A logical approach to specify service-oriented application

Stefania Gnesi

ISTI-CNR

IFIP WP 1.3

Sierra Nevada - January 17, 2008

Joint work with Alessandro Fantechi, Alessandro Lapadula, Franco Mazzanti, Rosario Pugliese and Francesco Tiezzi

Outline

Outline

A General Theory of Services

Several formalisms for service description that can lay the mathematical basis for analysing and experimenting with components interactions, and for combining Service-Oriented application have been recently developed (see e.g. the Sensoria project) .

Core calculi

Some calculi based on process algebras for service specifications have been recently proposed that:

- comply with a service-oriented approach to business modelling;
- allow for modular description of services;
- support dynamic, ad-hoc, "just-in-time" composition.

A General Theory of Services

Several formalisms for service description that can lay the mathematical basis for analysing and experimenting with components interactions, and for combining Service-Oriented application have been recently developed (see e.g. the Sensoria project) .

Core calculi

Some calculi based on process algebras for service specifications have been recently proposed that:

- comply with a service-oriented approach to business modelling;
- allow for modular description of services;
- support dynamic, ad-hoc, "just-in-time" composition.

Outline

A more abstract view for Services

- Using calculi, services are specified according to a behavioral description
- Not many analytical tools for checking that services enjoy desirable properties and do not manifest unexpected behaviors are available for the calculi

Logic as specification language

Logics have been since long proved able to reason about such complex systems as SOC applications

- they provide abstract specifications of complex systems;
- can be used for describing system properties rather than system behaviors;

Logic verification framework

- We introduce a logical verification framework for describing functional requirements of services by abstracting away from the computational contexts in which they are operating;
- services are abstractly considered as entities capable of *accepting requests, delivering corresponding responses* and, on-demand, *cancelling requests*.

A more abstract view for Services

- Using calculi, services are specified according to a behavioral description
- Not many analytical tools for checking that services enjoy desirable properties and do not manifest unexpected behaviors are available for the calculi

Logic as specification language

Logics have been since long proved able to reason about such complex systems as SOC applications

- they provide abstract specifications of complex systems;
- can be used for describing system properties rather than system behaviors;

Logic verification framework

- We introduce a logical verification framework for describing functional requirements of services by abstracting away from the computational contexts in which they are operating;
- services are abstractly considered as entities capable of *accepting requests, delivering corresponding responses* and, on-demand, *cancelling requests*.

A more abstract view for Services

- Using calculi, services are specified according to a behavioral description
- Not many analytical tools for checking that services enjoy desirable properties and do not manifest unexpected behaviors are available for the calculi

Logic as specification language

Logics have been since long proved able to reason about such complex systems as SOC applications

- they provide abstract specifications of complex systems;
- can be used for describing system properties rather than system behaviors;

Logic verification framework

- We introduce a logical verification framework for describing functional requirements of services by abstracting away from the computational contexts in which they are operating;
- services are abstractly considered as entities capable of *accepting requests, delivering corresponding responses* and, on-demand, *cancelling requests*.

Outline

Abstract properties of services

A service may be:

- **available**: if it is always capable to accept a request.
- **reliable**: if, when a request is accepted, a final successful response is guaranteed.
- **responsive**: if it always guarantees a response to each received request.
- **broken**: if, after accepting a request, it does not provide the (expected) response.
- **unavailable**: if it refuses all requests.
- **fair**: if it is possible to cancel a request before the response.
- **non-ambiguous**: if, after accepting a request, it provides no more than one response.
- **sequential**: if, after accepting a request, no other requests may be accepted before giving a response.
- **asynchronous**: if, after accepting a request, other requests may be accepted before giving a response.
- **non-persistent**: if, after accepting a request, no other requests can be accepted.

SocL logic

SocL is a logic specifically designed to capture peculiar aspects of services and it has been introduced to formalize the properties above.

- SocL is a variant of the logic UCTL [fmics07], originally introduced to express properties of UML statecharts.
- UCTL and SocL have many commonalities: they share the same temporal logic operators, they are both state and action based branching-time logics, they are both interpreted on Doubly Labeled Transition Systems by exploiting the same on-the-fly model-checking engine.

The two logics mainly differ for the syntax and semantics of state predicates and action formulae, and for the fact that SocL also permits to specify parametric formulae.

SocL logic

SocL is a logic specifically designed to capture peculiar aspects of services and it has been introduced to formalize the properties above.

- SocL is a variant of the logic UCTL [fmics07], originally introduced to express properties of UML statecharts.
- UCTL and SocL have many commonalities: they share the same temporal logic operators, they are both state and action based branching-time logics, they are both interpreted on Doubly Labeled Transition Systems by exploiting the same on-the-fly model-checking engine.

The two logics mainly differ for the syntax and semantics of state predicates and action formulae, and for the fact that SocL also permits to specify parametric formulae.

SocL logic

SocL is a logic specifically designed to capture peculiar aspects of services and it has been introduced to formalize the properties above.

- SocL is a variant of the logic UCTL [fmics07], originally introduced to express properties of UML statecharts.
- UCTL and SocL have many commonalities: they share the same temporal logic operators, they are both state and action based branching-time logics, they are both interpreted on Doubly Labeled Transition Systems by exploiting the same on-the-fly model-checking engine.

The two logics mainly differ for the syntax and semantics of state predicates and action formulae, and for the fact that SocL also permits to specify parametric formulae.

