



Behavioural Self-Adaptation of Services in Ubiquitous Computing Environments

Javier Cámara, Carlos Canal, and Gwen Salaün

Software Engineering Group

Department of Computer Science

Universidad de Málaga



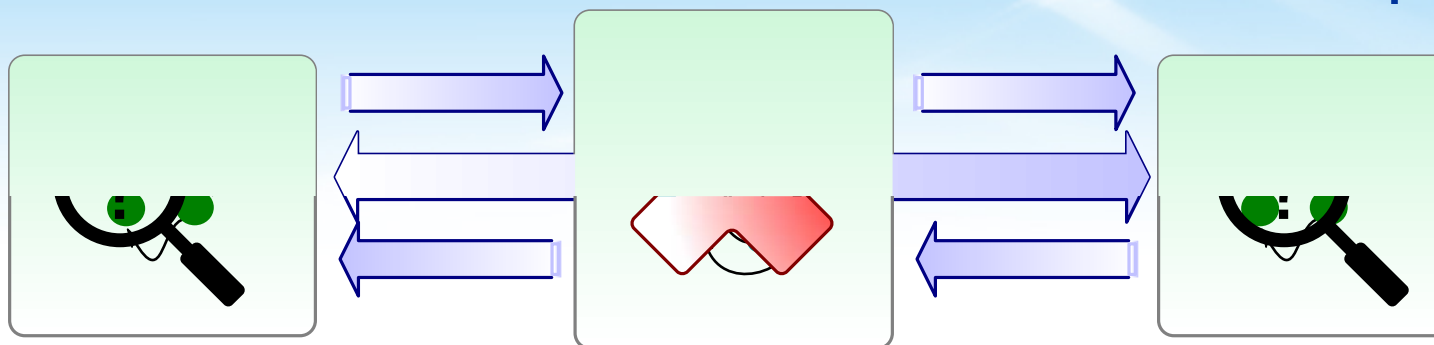
ICSE 2009 Workshop
Software Engineering for
Adaptive and Self-Managing Systems

- ▶ Problem Context.
 - ▶ Case Study: Airport Services.

- ▶ Description of the proposal.
 - ▶ Interface model.
 - ▶ Run-time execution engine.

- ▶ Conclusions and future work.

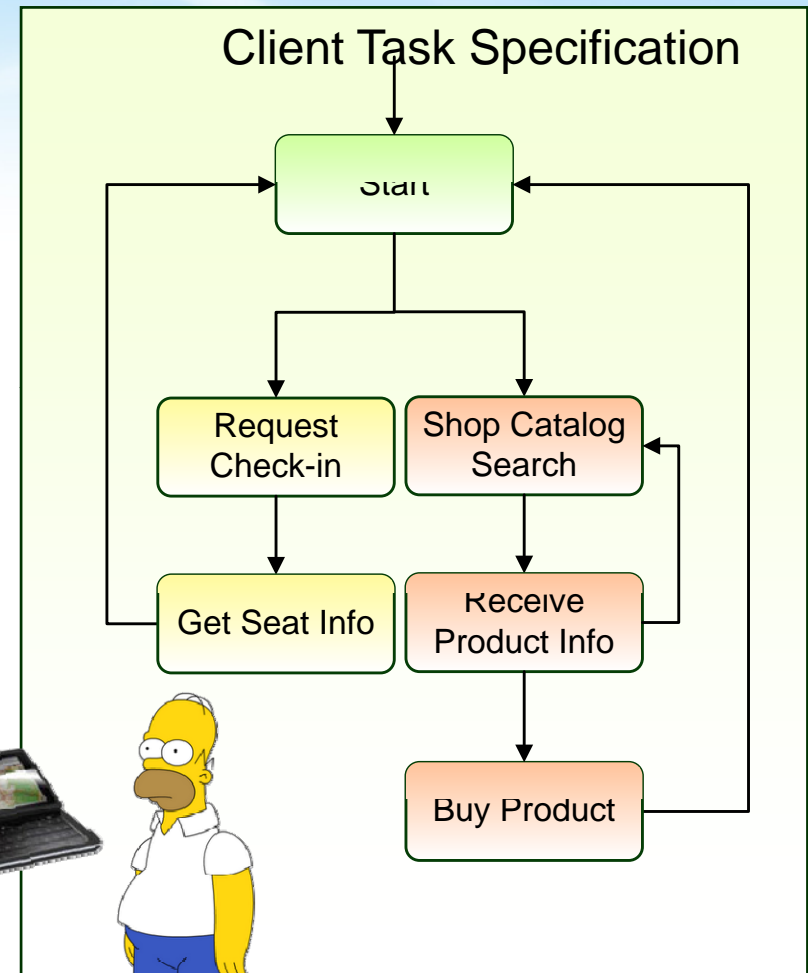
Problem Context: Service Behavioural Adaptation



