A Framework for Requirements Monitoring of Service-Based Systems

George Spanoudakis

Software Engineering Group
Department of Computing
City University

Outline

- The problem
- Current approaches
- Complications of service-based systems
- Our approach
  - architecture
  - types of requirements
  - specification/extraction of requirements
  - types of deviations and examples
  - consistency checking
- Future work

The problem

Run-time system requirements monitoring

Check whether the run-time behaviour of a software system satisfies the requirements set for it during development

Requirements

- Functional requirements
- Quality constraints (e.g. performance, availability constraints)
- Assumptions about the behaviour of agents in the environment of a system

Why requirements may fail at run-time?

- Unforeseen interactions between system components
- Unexpected behaviour of agents interacting with the system

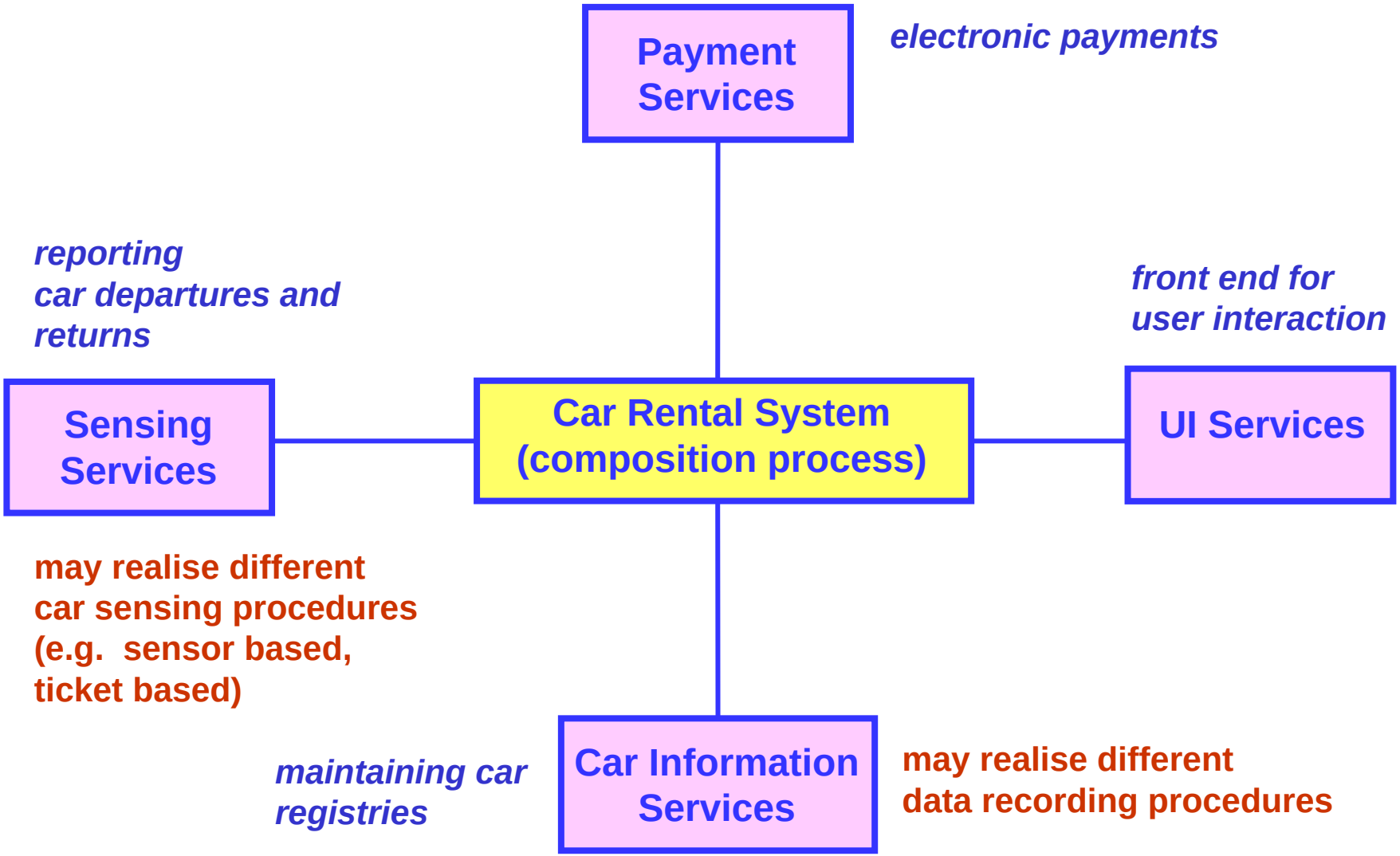
Current Approaches

- Specify requirements in some high level formal language (typically based on temporal logic)
- Translate the specification of each requirement into a pattern of events that can be observed at run-time