L^{2}TS definition

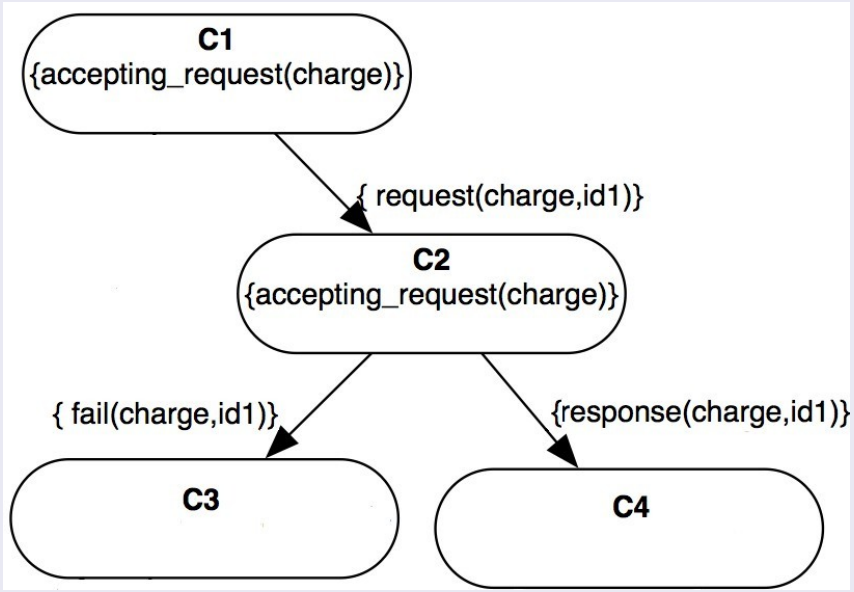
An L^2TS is a tuple $\langle Q, q_0, Act, R, AP, L \rangle$, where:

- Q is a set of states;
- $q_0 \in Q$ is the initial state;
- Act is a finite set of observable events (actions) with α ranging over 2^{Act} and ϵ denoting the empty set;
- $R \subseteq Q \times 2^{Act} \times Q$ is the transition relation.
- AP is a set of atomic propositions with π ranging over AP;
- $L : Q \longrightarrow 2^{AP}$ is a labelling function that maps each state in Q to a subset of AP.

- Basically, an L^{2}TS is an LTS (defined as the quadruple $\langle Q, q_0, Act, R \rangle$), extended with a labelling function from states to sets of atomic propositions.
- By means of an L^{2}TS, a system can be characterized by states and state changes and by the events (actions) that are performed when moving from one state to another.

An example of L^2TS

- Act ranges over by observable actions, such as: request(i, c), response(i, c),cancel(i, c) and fail(i, c), where
- i indicates the interaction to which the operation performed by a service belongs c denotes a tuple of correlation values that identifies a particular invocation of the operation
- AP is a finite set of atomic propositions, parameterized by interactions and correlation tuples, like accepting_request(i) and accepting_cancel(i), that can be true over a state of an L^2TS.



SocL logic

Action formulae syntax

Given *Act*\$ as *Act* plus correlation variable names *a*\$ $\mathcal{AF}(Act\$)$ is defined as follows:

$$\gamma ::= a\$ \mid \chi \qquad \chi ::= tt \mid a\% \mid \tau \mid \neg\chi \mid \chi \wedge \chi$$

SocL syntax

$$\begin{array}{lll} \textit{(state formulae)}\ \phi & ::= & \textit{true} \mid \pi \mid \neg\phi \mid \phi \wedge \phi' \mid E\Psi \mid A\Psi \\ \textit{(path formulae)}\ \Psi & ::= & X_\gamma \phi \mid \phi\ {}_\chi U\ \phi' \mid \phi\ {}_\chi U_\gamma\ \phi' \mid \phi\ {}_\chi W\ \phi' \mid \phi\ {}_\chi W_\gamma\ \phi' \end{array}$$

Some derived modalities

$< \gamma > \phi$ stands for $EX_\gamma\ \phi$

$[\gamma]\ \phi$ 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 semantics

- *Substitutions*, ranged over by ρ, are functions mapping correlation variables to values and are written as collections of pairs of the form var/val.
- The empty substitution is denoted by $\emptyset$.
- Application of substitution ρ to a formula ϕ, written $\phi \cdot \rho$, has the effect of replacing every occurrence $\%var$ in ϕ with val, for each $var/val \in \rho$.
- The partial function $\mathrm{m}(\_,\_)$ from pairs of actions to substitutions, that permits performing *pattern-matching*, is defined by the following rules:

$$\mathrm{m}(request(i,c), request(i,c')) = \mathrm{m}(c,c') \qquad \mathrm{m}(\$var, val) = \{var/val\}$$

$$\mathrm{m}(response(i,c), response(i,c')) = \mathrm{m}(c,c') \qquad \mathrm{m}(val, val) = \emptyset$$

$$\mathrm{m}(cancel(i,c), cancel(i,c')) = \mathrm{m}(c,c') \qquad \mathrm{m}(fail(i,c), fail(i,c')) = \mathrm{m}(c,c')$$

$$\mathrm{m}((e_1 \cdot c_1), (e_2 \cdot c_2)) = \mathrm{m}(e_1, e_2) \cup \mathrm{m}(c_1, c_2)$$

where notation $e \cdot c$ stands for a tuple with first element e.

