SCEL: Service-Component Ensemble Language

Rosario Pugliese

Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze

Awareness Virtual Lecture Series (AVLS) — Autumn 2011

Work developed within the EU project ASCENS (257414)
Outline

1 Motivations
2 SCEL: Design Principles
3 SCEL: Syntax
4 SCEL: Operational Semantics (sketch)
5 Ongoing & Future Work
Ensembles

Future generation of software-intensive systems featuring

- massive numbers of components
- complex interactions among components, and with humans and other systems
- operating in open and non-deterministic environments
- dynamically adapting to new requirements, technologies and environmental conditions

Definition introduced in the IST Coordinated Action InterLink [2007]
Challenges

- *Adaptation* has been proposed as a powerful mean for taming the ever-increasing complexity of today’s computer systems, networks and applications.

- Developing software for ensembles creates new difficulties related to e.g.
  - the large dimension of the systems
  - the need to adapt to changing environments and requirements
  - the emergent behaviours resulting from complex interactions
  - the uncertainty during design-time and run-time

- Formalisms, linguistic constructs and programming tools should be designed featuring high level of autonomous and adaptive system behavior.

- Rigorous and sound concepts should be devised to reason and prove system properties.
Autonomic Service-Component Ensembles (ASCENS)

- Researching ways to build ensembles that combine
  - *traditional software engineering approaches*
  - techniques from *autonomic, adaptive, knowledge-based and self-aware systems*
  - the assurance about functional and non-functional properties provided by *formal methods*

---

Motivations

SCEL: Service-Component Ensemble Language

© Rosario Pugliese
Service Components (SCs)

- The notions of service components and service-component ensembles are a means to dynamically structure independent and distributed system entities.

- SCs are autonomic entities that can cooperate, with different roles, in open and non-deterministic environments.
  - These basic properties, already satisfied by, e.g., current service-oriented architectures, are enriched with awareness.

- Awareness is achieved by
  - equipping SCs with information about their own state and behavior
  - enabling SCs to collect and store information about their working environment
  - allowing SCs to use this information for redirecting and adapting their behavior

- Awareness-rich behavior makes SCs adaptable, connectable and composable.
Service-Component Ensemble (SCE)

A set of service components (and possibly simpler SCEs)
- with dedicated *knowledge units*, to represent shared, local and global knowledge
- interconnected via highly dynamic *infrastructures*
- featuring goal-oriented, safe and secure execution, and efficient resource management
To develop SCEs whose properties go beyond the state of the art in current software engineering, we are investigating different issues:

- **Languages** for modelling and programming SCEs
- **Foundational models** for adaptation, dynamic self-expression and reconfiguration
- **Formal methods** for the design and verification of reliable SCs and SCEs
- **Software infrastructures** supporting deployment and execution of SCE-based applications
- ...
Our aim

- To model the behavior of service components and of their ensembles, their interactions, their sensitivity and adaptivity to the environment

- We are designing SCEL, a specific language with
  - programming abstractions for
    - directly representing Knowledge, Behaviors and Aggregations according to specific Policies
    - naturally programming interaction, adaptation and self- and context-awareness
  - a small set of basic constructs with solid semantic grounds
    - so that logics, tools and methodologies can be developed for formal reasoning on systems behavior
    - in order to establish qualitative and quantitative properties of both the individual components and the ensembles
SCEL: programming abstractions

The Service-Component Ensemble Language (SCEL) provides primitives and constructs for describing the following programming abstractions:

- **Knowledge**: describe how data, information and knowledge is managed
- **Behaviours**: describe how systems of components progress
- **Aggregations**: describe how different entities are brought together to form *components, systems* and, possibly, *ensembles*
- **Policies**: deal with the way properties of computations are represented and enforced
Our Starting Points (1980 — …)

