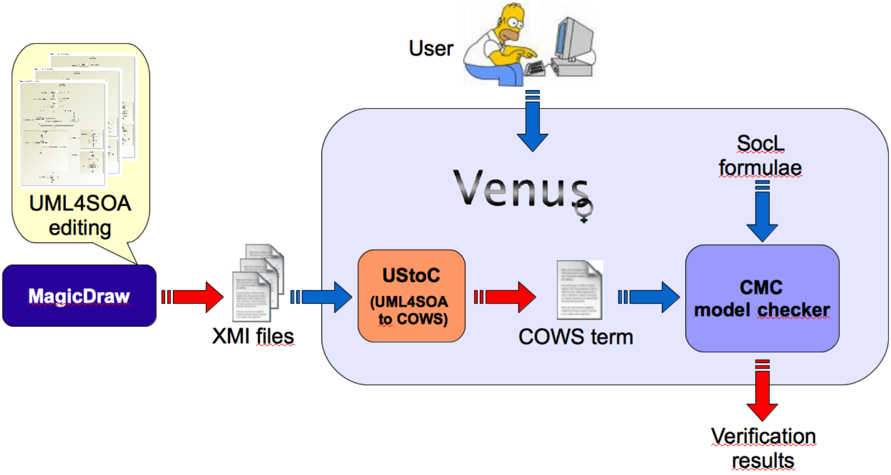
How to reconcile



Our proposal



Our proposal

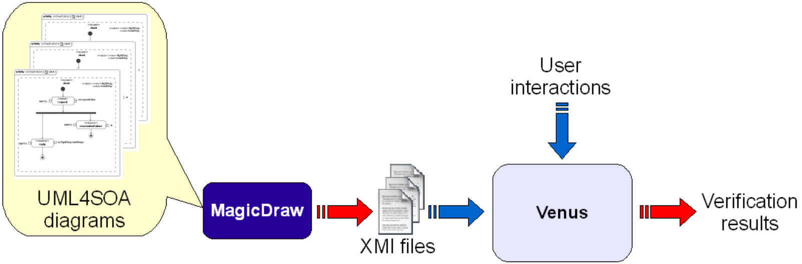


Our proposal

Venus: a Verification ENvironment for UML models of Services

A software environment for verifying behavioural properties of UML models of services by exploiting process calculi and temporal logics

- UML models of services: UMLSOA activity diagrams
- Venus shepherds the (non-expert) users to set the behavioural service properties they want to verify
- It is a proof-of-concept implementation

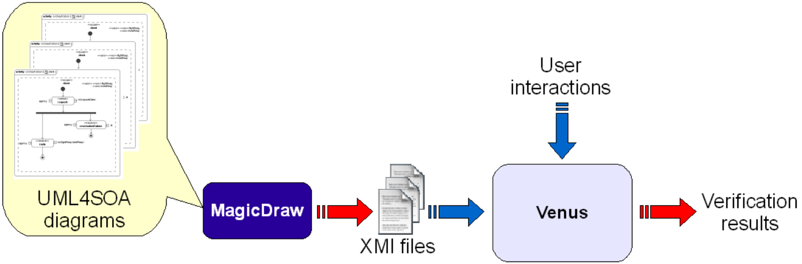


Our proposal

Venus: a Verification ENvironment for UML models of Services

A software environment for verifying behavioural properties of UML models of services by exploiting process calculi and temporal logics

- UML models of services: UMLSOA activity diagrams
- Venus shepherds the (non-expert) users to set the behavioural service properties they want to verify
- It is a proof-of-concept implementation

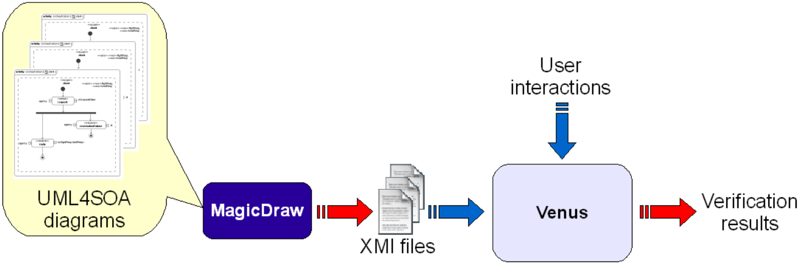


Our proposal

Venus: a Verification ENvironment for UML models of Services

A software environment for verifying behavioural properties of UML models of services by exploiting process calculi and temporal logics

- UML models of services: UMLSOA activity diagrams
- Venus shepherds the (non-expert) users to set the behavioural service properties they want to verify
- It is a proof-of-concept implementation