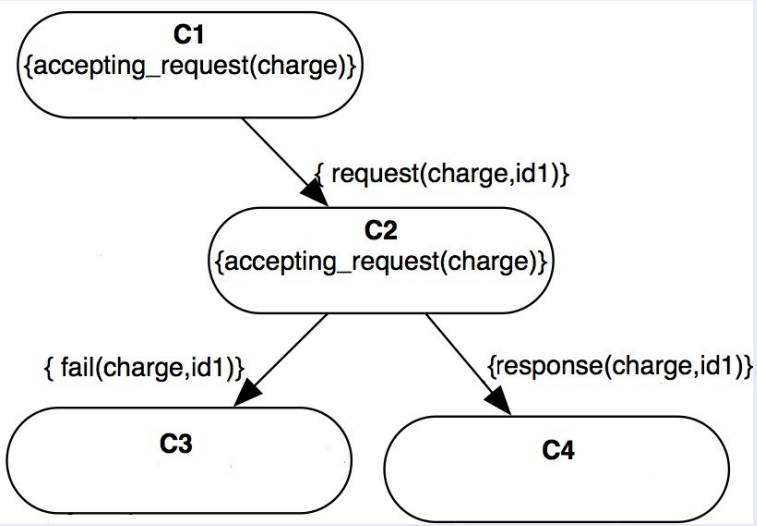
Parametric SocL formulae evaluation

$$\phi = EX_{request(charge,\langle \$id \rangle)}\, AX_{response(charge,\langle \%id \rangle)}\, true$$

$$request(charge, \langle id1 \rangle) \models request(charge, \langle \$id \rangle) \rhd \rho$$

where the produced substitution ρ is

$$\rho = \mathrm{m}\,(request(charge, \langle \$id \rangle), request(charge, \langle id1 \rangle)) = \mathrm{m}\,(\$id, id1) = \{id/id1\}$$



SocL semantics - Action formulae semantics

The satisfaction relation $\models$ for action formulae ($\alpha \models \gamma \rhd \rho$) is defined over sets of observable actions in *Act*\$, and over a substitution.

- $\alpha \models a\$ \rhd \rho$ iff $\exists!\, b \in \alpha$ such that $\mathsf{m}(a\$, b) = \rho$;
- $\alpha \models \chi \rhd \emptyset$ iff $\alpha \models \chi$

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

  - $\alpha \models tt$ holds always;
  - $\alpha \models a\%$ iff $\exists!\, b \in \alpha$ such that $\mathsf{m}(a\%, b) = \emptyset$;
  - $\alpha \models \tau$ iff $\alpha = \epsilon$;
  - $\alpha \models \neg\chi$ iff not $\alpha \models \chi$;
  - $\alpha \models \chi \wedge \chi'$ iff $\alpha \models \chi$ and $\alpha \models \chi'$.

- The notation $\alpha \models \chi \rhd \rho$ means: the formula γ is satisfied over the action (set) α (only) under substitution ρ.
- We assume that inside a single evolution step two or more actions with the same type and interaction do not occur.

SocL semantics

The satisfaction relation of closed SocL formulae, i.e. formulae without unbound variables, over a L^2TS is defined as follows:

- $q \models \mathit{true}$ 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 $\exists\, \sigma \in \mathit{path}(q)$ such that $\sigma \models \Psi$;
- $q \models A\Psi$ iff $\forall\, \sigma \in \mathit{path}(q) \quad \sigma \models \Psi$;
- $\sigma \models X_\gamma \phi$ iff $\sigma = (q, \alpha, q')\sigma'$, $\alpha \models \gamma \rhd \rho$, and $q' \models \phi \cdot \rho$;
- $\sigma \models \phi \ {}_\chi U \phi'$ iff there exists $j \geq 0$ such that $\sigma(j) \models \phi'$ and for all $0 \leq i < j$:
  $\sigma = \sigma'(\sigma(i), \alpha_{i+1}, \sigma(i+1))\sigma''$ implies $\sigma(i) \models \phi$ and $\alpha_{i+1} = \epsilon$ or $\alpha_{i+1} \models \chi$;

SocL semantics

- $\sigma \models \phi \ {}_{\chi}U_{\gamma} \phi'$ iff
  there exists $j \geq 1$ such that $\sigma = \sigma'(\sigma(j-1), \alpha_j, \sigma(j))\sigma''$ and $\alpha_j \models \gamma \rhd \rho$ and $\sigma(j) \models \phi' \cdot \rho$ and $\sigma(j-1) \models \phi$, and for all $0 < i < j$:
  $\sigma = \sigma'_i(\sigma(i-1), \alpha_i, \sigma(i))\sigma''_i$ implies $\sigma(i-1) \models \phi$, and $\alpha_i = \epsilon$ or $\alpha_i \models \chi$;

- $\sigma \models \phi \ {}_{\chi}W \phi'$ iff either
  there exists $j \geq 0$ such that $\sigma(j) \models \phi'$ and for all $0 \leq i < j$:
  $\sigma = \sigma'(\sigma(i), \alpha_{i+1}, \sigma(i+1))\sigma''$ implies $\sigma(i) \models \phi$ and $\alpha_{i+1} = \epsilon$ or $\alpha_{i+1} \models \chi$
  or for all $0 \leq i$:
  $\sigma = \sigma'(\sigma(i), \alpha_{i+1}, \sigma(i+1))\sigma''$ implies $\sigma(i) \models \phi$, and $\alpha_{i+1} = \epsilon$ or $\alpha_{i+1} \models \chi$;

