Domain Specific Formal Languages

– Analysing Service-Oriented Systems with COWS –

Francesco Tiezzi

School of Science and Technology

Computer Science Division

University of Camerino

A.A. 2016/2017

Analysis techniques for COWS specifications

- A bisimulation-based observational semantics [ICALP'09]
- A type system for checking confidentiality properties [FSEN'07]
- A logical verification methodology [FASE'08,TOSEM'12]

Analysis techniques for COWS specifications

- A bisimulation-based observational semantics [ICALP'09]

- A type system for checking confidentiality properties [FSEN'07]

- A logical verification methodology [FASE'08,TOSEM'12]

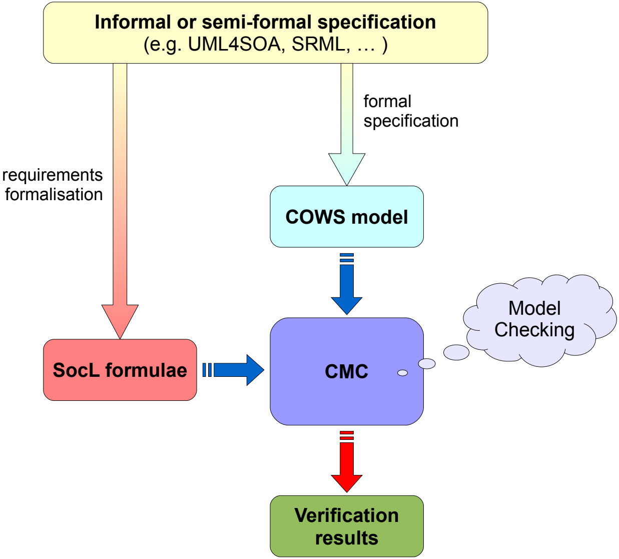
Logics and Model checking

- 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



Informal or semi-formal specification
(e.g. UML4SOA, SRML, …)

requirements formalisation

formal specification

COWS model

SocL formulae

CMC

Model Checking

Verification results

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^{2}TS) 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

⇓

Doubly Labelled Transition Systems (L^{2}TS) 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 $type/name(interaction, corrTuple)$

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

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

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

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

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

Action formulae syntax

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

$\underline{a}$ 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

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

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

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

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

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

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 $EX_{\gamma}\,\phi$

$E(\phi\,{}_{\chi}U\,\phi')$ stands for $\phi' \vee E(\phi_{\chi} U_{\chi\vee\tau}\,\phi')$

$AF_{\gamma}\,\textit{true}$ stands for $A(\textit{true}\,{}_{tt}U_{\gamma}\,\textit{true})$

$[\gamma]\,\phi$ stands for $\neg < \gamma > \neg\,\phi$

$EF\phi$ stands for $E(\textit{true}\,{}_{tt}U\phi)$

$AG\,\phi$ 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
- $\mathit{EF}\phi$ means that there is some path that leads to a state at which ϕ holds; that is, ϕ *eventually* holds on some path
- $\mathit{AF}_\gamma\,\phi$ 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
- $\mathit{AG}\,\phi$ states that ϕ holds at every state on every path; that is, ϕ holds *globally*

Some derived modalities

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

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

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

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

- $q \models true$ 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 $\sigma(2) \models \phi\, \rho$
- ...

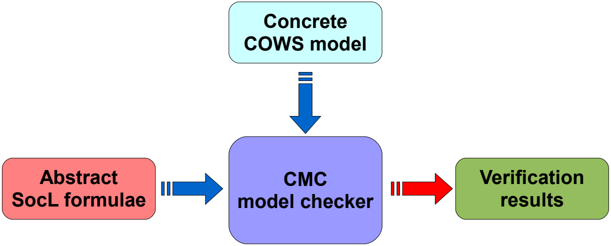
SocL semantics

- 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

- $\ldots$
- $\sigma \models \phi \,_{\chi}U_{\gamma}\phi'$ iff $\exists\, j \geq 1$
  $\sigma(j) \models \phi$, and $\exists\, \rho : \sigma\{j\} \models \gamma \triangleright \rho$ 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 $\sigma\{i\} \models \chi$

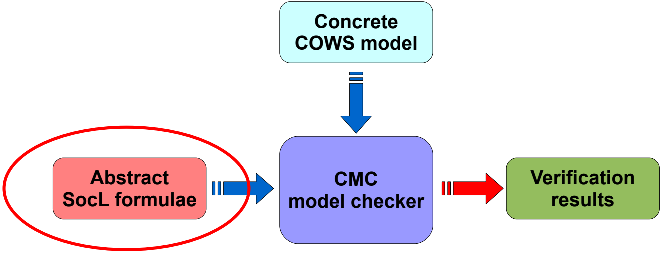
A novel verification methodology of service properties

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



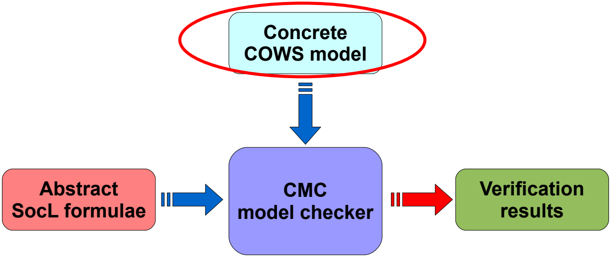
A novel verification methodology of service properties

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

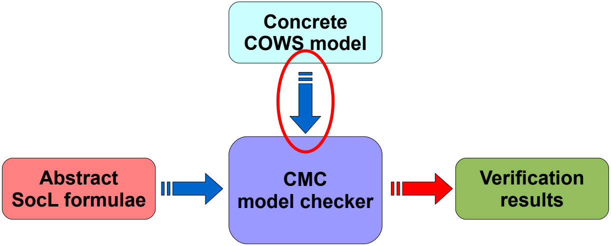
We resort to a linguistic formalism rather than directly using L^{2}TSs because

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

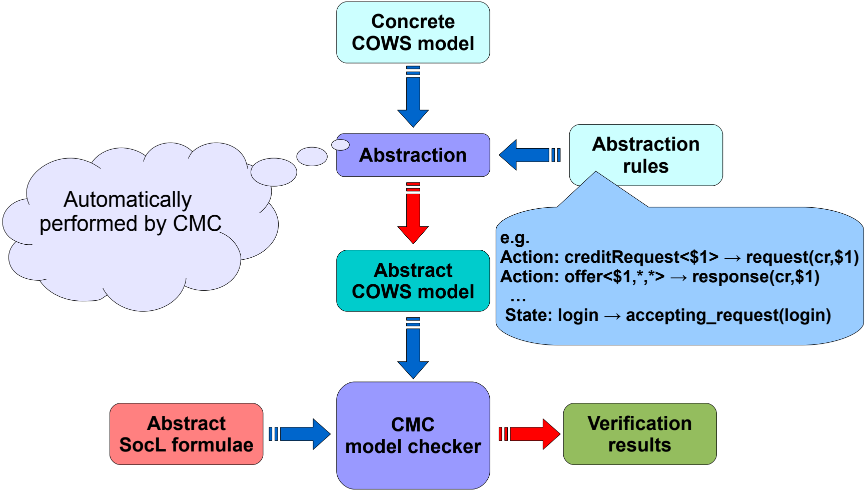
  i.e. they offer no means for constructing the L^{2}TS of a composed service in terms of the L^{2}TSs of its components
- linguistic terms are more intuitive and concise notations
- using linguistic terms, services are built in a compositional way
- linguistic terms are syntactically finite, even when the corresponding semantic model (i.e. L^{2}TSs) 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