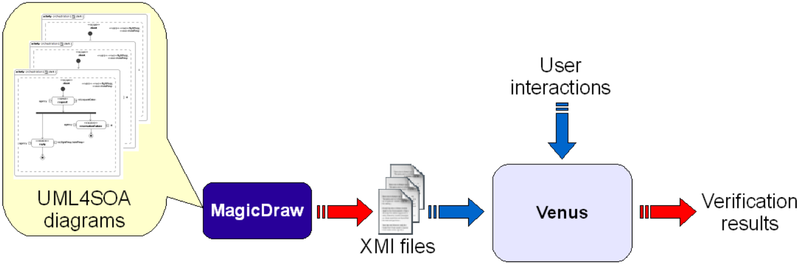


Our proposal

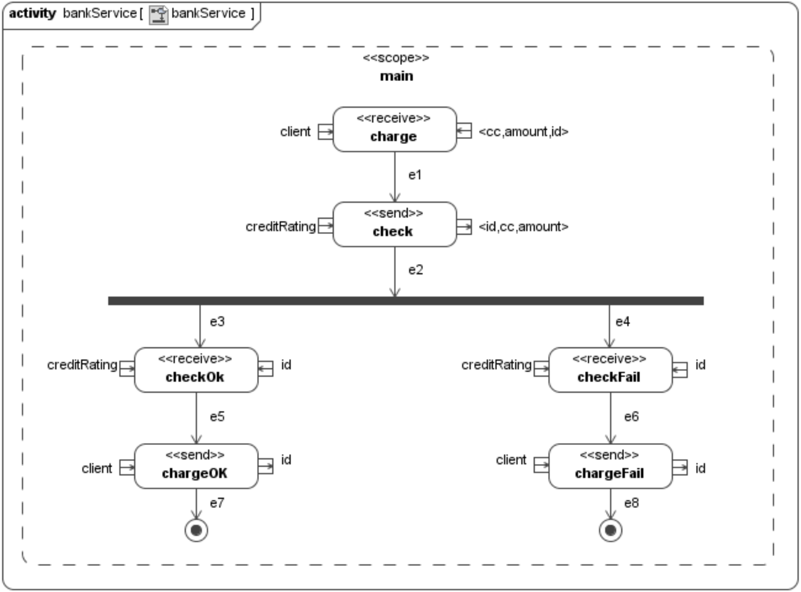
Venus: a **V**erification **EN**vironment for **U**ML models of **S**ervices

A software environment for verifying behavioural properties of UML models of services by exploiting process calculi and temporal logics

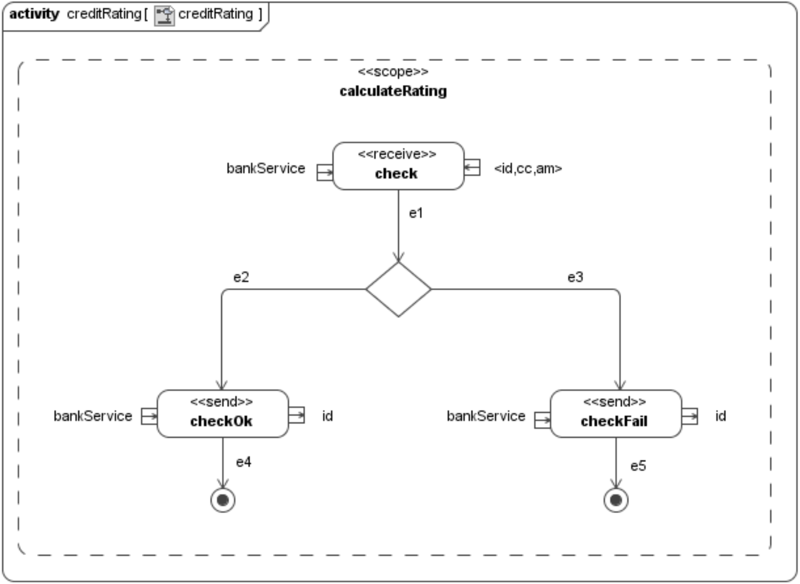
- UML models of services: UMLSOA activity diagrams
- Venus shepherds the (non-expert) users to set the behavioural service properties they want to verify
- It is a proof-of-concept implementation