- $\sigma \models \phi \ {}_{\chi}W_{\gamma} \phi'$ iff either
  there exists $j \geq 1$ such that $\sigma = \sigma'(\sigma(j-1), \alpha_j, \sigma(j))\sigma''$ and
  $\alpha_j \models \gamma \rhd \rho$ and $\sigma(j) \models \phi' \cdot \rho$ and $\sigma(j-1) \models \phi$, and for all $0 < i < j$:
  $\sigma = \sigma'_i(\sigma(i-1), \alpha_i, \sigma(i))\sigma''_i$ implies $\sigma(i-1) \models \phi$, and $\alpha_i = \epsilon$ or $\alpha_i \models \chi$
  or for all $0 \leq i$:
  $\sigma = \sigma'_i(\sigma(i-1), \alpha_i, \sigma(i))\sigma''_i$ implies $\sigma(i-1) \models \phi$, and $\alpha_{i+1} = \epsilon$ or $\alpha_{i+1} \models \chi$.

SocL description of abstract properties

- Availability:if it is always capable to accept a request
  $AG(accepting\_request(i))$.
  $AGAF(accepting\_request(i))$ (weaker).

- Reliability: if, when a request is accepted, a final successful response is guaranteed.
  $AG[request(i,\$v)]AF_{response(i,\%v)}\ true$.

  Notably, the response belongs to the same interaction i of the accepted request and they are correlated by the variable v.

- Responsiveness: if it always guarantees a response to each received request.
  $AG[request(i,\$v)]\ AF_{response(i,\%v)\vee fail(i,\%v)}\ true$.

- Broken service: if it does not provide the (expected) response
  $\neg AG[request(i,\$v)]\ AF_{response(i,\%v)\vee fail(i,\%v)}\ true$. *(temporarily broken)*

  $AG[request(i,\$v)]\ \neg EF_{response(i,\%v)\vee fail(i,\%v)}\ true$. *(permanently broken)*

- Unavailability: if it refuses all requests.
  $AG[request(i,\$v)]\ AF_{fail(i,\%v)}\ true$.

SocL description of abstract properties

- Fairness :if it is possible to cancel a request before the response
  $AG[request(i,\$v)]\, A(accepting\_cancel(i,\%v)\ {}_{tt}W_{response(i,\%v)\vee fail(i,\%v)}\ true)$.
  (fairness towards the client);
  $AG[response(i,\$v)]\, \neg EF < cancel(i,\%v) > true$
  (fairness towards the server).

- Unambiguity: if, after accepting a request, it provides no more than one response.
  $AG[request(i,\$v)]\, \neg EF < response(i,\%v) >$ EF $< response(i,\%v) > true$.

- Sequentiality: if, after accepting a request, no other requests may be accepted before giving a response
  $AG[request(i,\$v)]\, A(\neg\, accepting\_request(i)\ {}_{tt}U_{response(i,\%v)\vee fail(i,\%v)}\ true)$.

- Asynchronicity: if, after accepting a request, other requests may be accepted before giving a response.
  $AG[request(i,\$v)]\, EF < response(i,\%v) \vee fail(i,\%v) > true$.

- Non-persistency: if, after accepting a request, no other requests can be accepted. $AG[request(i,\$v)]\, AG\, \neg\, accepting\_request(i)$.

Outline

COWS: Calculus for Orchestration of Web Services

- Services can create new instances to serve specific requests
- Instances contain concurrent threads (possibly with a *shared state*)
- Services and instances communicate through endpoints
- Endpoint's names can be communicated (only the 'send capability')
- Communication is regulated by a pattern-matching mechanism
  - that permits correlating, by means of their same contents, different service interactions logically forming a same 'session'
- The only binder is the delimitation operator; it can generate fresh names (like the restriction operator of the π-calculus) and also regulate the range of application of substitutions generated by communication
- Termination of parallel activities can be forced by using a kill, but sensitive code can be protected from the effect of a forced termination

COWS: Syntax

s ::=	(services)	(notations)
$\mathbf{kill}(k)$	(kill)	k *(killer) labels*
$\mid\ u \bullet u'!\bar{e}$	(invoke)	e *expressions*
$\mid\ \sum_{i=0}^{r} p_i \bullet o_i?\bar{w}_i.s_i$	(receive-guarded choice)	x *variables*
$\mid\ s \mid s$	(parallel composition)	v *values*
$\mid\ \{\!\mid s \mid\!\}$	(protection)	n, p, o *names*
$\mid\ [d]\, s$	(delimitation)	u: names|vars
$\mid\ * s$	(replication)	w: values|vars
		d: labels|names|vars

Only one binding construct: $[d]\, s$ binds d in the scope s
(free/bound names/variables/labels and closed terms defined accordingly)

