INSTITUTE
FOR ADVANCED
STUDIES
LUCCA

A formal approach to autonomic systems programming: The SCEL Language

Italian Conference on TCS
17-19 September 2014, Perugia, Italy

R. De Nicola - IMT Lucca

Presenting work done in collaboration with:
M. Loreti, R. Pugliese, F. Tiezzi

Contents

Ensembles and their Programming

Ensembles are software-intensive systems featuring

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

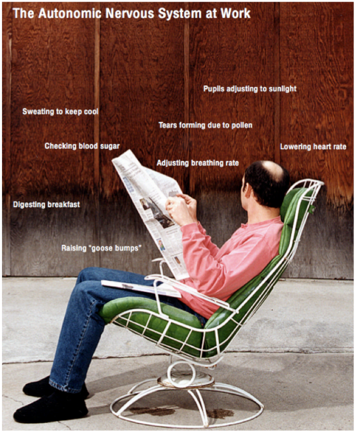
Challenges for software development for ensembles

- the dimension of the systems
- the need to adapt to changing environments and requirements
- the emergent behaviour resulting from complex interactions
- the uncertainty during design-time and run-time

The Autonomic Computing paradigm is in our view a possible approach to facing the challenges posed by ensembles

Autonomic Computing

To master the complexity of massively complex systems inspiration has come from the human body and its autonomic nervous system

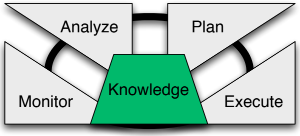


IBM vision

The IBM MAPE-K loop

Systems can manage themselves by continuously

- monitoring their behaviour (self-awareness) and their working environment (context-awareness)
- analysing the acquired knowledge to identify changes
- planning reconfigurations
- executing plan actions

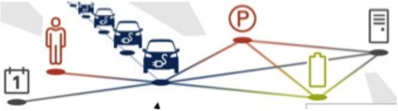


Autonomic Systems: examples

Robot swarms

Clouds

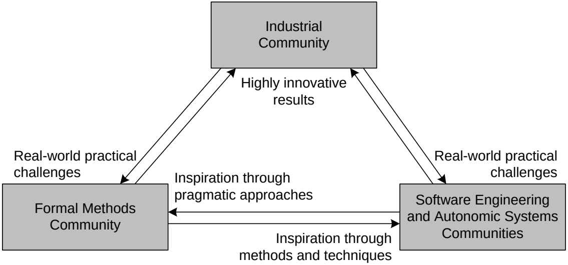
Cooperative e-vehicles



The ASCENS Projects

The ASCENS (Autonomic Service-Component Ensembles) project aims at finding ways to build ensembles that combine

- traditional software engineering approaches
- techniques from the areas of autonomic, adaptive, knowledge-based and self-aware systems
- formal methods to guarantee systems properties



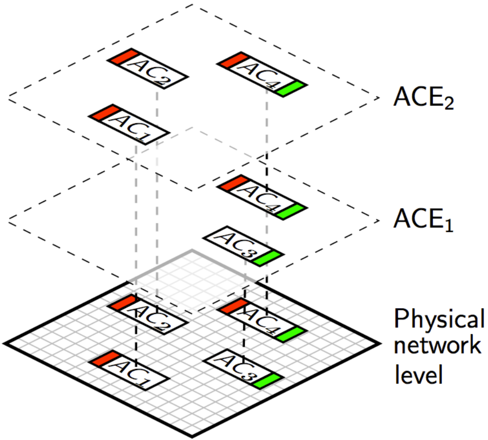
ascens autonomic service-component ensembles

AS as ensembles

Systems are structured as Autonomic Components (AC) dynamically forming interacting AC ensembles

- Autonomic Components have an interface exposing component attributes
- AC ensembles are not rigid networks but highly flexible structures where components linkages are dynamically established
- Interaction between ACs is based on attributes and predicates over AC attributes dynamically specify ACE as targets of communication actions

Ensemble Formation



Ensembles are determined by components attributes and by predicates validated by each component.

A formal approach to engineering AS

Basic ingredients of the approach:

1. Specification language
   - equipped with a formal semantics
   - the semantics associates mathematical models to language terms
2. Verification techniques
   - built on top of the models
   - logics used to express properties of interest for the considered application domain
3. Software support
   - runtime environment
   - programming framework
   - verification tools for (qualitative and quantitative) analysis

Our approach to engineering AS

Basic ingredients of the approach:

1. Specification language
   - SCEL - A Service Component Ensemble Language
2. Verification techniques
   - Model checking with Spin
   - Translation into BIP
   - Simulation and statistical model checking
3. Software support
   - jRESP - http://jresp.sourceforge.net/ - the runtime environment for the SCEL paradigm provides
     - an API permitting using SCEL constructs in Java programs
     - a simulation module permitting to simulate SCEL programs and collect relevant data for analysis

Importance of languages

Languages play a key role in the engineering of AS.