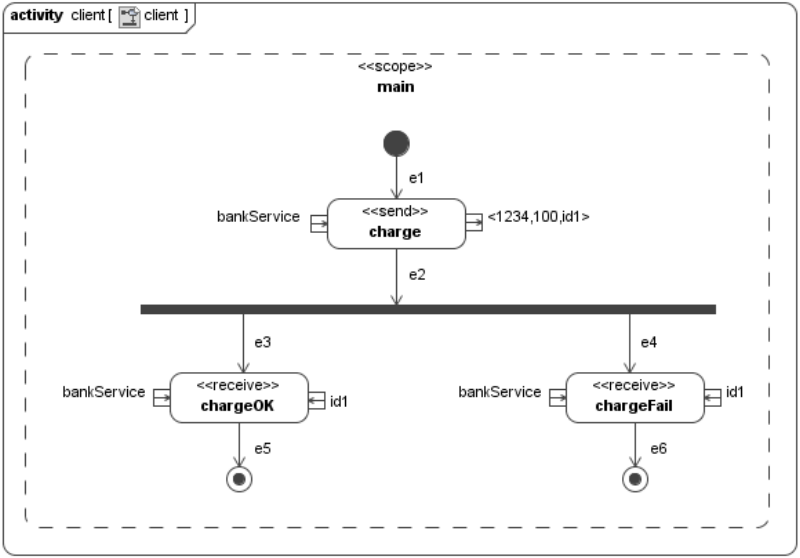
Bank scenario: bank service



Bank scenario: credit rating service

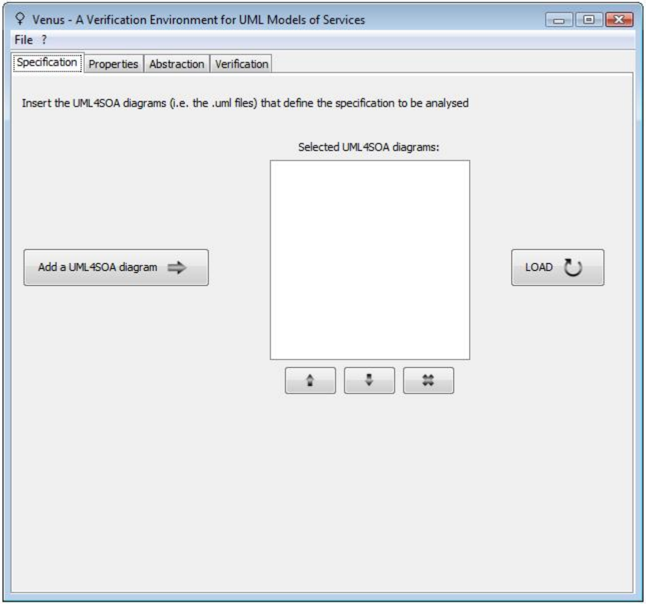


Bank scenario: client service

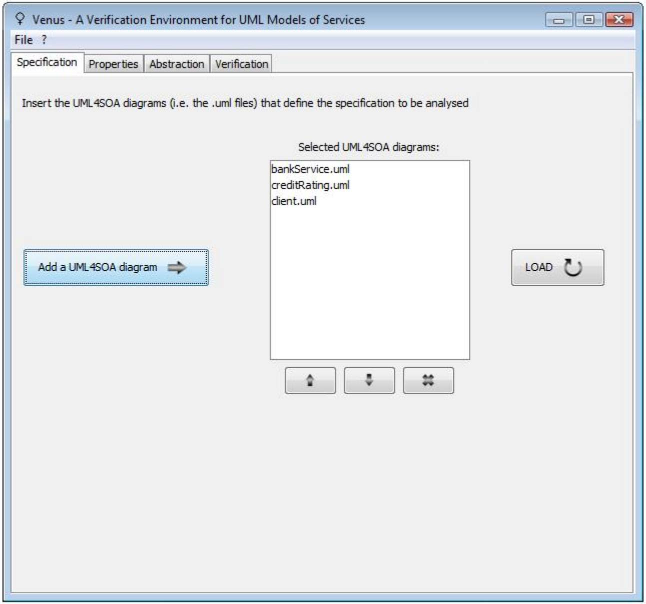


Venus demo 1/16

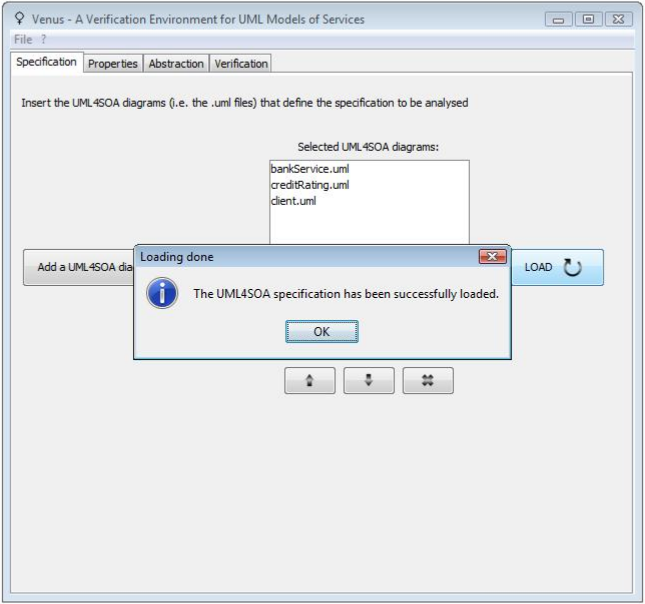
Venus demo 2/16



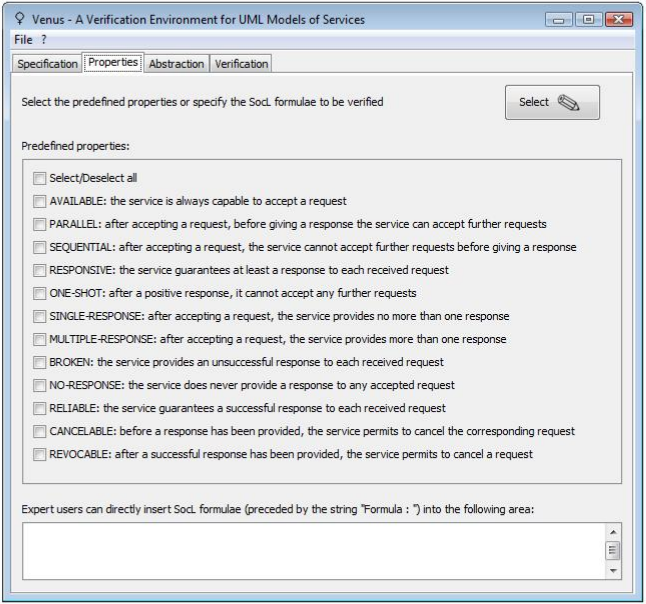
Venus demo 3/16



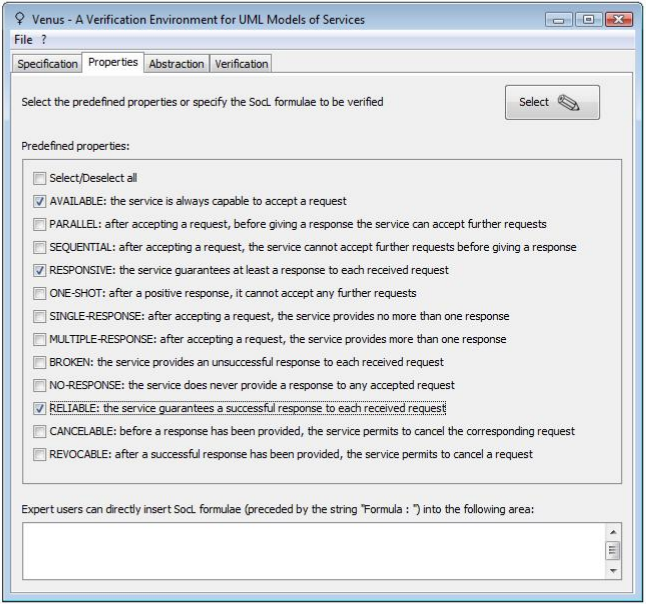
Venus demo 4/16



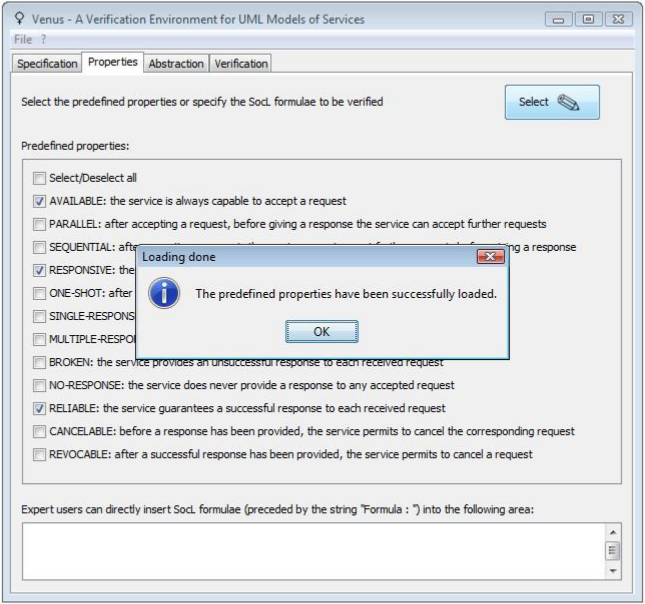
Venus demo 5/16



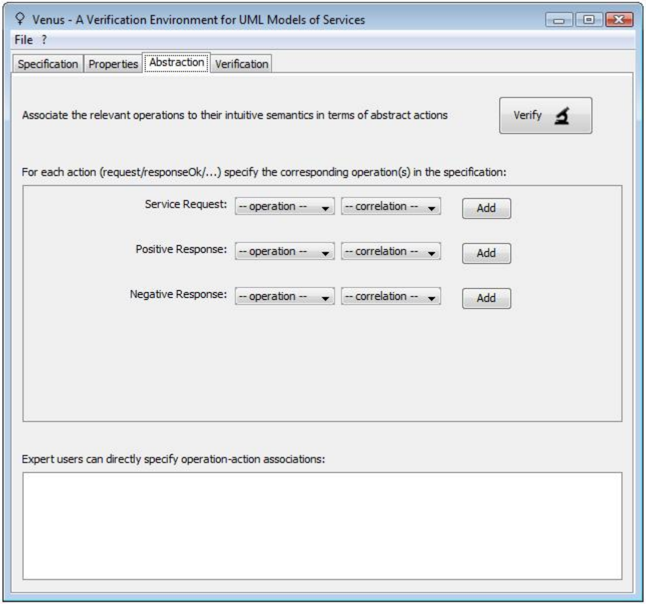
Venus demo 6/16



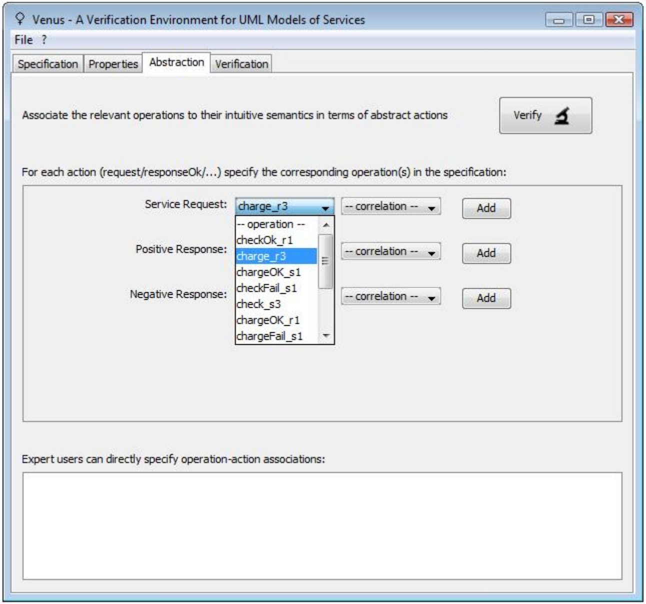
Venus demo 7/16