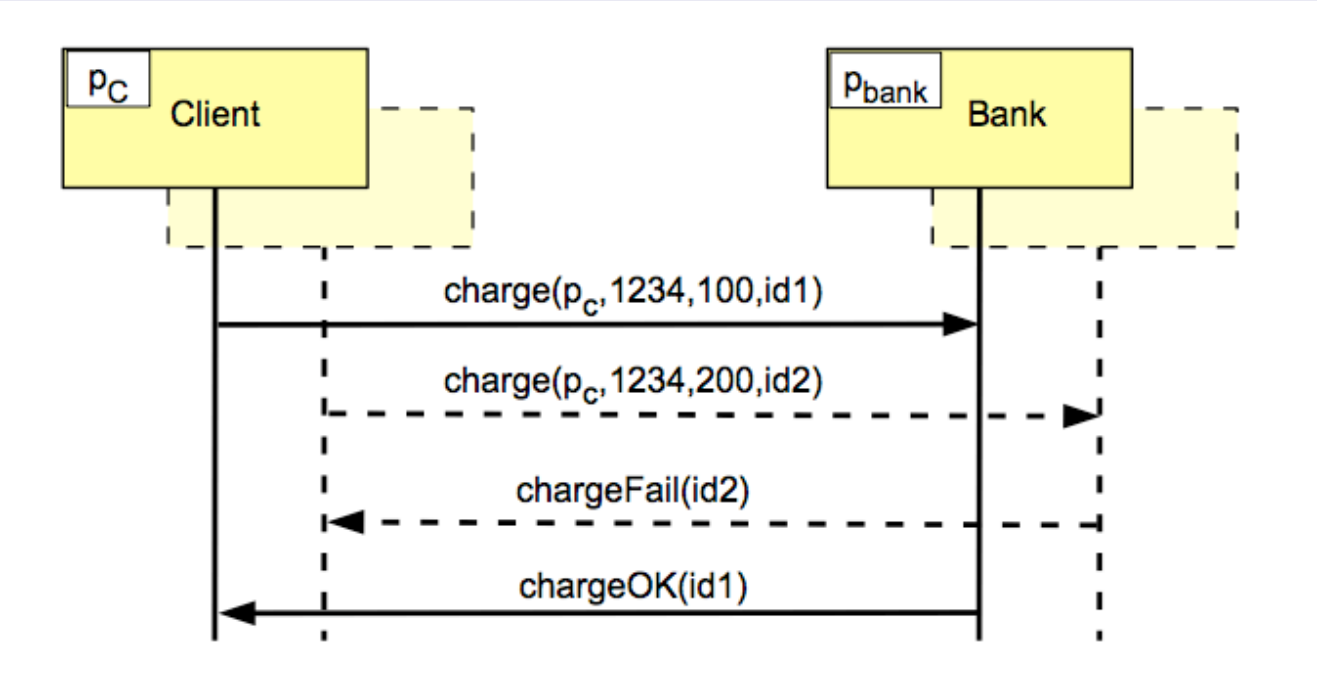
An example

BankInterface; *CreditRating*

A bank service is composed of two persistent subservices: *BankInterface* and *CreditRating*.

The scenario also involves the processes $Client_1$ and $Client_2$ that model requests for charging the customer's credit card with some amount.

Bank service



Bank service scenario

$$[o_{check}, o_{checkOK}, o_{checkFail}]\,(\,*\,\mathit{BankInterface} \mid *\,\mathit{CreditRating}\,) \mid \mathit{Client}_1 \mid \mathit{Client}_2$$

$$\begin{aligned}
\mathit{BankInterface} \triangleq\ & [x_{cust}, x_{cc}, x_{amount}, x_{id}] \\
& p_{bank} \bullet o_{charge}?\langle x_{cust}, x_{cc}, x_{amount}, x_{id}\rangle. \\
& (\, p_{bank} \bullet o_{check}!\langle x_{id}, x_{cc}, x_{amount}\rangle \\
& \quad \mid p_{bank} \bullet o_{checkOK}?\langle x_{id}\rangle.\ x_{cust} \bullet o_{chargeOK}!\langle x_{id}\rangle \\
& \quad + p_{bank} \bullet o_{checkFail}?\langle x_{id}\rangle.\ x_{cust} \bullet o_{chargeFail}!\langle x_{id}\rangle\,)
\end{aligned}$$

$$\begin{aligned}
\mathit{CreditRating} \triangleq\ & [x_{id}, x_{cc}, x_a] \\
& p_{bank} \bullet o_{check}?\langle x_{id}, x_{cc}, x_a\rangle. \\
& [p, o]\,(\, p \bullet o!\langle\rangle \mid p \bullet o?\langle\rangle.\ p_{bank} \bullet o_{checkOK}!\langle x_{id}\rangle \\
& \qquad + p \bullet o?\langle\rangle.\ p_{bank} \bullet o_{checkFail}!\langle x_{id}\rangle\,)
\end{aligned}$$

$$\mathit{Client}_1 \triangleq p_{bank} \bullet o_{charge}!\langle p_C, 1234, 100, id_1\rangle \mid p_C \bullet o_{chargeOK}?\langle id_1\rangle + p_C \bullet o_{chargeFail}?\langle id_1\rangle$$

$$\mathit{Client}_2 \triangleq p_{bank} \bullet o_{charge}!\langle p_C, 1234, 200, id_2\rangle \mid p_C \bullet o_{chargeOK}?\langle id_2\rangle + p_C \bullet o_{chargeFail}?\langle id_2\rangle$$