- Instrument the source code of the system to make it produce the required events
- Check the adherence of system behaviour to individual event patterns (and requirements)

Complications with service based systems

- autonomous components (web-services) within the boundaries of the system
  - monitoring individual requirements in isolation cannot capture all potential problems
  - top-down mapping of requirements onto monitorable events and code instrumentation are not generally viable options (due to diverse ownership and dynamic assembling of component services)
  - need to reason in the presence of incomplete information (services may have failed without any form of notification)
- different realisations of same functional roles and incomplete behavioural specifications
  - specification of assumptions for individual services

An example



Examples of problems for CRS

Sensing Services may fail to report departures/arrivals of cars

➡ difficult to distinguish failure from inactivity

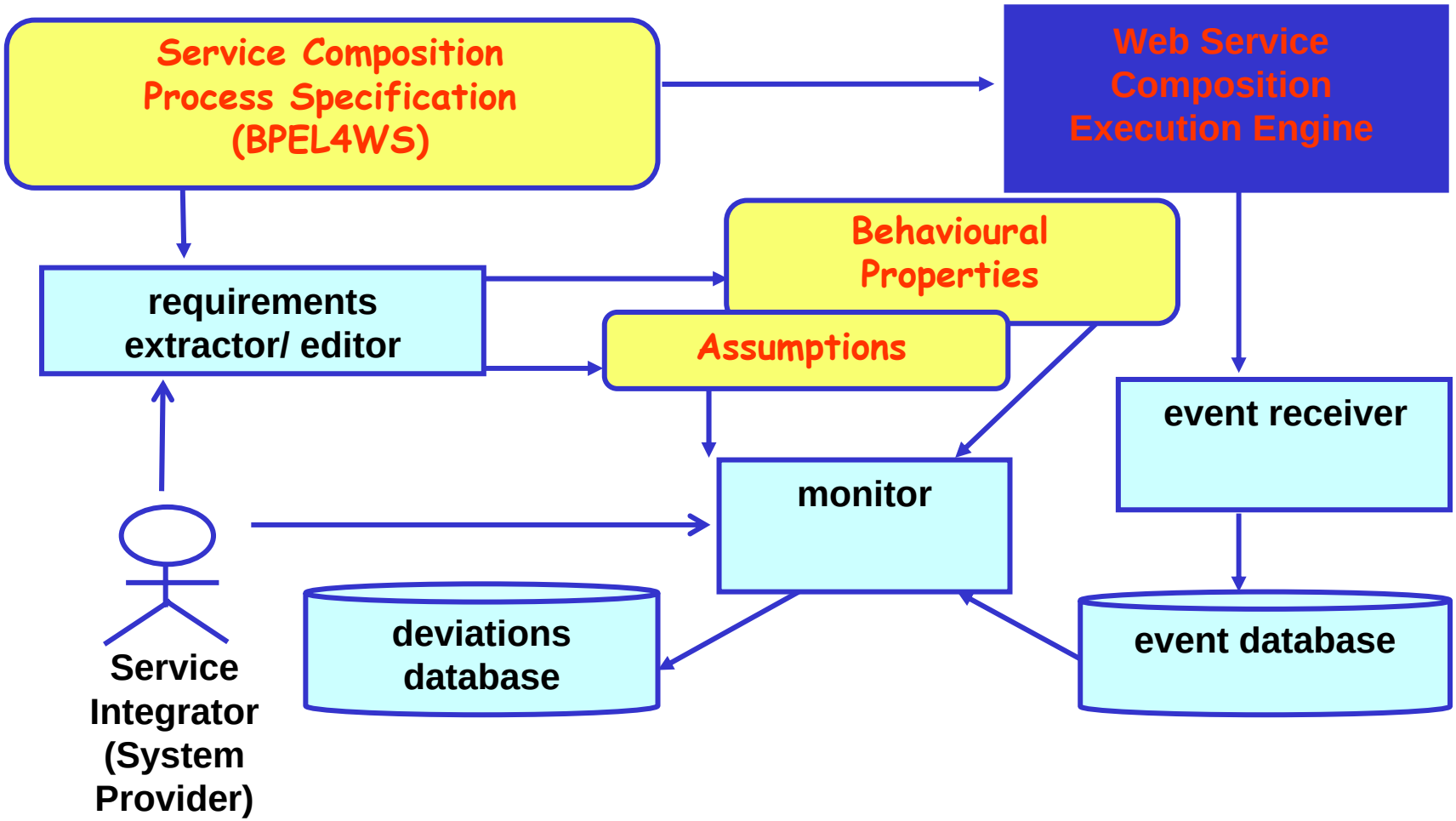
➡ car information services may fail to keep accurate records of car availability if they update car availability data prompted by notifications from sensing services