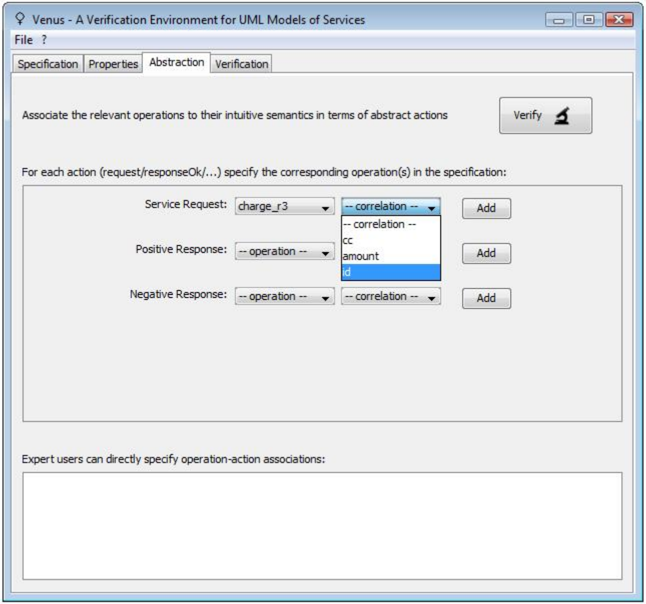
Venus demo 8/16



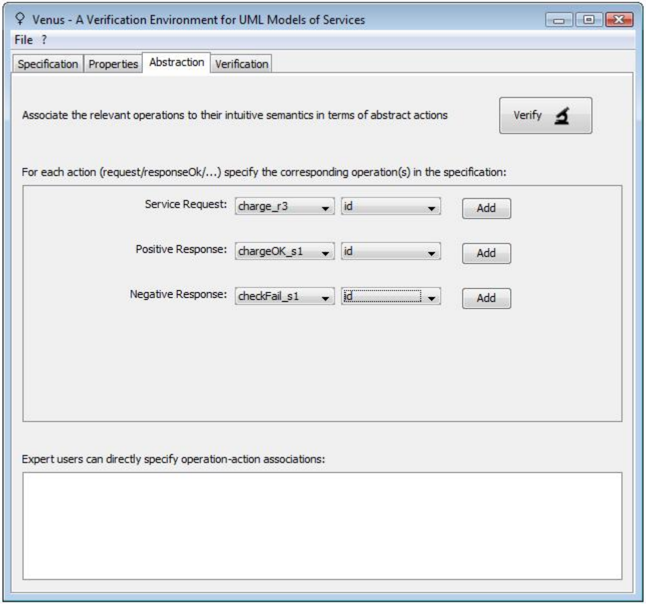
Venus demo 9/16



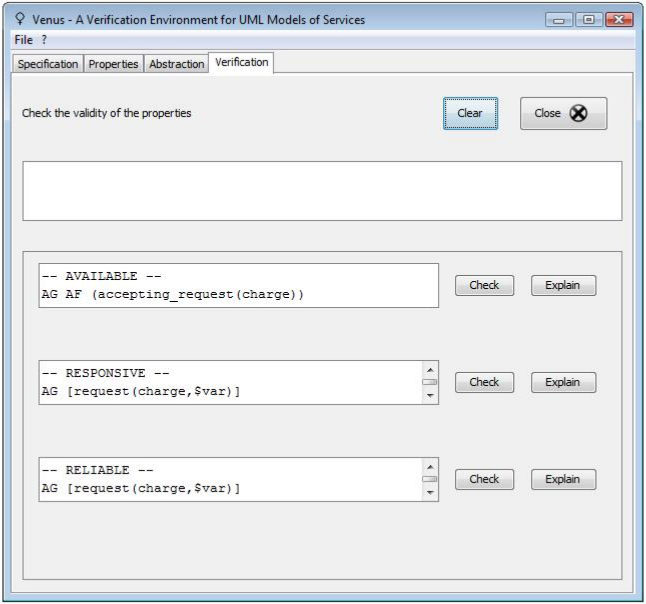
Venus demo 10/16



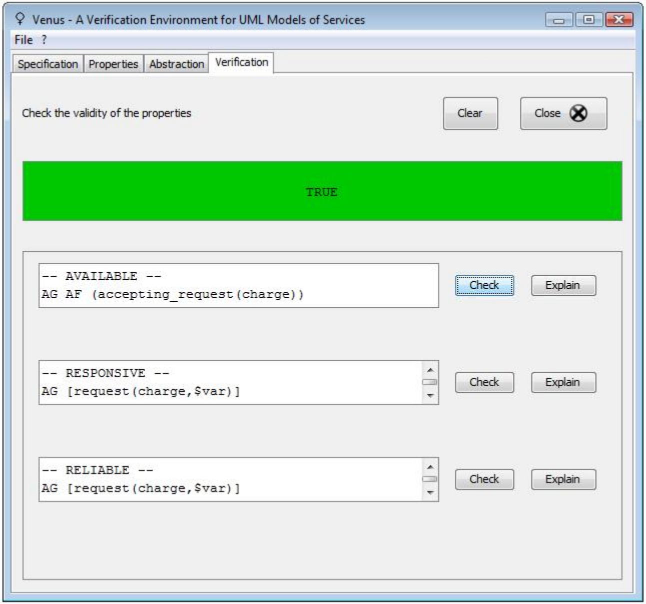
Venus demo 11/16



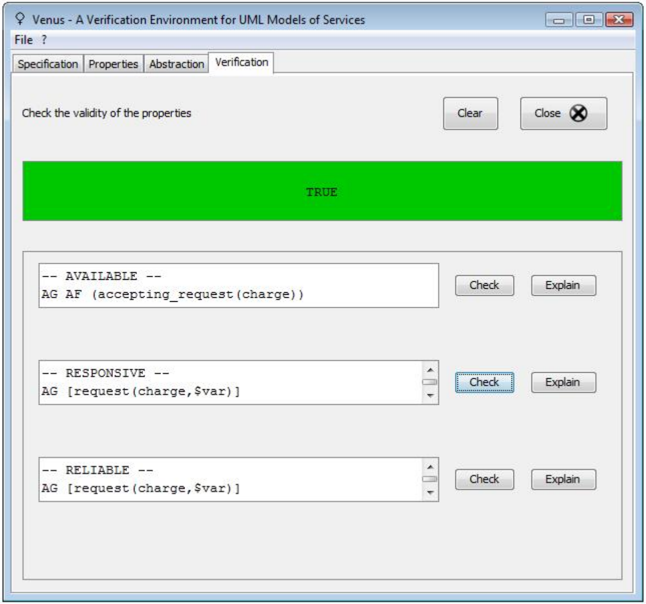
Venus demo 12/16



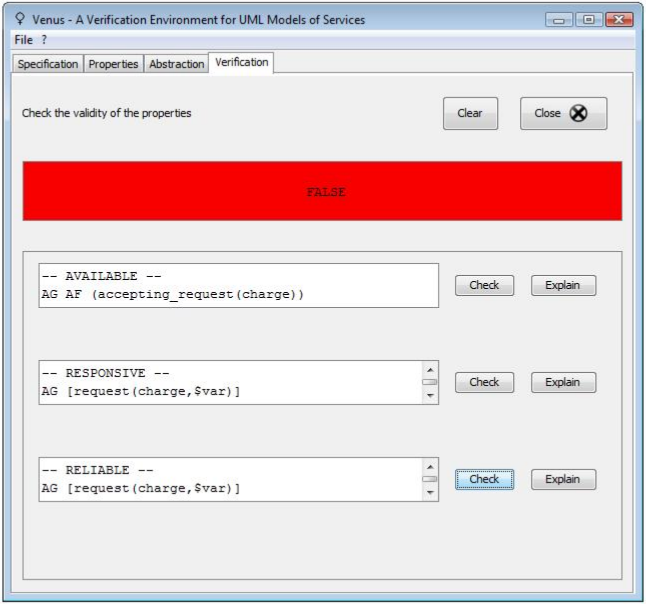
Venus demo 13/16



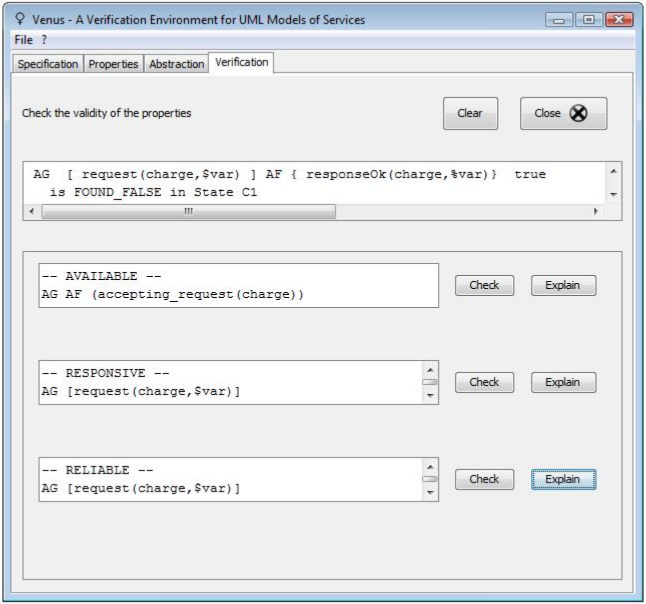
Venus demo 14/16



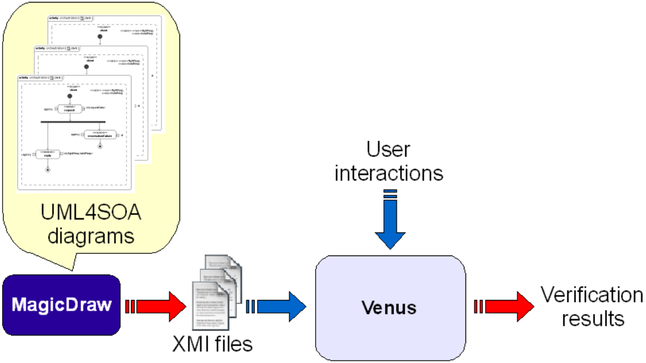
Venus demo 15/16



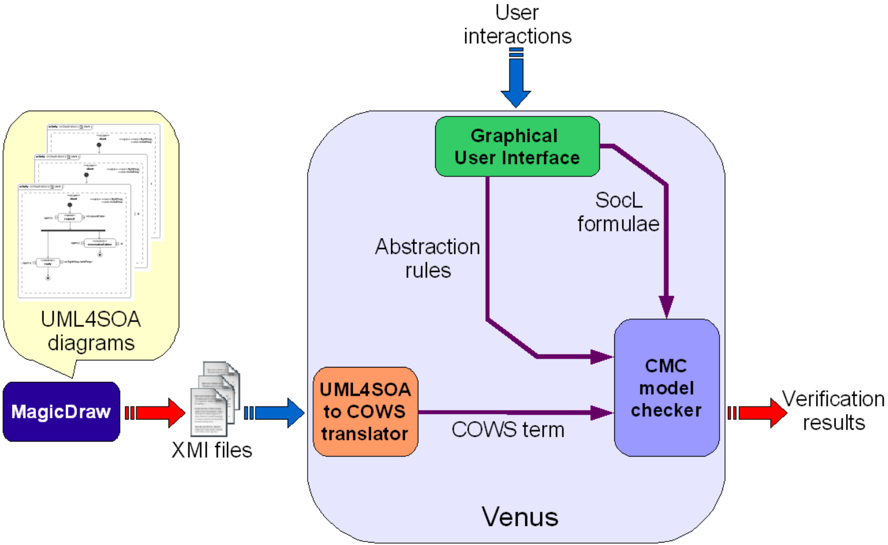
Venus demo 16/16



Venus architecture



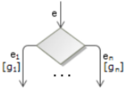