Outline

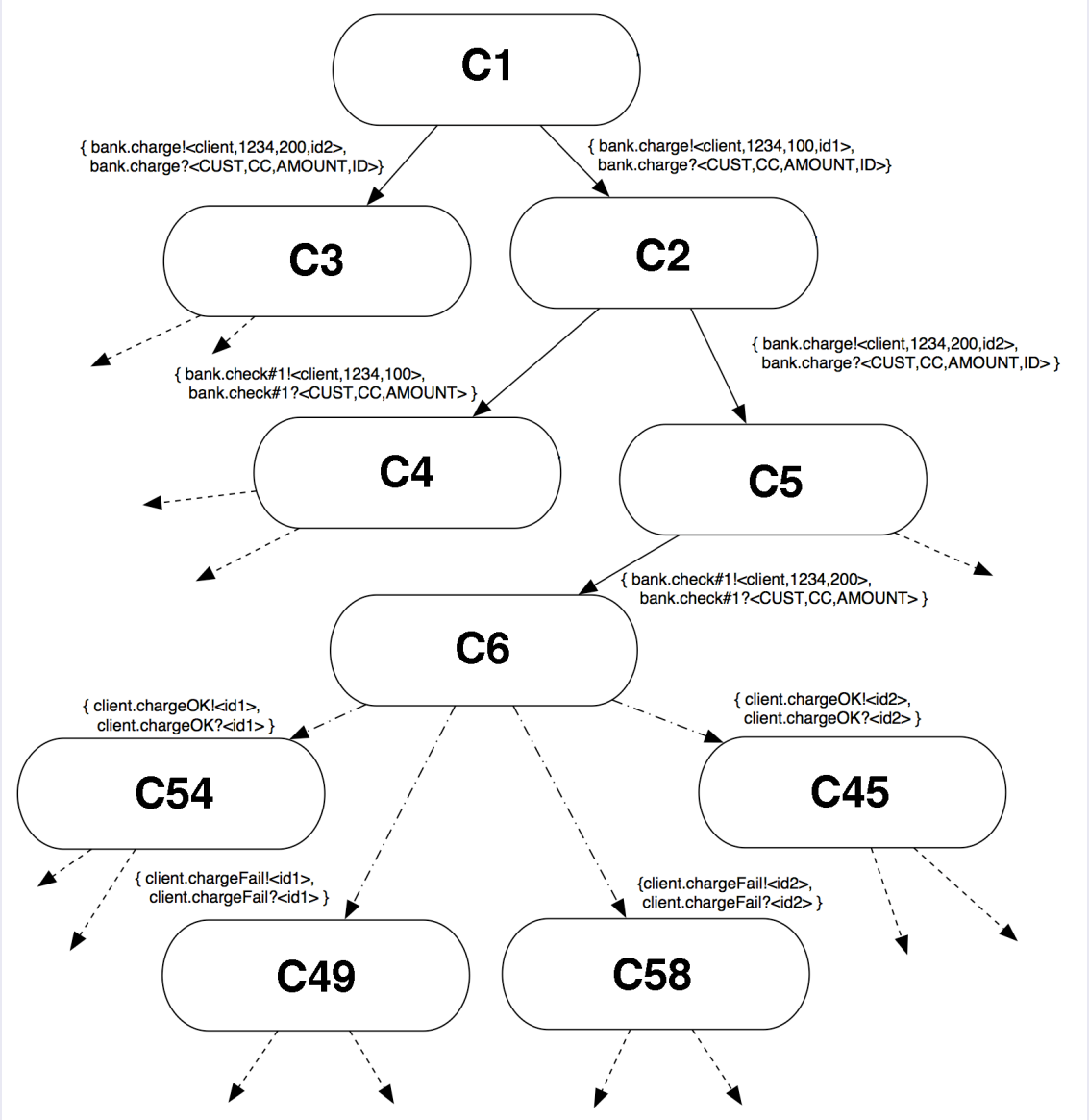
SocL and calculi for services

- SocL my be also used to describe properties of services behaviorally specified using a calculus
- The model checker developed for SocL over L^2TSmay be used to check the properties

USING socL on COWS

1. the semantics of a COWS term is defined by using a Labelled Transition System (LTS).
2. this LTS is transformed into an L^2TS by labelling each state with the set of actions the COWS term is able to perform immediately from that state.
3. by applying a set of application-dependent abstraction rules over the actions, the concrete L^2TS is abstracted into a simpler L^2TS.
4. SocL formulae are checked over this abstract L^2TS.