➡ the system composition process may respond wrongly to car hire requests etc

Our approach

- Introduction of types of deviation beyond classical inconsistencies
- Identification of primitive monitorable events & extraction of behavioural properties from the specification of the composition process of a service-based system
- Specification of additional monitorable assumptions in terms of the identified primitive monitorable events (bottom-up approach)
- Deployment of event calculus as the specification framework of the monitorable properties
- Continual monitoring of requirements in parallel with system operations

Architecture



Types of requirements

- ***behavioural properties*** of the system composition process
  - automatically extracted from the specification of the composition process
- ***assumptions*** about the behaviour of services and actors interacting with the system
  - specified by system providers using events that can be captured during the execution of the composition process

behavioural properties and assumptions are specified in event calculus

Event Calculus

A first-order formal language for specifying properties of dynamic systems which change over time using predefined predicates, including:

- **Happens(e, t, ℜ(t1,t2))** – occurrence of an event ***e*** of instantaneous duration at some time ***t*** within the time range **ℜ(t1,t2)**
- **HoldsAt(f,t)** – fluent ***f*** holds at time **t**.
- **Initiates(e,f,t)** – fluent ***f*** starts to hold after the event ***e*** at time ***t***.
- **Terminates(e,f,t)** – fluent ***f*** ceases to hold after the event ***e*** occurs at time ***t***

Types of Events and Fluents

- 5 types of **events** signifying
  - requests for execution of operations in individual services

    ic:S:OperationName(operation-parameters)
  - returns from the execution of service operations

    ir:S:OperationName (operation-parameters)
  - requests for execution of operations by the composition process

    rc:S:OperationName(operation-parameters)
  - replies generated by the composition process

    re:S:OperationName(operation-parameters)
  - assignments of values to variables of the system composition process

    as:AssignmentName(var)
- **fluents** signifying the state of the system (e.g. in terms of values of system variables) at different instances of time

  {equalTo | lessThan | greaterThan} (v1,v2) + ...