- Process Algebras
  - CCS, CSP, ACP, …

- Calculi and Languages for Mobility
  - Pi-calculus, Obliq, Ambients, …

- Tuple-space-based Coordination Languages
  - Linda, KLAIM, MARS, …
SCEL: Syntax (at once)

**Systems:** \[ S ::= C \mid S_1 \parallel S_2 \mid (\nu n)S \]

**Components:** \[ C ::= \mathcal{I}[K, \Pi, P] \]

**Knowledge:** \[ K ::= \langle t \rangle \mid K_1 \parallel K_2 \]

**Processes:** \[ P ::= \text{nil} \mid a.P \mid P_1 + P_2 \mid P_1[P_2] \mid X \mid A(\bar{p}) \quad (A(\bar{f}) \triangleq P) \]

**Actions:** \[ a ::= \text{get}(T)@c \mid \text{rtv}(T)@c \mid \text{put}(t)@c \mid \text{exec}(P) \mid \text{new}(\mathcal{I}, K, \Pi, P) \]

**Targets:** \[ c ::= n \mid x \mid \text{self} \]

**Items:** \[ t ::= \ldots \]

**Templates:** \[ T ::= \ldots \]
Knowledge

SCEL is *parametric* wrt the means of managing knowledge as we don’t want to take a definite stand about

- **Knowledge representation**
  
  Tuples, Records, Clauses, Constraints, . . .

- Corresponding *knowledge handling* mechanisms
  
  Pattern-matching, Reactive TS, Queries, Logic programming, Concurrent constraint programming, . . .

They may depend on the application domain
Knowledge

\[ K ::= \langle t \rangle \mid K_1 \parallel K_2 \]

- Knowledge repositories are (multi)sets of items \( \langle t \rangle \)
  - **Application data**: data used for the progress of the computation
  - **Control data**: data providing information about
    - the environment in which a component is running (e.g. monitored data from sensors)
    - the current status of a component (e.g. its position or its battery level)

- The knowledge handling mechanism provides three operations
  - for *adding* information to a repository
  - for *retrieving* information from a repository
  - for *withdrawing* information from a repository

which can be triggered by (the abstractions relative to) behaviours
Knowledge

\[
K ::= \langle t \rangle \mid K_1 \parallel K_2
\]

- Knowledge repositories are (multi)sets of items \( \langle t \rangle \)
  - \textit{Application data}: data used for the progress of the computation
  - \textit{Control data}: data providing information about
    - the environment in which a component is running (e.g. monitored data from sensors)
    - the current status of a component (e.g. its position or its battery level)

- The knowledge handling mechanism provides three operations
  - for \textit{adding} information to a repository
  - for \textit{retrieving} information from a repository
  - for \textit{withdrawing} information from a repository

  which can be triggered by (the abstractions relative to) behaviours
Behaviors

Describe how systems of components progress

- Are modeled as processes in the style of process calculi
- *Interaction* is rendered by allowing processes to access to knowledge repositories, possibly of other components
- *Adaptation* is modeled by retrieving from the knowledge repositories both information about the changing environment and the component state and (suggestions about) the code to execute for reacting to these changes
Processes

\[ P ::= \text{nil} \mid a.P \mid P_1 + P_2 \mid P_1[P_2] \mid X \mid A(\bar{p}) \quad (A(\bar{f}) \triangleq P) \]

Processes: the SCEL active computational units

Processes are built up from the inert process \text{nil} via

- \textit{action prefixing}: \( a.P \),
- \textit{nondeterministic choice}: \( P_1 + P_2 \),
- \textit{controlled composition}: \( P_1[P_2] \),
- \textit{process variable}: \( X \),
- \textit{parameterised process invocation}: \( A(\bar{p}) \)
- \textit{parameterised process definition}: \( A(\bar{f}) \triangleq P \)
Processes

\[ P ::= \text{nil} \mid a.P \mid P_1 + P_2 \mid P_1[P_2] \mid X \mid A(\overline{p}) \ (A(\overline{f}) \triangleq P) \]