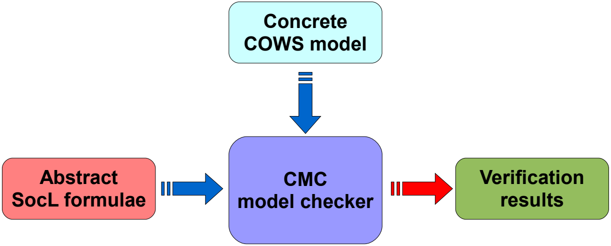
A novel verification methodology of service properties



e.g.
Action: creditRequest<$1> → request(cr,$1)
Action: offer<$1,*,*> → response(cr,$1)
…
State: login → accepting_request(login)

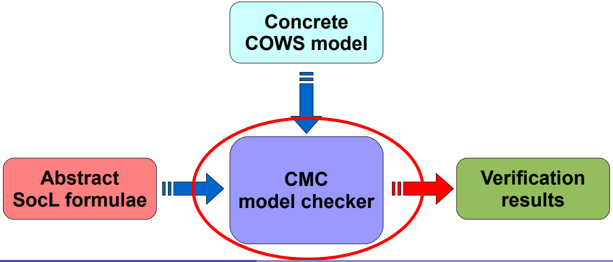
A novel verification methodology of service properties

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

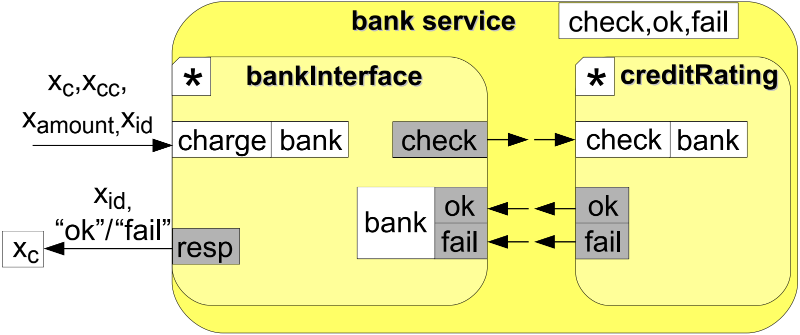
To assist the verification process of SocL formulae over L^{2}TS

- CMC is an efficient on-the-fly model checker
- The basic idea behind CMC is that, given a state of an L^{2}TS, 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*

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

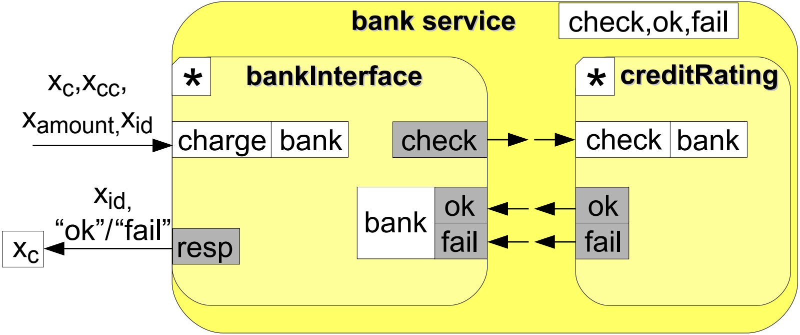
The response and the failure notification belong to the same interaction *charge* as the accepted request and they are correlated by the variable *v*

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



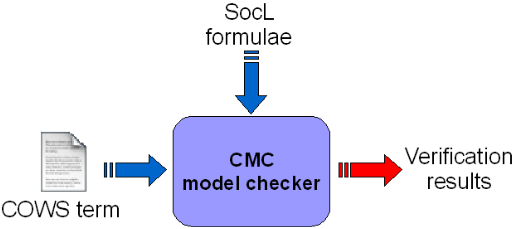
Abstraction rules

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

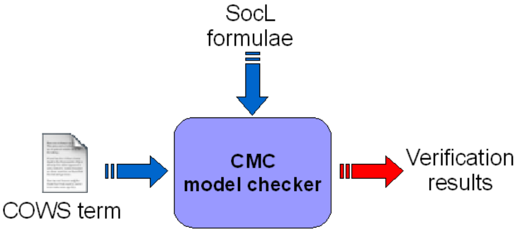


People in charge of verifying systems are required to understand and deal with calculi and logics.

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



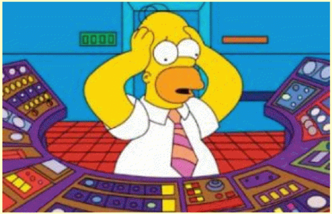
People in charge of verifying systems are required to understand and deal with calculi and logics.

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...

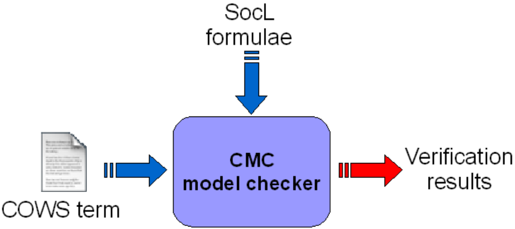


ification
esults

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

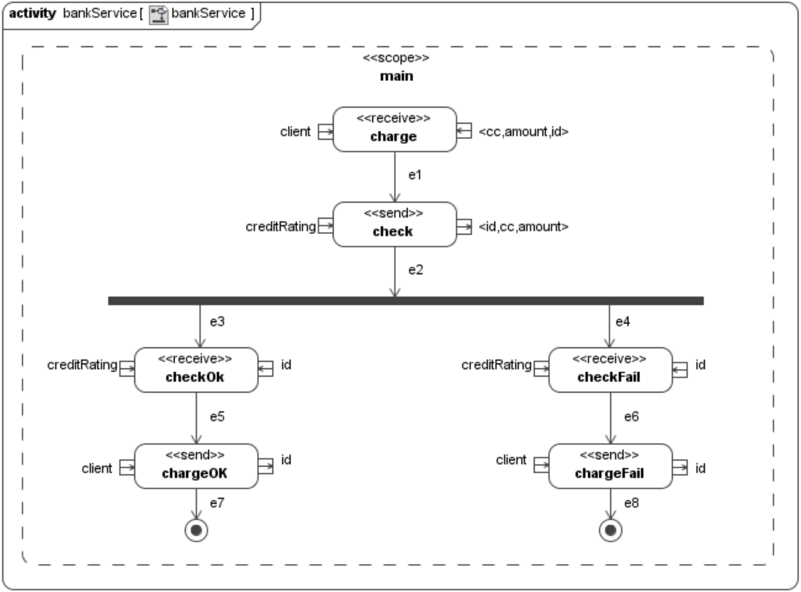
- The most widely used language for modelling sw systems is UML

- UML4SOA is a UML 2.0 profile, inspired by WS-BPEL, that has been expressly designed for modeling service-oriented applications

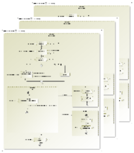
- UML4SOA activity diagrams express the behavioral aspects of services
  - integrate UML with specialized actions for exchanging messages, specialized structured activity nodes and activity edges for representing scopes with event, fault and compensation handlers

- Since UML4SOA specifications are static models, they are not suitable for direct automated analysis