Venus architecture



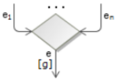
From UML4SOA to COWS

$$creditRequest \bullet initialize?\langle x_{portal}, x_{id}, x_{name}, x_{pwd}\rangle$$

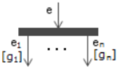
$$x_{portal} \bullet initialize!\langle creditRequest, x_{id}, x_{userOk}\rangle$$



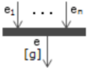
$$* \, e?\langle \mathbf{true}\rangle. \; [n_1, \ldots, n_n] \, (\, n_1!\langle g_1\rangle \mid \ldots \mid n_n!\langle g_n\rangle \mid n_1?\langle \mathbf{true}\rangle. \, e_1!\langle \mathbf{true}\rangle + \ldots + n_n?\langle \mathbf{true}\rangle. \, e_n!\langle \mathbf{true}\rangle \,)$$



$$* \, (\, e_1?\langle \mathbf{true}\rangle. \, e!\langle g\rangle + \ldots + e_n?\langle \mathbf{true}\rangle. \, e!\langle g\rangle \,)$$



$$* \, e?\langle \mathbf{true}\rangle. \, (\, e_1!\langle g_1\rangle \mid \ldots \mid e_n!\langle g_n\rangle \,)$$



$$* \, e_1?\langle \mathbf{true}\rangle. \, \ldots \, . \, e_n?\langle \mathbf{true}\rangle. \, e!\langle g\rangle$$

From UML4SOA to COWS

$$[\mathtt{r}, \mathit{stack}]\ \big([k]\ (\mathrm{GRAPH} ; \{\!|\, c \bullet \mathit{main}?\langle\rangle . \mathrm{GRAPH}_c \,|\!\} \mid \{\!|\mathit{Stack}|\!\} \mid * \mathrm{GRAPH}_{ev}) \mid \mathtt{r}?\langle\rangle . \{\!|\mathrm{GRAPH}_e|\!\} \big)$$

$$c \bullet \mathit{main}!\langle\rangle$$

$$\mathbf{kill}(k) \mid \{\!|\mathtt{r}!\langle\rangle|\!\}$$

$$\mathit{stack} \bullet \mathit{compAll}!\langle\rangle$$

Our COWS implementation of UML4SOA constructs follows a compositional approach

From UML4SOA to COWS

$$[\mathtt{r}, \mathit{stack}]\,(\,[k]\,(\mathrm{GRAPH}\,;\,\{\!|\,c \bullet \mathit{main}?\langle\rangle.\,\mathrm{GRAPH_c}\,|\!\} \mid \{\!|\mathit{Stack}|\!\} \mid *\,\mathrm{GRAPH_{ev}}\,) \mid \mathtt{r}?\langle\rangle.\,\{\!|\mathrm{GRAPH_e}|\!\}\,)$$