Processes: the SCEL active computational units

Controlled composition \( P_1[P_2] \) can be seen as a generalisation of the various forms of “parallel composition of \( P_1 \) and \( P_2 \)” commonly used in process calculi.
Actions

\[ a ::= \text{get}(T)_c \mid \text{rtv}(T)_c \mid \text{put}(t)_c \mid \text{exec}(P) \mid \text{new}(I, K, \Pi, P) \]

Actions: processes can perform five different kinds of actions

- Actions \text{get}(T)_c, \text{rtv}(T)_c and \text{put}(t)_c manage knowledge repositories by withdrawing/retrieving/adding information items \( t \) from/to the knowledge repository \( c \)
  - Exploit templates \( T \) as patterns to select knowledge items
  - Trigger execution of the corresponding operation provided by the knowledge handling mechanism of the target repository \( c \)
- Action \text{exec}(P) triggers a controlled execution of process \( P \)
- Action \text{new}(I, K, \Pi, P) creates a new component \( I[K, \Pi, P] \)
Aggregations

Describe how different entities are brought together to form components, systems and, possibly, ensembles

- Model resource allocation and distribution
- Reflect the idea of administrative domains, i.e. resources and computations of a given entity are under the control of a specific authority

Compositionality and interoperability are supported by component interfaces that specify attributes and functionalities provided by components
Components

\[ C ::= I[\mathcal{K}, \Pi, P] \]

- An interface \( I \) containing information about the component itself
- A knowledge manager \( \mathcal{K} \) providing local (and possibly part of the global) knowledge (i.e. control data) in addition to the application data, together with a specific handling mechanism
- A set of policies \( \Pi \) regulating the interaction among the different internal parts of the component and its interaction with the other components
- A process term \( P \)
  - Some of the processes composing \( P \) perform the local computation
  - Some other ones may coordinate processes interaction with the knowledge repository and/or deal with the issues related to adaptation and reconfiguration
Components

\[ C ::= \mathcal{I}[\mathcal{K}, \Pi, P] \]

- An *interface* \( \mathcal{I} \) containing information about the component itself
- A *knowledge manager* \( \mathcal{K} \) providing local (and possibly part of the global) knowledge (i.e. control data) in addition to the application data, together with a specific handling mechanism
- A set of *policies* \( \Pi \) regulating the interaction among the different internal parts of the component and its interaction with the other components
- A *process* term \( P \)
  - Some of the processes composing \( P \) perform the local computation
  - Some other ones may coordinate processes interaction with the knowledge repository and/or deal with the issues related to adaptation and reconfiguration
Components

\[ C ::= \mathcal{I}[\mathcal{K}, \Pi, P] \]

- An interface \( \mathcal{I} \) containing information about the component itself
- A knowledge manager \( \mathcal{K} \) providing local (and possibly part of the global) knowledge (i.e. control data) in addition to the application data, together with a specific handling mechanism
- A set of policies \( \Pi \) regulating the interaction among the different internal parts of the component and its interaction with the other components
- A process term \( P \)
  - Some of the processes composing \( P \) perform the local computation
  - Some other ones may coordinate processes interaction with the knowledge repository and/or deal with the issues related to adaptation and reconfiguration
### Components

\[
C ::= \mathcal{I}[\mathcal{K}, \Pi, P]
\]

#### Components

- An *interface* \( \mathcal{I} \) containing information about the component itself
- A *knowledge manager* \( \mathcal{K} \) providing local (and possibly part of the global) knowledge (i.e. control data) in addition to the application data, together with a specific handling mechanism
- A set of *policies* \( \Pi \) regulating the interaction among the different internal parts of the component and its interaction with the other components
- A *process* term \( P \)
  - Some of the processes composing \( P \) perform the local computation
  - Some other ones may coordinate processes interaction with the knowledge repository and/or deal with the issues related to adaptation and reconfiguration
Interfaces