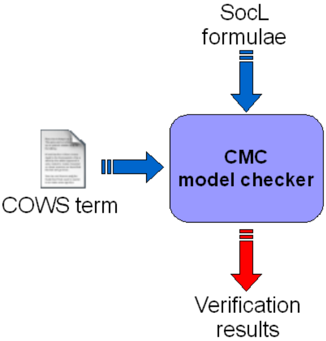
UML4SOA: diagram example



How to reconcile



UML4SOA diagrams



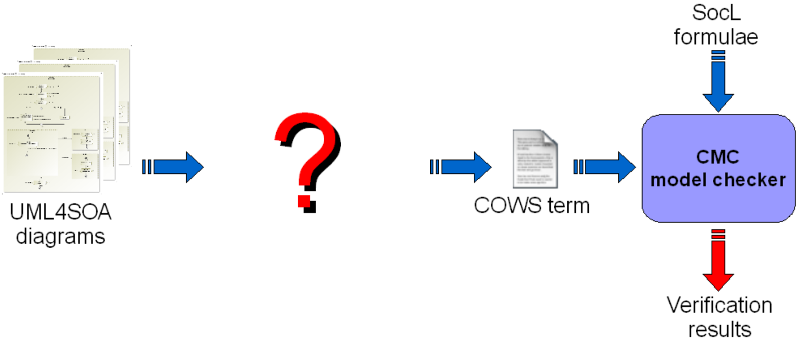
SocL formulae

COWS term

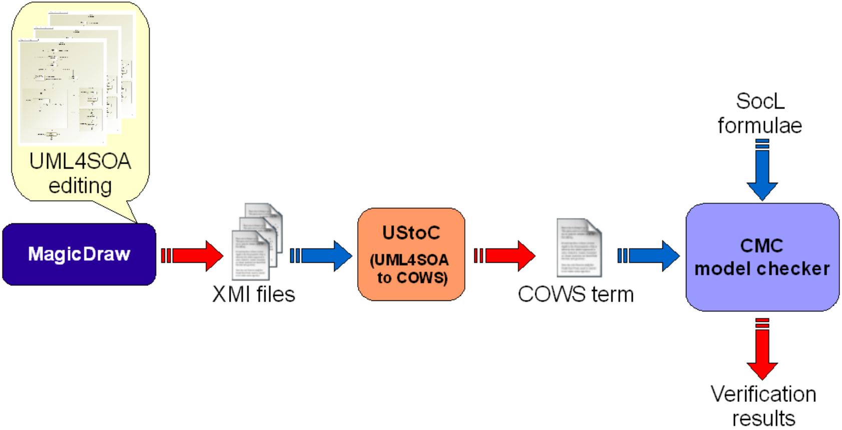
CMC model checker

Verification results

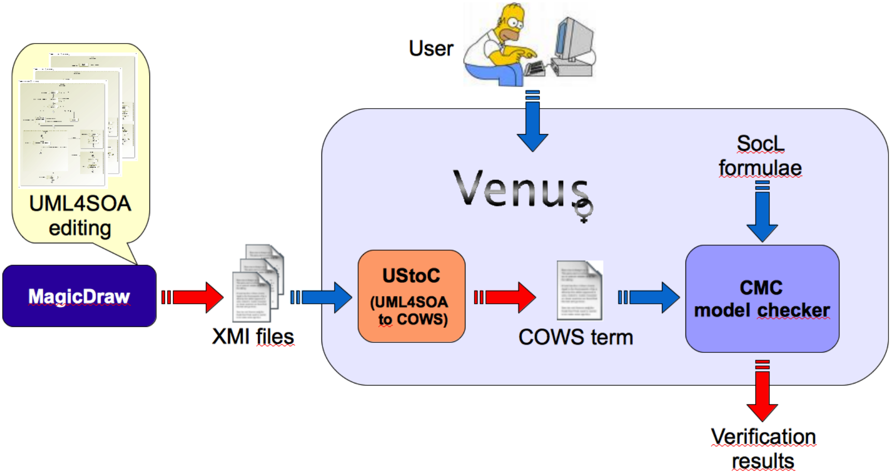
How to reconcile



Our proposal



Our proposal

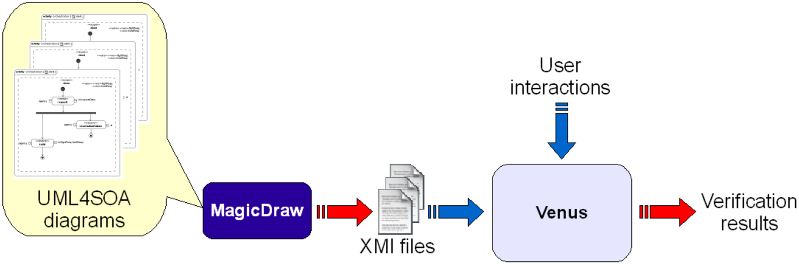


Our proposal

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

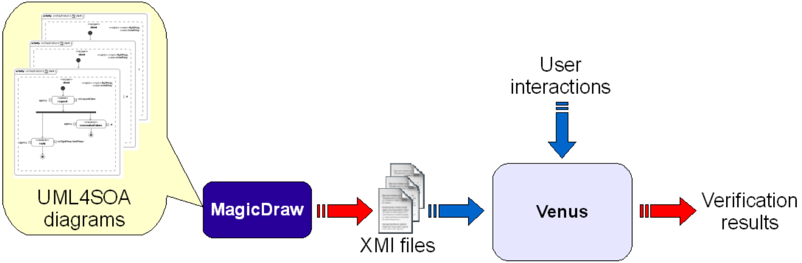


Our proposal

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

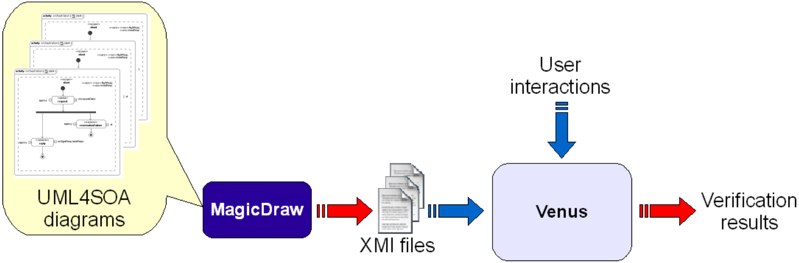


Our proposal

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

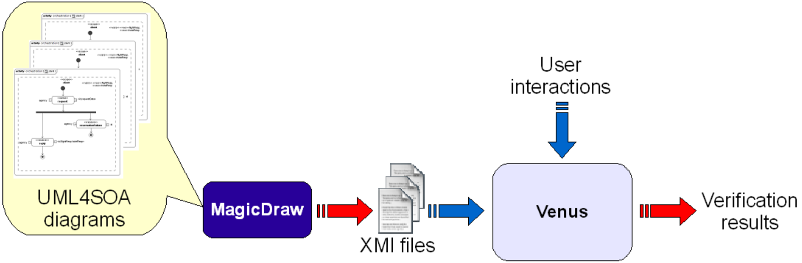


Our proposal

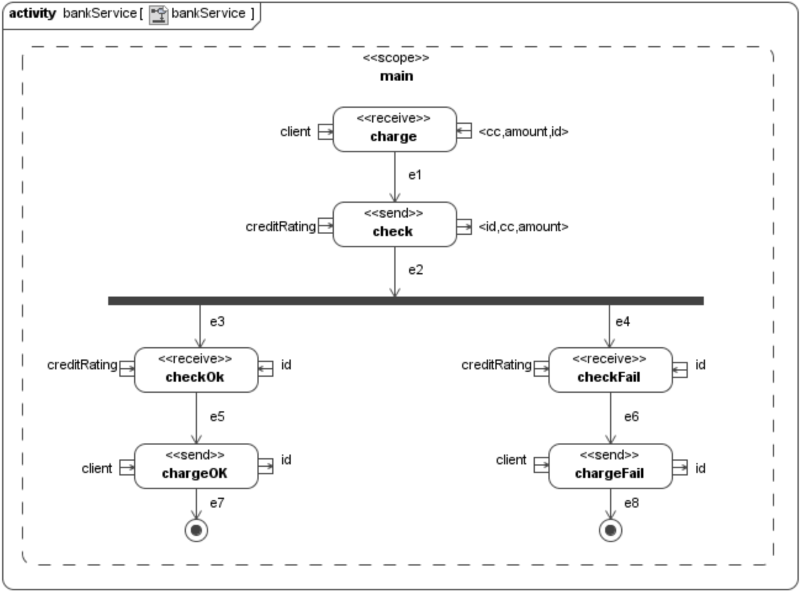
Venus: a **V**erification **EN**vironment for **U**ML models of **S**ervices