Interoperability Problems:

- Signature
- Behavioural
- Service
- Semantic

Case Study: Airport Services

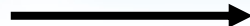


Case Study: Airport Services

Check-in Kiosk

Airline Information System

Duty-Free Shop



- ❏ Software services **not designed to interoperate** with each other.
- ❏ Concrete **scenarios cannot be envisioned *a priori***.
 - Services may join/dissapear.
 - Lack of a predefined architectural description.
 - Property verification at run-time.
- ❏ **Relationships** among services **are short-lived**.
 - Computational **cost**.

- Tackle the aforementioned problems by providing:
 - **Service interface model** to enable the derivation of an architectural description of the system.
 - **Run-time composition and adaptation engine.**
 - Deadlock-freedom.
 - User-defined properties.

Interfaces consist of:

- ❑ Operation signatures
- ❑ Protocol Description (Symbolic Transition Systems)
- ❑ Correspondences between Operations (Vectors)
- ❑ Composition constraints (LTL Formulae)

□ An STS is a tuple
 (A, S, I, B, T) :

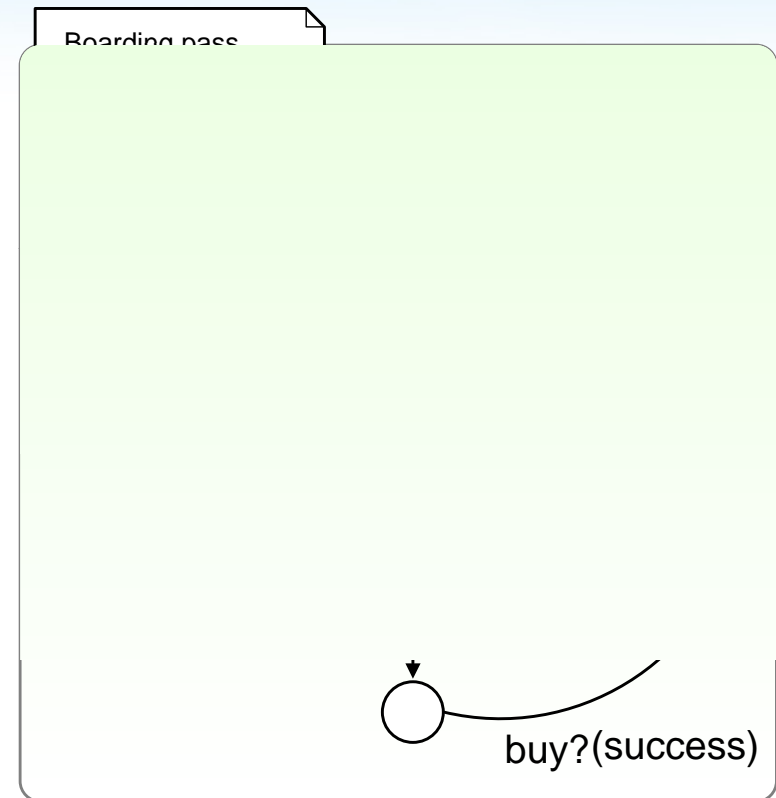
A : Alphabet set of communication actions $a=(M,D,PL)$