Contain attributes and functionalities provided by a component

**Attribute: a name to which a values is associated**

- Attribute values can be dynamically changed
- Necessary attributes of any component $C$ are
  - $id$: the name of the component $C$
  - $ensemble$: a predicate on interfaces used to determine the actual components of the ensemble created and coordinated by $C$
  - $membership$: a predicate on the interfaces used to determine the ensembles which $C$ is willing to be member of

**Functionality: a behaviour made available for external invocation**
Systems

\[ S ::= C \mid S_1 \parallel S_2 \mid (\nu n)S \]

Systems: aggregations of components where some names are restricted

- Single component \( C \)
- Parallel composition \( \_ \parallel \_ \)
- Name restriction \( \nu n \_ \) (to delimit the scope of name \( n \))
  - In a system of the form \( S_1 \parallel (\nu n)S_2 \), the effect of the operator is to make name \( n \) invisible from within \( S_1 \)
Ensembles

- No specific syntactic construct for making ensembles
- They are dynamically formed by exploiting attributes, e.g. *ensemble* and *membership*, to put requirements on the other partners
- The design choice of having ‘synthesized’ ensembles dynamically determined
  - supports an high dynamicity and flexibility in forming, joining and disjoining ensembles
  - permits avoiding to structure ensembles through rigid syntactic constructs
  - affects the communication abilities of components (and of the processes therein)
Ensembles

- No specific syntactic construct for making ensembles
- They are dynamically formed by exploiting attributes, e.g. *ensemble* and *membership*, to put requirements on the other partners
- The design choice of having ‘synthesized’ ensembles dynamically determined
  - supports an high dynamicity and flexibility in forming, joining and disjoining ensembles
  - permits avoiding to structure ensembles through rigid syntactic constructs
  - affects the communication abilities of components (and of the processes therein)

**ensemble predicates**

- \( \mathcal{I}.id \in \{n, m, p\} \)
- \( \mathcal{I}.active = yes \land \mathcal{I}.battery\_level \geq 30\% \)
- \( range_{max} \geq \sqrt{(self.x - \mathcal{I}.x)^2 + (self.y - \mathcal{I}.y)^2} \)
Ensembles

- No specific syntactic construct for making ensembles
- They are dynamically formed by exploiting attributes, e.g. *ensemble* and *membership*, to put requirements on the other partners
- The design choice of having ‘synthesized’ ensembles dynamically determined
  - supports an high dynamicity and flexibility in forming, joining and disjoining ensembles
  - permits avoiding to structure ensembles through rigid syntactic constructs
  - affects the communication abilities of components (and of the processes therein)

**membership predicates**

- *true*
- *false*
- $I.trust\_level > medium$
Policies

Deal with the way properties of computations are represented and enforced

- Interaction: interaction predicates, ...
- Resource usage: accounting, leasing, ...
- Security: access control, trust, reputation, ...

SCEL is *parametric* wrt the actual language used to express *policies*
SCEL: Operational Semantics

- The operational semantics is given in the Structural Operational Semantics (SOS) style by relying on the notion of Labelled Transition System (LTS)

LTS: a triple \( \langle S, \mathcal{L}, \rightarrow \rangle \)

- A set of states \( S \)
- A set of transition labels \( \mathcal{L} \)
- A labelled transition relation \( \rightarrow \subseteq S \times \mathcal{L} \times S \) accounting for the actions that can be performed from each state and the new state reached after each such transition

- The semantics is defined in two steps
  - First, the semantics of processes specifies process commitments, i.e. the actions that processes can initially perform, ignoring process allocation, available data, regulating policies, …
  - Then, by taking process commitments and systems configuration into account, the semantics of systems provides a full description of systems behaviour
Operational Semantics of Processes

**Processes:**

\[ P ::= \text{nil} \mid a.P \mid P_1 + P_2 \mid P_1[P_2] \mid X \mid A(p) \mid (A(f) \triangleq P) \]