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

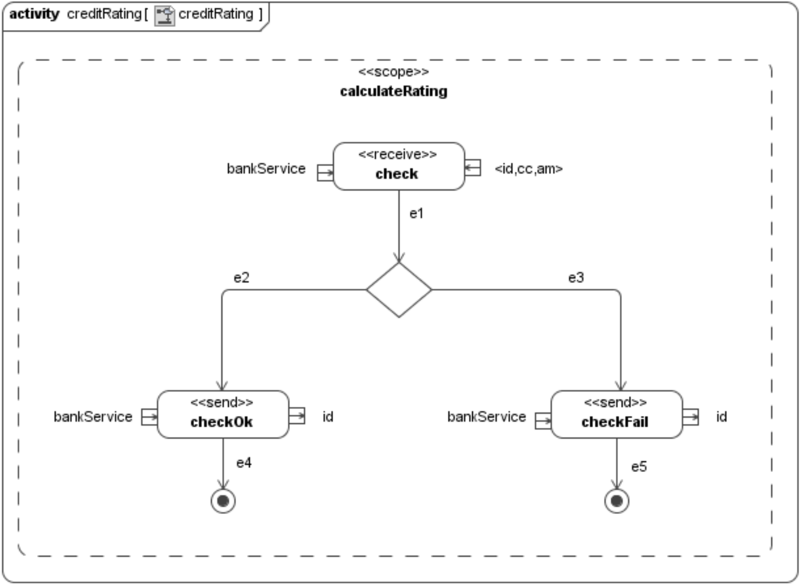
- UML models of services: UMLSOA activity diagrams
- Venus shepherds the (non-expert) users to set the behavioural service properties they want to verify
- It is a proof-of-concept implementation



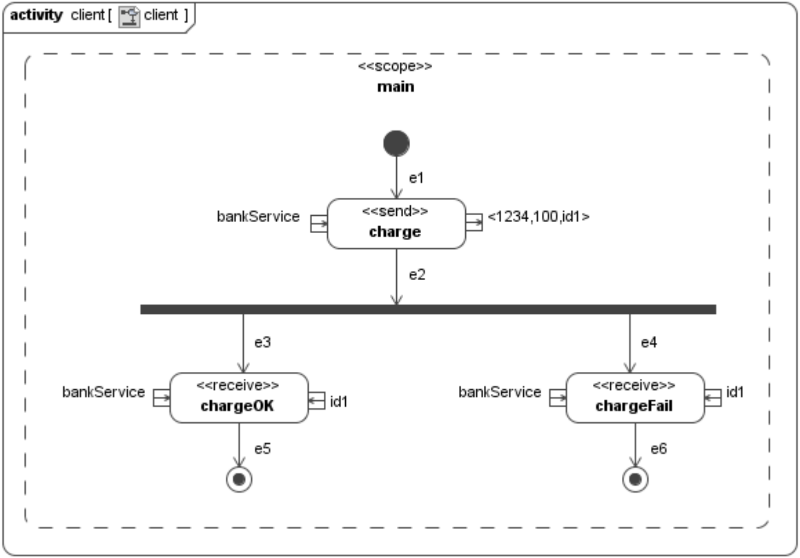
Bank scenario: bank service



Bank scenario: credit rating service

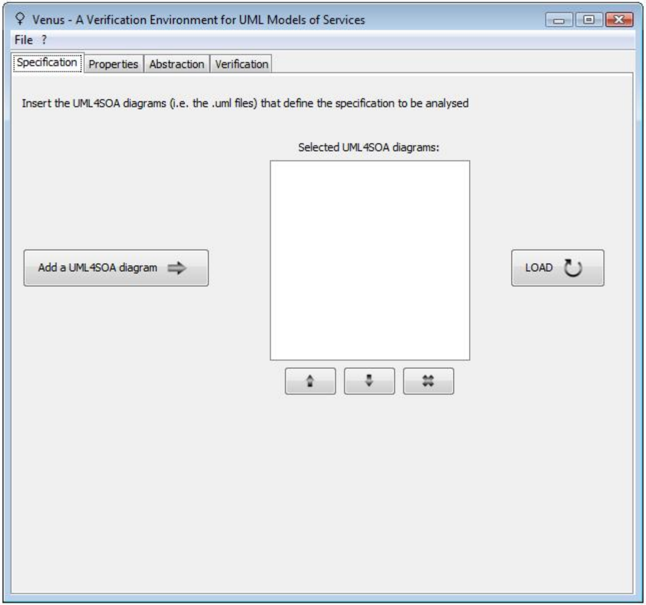


Bank scenario: client service



Venus demo 1/16

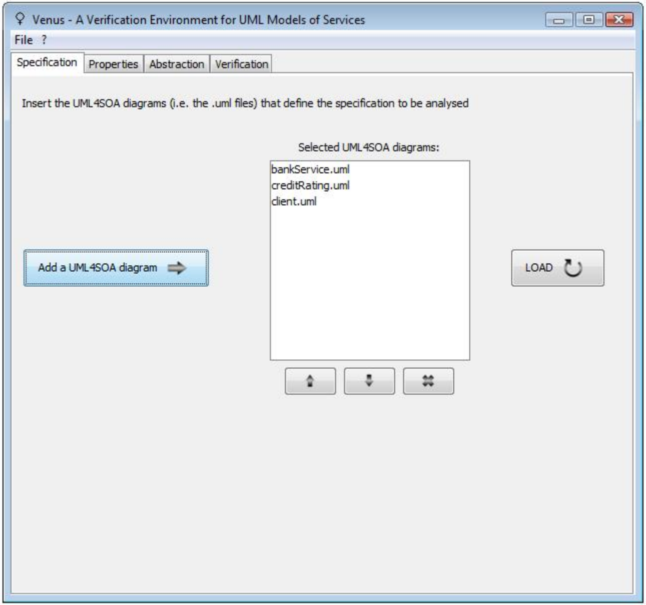
Venus demo 2/16


Venus - A Verification Environment for UML Models of Services
File
?
Specification
Properties
Abstraction
Verification

Insert the UML4SOA diagrams (i.e. the .uml files) that define the specification to be analysed