S: Set of states

I : Initial state

B: Set of *stable* states

T: Transition function $T: S \times A \rightarrow S$



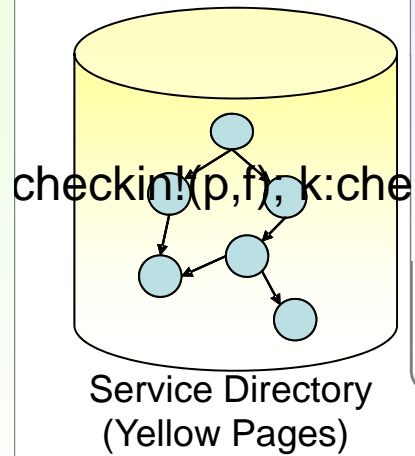
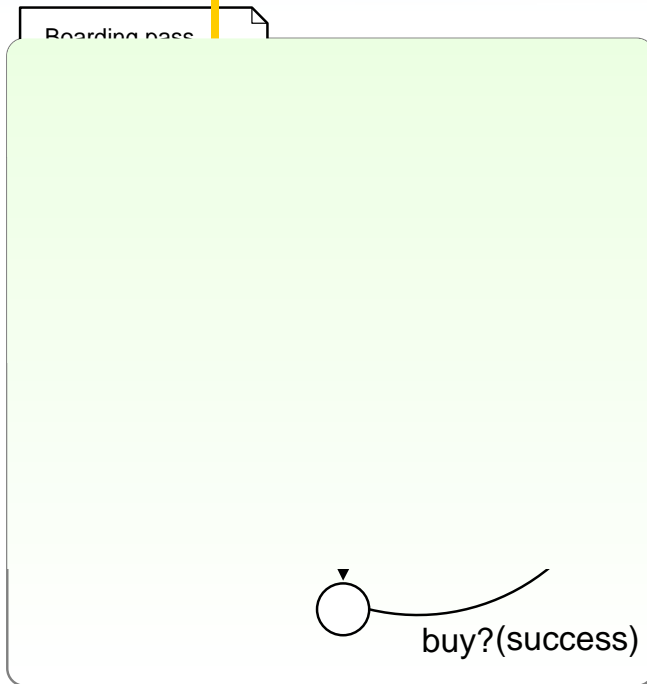


G I S U M

Vectors

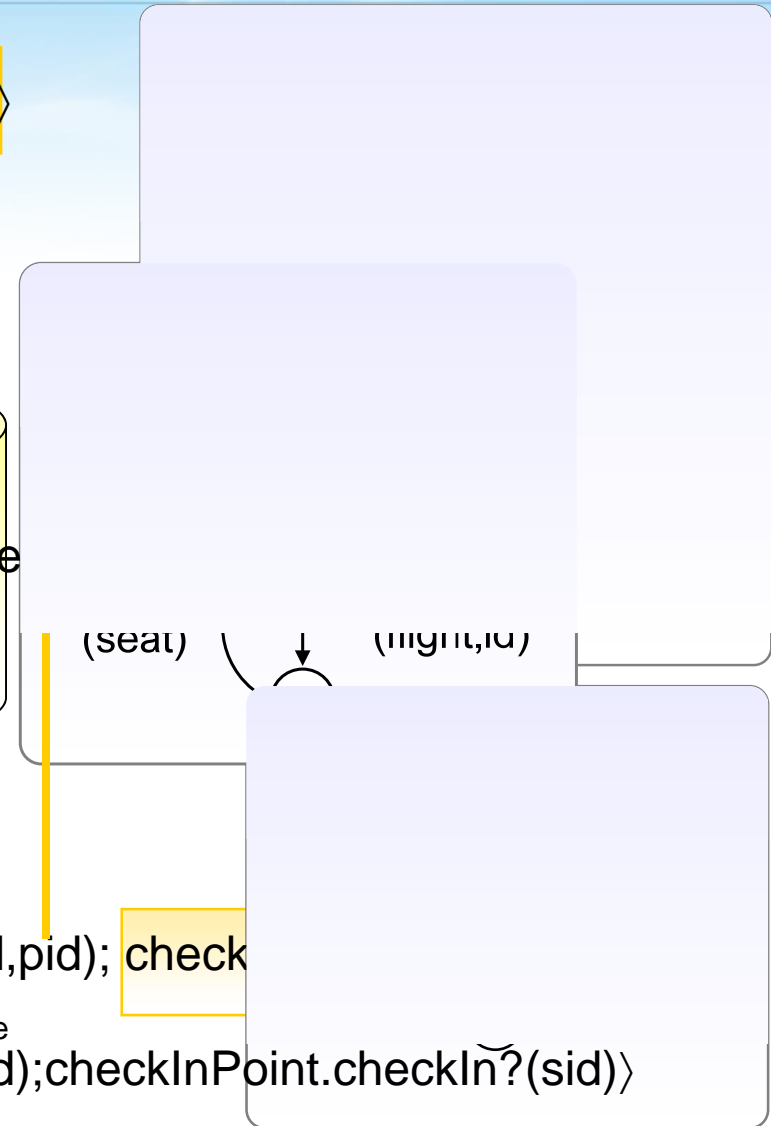
V_{checkin}
 $\langle \text{checkin}!(\text{pid}, \text{fid}); \text{checkInPoint.checkIn}?(\text{pid}, \text{fid}) \rangle$