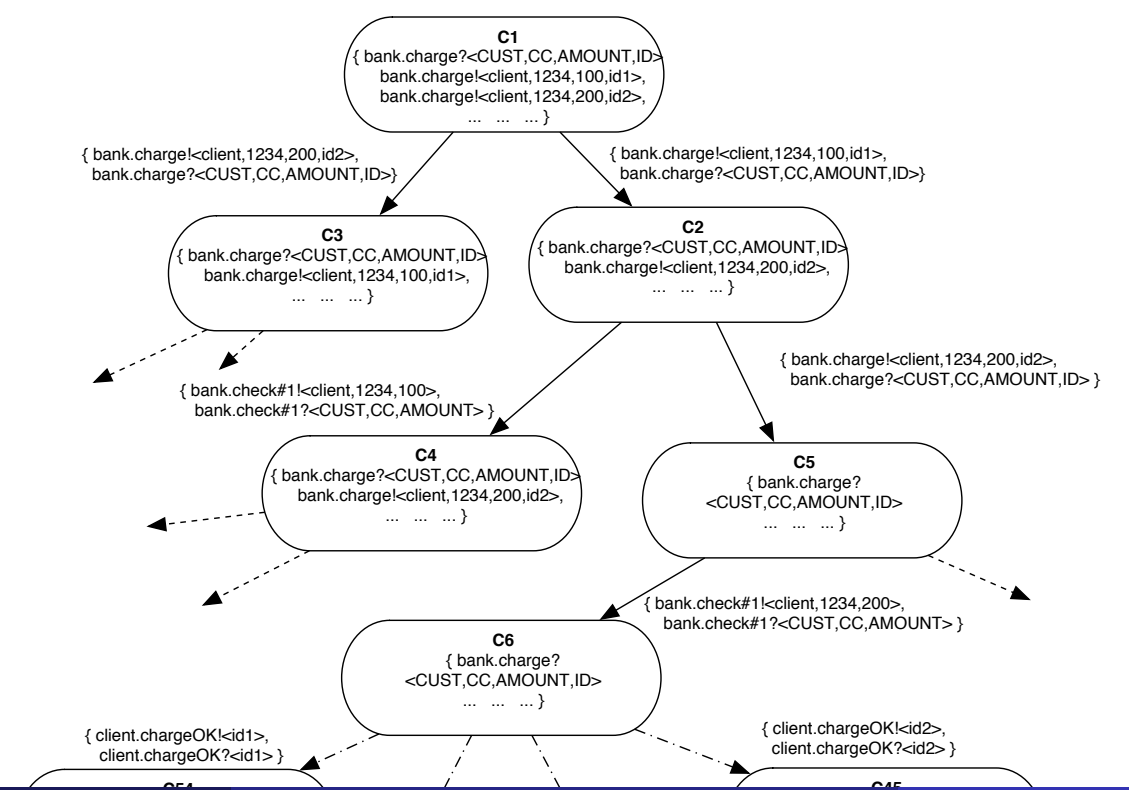
Concrete LTS for the COWS specification



COWS and SocL

The semantics of COWS bank scenario is given over a L^2TS:

- transitions are labelled by ‘concrete’ actions, i.e. those actions occurring in the COWS term.
- each state is labelled with the set of actions that each active subterm of the COWS term would be able to perform immediately.



COWS and SocL

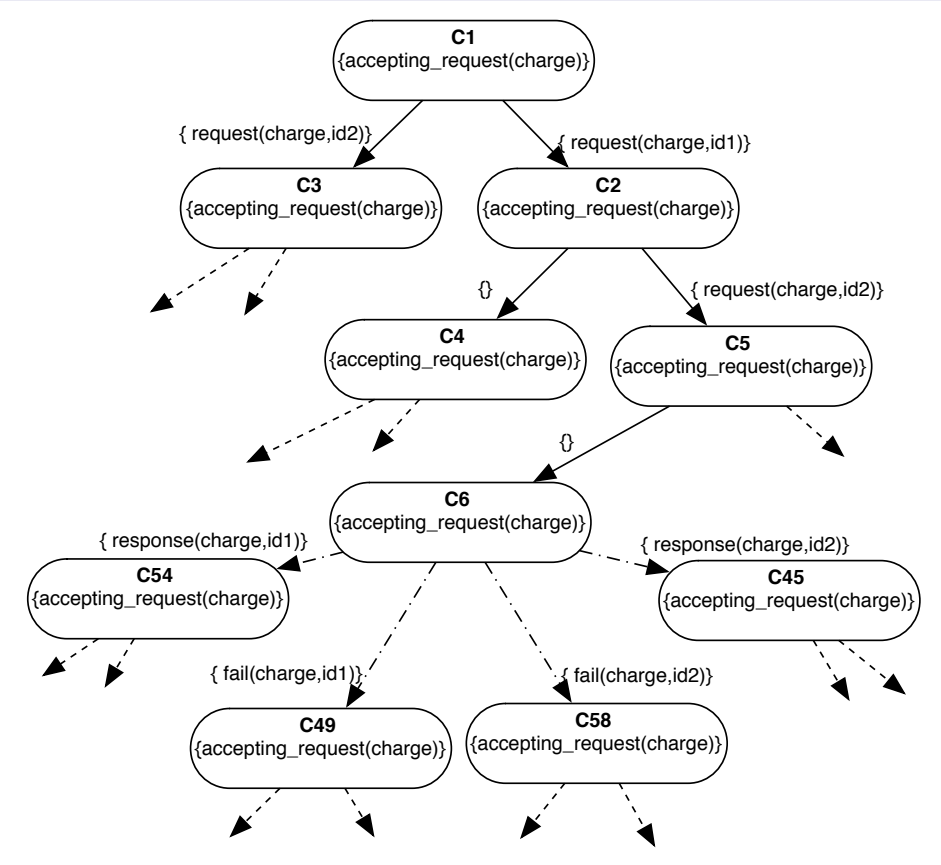
The semantics of COWS bank scenario is given over an abstract L^2TS

- applying a set of suitable abstraction rules to the concrete action.
- replacing concrete labels on the transitions with actions belonging to the set *Act*, i.e. $request(i, c)$, $response(i, c)$, $cancel(i, c)$ and $fail(i, c)$

$$
\begin{array}{rlcl}
Action: & charge\langle *, *, *, \$1\rangle & \rightarrow & request(charge, \langle \$1\rangle) \\
Action: & chargeOK\langle \$1\rangle & \rightarrow & response(charge, \langle \$1\rangle) \\
Action: & chargeFail\langle \$1\rangle & \rightarrow & fail(charge, \langle \$1\rangle) \\
State: & charge & \rightarrow & accepting\_request(charge)
\end{array}
$$

COWS and SocL

The set of "*Action* :" and "*State* :" rules is not defined once and for all, but is application-dependent and, thus, must be defined from time to time.



Outline

Model checking COWS

- the model checker CMC is implemented by exploiting an on-the-fly algorithm.
- the instantiation of the generic patterns of formulae over the bank service has been obtained by just replacing any occurrence of *i* with *charge*

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

Property	Result	States
Available	TRUE	274
Reliable	FALSE	37
Responsive	TRUE	274
Permanently Broken	FALSE	12
Temporarily Broken	FALSE	274
Unavailable	FALSE	18
Fair 2	TRUE	274
Non-ambiguous	TRUE	274
Sequential	FALSE	3
Asynchronous	TRUE	274
Non-persistent	FALSE	3

Generated counterexample

Non-persistent service:
$AG[request(charge, \$v)]\, AG\, \neg\, accepting\_request(charge)$.