Examples: Behavioural Properties

CRS will accept a car hire request if it can find a car at the relevant car park

(B1) **Happens**(rc:UI:CarRequest(oID1),t1,$\Re$(t1,t1)) $\wedge$
Initiates(rc:UI:CarRequest(oID1), equalTo(p,pID),t1) $\wedge$
Happens(ic:IS:FindAvailable(oID2, pID),t2, $\Re$(t1,t2)) $\wedge$
Happens(ir:IS:FindAvailable(oID2),t3, $\Re$(t2,t3)) $\wedge$
Initiates(ir:IS:FindAvailable(oID2), equalTo(res,vID),t3)
$\Rightarrow$ ($\exists$t4:Time)
Happens(re:UI:CarHire(oID3,vID), t4, $\Re$(t3,t3+tu))

Examples: Assumptions

Between two events signifying the entrance of a car C in a car park P1 and later the entrance of the same car in a car park P2 there must be an event signifying the departure of C from P1 (assumption about the SS service)

(A1)

Happens(rc:SS:Enter(oID1),t1,$\Re$(t1,t1)) $\wedge$
Initiates(rc:SS:Enter(oID1), equalTo(v1, vID), t1) $\wedge$
Initiates(rc:SS:Enter(oID1), equalTo(p1, pID1), t1) $\wedge$
Happens(rc:SS:Enter(oID2),t2,$\Re$(t1+tu,t2)) $\wedge$
Initiates(rc:SS:Enter(oID2), equalTo(v2, vID), t2)
$\Rightarrow$
($\exists$t3:Time) **Happens**(rc:SS:Depart(oID3),t3,$\Re$(t1+tu,t2−tu)) $\wedge$
Initiates(rc:SS:Depart(oID3), equalTo(v3, vID), t3) $\wedge$
Initiates(rc:SS:Depart(oID3), equalTo(p3, pID1), t3)

Extraction from BPEL4WS basic activities

operation invocation in partner service

```
<invoke partner="P”
portType= "a:Pport”
operation= "O”
inputVariable = "X”
outputVariable= "Y"/>
```

Happens(ic:P:*O*(vID,vX),t1, ℜ(t1,t1)) ∧
(∃t2) **Happens**(ir:P:*O*(vID),t2, ℜ(t1,t2)) ∧
Initiates(ir:P:*O*(vID),equalTo(Y,vY),t2)

assignment of variable value

```
<assign name ="A"> <copy>
<from variable="X" part="a"/>
<to variable="Y" part="b"/>
</copy> </assign>
```

Happens(as:*A*(vID),t1, ℜ(t1,t1)) ∧ (∃t2)
Initiates(as:*A*(vID), equalTo(Y.b,vX.a),t2)
∧ (t1 < t2)

Extraction from BPEL4WS structured activities

```
<flow>
<links> <link name="AtoB"/>
        <link name="AtoC"/> ...
</links>
<actType name="A">
 <source linkName="AtoB"
 transitionCondition="P=v1"/>
<source linkName="AtoC"/>
...
</actType>
<actType name="B">
 <target linkName="AtoB" /> ...
</actType>
<actType name="C">
 <target linkName="AtoC" /> ...
 </actType>
</flow>
```

activity flow

$EC(A,[]) \wedge HoldsAt(equalTo(P,v_1),t_1) \wedge max_t(A) < t_2$
$\Rightarrow$
$EC(B,[min_t(B)]) \wedge t_2 < min_t(B)$

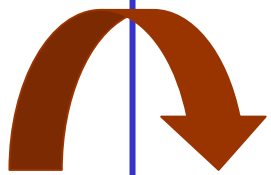
$EC(A,[]) \Rightarrow$
$EC(C,[min_t(c)]) \wedge max_t(A) < min_t(C)$

$EC(D)$

- **actType:** any type of BPEL4WS activity;
- **EC(A, [t_1,…,t_n]):** the formulas *A* is transformed to are re-written with the quantifiers of universally quantified time vars reversed ($\forall \rightarrow \exists$);
- **min_t(X):** time of the *earliest* predicate in the formulas of activity X, and
- **max_t(X):** time of the *latest* predicate in the formulas of activity X.