- Systems must be specified as naturally as possible
- distinctive aspects of the domain need to be first-class citizens to guarantee intuitive/concise specifications and avoid encodings
- high-level abstract models guarantee feasible analysis
- the analysis of results is based on system features, not on their low-level representation to better exploit feedbacks

The big challenge for language designers is to devise appropriate abstractions and linguistic primitives to deal with the specificities of the systems under consideration

A Language for Ensembles

We aim at at developing linguistic supports for modelling (and programming) the service components and their ensembles, their interactions, their sensitivity and adaptivity to the environment

SCEL

We aim at designing a specific language with

- programming abstractions necessary for
  - directly representing Knowledge, Behaviors and Aggregations according to specific Policies
  - naturally programming interaction, adaptation and self- and context- awareness
- linguistic primitives with solid semantic grounds
  - To develop logics, tools and methodologies for formal reasoning on systems behavior
  - to establish qualitative and quantitative properties of both the individual components and the ensembles

Key Notions

We need to enable programmers to model and describe the behavior of service components ensembles, their interactions, and their sensitivity and adaptivity to the environment.

Notions to model

1. The behaviors of components and their interactions
2. The topology of the network needed for interaction, taking into account resources, locations, visibility, reachability issues
3. The environment where components operate and resource-negotiation takes place, taking into account open ended-ness and adaptation
4. The global knowledge of the systems and of its components
5. The tasks to be accomplished, the properties to guarantee and the constraints to respect.

Programming abstractions for AS

The Service-Component Ensemble Language (SCEL) currently provides primitives and constructs for dealing with 4 programming abstractions.

1. Knowledge: to describe how data, information and (local and global) knowledge is managed
2. Behaviours: to describe how systems of components progress
3. Aggregations: to describe how different entities are brought together to form *components*, *systems* and, possibly, *ensembles*
4. Policies: to model and enforce the wanted evolutions of computations.

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

1. Knowledge

SCEL is *parametric* wrt the means of managing knowledge that would depend on the specific class of application domains.

Knowledge representation

- Tuples, Records
- Horn Clause Clauses,
- Concurrent Constraints,
- ...

Knowledge handling mechanisms

- Pattern-matching, Reactive Tuple Spaces
- Data Bases Querying
- Resolution
- Constraint Solving
- ...

1. Knowledge (and Adaptation)

Application and Control Data

No definite stand is taken about the kind of knowledge that might depend on the application domain. To guarantee adaptivity, we, however, require there be some specific components.

- Application data: used for the progress of the computation.
- Control data: which provide information about the environment in which a component is running (e.g. data from sensors) and about its current status (e.g. its position or its battery level).

Knowledge handling mechanisms

- Add information to a knowledge repository
- Retrieve information from a knowledge repository
- Withdraw information from a knowledge repository

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

2. Behaviors

Components behaviors are modeled as terms of process calculi

- Adaptation is obtained by retrieving from knowledge repositories
  - information about the changing environment and the component status
  - the code to execute for reacting to these changes - local adaptation.
- Interaction is obtained by allowing processes to access knowledge repositories, (also) of other components and is exploited to guarantee system adaptation

Processes

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

The operators have the expected semantics. $P_1[\,P_2\,]$ (Controlled Composition) can be seen as a generalization of "parallel compositions" of process calculi. For the meaning of $a.-$, see next.

2. Behaviours (and Actions)

Actions operate on knowledge repository c and use T as a pattern to select knowledge items:

- manage knowledge repositories by
  - withdrawing information - $\mathbf{get}(T)@c$,
  - retrieving information - $\mathbf{qry}(T)@c$
  - adding information - $\mathbf{put}(t)@c$
- create new names or new components $\mathcal{I}[\mathcal{K}, \Pi, P]$ - $\mathbf{new}(\mathcal{I}, \mathcal{K}, \Pi, P)$

Actions

$$a \ ::= \ \mathbf{get}(T)@c \mid \mathbf{qry}(T)@c \mid \mathbf{put}(t)@c \mid \mathbf{fresh}(n) \mid \mathbf{new}(\mathcal{I}, \mathcal{K}, \Pi, P)$$

Action Targets

$$c \ ::= \ n \mid x \mid \mathsf{self} \mid \mathit{ensemble}(?)$$

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

3. Aggregations

Aggregations describe how different entities are brought together to form *ensembles* and

- Model resource *allocation* and *distribution*
- Reflect the idea of *administrative domains*, i.e. the authority controlling a given set of resources and computing agents.
- are modelled by resorting to the notions of system, component and ensemble.