———————————————-
The formula: AG [request(charge,id1)] AG not (accepting_request(charge))
  is FOUND_FALSE in State C1
because
 the formula: [request(charge,id1)] AG not (accepting_request(charge))
  is FOUND_FALSE in State C1
because
 C1 –> C2 { bank.charge!,bank.charge? } {{ request(charge,id1)}}
 and the formula: AG not accepting_request(charge)
 is FOUND_FALSE in State C2
because
 the formula: not accepting_request(charge)
 is FOUND_FALSE in State C2
because
 the formula: (accepting_request(charge))
  is FOUND_TRUE in State C2
———————————————-

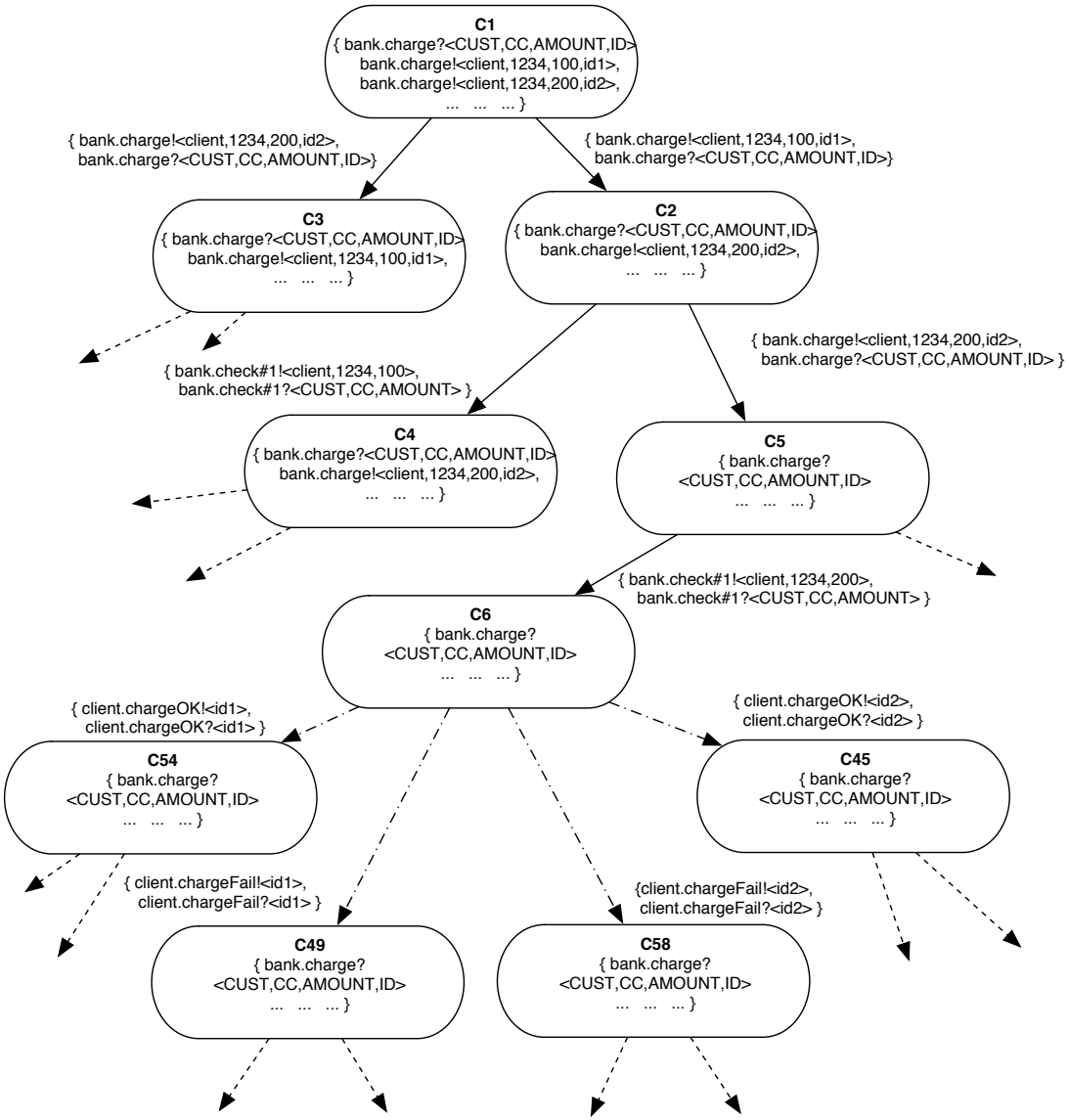
Outline

Conclusions

- The logic interpretation model (i.e. L^2TSs) is independent from the service specification language (i.e. COWS), it can be easily tailored to be used in conjunction with other SOC specification languages.
- SocL permits expressing properties about any kind of interaction pattern, such as *one–way*, *request–response*, *one request–multiple responses*, *one request-none of two possible responses*
- The use of L^2TSs as model of the logic helps to reduce the state space and, hence, the memory used and the time spent for verification.
- We are currently only able to analyse systems of services 'as a whole' i.e. we cannot analyse isolated services

Thank you for your attention!

Concrete L^2TS of the same COWS term



Abstract L^{2}TSof the same COWS term