Extraction: an example

```
<process name="CRS" .../>
<partners> ... </partners> ...
<flow>
<links> <link name="receive-to-auth"/>
... </links>
<receive name="receiveRequest"
  partner="UI"
  portType="sns:CRSUI"
  operation="CarRequest" variable="Req"
  createInstance="yes">
 <source linkName="receive-to-auth"/>
 <correlations> ... </correlations>
</receive>
<sequence>
<target linkName="receive-to-auth"/>
<assign name="a1">
  <copy>
    <from variable="Req" part="Loc"/>
    <to variable="Q" part="Loc"/> </copy>
</assign>
<invoke name="findCar"  partner="IS"
  portType="crns:CRSIS" operation="FindAvailable"
  inputVariable="Q" outputVariable="Res"> ...
</invoke> </sequence> ... </flow> ...</process>
```



Happens(rc:UI:CarRequest(oID1),t1,R(t1,t1)) ∧
Initiates(rc:UI:CarRequest(oID1),
equalTo(Req.Loc,vReq.Loc), t1) ∧
Initiates(rc:UI:CarRequest(oID1),
equalTo(Req.CId, vReq.CId),t1) ⟹ ((∃t2) (t1 < t2) ∧
Happens(as:a1(aID),t2,ℜ(t2,t2)) ∧ (∃t3) (t2 < t3) ∧
Initiates(as:a1(aID),equalTo(Q.Loc, vReq.loc),t3)
⟹ (∃t4)
Happens(ic:IS:FindAvailable(oID2,vQ),t4,ℜ(t3,t4)) ∧
(∃t5) **Happens**(ir:IS:FindAvailable(oID2,vQ),t5,
ℜ(t4,t5)) ∧
Initiates(ir:IS:FindAvailable(oID2,vQ),
equalTo(Res,vRes), t5))

Three Types of Deviation

- Inconsistency w.r.t recorded behaviour
- Inconsistency w.r.t expected behaviour
- Unjustified behaviour

Inconsistency w.r.t Recorded Behaviour

An assumption of the form f: $\mathbf{C \Rightarrow A}$ is inconsistent with the *recorded behaviour* of a system *S* at time *T* if and only if:

$$\{E_R(T)\} \models_{nf} \neg f$$

where

- $\models_{nf}$ signifies entailment using the normal rules of inference of first-order logic and the principle of negation as failure

- $E_R(T)$ is the set of the events recorded by the system from the start of its operation until time T

Inconsistency w.r.t Recorded Behaviour: Example

Assumption: Between two events signifying the entrance of a car C in a car park P1 and later the entrance of the same car in a car park P2 there must be an event signifying the departure of C from P1

Happens(rc:SS:Enter(oID1),t1,$\Re$(t1,t1)) $\wedge$
Initiates(rc:SS:Enter(oID1), equalTo(v1, vID), t1) $\wedge$
Initiates(rc:SS:Enter(oID1), equalTo(p1, pID1), t1) $\wedge$
Happens(rc:SS:Enter(oID2),t2,$\Re$(t1+tu,t2)) $\wedge$
Initiates(rc:SS:Enter(oID2), equalTo(v2, vID), t2) $\Rightarrow$
($\exists$t3:Time) **Happens**(rc:SS:Depart(oID3),t3,$\Re$(t1+tu,t2−tu)) $\wedge$
Initiates(rc:SS:Depart(oID3), equalTo(v3, vID), t3) $\wedge$
Initiates(rc:SS:Depart(oID3), equalTo(p3, pID1), t3)

Recorded events:

Happens(rc:SS:Enter(op1),1,$\Re$(1,1))
Initiates(rc:SS:Enter(op1), equalTo(v1,veh1),1)
Initiates(rc:SS:Enter(op1), equalTo(p1,loc1),1)
Happens(rc:SS:Enter(op2),27,$\Re$(27,27))
Initiates(rc:SS:Enter(op2), equalTo(v1,veh1),27)
Initiates(rc:SS:Enter(op2), equalTo(p1,loc3),27)

entail the negation of

*(no **depart** event for **veh1** from T=2 to T=26)*

Inconsistency of Expected Behaviour

A behavioural property or assumption of the form $f: C \Rightarrow A$ is inconsistent with the *expected behaviour* of a system *S* at time *T* if and only if:

$$\{E_R(T), E_U(dep(f),T), EC_A\} \models_{nf} \neg f$$

where

- $E_u(dep(f), T)$ is the set of events that may be generated by the formulas which *f* depends on (i.e., the formulas in *dep(f)*) at time T
- a formula $F: B \Rightarrow H$ belongs to $dep(f)$, if its head *H* has a literal *L* that unifies with: (i) some literal *K* in the body *C* of *f*, or (ii) some literal *K* in the body *B''* of another formula *F''* that belongs to $dep(f)$
- EC_A is the set of event calculus axioms