Systems

$$S \quad ::= \quad C \quad | \quad S_1 \parallel S_2 \quad | \quad (\nu n)S$$

- Single component C
- Parallel composition $\_ \parallel \_$
- Name restriction $\nu n\_$ (to delimit the scope of name n), thus in $S_1 \parallel (\nu n)S_2$, name n is invisible from within S_1

3. Aggregations (Components)

Components consist of:

- An interface $\mathcal{I}$ containing information about the component itself. In particular, each component C has attributes:
  - *id*: the name of the component C
- A knowledge manager $\mathcal{K}$ providing control data (i.e. the local and (part of the) global knowledge) and application data; together with a specific knowledge handling mechanism
- A set of policies Π regulating inter-component and intra-component interactions
- A process term P that performs the local computation, coordinates their interaction with the knowledge repository and deals with adaptation and reconfiguration

Components

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

4. Policies

Policies deal with the way properties of computations are represented and enforced

- Interaction policies: interaction predicates, for modeling interleaving, monitoring, ...
- Authorization policies: accounting, leasing, trust, reputation ...
- Policies for access control, resource usage, adaptation, ...

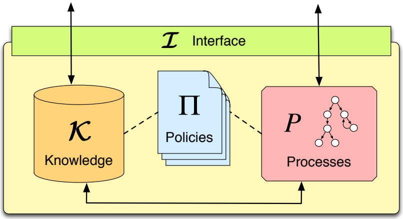
SCEL is *parametric* wrt the actual language used to express policies. Currently we use FACPL.

- simple and unambiguous syntax (declarative style)
- industry basis (OASIS standard XACML)
- formal semantics
- Java implementation (http://rap.dsi.unifi.it/facpl/)

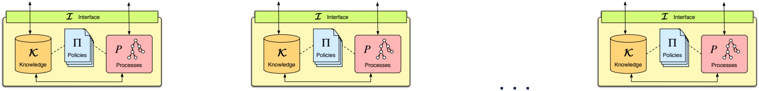
Components and Systems

Aggregations describe how different entities are brought togheter and controlled:

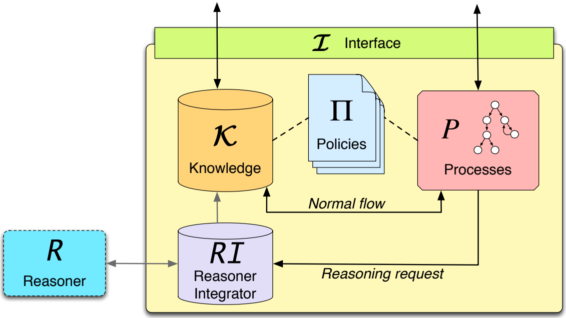
- Components:



- Systems:



A reasoning SCEL component



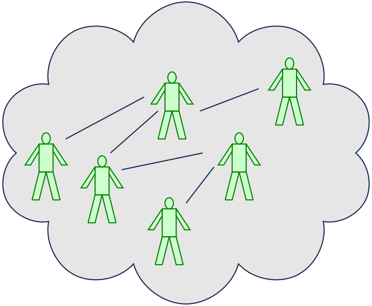
Providing Reasoning Capabilities

SCEL programs to take decisions may resort to external reasoners that can have a fuller view of the environment in which single components are operating.

SCEL: Syntax (in one slide)

SYSTEMS:	S	$::=$	$C \mid S_1 \parallel S_2 \mid (\nu n)S$
COMPONENTS:	C	$::=$	$\mathcal{I}[\mathcal{K}, \Pi, P]$
KNOWLEDGE:	K	$::=$	... currently, just tuple spaces
POLICIES:	Π	$::=$	... currently, interaction and FACPL policies
PROCESSES:	P	$::=$	$\mathbf{nil} \mid a.P \mid P_1 + P_2 \mid P_1[P_2] \mid X \mid A(\bar{p}) \quad (A(\bar{f}) \triangleq P)$
ACTIONS:	a	$::=$	$\mathbf{get}(T)@c \mid \mathbf{qry}(T)@c \mid \mathbf{put}(t)@c \mid \mathbf{fresh}(n) \mid \mathbf{new}(\mathcal{I}, \mathcal{K}, \Pi, P)$
TARGETS:	c	$::=$	$n \mid x \mid \mathsf{self} \mid \mathcal{P}$
ITEMS:	t	$::=$	... currently, tuples
TEMPLATES:	T	$::=$	... currently, tuples with variables

An ensemble



Where are ensembles in SCEL?

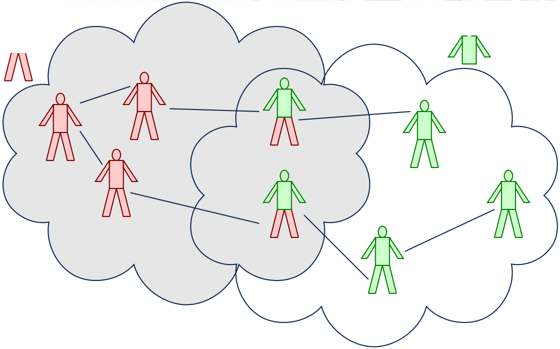
- SCEL syntax does not have specific syntactic constructs for building ensembles.
- Components Interfaces specify (possibly dynamic) attributes (features) and functionalities (services provided).
- Predicate-based communication tests attributes to select the communication targets among those enjoying specific properties.