$V_{\text{checkinResponse}}$
 $\langle \text{checkin}?(\text{sid}); \text{checkInPoint.checkIn}!(\text{sid}) \rangle$



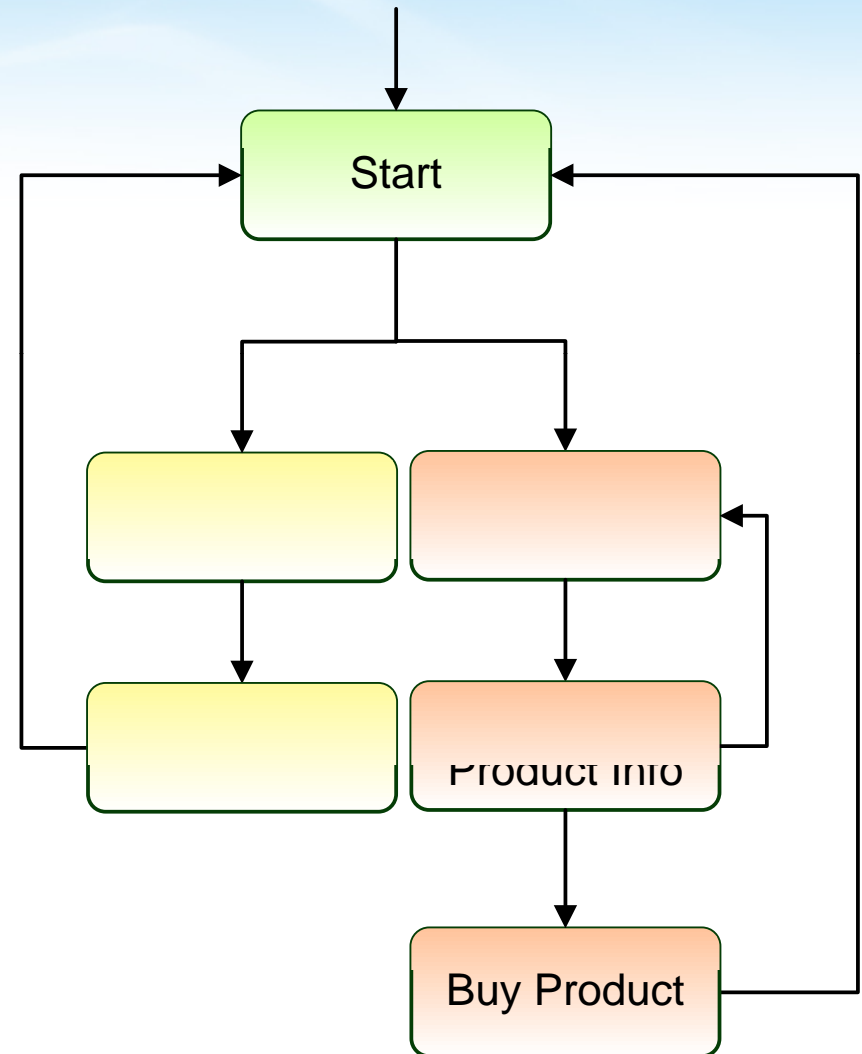
$V_{\text{checkinRequest}}$
 $\langle \text{checkin}?(\text{fid}, \text{pid}); \text{check}$

$V_{\text{checkinResponse}}$
 $\langle \text{checkin}!(\text{sid}); \text{checkInPoint.checkIn}?(\text{sid}) \rangle$