Inconsistency of Expected Behaviour: Example

Recorded events

Happens(rc:SS:Enter(op1),1,ℜ(1,1))
…
Happens(ic:UI:RelKey(op3, veh2),28, ℜ(28,28))
Happens(ir:UI:RelKey(op3), 29, ℜ(29,29))
Happens(rc:UI:CarRequest(op4),49, ℜ(49,49))
Initiates(rc:UI:CarRequest(op4),equalTo(p,loc2),49)
Happens(ic:IS:FindAvailable(op5,loc2),50, ℜ(50,50))
Happens(ir:IS:FindAvailable(op5), 51, ℜ(51,51))
Initiates(ir:IS:FindAvailable(op5), equalTo(Res,veh2),51)
Happens(re:UI:CarHire(op6,veh2,loc2), 52, R(52,52))
…
Happens(rc:UI:RetKey(op8),54,ℜ(54,54))
Initiates(rc:UI:RetKey(op8), equalTo(v, veh2), 54)

(A3)
Happens(ic:UI:RelKey(oID1,vID),t1, ℜ(t1, t1)) ∧
Happens(ir:UI:RelKey(oID1), t2, ℜ(t1, t2)) ∧
Happens(rc:UI:RetKey(oID2), t3, ℜ(t2, t3)) ∧
Initiates(rc:UI:RetKey(oID2), equalTo(v, vID), t3) ⇒
(∀t4:Time) ((t1 < t4)∧ (t4 < t3)
HoldsAt(equalTo(availability(vID), "unavailable"),t4)

HoldsAt(equalTo(availability(veh2), "unavailable"), 50)

entail the negation of

(A2) **Happens**(ic:IS:FindAvailable(oID,pID), t1, ℜ(t1,t1)) ∧
Happens(ir:IS:FindAvailable(oID), t2, ℜ(t1,t2)) ∧
HoldsAt(equalTo(availability(vID1),"unavailable"), t2−tu) ⇒
¬**Initiates** (ir:IS:FindAvailable (oID), equalTo(vID2, vID1), t2)

Unjustified Behaviour

A behavioural property of the form f: $\boldsymbol{C \Rightarrow A}$ is said to generate ***unjustified behaviour*** if and only if there is a literal ***e*** such that

$e \in E_R(T)$

e can be unified with A

$\{E_R(T) - \{e\}, f\} \models_{nf} e$

$\{E_R(T) - \{e\}, B_S - \{f\}\} \not\models_{nf} e$ and

there is a literal L in C for which,

$\{E_R(T), E_U(dep(f), T), EC_A)\} \models_{nf} \neg L$

where $\mathbf{B_S}$ is the set of the behavioural properties of a service based system

Unjustified Behaviour

A behavioural property of the form f: $\boldsymbol{C \Rightarrow A}$ is said to generate ***unjustified behaviour*** if and only if there is a literal ***e*** such that

$e \in E_R(T)$
e can be unified with A
$\{E_R(T) - \{e\}, f\} \models_{nf} e$

⇨ e can be generated f

$\{E_R(T) - \{e\}, B_S - \{f\}\} \not\models_{nf} e$ and
there is a literal L in C for which,
$\{E_R(T), E_U(dep(f), T), EC_A)\} \models_{nf} \neg L$

where $\mathbf{B_S}$ is the set of the behavioural properties of a service based system

Unjustified Behaviour

A behavioural property of the form f: $\boldsymbol{C \Rightarrow A}$ is said to generate ***unjustified behaviour*** if and only if there is a literal ***e*** such that