Communication targets are predicates!!

$$\textsc{Targets:} \quad c \quad ::= \quad n \mid x \mid \mathsf{self} \mid P$$

By sending to, or retrieving and getting from predicate P one components interacts with all the components that satisfy the same predicate.

Predicate-based ensembles



- Ensembles are determined by the predicates validated by each component.
- There is no coordinator, hence no bottleneck or critical point of failure
- A component might be part of more than one ensemble

Example Predicates

- $id \in \{n, m, p\}$
- $active = \text{yes} \wedge battery\_level > 30\%$
- $\text{range}_{\text{max}} > \sqrt{(this.x - x)^2 + (this.y - y)^2}$
- true
- $trust\_level > \text{medium}$
- $\dots$
- $trousers = \text{red}$
- $shirt = \text{green}$

Alternative characterization of ensembles

Apart for using predicates as targets of interaction actions (send, retrieve and get) to identify those components that form an ensemble and guarantee general communication between members of the same ensemble we have experimented with two additional alternatives:

- Adding a specific syntactic category for ensembles that would define static ensembles
- Enriching interfaces of components with special attributes, ensemble and membership, to single out groups of components forming an ensemble; each ensemble would then have an initiator but would be more dynamic.

Static ensembles

Adding a specific syntactic category

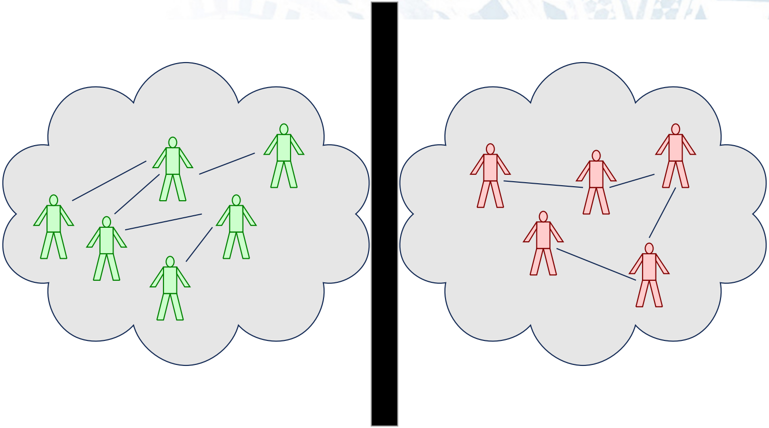
We explicitly declare the component that represents an ensemble, and whenever the target of an operation contains the name e of an ensemble it will impact on all its components.

$$\textsc{Ensembles:}\quad \begin{array}{rcl} E & ::= & e[S] \\ S & ::= & E \mid C \mid S_1 \| S_2 \end{array}$$

Ensembles may have a hierarchical structure

This is the approach taken in process algebras with explicit localities or in programming language with distributed tuple space (e.g. Klaim).

Static ensembles



Drawback

- The structure of the aggregated components is static, defined once and for all.
- a component can be part of just one ensemble.

Dynamic ensembles

Ensembles are dynamically formed by exploiting components interfaces and distinguished attributes

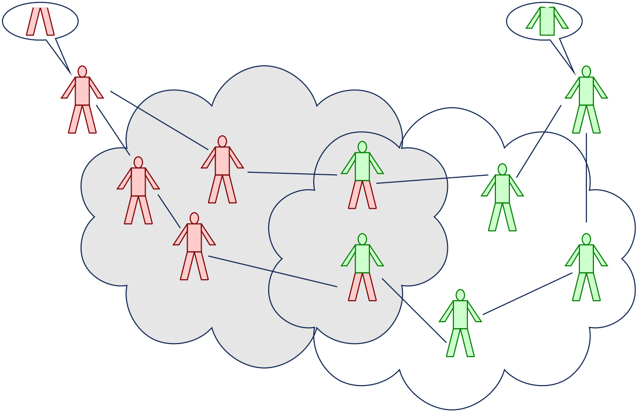
- *ensemble*: a predicate on interfaces used to determine the actual components of the ensemble created and coordinated by C, e.g. $id \in \{n, m, p\}$ or true.
- *membership*: a predicate on the interfaces used to determine the ensembles which C is willing to be member of, e.g. $trust\_level > \mathrm{medium}$ or false.

Allowing ensemble as targets

By sending to, or retrieving and getting from super one components interacts with all the components of the same ensemble it is in.

$$\textsc{Targets:} \quad c \quad ::= \quad n \mid x \mid \mathsf{self} \mid \mathsf{super}$$

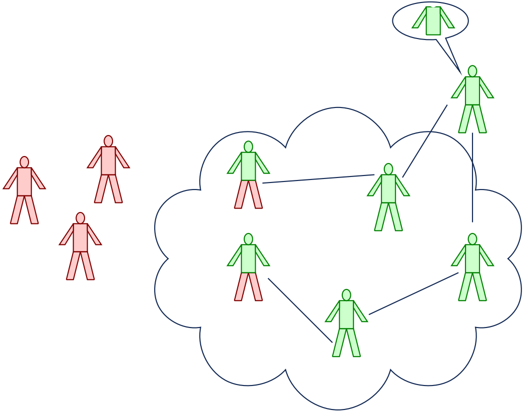
Dynamic ensemble



Drawback

An ensemble dissolves if its coordinator disappears: single point of failure.