Selected UML4SOA diagrams:
Add a UML4SOA diagram
LOAD

Venus demo 3/16



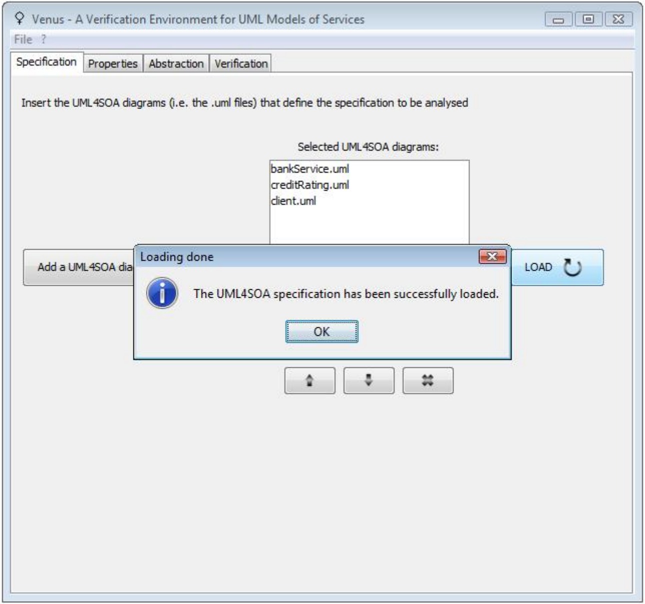
Venus - A Verification Environment for UML Models of Services
File ?
Specification
Properties
Abstraction
Verification

Insert the UML4SOA diagrams (i.e. the .uml files) that define the specification to be analysed

Selected UML4SOA diagrams:

bankService.uml
creditRating.uml
client.uml

Venus demo 4/16



Venus - A Verification Environment for UML Models of Services
File ?
Specification
Properties
Abstraction
Verification

Insert the UML4SOA diagrams (i.e. the .uml files) that define the specification to be analysed

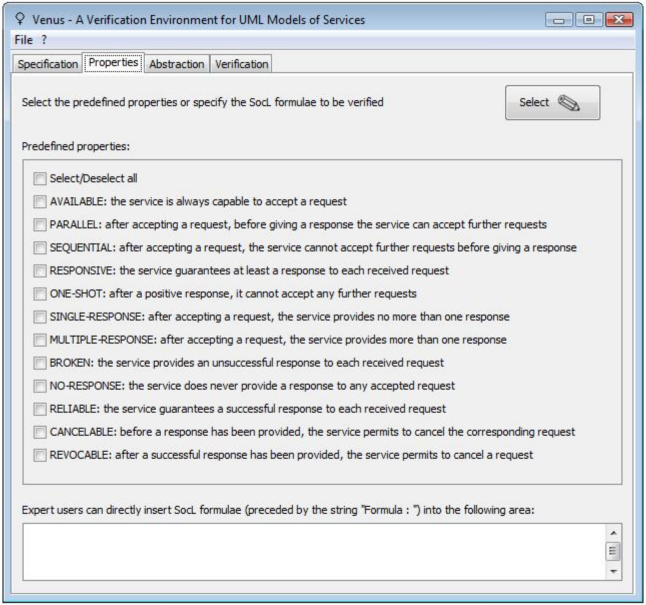
Selected UML4SOA diagrams:

bankService.uml
creditRating.uml
client.uml

Loading done

The UML4SOA specification has been successfully loaded.

Venus demo 5/16


Venus - A Verification Environment for UML Models of Services
File ?
Specification
Properties
Abstraction
Verification
Select

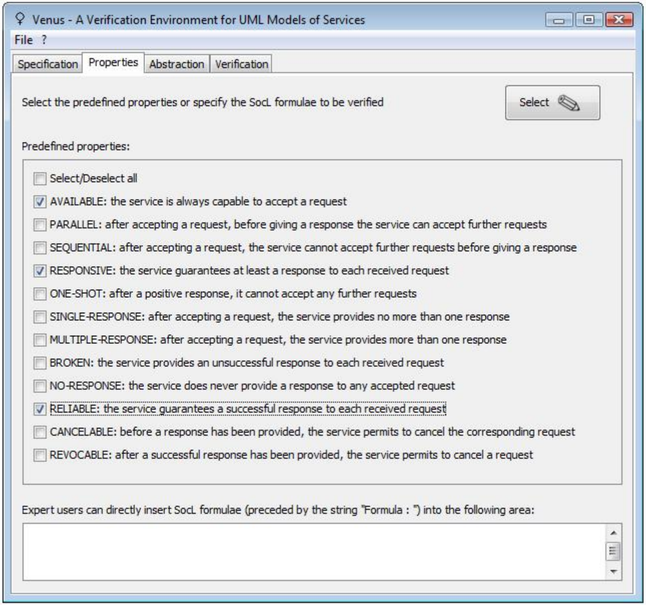
Select the predefined properties or specify the SocL formulae to be verified

Predefined properties:

- □ Select/Deselect all
- □ AVAILABLE: the service is always capable to accept a request
- □ PARALLEL: after accepting a request, before giving a response the service can accept further requests
- □ SEQUENTIAL: after accepting a request, the service cannot accept further requests before giving a response
- □ RESPONSIVE: the service guarantees at least a response to each received request
- □ ONE-SHOT: after a positive response, it cannot accept any further requests
- □ SINGLE-RESPONSE: after accepting a request, the service provides no more than one response
- □ MULTIPLE-RESPONSE: after accepting a request, the service provides more than one response
- □ BROKEN: the service provides an unsuccessful response to each received request
- □ NO-RESPONSE: the service does never provide a response to any accepted request
- □ RELIABLE: the service guarantees a successful response to each received request
- □ CANCELABLE: before a response has been provided, the service permits to cancel the corresponding request
- □ REVOCABLE: after a successful response has been provided, the service permits to cancel a request

Expert users can directly insert SocL formulae (preceded by the string "Formula : ") into the following area:

Venus demo 6/16



Venus - A Verification Environment for UML Models of Services
File ?
Specification
Properties
Abstraction
Verification
Select

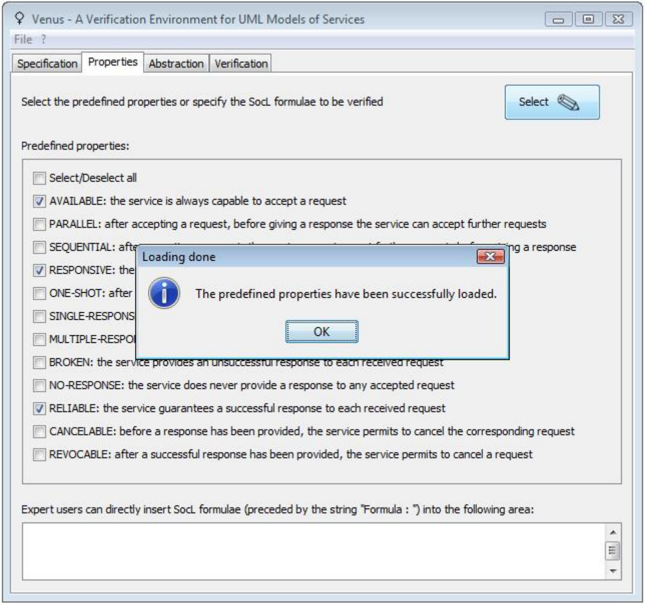
Select the predefined properties or specify the SocL formulae to be verified

Predefined properties:

- □ Select/Deselect all
- ☑ AVAILABLE: the service is always capable to accept a request
- □ PARALLEL: after accepting a request, before giving a response the service can accept further requests
- □ SEQUENTIAL: after accepting a request, the service cannot accept further requests before giving a response
- ☑ RESPONSIVE: the service guarantees at least a response to each received request
- □ ONE-SHOT: after a positive response, it cannot accept any further requests
- □ SINGLE-RESPONSE: after accepting a request, the service provides no more than one response
- □ MULTIPLE-RESPONSE: after accepting a request, the service provides more than one response
- □ BROKEN: the service provides an unsuccessful response to each received request
- □ NO-RESPONSE: the service does never provide a response to any accepted request
- ☑ RELIABLE: the service guarantees a successful response to each received request
- □ CANCELABLE: before a response has been provided, the service permits to cancel the corresponding request
- □ REVOCABLE: after a successful response has been provided, the service permits to cancel a request