$e \in E_R(T)$
e can be unified with A
$\{E_R(T) - \{e\}, f\} \models_{nf} e$

⇒ e can be generated f

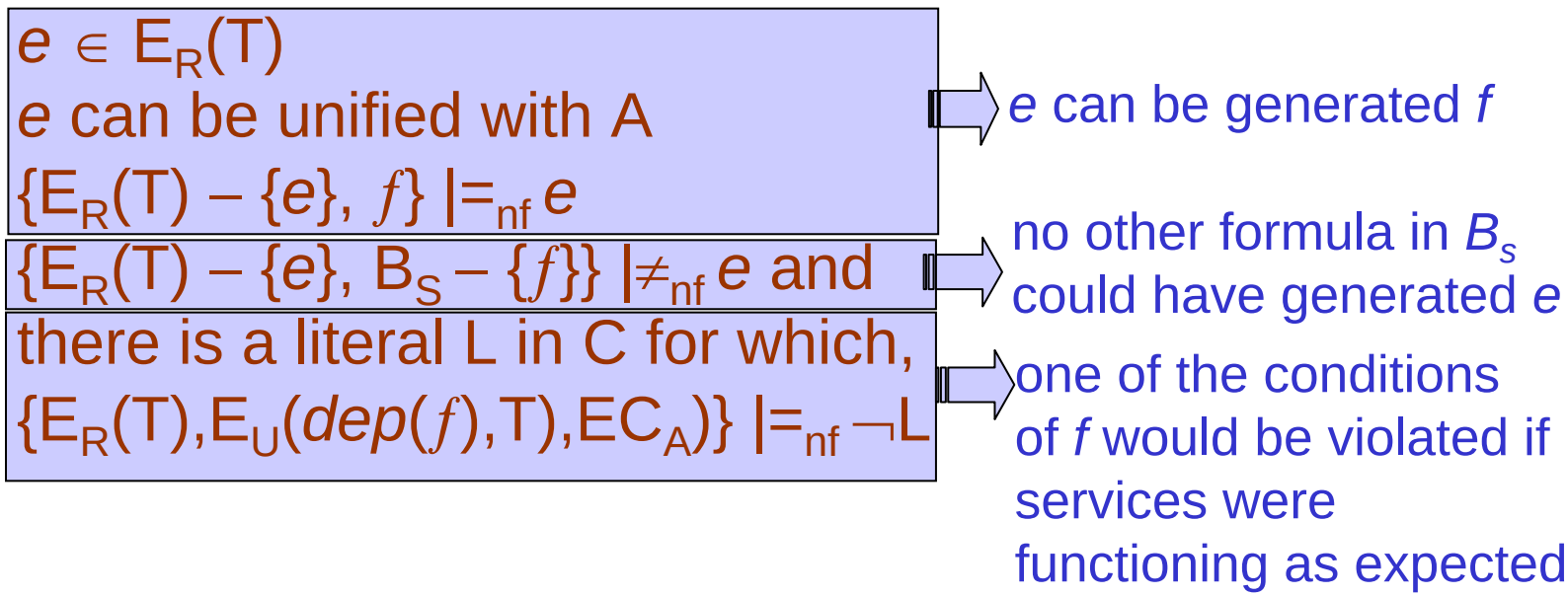
$\{E_R(T) - \{e\}, B_S - \{f\}\} \not\models_{nf} e$ and there is a literal L in C for which,
$\{E_R(T), E_U(dep(f), T), EC_A)\} \models_{nf} \neg L$

⇒ no other formula in B_s could have generated e

where $\mathbf{B_S}$ is the set of the behavioural properties of a service based system

Unjustified Behaviour

A behavioural property of the form f: $\boldsymbol{C \Rightarrow A}$ is said to generate ***unjustified behaviour*** if and only if there is a literal ***e*** such that



where $\mathbf{B_S}$ is the set of the behavioural properties of a service based system

Unjustified behaviour: an example

Recorded events

Happens(rc:SS:Enter(op1),1,ℜ(1,1))