Dynamic ensemble



Drawback

An ensemble dissolves if its coordinator disappears: single point of failure.

SCEL: Operational Semantics

Structural operational semantics relies on the notion of Labelled Transition System (LTS)

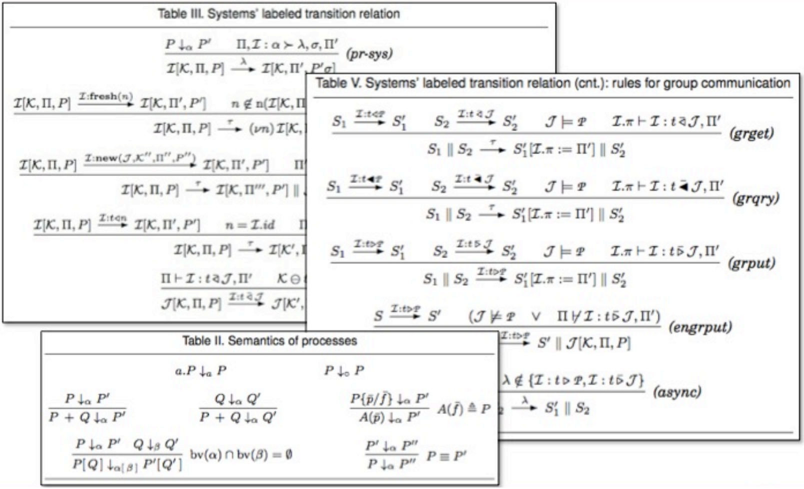
LTS: a triple $\langle \mathcal{S}, \mathcal{L}, \rightarrow \rangle$

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

Semantics is structured in two layers:

1. Processes semantics specifies process commitments, i.e. the actions that processes can initially perform, while ignoring process allocation, available data, regulating policies, ...
2. Systems semantics, builds on process commitments and systems configuration to provide a full description of systems behavior.

Operational Semantics: A flavour



Semantics of Processes

Rules for Processes (excerpt)

$$a.P \downarrow_a P \qquad\qquad P \downarrow_\circ P$$

$$\frac{P \downarrow_\alpha P' \qquad Q \downarrow_\beta Q'}{P[\,Q\,] \downarrow_{\alpha[\,\beta\,]} P'[\,Q'\,]}$$

- $a.P$ executes action a and then behaves like process P
- $\downarrow_\circ$ indicates that process P may always decide to stay idle
- The semantics of $P[\,Q\,]$ at process level is very permissive and generates all combinations of the commitments of the involved processes; its behaviour is refined at systems level when policies enter the game.

SOS Rules for Systems (excerpt)

From process actions to component actions

$$\frac{P \downarrow_{\alpha} P' \qquad \Pi, \mathcal{I} : \alpha \succ \lambda, \sigma, \Pi'}{\mathcal{I}[\mathcal{K}, \Pi, P] \xrightarrow{\lambda} \mathcal{I}[\mathcal{K}, \Pi', P'\sigma]}$$

SOS Rules for Systems (excerpt)

From process actions to component actions

$$\frac{P \downarrow_{\alpha} P' \qquad \Pi, \mathcal{I} : \alpha \succ \lambda, \sigma, \Pi'}{\mathcal{I}[\mathcal{K}, \Pi, P] \xrightarrow{\lambda} \mathcal{I}[\mathcal{K}, \Pi', P'\sigma]}$$

Interaction Predicates: Action Transformation

$$\frac{\mathcal{E}[\![\, T \,]\!]\mathcal{I} = T' \qquad \mathcal{N}[\![\, c \,]\!]_{\mathcal{I}} = c' \qquad match(T', t) = \sigma}{\Pi_{\oplus}, \mathcal{I} : \mathbf{get}(T)@c \succ \mathcal{I} : t \triangleleft c', \sigma, \Pi_{\oplus}}$$

Interaction Predicates: Actions interleaving

$$\frac{\Pi_{\oplus}, \mathcal{I} : \alpha \succ \lambda, \sigma, \Pi_{\oplus}}{\Pi_{\oplus}, \mathcal{I} : \alpha[\circ] \succ \lambda, \sigma, \Pi_{\oplus}} \qquad \frac{\Pi_{\oplus}, \mathcal{I} : \alpha \succ \lambda, \sigma, \Pi_{\oplus}}{\Pi_{\oplus}, \mathcal{I} : \circ[\alpha] \succ \lambda, \sigma, \Pi_{\oplus}}$$