Expert users can directly insert SocL formulae (preceded by the string "Formula : ") into the following area:

Venus demo 7/16



Venus - A Verification Environment for UML Models of Services
File ?
Specification
Properties
Abstraction
Verification
Select

Select the predefined properties or specify the SocL formulae to be verified

Predefined properties:

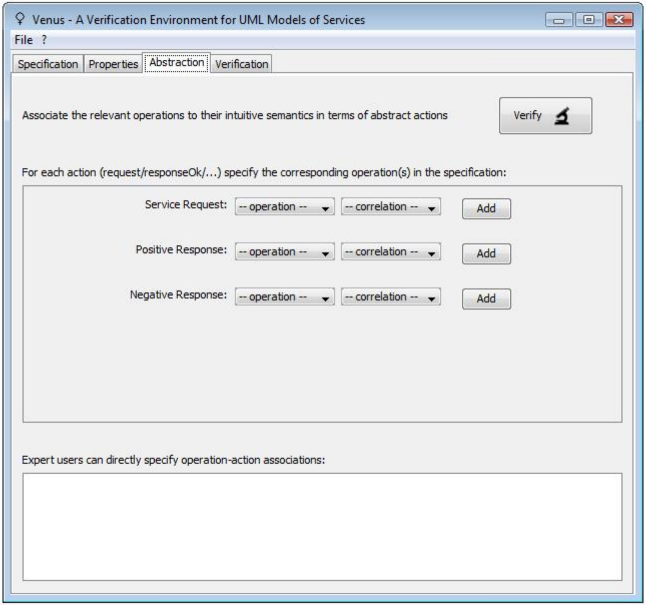
- □ Select/Deselect all
- ☑ AVAILABLE: the service is always capable to accept a request
- □ PARALLEL: after accepting a request, before giving a response the service can accept further requests
- □ SEQUENTIAL: afte ng a response
- ☑ RESPONSIVE: the
- □ ONE-SHOT: after
- □ SINGLE-RESPONS
- □ MULTIPLE-RESPO
- □ BROKEN: the service provides an unsuccessful response to each received request
- □ NO-RESPONSE: the service does never provide a response to any accepted request
- ☑ RELIABLE: the service guarantees a successful response to each received request
- □ CANCELABLE: before a response has been provided, the service permits to cancel the corresponding request
- □ REVOCABLE: after a successful response has been provided, the service permits to cancel a request

Loading done

The predefined properties have been successfully loaded.

Expert users can directly insert SocL formulae (preceded by the string "Formula : ") into the following area:

Venus demo 8/16



Venus - A Verification Environment for UML Models of Services
File ?
Specification
Properties
Abstraction
Verification
Verify

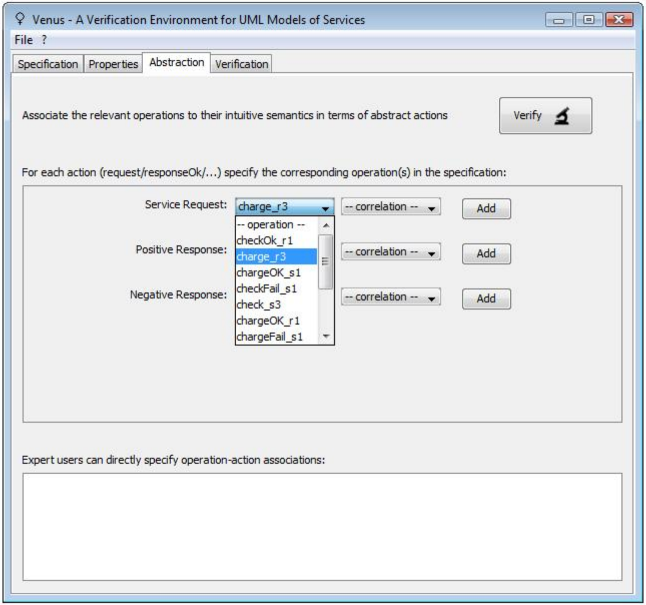
Associate the relevant operations to their intuitive semantics in terms of abstract actions

For each action (request/responseOk/...) specify the corresponding operation(s) in the specification:

Service Request:	-- operation --	-- correlation --	Add
Positive Response:	-- operation --	-- correlation --	Add
Negative Response:	-- operation --	-- correlation --	Add

Expert users can directly specify operation-action associations:

Venus demo 9/16



Venus - A Verification Environment for UML Models of Services

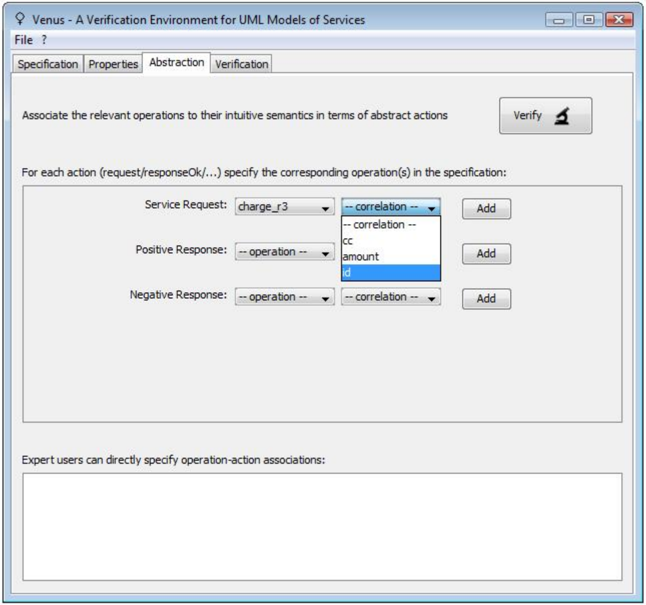
Associate the relevant operations to their intuitive semantics in terms of abstract actions

For each action (request/responseOk/...) specify the corresponding operation(s) in the specification:

Service Request: charge_r3

Expert users can directly specify operation-action associations:

Venus demo 10/16



Associate the relevant operations to their intuitive semantics in terms of abstract actions

For each action (request/responseOk/...) specify the corresponding operation(s) in the specification:

Service Request: charge_r3 -- correlation --

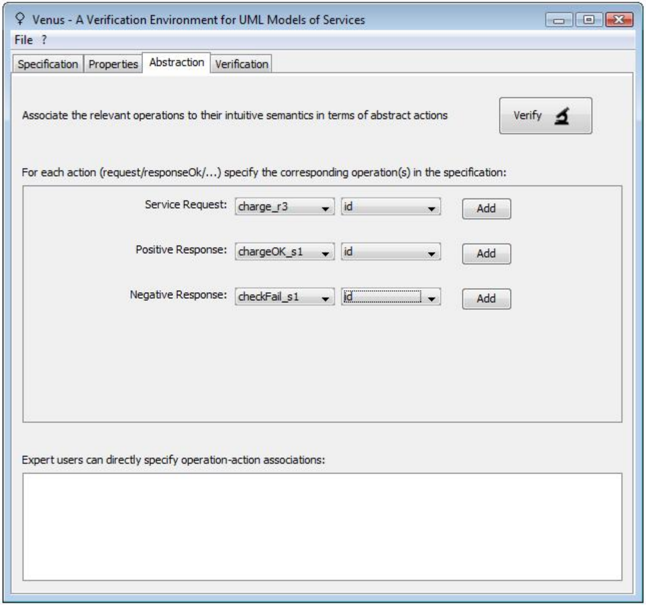
-- correlation --
cc
amount
id

Positive Response: -- operation --

Negative Response: -- operation -- -- correlation --

Expert users can directly specify operation-action associations:

Venus demo 11/16



Associate the relevant operations to their intuitive semantics in terms of abstract actions

For each action (request/responseOk/...) specify the corresponding operation(s) in the specification:

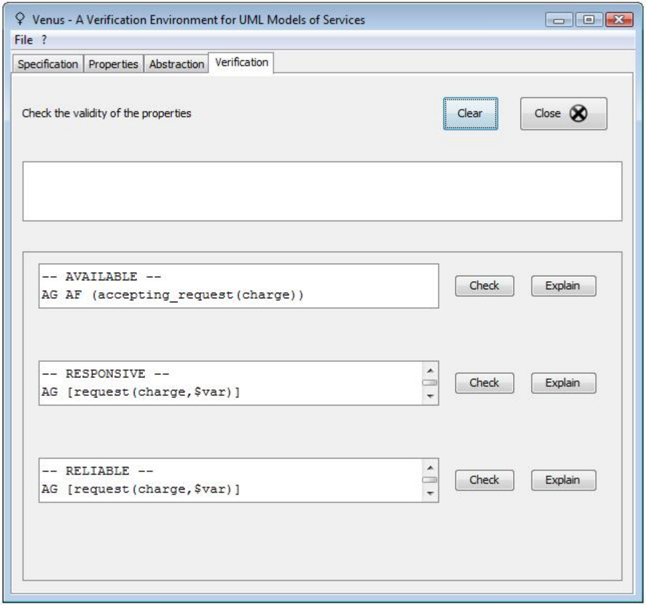
Service Request: charge_r3 id

Positive Response: chargeOK_s1 id

Negative Response: checkFail_s1 id

Expert users can directly specify operation-action associations:

Venus demo 12/16



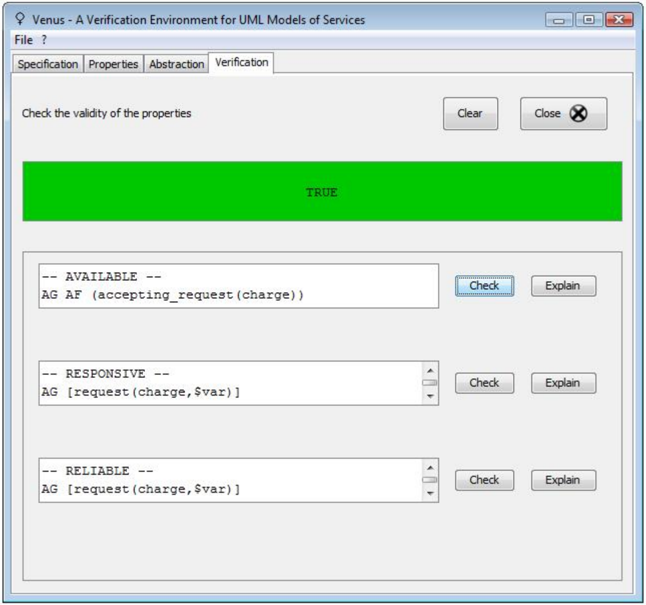
Check the validity of the properties

```
-- AVAILABLE --
AG AF (accepting_request(charge))
```

```
-- RESPONSIVE --
AG [request(charge,$var)]
```

```
-- RELIABLE --
AG [request(charge,$var)]
```

Venus demo 13/16



Check the validity of the properties

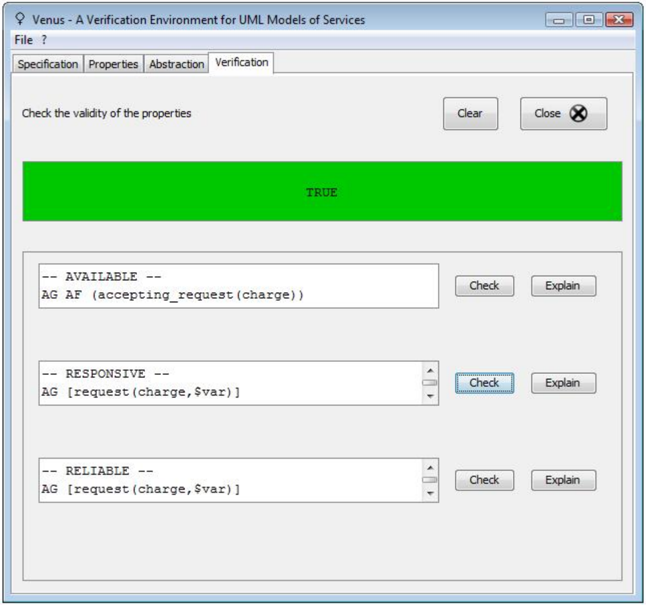
TRUE

```
-- AVAILABLE --
AG AF (accepting_request(charge))
```

```
-- RESPONSIVE --
AG [request(charge,$var)]
```

```
-- RELIABLE --
AG [request(charge,$var)]
```

Venus demo 14/16



Check the validity of the properties

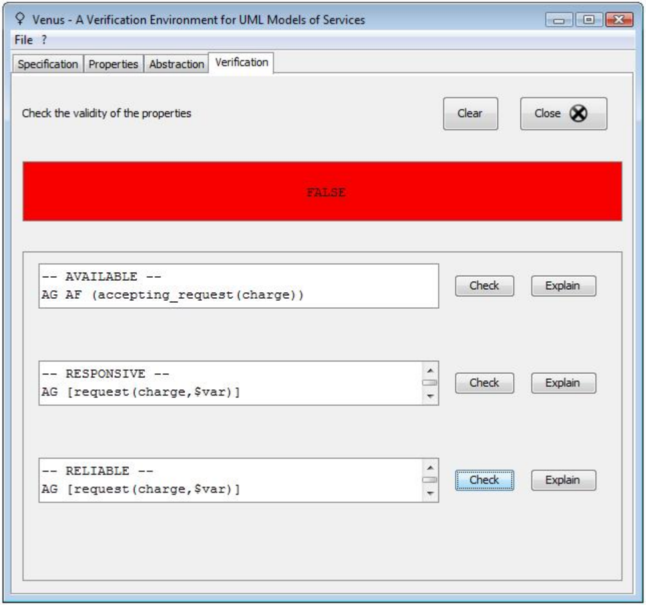
TRUE

```
-- AVAILABLE --
AG AF (accepting_request(charge))
```

```
-- RESPONSIVE --
AG [request(charge,$var)]
```

```
-- RELIABLE --
AG [request(charge,$var)]
```