…

Happens(ic:UI:RelKey(op3, veh2),28, ℜ(28,28))

Happens(ir:UI:RelKey(op3), 29, ℜ(29,29))

…

Happens(ic:IS:FindAvailable(op5,loc2),50, ℜ(50,50))

Happens(ir:IS:FindAvailable(op5), 51, ℜ(51,51))

Initiates(ir:IS:FindAvailable(op5), equalTo(Res,veh2),51)

Happens(re:UI:CarHire(op6,veh2,loc2), 52, ℜ(52,52))

Happens(rc:UI:RetKey(op8),54,ℜ(54,54))

Initiates(rc:UI:RetKey(op8), equalTo(v, veh2), 54)

(A3)

Happens(ic:UI:RelKey(oID1,vID),t1, ℜ(t1, t1)) ∧

Happens(ir:UI:RelKey(oID1), t2, ℜ(t1, t2)) ∧

Happens(rc:UI:RetKey(oID2), t3, ℜ(t2, t3)) ∧

Initiates(rc:UI:RetKey(oID2), equalTo(v, vID), t3) ⇒

(∀t4:Time) ((t1 < t4)∧ (t4 < t3)

HoldsAt(equalTo(availability(vID), "unavailable"),t4)

HoldsAt(equalTo(availability(veh2), "unavailable"), 50)

(A2) **Happens**(ic:IS:FindAvailable(oID,pID), t1, ℜ(t1,t1)) ∧

Happens(ir:IS:FindAvailable(oID), t2, ℜ(t1,t2)) ∧

HoldsAt(equalTo(availability(vID1),"unavailable"), t2−tu) ⇒

¬**Initiates** (ir:IS:FindAvailable (oID), equalTo(vID2, vID1), t2)

¬**Initiates**(ir:IS:FindAvailable(op5), equalTo(Res,veh2),51)

Unjustified behaviour: an example (cont'd)

Recorded events

Happens(rc:SS:Enter(op1),1,ℜ(1,1))
…
Happens(ic:UI:RelKey(op3, veh2),28, ℜ(28,28))
Happens(ir:UI:RelKey(op3), 29, ℜ(29,29))
…
Happens(ic:IS:FindAvailable(op5,loc2),50, ℜ(50,50))
Happens(ir:IS:FindAvailable(op5), 51, ℜ(51,51))
Initiates(ir:IS:FindAvailable(op5), equalTo(Res,veh2),51)
Happens(re:UI:CarHire(op6,veh2,loc2), 52, ℜ(52,52))
Happens(rc:UI:RetKey(op8),54,ℜ(54,54))
Initiates(rc:UI:RetKey(op8), equalTo(v, veh2), 54)

(A2, A3)

¬**Initiates**(ir:IS:FindAvailable(op5), equalTo(Res,veh2),51)

violates

(B2) **Happens**(rc:UI:CarRequest(oID1),t1,ℜ(t1,t1)) ∧
Initiates(rc:UI:CarRequest(oID1), equalTo(p,pID),t1) ∧
Happens(ic:IS:FindAvailable(oID2, pID),t2, ℜ(t1,t2)) ∧
Happens(ir:IS:FindAvailable(oID2),t3, ℜ(t2,t3)) ∧
Initiates(ir:IS:FindAvailable(oID2), equalTo(res,vID),t3)
⇒ (∃t4:Time) **Happens**(re:UI:CarHire(oID3,vID), t4, ℜ(t3,t3+tu))

Consistency Checking

- restricted quantification of formulas (fixed time boundaries for existentially quantified time vars)
- catching of events from the composition process execution engine using regular expressions
- past and future formula checks using a variant of an algorithm used for consistency checking in temporal deductive databases

Future work

- incorporation of abductive reasoning (to reason in the presence of incomplete information - cases that may have been missed)
- experimental evaluation (e.g. delays in detection?, usefulness of warnings
- diagnosis of the cause of deviations (requires detailed models of service behaviour)
- handling of deviations (e.g. replacement of continually malfunctioning services)
- integration of requirement monitors in run-time service discovery frameworks