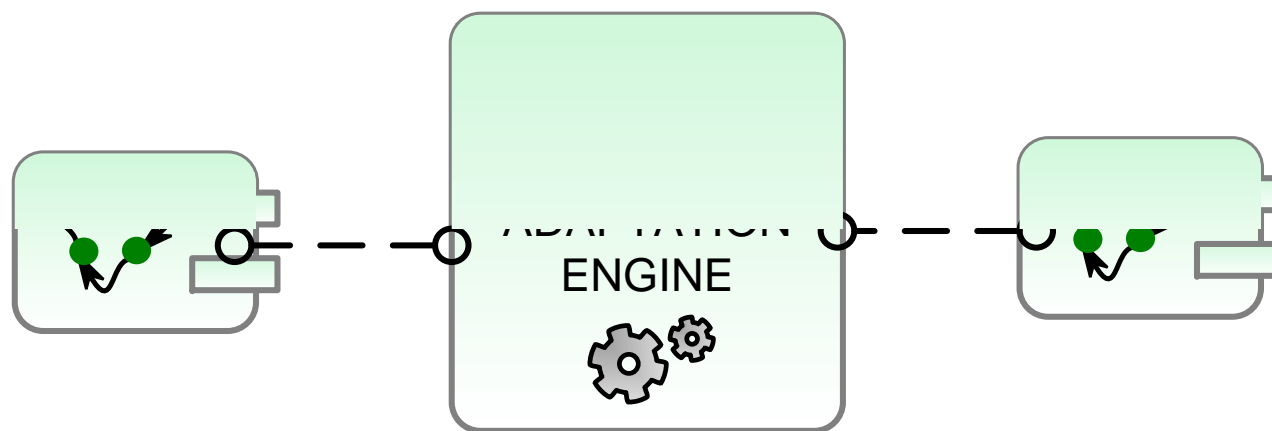


- ▣ Specification of user-defined safety and liveness properties.
- ▣ Expressed in LTL-X:
 - ▣ Finite trace semantics.
 - ▣ Atomic propositions correspond to vector executions.
- ▣ Example:

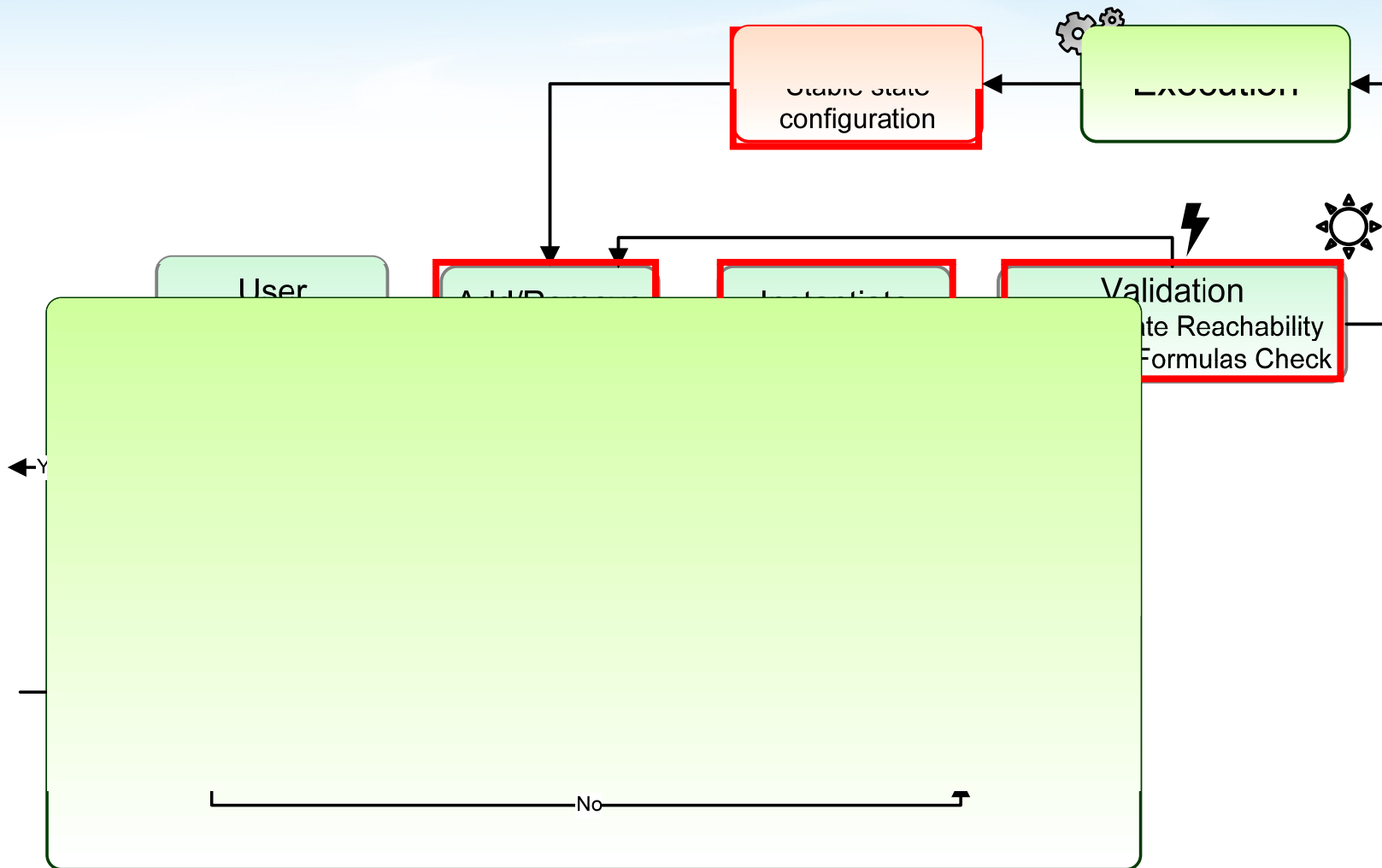
$$\mathbf{G}(v_{\text{checkin}} \rightarrow \mathbf{!F}(v_{\text{checkin}}))$$