SOS Rules for Systems (excerpt)

Intra-component withdrawal

$$\frac{\mathcal{I}[\mathcal{K},\Pi,P] \xrightarrow{\mathcal{I}:t\lhd n} \mathcal{I}[\mathcal{K},\Pi',P'] \quad n=\mathcal{I}.id \quad \mathcal{K}\ominus t=\mathcal{K}' \quad \Pi'\vdash\mathcal{I}:t\,\bar{\lhd}\,\mathcal{I},\Pi''}{\mathcal{I}[\mathcal{K},\Pi,P] \xrightarrow{\tau} \mathcal{I}[\mathcal{K}',\Pi'',P']}$$

Component n

- reads (and removes) from its knowledge repository
- after checking that tuple t is present and removes it
- asks the authorization to perform the action

SOS Rules for Systems (excerpt)

Inter-component, point-to-point withdrawal

$$\frac{S_1 \xrightarrow{\mathcal{I}:t \lhd n} S_1' \qquad S_2 \xrightarrow{\mathcal{I}:t \,\bar{\lhd}\, \mathcal{J}} S_2' \qquad \mathcal{J}.id = n \qquad \mathcal{I}.\pi \vdash \mathcal{I} : t \,\bar{\lhd}\, \mathcal{J}, \Pi'}{S_1 \parallel S_2 \xrightarrow{\tau} S_1'[\mathcal{I}.\pi := \Pi'] \parallel S_2'}$$

Component $\mathcal{I}$.id

- reads tuple t from the knowledge repository of $\mathcal{J}.\mathrm{id} = \mathrm{n}$
- after checking that component n is willing to provide it
- and after checking that it has the appropriate authorizations

More SOS Rules for Systems

Inter-component, group-oriented withdrawal

$$\frac{S_1 \xrightarrow{\mathcal{I}:t \triangleleft \mathrm{P}} S_1' \qquad S_2 \xrightarrow{\mathcal{I}:t \,\bar{\triangleleft}\, \mathcal{J}} S_2' \qquad \boxed{\mathcal{J} \models \mathrm{P}} \qquad \mathcal{I}.\pi \vdash \mathcal{I} : t \,\bar{\triangleleft}\, \mathcal{J}, \Pi'}{S_1 \parallel S_2 \xrightarrow{\tau} S_1'[\mathcal{I}.\pi := \Pi'] \parallel S_2'}$$

Component $\mathcal{I}$.id

- reads tuple t from the knowledge repository of a component $\mathcal{J}$ satisfying predicate $\boxed{P}$.
- after checking that component n is willing to provide it
- and after checking that it has the appropriate authorisations

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

Robotics scenario in SCEL

Robot Swarms

Robots of a swarm have to reach different target zones according to their assigned tasks (help other robots, reach a safe area, clear a minefield, etc.)

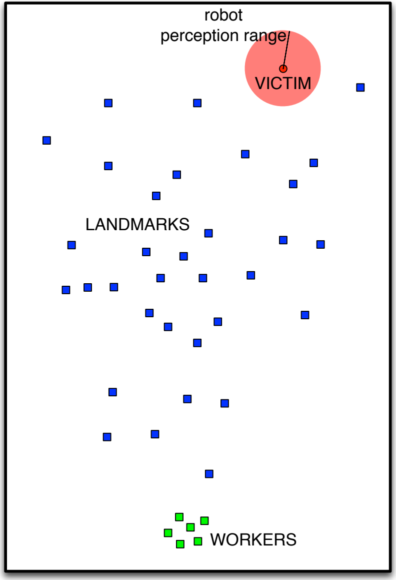
Robots:

- have limited battery lifetime
- can discover target locations
- can inform other robots about their location

The behaviour of each robot is implemented as $AM[ME]$ where the autonomic manager AM controls the execution of the managed element ME. A general scenario can be expressed in SCEL as a system:

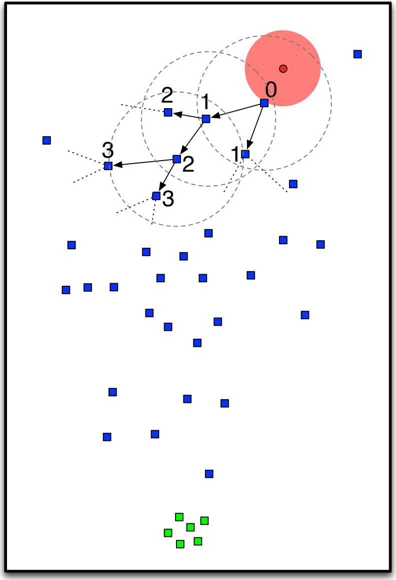
$$\mathcal{I}[\mathcal{K}_i, \Pi_i, P_i] \parallel \mathcal{J}[\mathcal{K}_j, \Pi_j, P_j] \ldots \mathcal{L}[\mathcal{K}_l, \Pi_l, P_l]$$

