SCEL: Service Component Ensemble Language

Jaroslav Keznikl
keznikl@d3s.mff.cuni.cz
1. Distributed Self-Aware Systems

2. Emergent Behavior & Communication
1. Specification

2. Property Verification

3. Compliance Checking
Basic Abstractions Overview

Knowledge (repository)
- Linda-based storage
- items = **tuples** of records
- add/retrieve/withdraw operations (+ pattern matching)
- **arbitrary** data

Aggregation
- component = \( C \equiv I[K, \Pi, P] \)
- interface = set of attributes in **knowledge** (sometimes including values)
- component vs. system vs. ensemble
Behavior

- process-level = regular process algebra
- system-level = process-level + policies

Policies

- restrictions on interaction and knowledge access
- cornerstone of system-level semantics
LogDemo Example
Basic LogDemo Example

Synchronous message transmission rather than procedure call

Tester

log(msg)

Logger

Synchronized::log("Hello World")

Logger::println("HelloWorld")

Tester::Sleep(10)
**LogDemo Components**

\[
S \quad ::= \quad C_{Test} \parallel C_{Logger}
\]

\[
C_{Logger} \quad ::= \quad \mathcal{I}_{Logger}[\mathcal{K}_{Logger}, \Pi, P_{Logger}]
\]

\[
C_{Test} \quad ::= \quad \mathcal{I}_{Test}[\mathcal{K}_{Test}, \Pi, P_{Test}]
\]

\[
K_{Logger} \quad ::= \quad \langle \text{“attr”}, \text{“id”}, \text{“logger”} \rangle
\parallel \langle \text{“attr”}, \text{“ensemble”}, \text{“false”} \rangle
\parallel \langle \text{“attr”}, \text{“membership”}, \text{“true”} \rangle
\parallel \langle \text{“pfun”}, \text{“log”}, \text{“in”} \rangle
\]

\[
K_{Test} \quad ::= \quad \langle \text{“attr”}, \text{“id”}, \text{“tester”} \rangle
\parallel \langle \text{“attr”}, \text{“ensemble”}, \mathcal{I}.active = \text{yes} \land \mathcal{I}.battery\_level \geq 30\% \rangle
\parallel \langle \text{“attr”}, \text{“membership”}, \text{“false”} \rangle
\]

**Attributes of** $I_{Logger}$

**Attributes of** $I_{Test}$

**Knowledge-based communication**
What about **system** semantics?

Back to the theory...
System semantics

based on

Process Semantics

+ Policies
SCEL System Semantics

System Semantics = **unlabeled** transition system

states = parallel composition of individual component states

System LTS with all transitions labeled with $\tau$

- System labels
- transitions derived from
- $\tau$ produced by
- Employ Policies
- Application of **production rules**

- Process LTSs
- applied on
- Temporary

- process transitions + non-$\tau$ system transitions
Production Rules

include

1. Translation of process transitions to system transitions
   (process tr.)

2. Local knowledge manipulation
   (system tr.)

3. Remote (inter-component) knowledge manipulation
   (system tr.)

4. Independent asynchronous system evolution
   (system tr.)
Production rules
vs.
Process transitions

Translation of process transitions to system transitions
Production Rules vs. Process Transitions

**System semantics (LTS) → System labels**

\[ \lambda ::= \tau \mid \mathcal{I} : \text{new}(\mathcal{J}, \mathcal{K}, \Pi, P) \mid \mathcal{I} \diamond \]

- \[ \mathcal{I} : t \triangleleft c \]
- \[ \mathcal{I} : t \triangleright c \]
- \[ \mathcal{I} : t \triangleleft \mathcal{J} \]
- \[ \mathcal{I} : t \triangleright \mathcal{J} \]

Determined by **POLICY** (interaction predicate)

**Process semantics (LTS) → Process labels**

\[ \text{get}(T)@c \mid \text{qry}(T)@c \mid \text{put}(t)@c \mid \text{exec}(P) \mid \text{new}(\mathcal{I}, \mathcal{K}, \Pi, P) \]

\[ \alpha, \beta ::= a \mid \circ \mid \alpha[\beta] \]
Tester Example

Policy (interaction predicate):
\[ \Pi ::= \Pi + \Pi, I_{Tester} : \text{put}("invoke", "log", "Hello world!")@logger > I_{Tester} : \langle "invoke", "log", "Hello world!" \rangle \triangleright logger, [] \]

Process LTS:

System LTS:
Production rules vs. System transitions

1. Local knowledge manipulation
2. Remote (inter-component) knowledge manipulation
Local Knowledge Manipulation
(read/withdraw/put)

τ transition to the state representing the component I after the knowledge has been manipulated

I intends to manipulate its local knowledge

I’s policy accepts such manipulation

Determined by POLICY (authorization predicate)

Remote Knowledge Manipulation (read/withdraw/put)

τ transition to the state representing both systems (S1, S2) after the knowledge has been manipulated

I from S1 intends to manipulate knowledge of J from S2

J from S2 accepts such manipulation

J is coordinator of I (or vice versa) or I', coordinator of both, accepts interaction of I and J

Determined by POLICY (AP)

\[ S_1 \parallel S_2 \xrightarrow{\tau} S'_1 \parallel S'_2 \]

\[ S_1 \xrightarrow{I.t < n} S'_1 \]

\[ J.id = n \]

\[ S_2 \xrightarrow{I.t \geq J} S'_2 \]

\[ \text{ens}(I, J) \]

or

\[ I \in I' \land J \in I' \]

\[ \Pi, I' \vdash I \diamond J \]
Logger Example

Policy
Interaction predicate:
\[ \Pi \quad ::= \quad \Pi \oplus \]
\[ \Pi, I_{\text{Logger}} : \text{get}(\text{"invoke"}, \text{"log"}, \!\!msg)@self > \]
\[ I_{\text{Logger}} : \langle \text{"invoke"}, \text{"log"}, \!\!msg \rangle < \text{logger}, [msg \mapsto \text{"Hello world!"}] \]

Authorization predicate:
\[ \Pi, I_{\text{Logger}} \vdash I_{\text{Tester}} : t \overset{\upt{\Pi}}{\rightarrow} I_{\text{Logger}}, \ t = \langle \text{"invoke"}, \text{"log"}, msg \rangle \]
\[ \Pi, I_{\text{Logger}} \vdash I_{\text{Logger}} : t \overset{\upt{\Pi}}{\leftarrow} I_{\text{Logger}}, \ \forall t \]

System LTS:
Remote access (Tester ➔ Logger)

Local access (Logger)
What about

**Ensemble-Wide broadcast?**
new target super

qry@super enabled if qry@self enabled

new production rules for transitions with super
What is

Our Role?
Lessons from visit in Florence

- Rocco DeNicola, Rosario Pugliese, Michele Loreti
- Looking for
  - Suggestions for improvement
  - Case studies
  - General Feedback
  - Keen to cooperate
- Willing to come over to our place
Role of Component Group

Enhancing SCEL by:
- **Component Architectures**
  - Multiple component interfaces
  - Component composition
- **Dynamic Architectures**
- **Case study**
  - Robotic traffic playground

Component model
- based on SCEL (GACR w. Brno)

Emergent connections
- Connectors + Alloy
1. System semantics derived from process semantics (LTS).
2. Policies
   - restrictions on interaction and knowledge access (connectors)
     = authorization predicate
   - interpretation of process (i.e., component) behavior with respect to
     the whole system = interaction predicate
3. Still open to further improvements
Thank You

Questions?

ASCENS wiki ➔ Admin wiki ➔ Deliverables ➔ D1.1