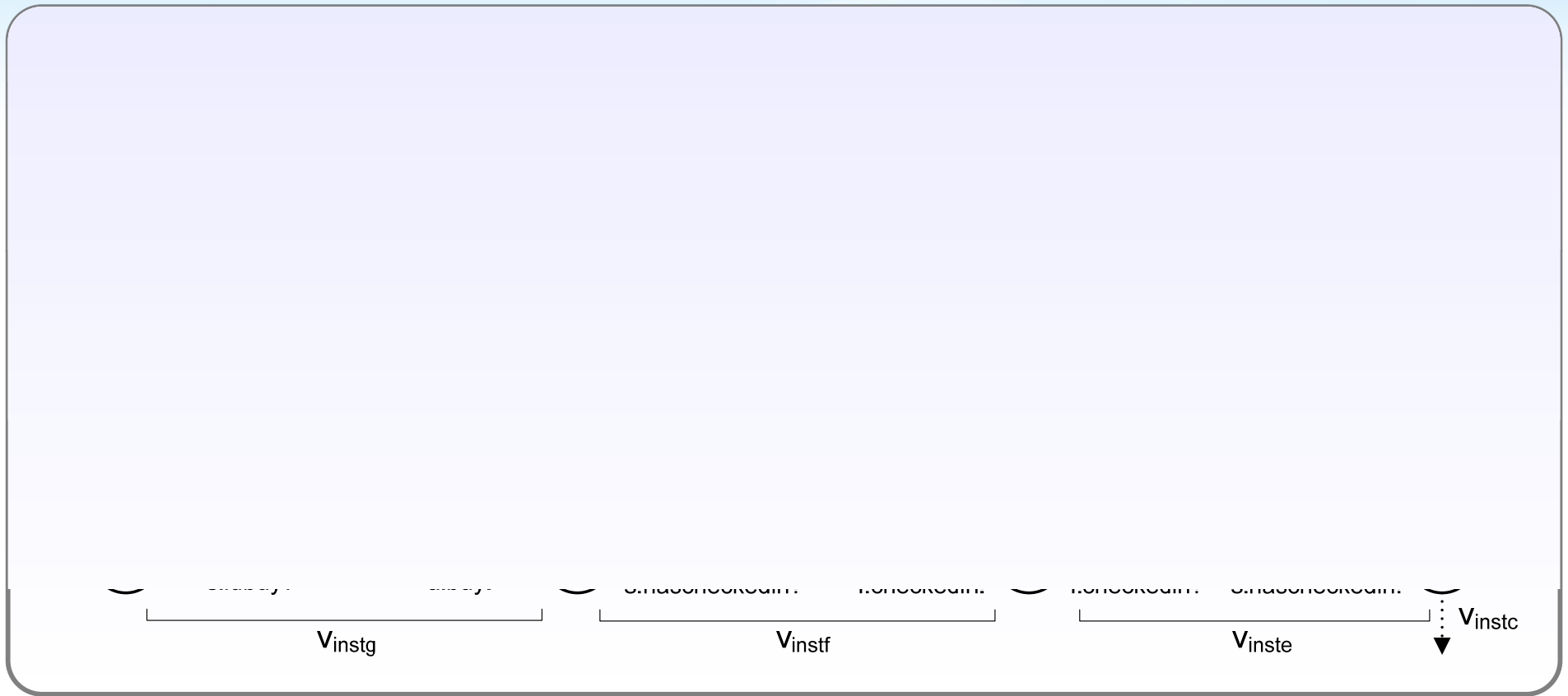
- ▶ Acts as an “interpreter” of the composition.
- ▶ Composition specification automatically derived at run-time.