Victim rescuing robotics scenario



- Two kind of robots (landmarks and workers) and one victim to be rescued
- No obstacles (except room walls)
- Landmarks randomly walk until victim is found; they choose a new random direction when a wall is hit
- Workers initially motionless; they move only when signalled by landmarks

Victim rescuing robotics scenario

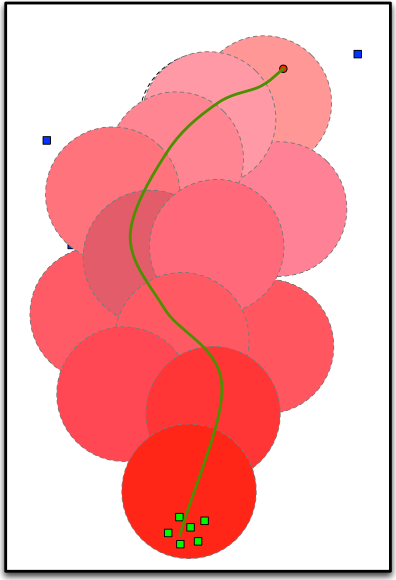


1. A landmark that perceives the victim stops and locally publishes the information that it is at ‘hop’ 0 from the victim

2. All the other landmarks in its range of communication stop and locally publish the information that they are at ‘hop’ 1 from victim

3. And so on . . .

Victim rescuing robotics scenario



- We obtain a sort of **computational fields** leading to the victim that can be exploited by workers
- When workers reach a landmark at hop d they look for a landmark at hop $d - 1$ until they find the victim

Victim rescuing robotics scenario: SCEL specification

LANDMARKS BEHAVIOUR:

$VictimSeeker[DataForwarder[RandomWalk]]$

$$
\begin{array}{l}
VictimSeeker = \\
\quad \textbf{qry}(\text{``}victimPerceived\text{''}, true)@self. \\
\quad \textbf{put}(\text{``}stop\text{''})@self. \\
\quad \textbf{put}(\text{``}victim\text{''}, self, 0)@self
\end{array}
$$

$$
\begin{array}{l}
DataForwarder = \\
\quad \textbf{qry}(\text{``}victim\text{''}, ?id, ?d)@(role = \text{``}landmark\text{''}). \\
\quad \textbf{put}(\text{``}stop\text{''})@self. \\
\quad \textbf{put}(\text{``}victim\text{''}, self, d+1)@self
\end{array}
$$

$$
\begin{array}{l}
RandomWalk = \\
\quad \textbf{put}(\text{``}direction\text{''}, 2\pi rand())@self. \\
\quad \textbf{qry}(\text{``}collision\text{''}, true)@self. \\
\quad RandomWalk
\end{array}
$$

WORKERS BEHAVIOUR: *GoToVictim*

$$
\begin{array}{l}
GoToVictim = \\
\quad \textbf{qry}(\text{``}victim\text{''}, ?id, ?d)@(role = \text{``}landmark\text{''}). \\
\quad \textbf{put}(\text{''}start\text{''})@self. \\
\quad \textbf{put}(\text{``}direction\text{''}, towards(id))@self. \\
\quad \textbf{while}(d > 0)\{\ d := d - 1. \\
\qquad\qquad \textbf{qry}(\text{``}victim\text{''}, ?id, d)@(role = \text{``}landmark\text{''}). \\
\qquad\qquad \textbf{put}(\text{``}direction\text{''}, towards(id))@self\ \} \\
\quad \textbf{qry}(\text{``}victimPerceived\text{''}, true)@self. \\
\quad \textbf{put}(\text{``}stop\text{''})@self
\end{array}
$$

Victim rescuing robotics scenario: SCEL specification

LANDMARKS BEHAVIOUR:

$VictimSeeker[DataForwarder[RandomWalk]]$

$$\begin{array}{l} VictimSeeker = \\ \quad \mathbf{qry}(\text{“}victimPerceived\text{”}, true)@self. \\ \quad \mathbf{put}(\text{“}stop\text{”})@self. \\ \quad \mathbf{put}(\text{“}victim\text{”}, self, 0)@self \end{array}$$

$$\begin{array}{l} DataForwarder = \\ \quad \mathbf{qry}(\text{“}victim\text{”}, ?id, ?d)@(role = \text{“}landmark\text{”}). \\ \quad \mathbf{put}(\text{“}stop\text{”})@self. \\ \quad \mathbf{put}(\text{“}victim\text{”}, self, d+1)@self \end{array}$$

$$\begin{array}{l} RandomWalk = \\ \quad \mathbf{put}(\text{“}direction\text{”}, 2\pi rand())@self. \\ \quad \mathbf{qry}(\text{“}collision\text{”}, true)@self. \\ \quad RandomWalk \end{array}$$