**Actions:**

\[ a ::= \text{get}(T)@c \mid \text{rtv}(T)@c \mid \text{put}(t)@c \mid \text{exec}(P) \mid \text{new}(I, K, \Pi, P) \]

**LTS rules (excerpt)**

\[
\begin{align*}
  a.P & \xrightarrow{a} P \quad (a \neq \text{exec}(P')) \\
  \text{exec}(P).P' & \xrightarrow{\text{exec}(P')} P'[P]
\end{align*}
\]

\[
\begin{align*}
  P & \xrightarrow{\alpha} P' \\
  Q & \xrightarrow{\beta} Q'
\end{align*}
\]

\[
\begin{align*}
  P[Q] & \xrightarrow{\alpha[\beta]} P'[Q']
\end{align*}
\]
Operational Semantics of Processes

Processes:
\[ P ::= \text{nil} \mid a.P \mid P_1 + P_2 \mid P_1[P_2] \mid X \mid A(\bar{p}) \quad (A(\bar{f}) \triangleq P) \]

Actions:
\[ a ::= \text{get}(T)@c \mid \text{rtv}(T)@c \mid \text{put}(t)@c \mid \text{exec}(P) \mid \text{new}(I, K, \Pi, P) \]

LTS rules (excerpt)

\[
\begin{align*}
\quad & a.P \xrightarrow{a} P \quad (a \neq \text{exec}(P')) & \quad & \text{exec}(P).P' \xrightarrow{\text{exec}(P)} P'[P] \\
\quad & P \xrightarrow{\alpha} P' & \quad & Q \xrightarrow{\beta} Q' \\
\quad & P[Q] \xrightarrow{\alpha[\beta]} P'[Q']
\end{align*}
\]

exec spawns a new process \( P \) whose execution can be controlled by the continuation \( P' \) of the process performing the action.
The semantics of $P[Q]$ at the level of processes is absolutely permissive and generates all possible combinations of the commitments of the involved processes; it is then refined at the level of systems to also take polices into account.
Operational Semantics of Systems: LTS rules (excerpt)

**SYSTEMS:** $S ::= I[K, \Pi, P] \mid S_1 \parallel S_2 \mid (\nu n)S$

From process actions to component actions

$$P \xrightarrow{\alpha} P' \quad \Pi, I : \alpha \succeq \lambda, \sigma$$

$$I[K, \Pi, P] \xrightarrow{\lambda} I[K, \Pi, P'\{\sigma\}]$$

Intra-component withdrawal

$$I[K, \Pi, P] \xrightarrow{I: t \triangleleft n} I[K, \Pi, P']$$

$$n = I{id} \quad \mathcal{K} \ominus t = \mathcal{K}' \quad \Pi, I \vdash I : t \triangleleft I$$

$$I[K, \Pi, P] \xrightarrow{\tau} I[K', \Pi, P']$$

Inter-component, intra-ensemble withdrawal

$$S_1 \xrightarrow{I: t \triangleleft n} S_1' \quad S_2 \xrightarrow{I: t \triangleleft J} S_2'$$

$$n = J.id \quad \text{ens}(I, J)$$

$$S_1 \parallel S_2 \xrightarrow{\tau} S_1' \parallel S_2'$$
We are assessing to which extent SCEL fits for modelling the behavior of service components and of their ensembles, their interactions, their sensitivity and adaptivity to the environment.

As testbeds we will use three case studies from different application domains:
- Robotics (collective transport),
- Cloud-computing (transiently available computers), and
- e-Mobility (cooperative e-vehicles).

This process might require tuning the language features.

We will *implement* SCEL, possibly by exploiting the distributed software framework IMC developed in previous EU projects (MIKADO & SENSORIA).

We will *develop* logics, tools and methodologies for formal reasoning on systems behavior, in order to establish qualitative and quantitative properties of both the individual components and the ensembles.
Many Thanks!