Composition Process Overview



Case study - Sample execution trace



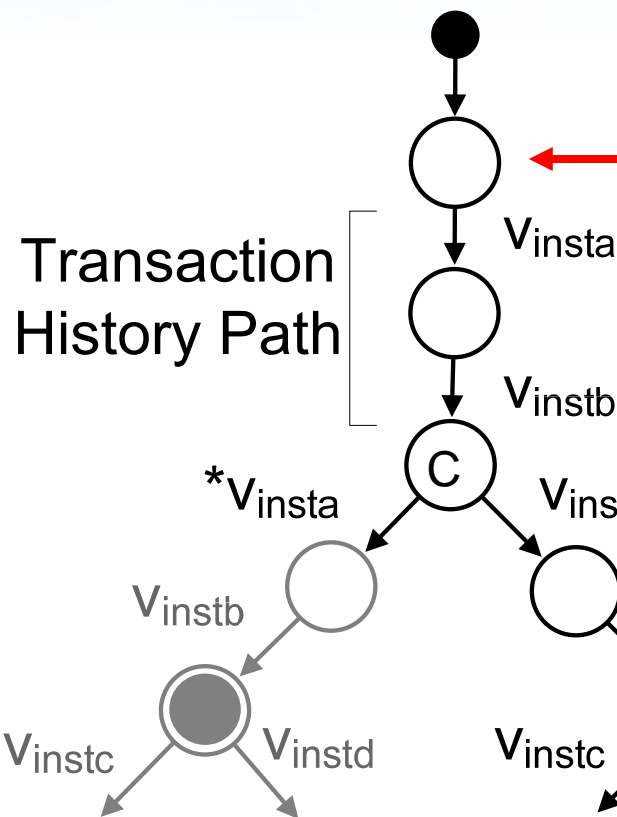
- ▶ Run potential execution traces through automata generated for the formulae:
 - ▶ **Fp**
 - ▶ Build the automaton for **Fp**.
 - ▶ At least one trace to a global stable state must satisfy the property.
 - ▶ **Gp = !F!p**
 - ▶ Build the automaton for **F!p**.
 - ▶ At least one trace to a global stable state must not violate the property.

Enforcing user-defined composition constraints

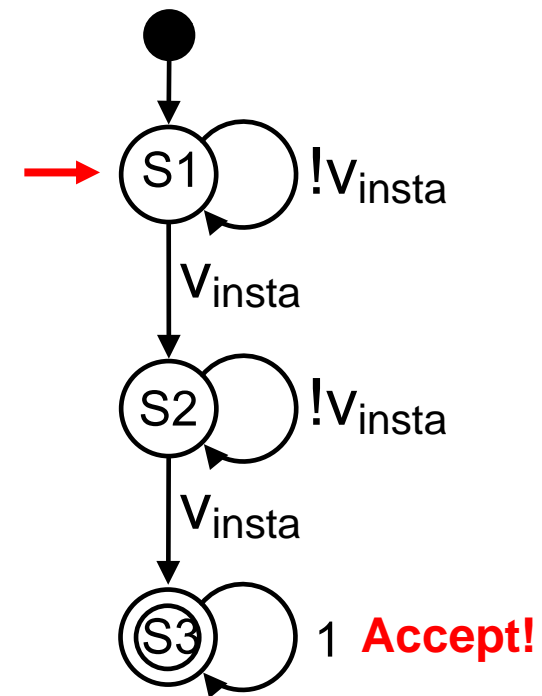
Constraint: $\mathbf{G}(v_{\text{checkin}} \rightarrow \mathbf{!F}v_{\text{checkin}})$

Automaton: $\mathbf{F!}(v_{\text{checkin}} \rightarrow \mathbf{!F}v_{\text{checkin}})$