$c \bullet \mathit{main}!\langle\rangle$

$\mathbf{kill}(k) \mid \{\!|\mathtt{r}!\langle\rangle|\!\}$

$\mathit{stack} \bullet \mathit{compAll}!\langle\rangle$

Our COWS implementation of UML4SOA constructs follows a compositional approach

Concluding remarks

Conclusions

- COWS permits modelling different and typical aspects of services and Web services technologies
  - multiple start activities, receive conflicts, routing of correlated messages, service instances and interactions among them

- COWS can express the most common workflow patterns and can encode many other process and orchestration languages

- COWS, with some mild linguistic additions, can model all the relevant phases of the life cycle of service-oriented applications
  - publication, discovery, negotiation, deployment, orchestration, reconfiguration and execution

Conclusions

- The observational semantics permits to check interchangeability of services and conformance against service specifications

- The type system permits specifying and forcing policies for constraining the services that can safely access any given datum
  - Types are just sets and operations on types are union, intersection, subset inclusion, . . .
  - The runtime semantics only involves efficiently implementable operations on sets

- The logical verification framework for checking functional properties of SOC applications has many advantages
  - It can be easily tailored to other service-oriented specification languages
  - SocL's parametric formulae permit expressing properties about many kinds of interaction patterns, e.g. *one-way*, *request-response*, *one request-multiple responses*, . . .

COWS

http://rap.dsi.unifi.it/cows/

References

References 1/4

- A WSDL-based type system for WS-BPEL
  A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of COORDINATION'06, LNCS 4038, 2006.

- A calculus for orchestration of web services
  A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of ESOP'07, LNCS 4421, 2007.
  go back

- Regulating data exchange in service oriented applications
  A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of FSEN'07, LNCS 4767, 2007.
  go back

- COWS: A timed service-oriented calculus
  A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of ICTAC'07, LNCS 4711, 2007. go back

- Stochastic COWS
  D. Prandi, P. Quaglia. Proc. of ICSOC'07, LNCS 4749, 2007.

References 2/4

- A model checking approach for verifying COWS specifications
  A. Fantechi, S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese, F. Tiezzi.
  Proc. of FASE'08, LNCS 4961, 2008. ▸ go back
- Service discovery and negotiation with COWS
  A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of WWV'07, ENTCS 200(3), 2008. ▸ go back
- Specifying and Analysing SOC Applications with COWS
  A. Lapadula, R. Pugliese, F. Tiezzi. In Concurrency, Graphs and Models, LNCS 5065, 2008.
- SENSORIA Patterns: Augmenting Service Engineering with Formal Analysis, Transformation and Dynamicity
  M. Wirsing, et al. Proc. of ISOLA'08, Communications in Computer and Information Science 17, 2008.
- A formal account of WS-BPEL
  A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of COORDINATION'08, LNCS 5052, 2008.

References 3/4

Formal analysis of BPMN via a translation into COWS
D. Prandi, P. Quaglia, N. Zannone. Proc. of COORDINATION'08, LNCS 5052, 2008.

Relational Analysis of Correlation
J. Bauer, F. Nielson, H.R. Nielson, H. Pilegaard. Proc. of SAS'08, LNCS 5079, 2008.

A Symbolic Semantics for a Calculus for Service-Oriented Computing
R. Pugliese, F. Tiezzi, N. Yoshida. Proc. of PLACES'08, ENTCS 241, 2009.

Specification and analysis of SOC systems using COWS: A finance case study
F. Banti, A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of WWV'08, ENTCS 235(C), 2009.

From Architectural to Behavioural Specification of Services
L. Bocchi, J.L. Fiadeiro, A. Lapadula, R. Pugliese, F. Tiezzi. Proc. of FESCA'09, ENTCS 253/1, 2009.

References 4/4

- On observing dynamic prioritised actions in SOC
  R. Pugliese, F. Tiezzi, N. Yoshida. Proc. of ICALP'09, LNCS 5556, 2009.
  go back

- On secure implementation of an IHE XUA-based protocol for authenticating healthcare professionals
  M. Masi, R. Pugliese, F. Tiezzi. Proc. of ICISS'09, LNCS 5905, 2009.

- Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing
  M. Wirsing and M. Hölzl Editors. LNCS, 2010. To appear.

- An Accessible Verification Environment for UML Models of Services
  F. Banti, R. Pugliese, F. Tiezzi. Journal of Symbolic Computation, 2010. To appear.

- A criterion for separating process calculi
  F. Banti, R. Pugliese, F. Tiezzi. Proc. of EXPRESS'10, 2010. go back