Venus demo 15/16



Check the validity of the properties

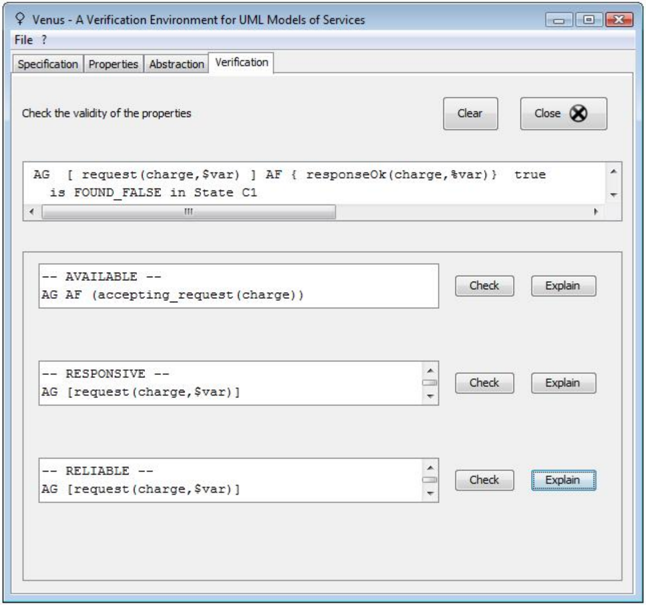
FALSE

```
-- AVAILABLE --
AG AF (accepting_request(charge))
```

```
-- RESPONSIVE --
AG [request(charge,$var)]
```

```
-- RELIABLE --
AG [request(charge,$var)]
```

Venus demo 16/16



Check the validity of the properties

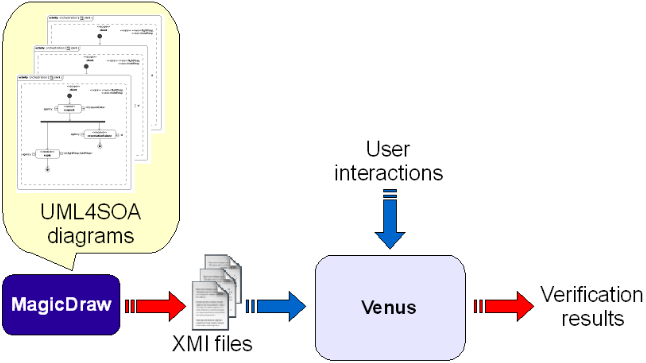
```
AG  [ request(charge,$var) ] AF { responseOk(charge,%var)}  true
  is FOUND_FALSE in State C1
```

```
-- AVAILABLE --
AG AF (accepting_request(charge))
```

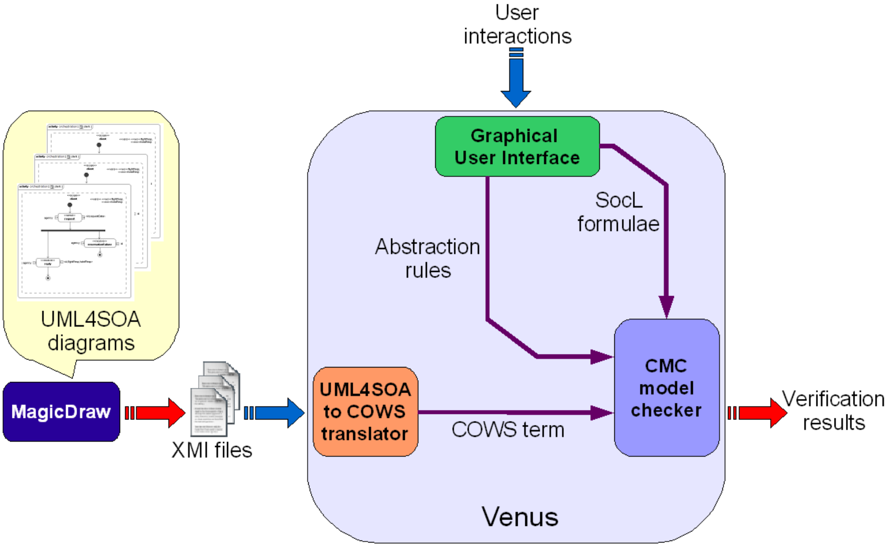
```
-- RESPONSIVE --
AG [request(charge,$var)]
```

```
-- RELIABLE --
AG [request(charge,$var)]
```

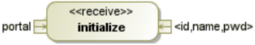
Venus architecture



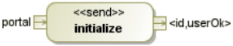
Venus architecture



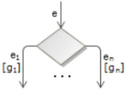
From UML4SOA to COWS



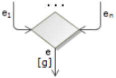
$creditRequest \bullet initialize?\langle x_{portal}, x_{id}, x_{name}, x_{pwd} \rangle$



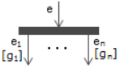
$x_{portal} \bullet initialize!\langle creditRequest, x_{id}, x_{userOk} \rangle$



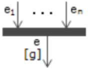
$$* \, e?\langle \mathbf{true} \rangle. \\ [n_1, \ldots, n_n] \, (\, n_1!\langle g_1 \rangle \mid \ldots \mid n_n!\langle g_n \rangle \\ \mid n_1?\langle \mathbf{true} \rangle. \, e_1!\langle \mathbf{true} \rangle \; + \; \ldots \; + \; n_n?\langle \mathbf{true} \rangle. \, e_n!\langle \mathbf{true}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



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

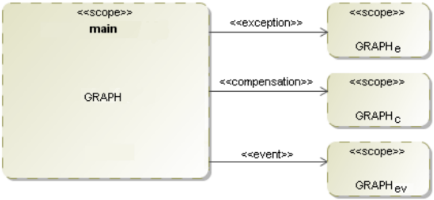


$$* \, e?\langle \mathbf{true} \rangle. \, (\, e_1!\langle g_1 \rangle \mid \ldots \mid e_n!\langle g_n \rangle \,)$$

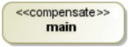


$$* \, e_1?\langle \mathbf{true} \rangle. \; \ldots \; . \, e_n?\langle \mathbf{true} \rangle. \, e!\langle g \rangle$$

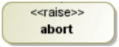
From UML4SOA to COWS



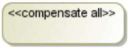
$$
\begin{array}{l}
[\mathtt{r}, \mathit{stack}] \\
(\,[k]\,(\mathrm{GRAPH}\,;\,\{\!|\, c \bullet \mathit{main}?\langle\rangle.\,\mathrm{GRAPH}_c\,|\!\} \\
\qquad \mid \{\!|\,\mathit{Stack}\,|\!\} \mid *\,\mathrm{GRAPH}_{ev}\,) \\
\quad \mid \mathtt{r}?\langle\rangle.\,\{\!|\,\mathrm{GRAPH}_e\,|\!\}\,)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



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



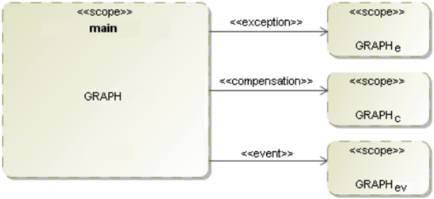
$\mathbf{kill}(k) \mid \{\!|\,\mathtt{r}!\langle\rangle\,|\!\}$



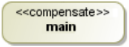
$\mathit{stack} \bullet \mathit{compAll}!\langle\rangle$

Our COWS implementation of UML4SOA constructs follows a compositional approach

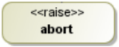
From UML4SOA to COWS



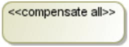
$$[\mathtt{r}, \mathit{stack}]\,(\,[k]\,(\mathrm{GRAPH}\,;\,\{\!|\,c \bullet \mathit{main}?\langle\rangle.\,\mathrm{GRAPH}_c\,|\!\}\;|\;\{\!|\,\mathit{Stack}\,|\!\}\;|\;*\,\mathrm{GRAPH}_{ev}\,)\;|\;\mathtt{r}?\langle\rangle.\,\{\!|\,\mathrm{GRAPH}_e\,|\!\}\,)$$



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



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



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

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*, . . .

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