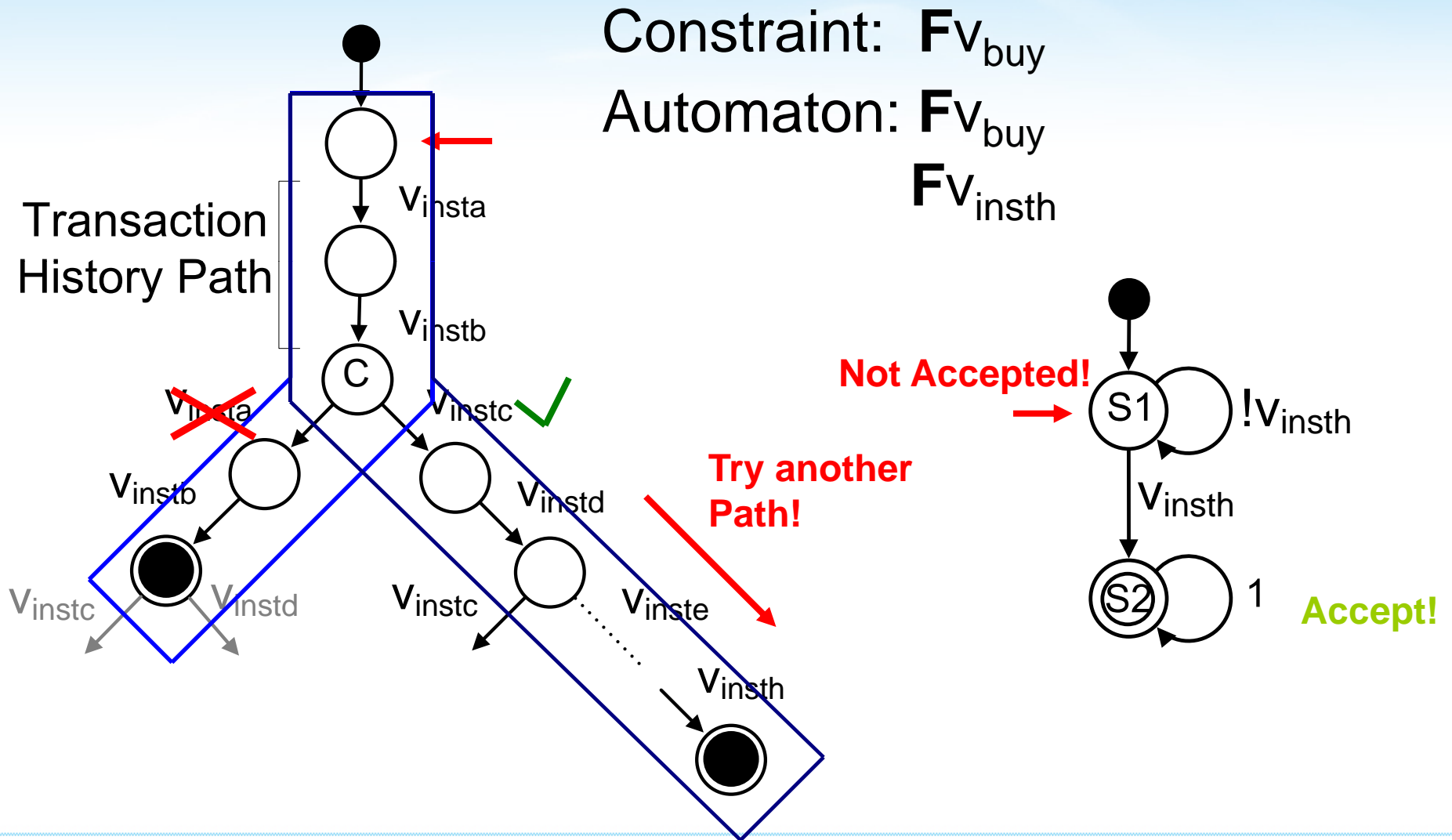
$\mathbf{F!}(v_{\text{insta}} \rightarrow \mathbf{!F}v_{\text{insta}})$



Try another Path!



Enforcing user-defined composition constraints



- Formal model and runtime execution for unanticipated behavioural adaptation of services in ubiquitous computing environments.
- Enforces user-defined constraints on the composition.
- Engine prototype integrated in ITACA (Demo at ICSE! – Thursday).

Future Work

- Exception handling mechanism (services may disappear at an unstable state).
- Centralized Engine imposes sequential processing.
- Implementation on JINI Services using AOP.

Thank you!