WORKERS BEHAVIOUR: *GoToVictim*

$$\begin{array}{l} GoToVictim = \\ \quad \mathbf{qry}(\text{“}victim\text{”}, ?id, ?d)@(role = \text{“}landmark\text{”}). \\ \quad \mathbf{put}(\text{”}start\text{”})@self. \\ \quad \mathbf{put}(\text{“}direction\text{”}, towards(id))@self. \\ \quad \mathbf{while}(d > 0)\{\ d := d - 1. \\ \qquad\qquad \mathbf{qry}(\text{“}victim\text{”}, ?id, d)@(role = \text{“}landmark\text{”}). \\ \qquad\qquad \mathbf{put}(\text{“}direction\text{”}, towards(id))@self\ \} \\ \quad \mathbf{qry}(\text{“}victimPerceived\text{”}, true)@self. \\ \quad \mathbf{put}(\text{“}stop\text{”})@self \end{array}$$

INSTITUTE FOR ADVANCED STUDIES LUCCA

Victim rescuing robotics scenario: jRESP code (an excerpt)

VictimSeeker =
qry("*victimPerceived*", *true*)@*self*.
put("*stop*")@*self*.
put("*victim*", *self*, 0)@*self*

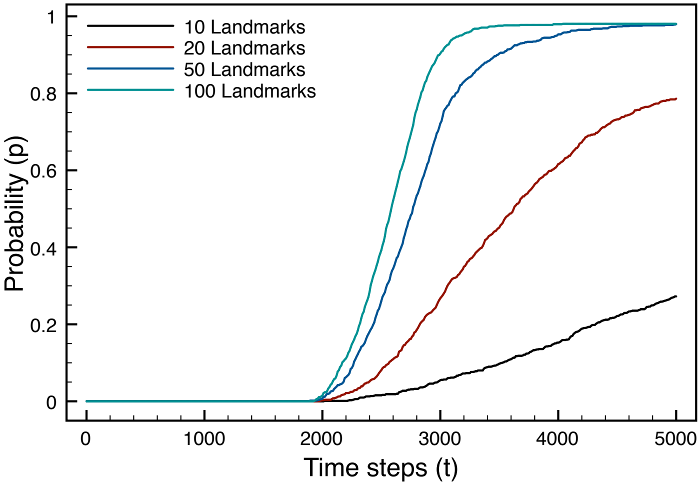
```
public class VictimSeeker extends Agent {
   private int robotId;

   protected void doRun() throws IOException, InterruptedException{
        query(new Template(new ActualTemplateField("VICTIM_PERCEIVED"),
                              new ActualTemplateField(true)) ,
              Self.SELF);
        put( new Tuple( "stop" ) , Self.SELF);
        put( new Tuple( "victim" , robotId , 0 ) , Self.SELF);
     }
   }
}
```

Victim rescuing robotics scenario: jRESP code simulation

DEMO: video...

Victim rescuing robotics scenario: analysis

Probability of rescuing the victim within a given time



Ongoing & Future Work

We have concentrated on modelling behaviors of components and their interactions. We are currently assessing this work and tackling other research items.

- working on interaction policies to study the possibility of modelling different forms of synchronization and communication
- considering different knowledge repositories and ways of expressing goals by analyzing different knowledge representation languages
- assessing the impact and the sensitivity of different adaptation patterns
- developping quantitative variants of SCEL to support components in taking decisions (e.g. via probabilistic model checking).
- distilling a core calculus with attribute based communication to fully understand the full impact of this novel paradigm.

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

Some papers

A formal approach to autonomic systems programming: The SCEL Language. R. De Nicola, M. Loreti, R. Pugliese, F. Tiezzi. ACM TAAS 9(2). ACM Press, 2014

- On Programming and Policing Autonomic Computing Systems. M. Loreti, A. Margheri, R. Pugliese, F. Tiezzi. In Proc. ISOLA, LNCS. Springer, 2014
- Self-expression and Dynamic Attribute-based Ensembles in SCEL. G. Cabri, N. Capodieci, L. Cesari, R. De Nicola, R. Pugliese, F. Tiezzi, F. Zambonelli. In Proc. of ISOLA, LNCS. Springer, 2014
- Programming and Verifying Component Ensembles. R. De Nicola, A. Lluch Lafuente, M. Loreti, A. Morichetta, R. Pugliese, V. Senni, F. Tiezzi. In Proc. FPS, LNCS 8415, 69–83. Springer, 2014
- Formalising Adaptation Patterns for Autonomic Ensembles. L. Cesari, R. De Nicola, R. Pugliese, M. Puviani, F. Tiezzi, F. Zambonelli. In Proc. of FACS, LNCS 8348, 100-118. Springer, 2014
- Reasoning on Service Component Ensembles in Rewriting Logic. L. Belzner, R. De Nicola, A. Vandin, M. Wirsing. In Proc. of SAS, LNCS 8373, 188–211. Springer, 2014
- Stochastically timed predicate-based communication primitives for autonomic computing. D. Latella, M. Loreti. M. Massink, V. Senni. In Proc. of QAPL, 1-16. EPTCS, 2014

Many thanks for your time.

Questions?