IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

Languages and Calculi for Collective Adaptive Systems

Rocco De Nicola

Joint work with
Y. A. Alrahman, M. Loreti, R. Pugliese and F. Tiezzi

27th Nordic Workshop on Programming Theory
Reykjavik – October 2015

Contents

Collective Adaptive Systems - CAS

CAS 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 CAS

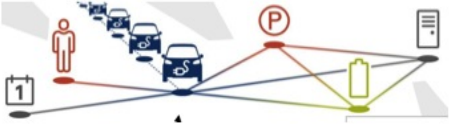
- the dimension of the systems
- the need to adapt to changing environments and requirements
- the emergent behaviour resulting from complex interactions
- the uncertainty during design-time and run-time

Examples of CAS

Robot swarms

Clouds

Cooperative e-vehicles



Importance of languages

Languages play a key role in the engineering of CAS.

- 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 systems features (not on their low-level representations) 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

Key Concepts of CAS

Our aim

We want to enable CAS programmers to model and describe as naturally as possible their behaviour, their interactions, and their sensitivity and adaptivity to the environment.

Key notions to model

1. The behaviours 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

Programming abstractions for CAS

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.

Collective Adaptive Systems as Ensembles

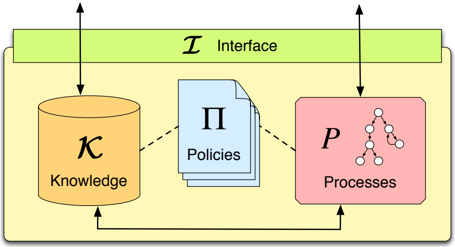
Systems are structured as sets of components dynamically forming interacting ensembles

- Components have an interface exposing component attributes
- Ensembles are not rigid networks but highly flexible structures where components linkages are dynamically established
- Interaction between components is based on attributes and predicates over attributes that permit dynamically specifying targets of communication actions

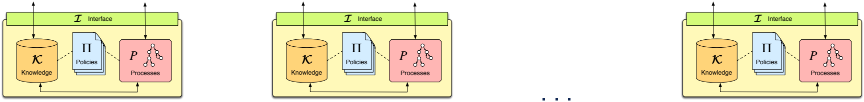
Components and Systems

Aggregations describe how different entities are brought togheter and controlled:

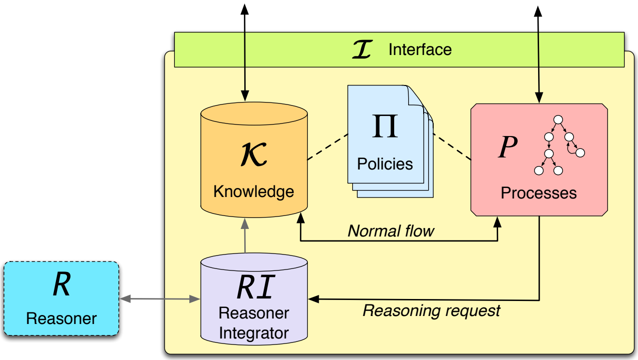
- Components:



- Systems:



A reasoning SCEL component



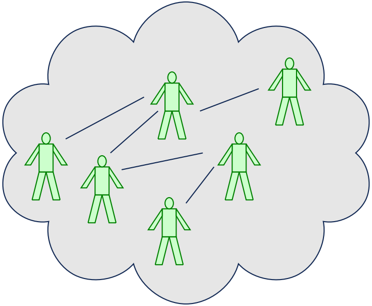
Providing Reasoning Capabilities

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	::=	$\textbf{nil} \mid a.P \mid P_1 + P_2 \mid P_1[P_2] \mid X \mid A(\bar{p}) \quad (A(\bar{f}) \triangleq P)$
Actions:	a	::=	$\textbf{get}(T)@c \mid \textbf{qry}(T)@c \mid \textbf{put}(t)@c \mid \textbf{fresh}(n) \mid \textbf{new}(\mathcal{I}, \mathcal{K}, \Pi, P)$
Targets:	c	::=	$n \mid x \mid \textbf{self} \mid \mathcal{P}$
Items:	t	::=	... currently, tuples
Templates:	T	::=	... currently, tuples with variables

An ensemble



Where are ensembles in SCEL?

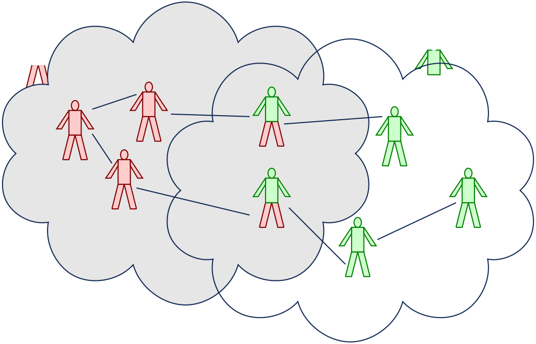
- SCEL syntax does not have specific syntactic constructs for building ensembles.
- Components Interfaces specify (possibly dynamic) attributes (features) and functionalities (services provided).
- Predicate-based communication tests attributes to select the communication targets among those enjoying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

Communication targets can be predicates!

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

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

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

Alternative rendering of ensembles

Static Ensembles

A specific syntactic category is added for representing ensembles. We then have static ensembles with a name; communication to the all elements of an ensemble would be possible using its name.

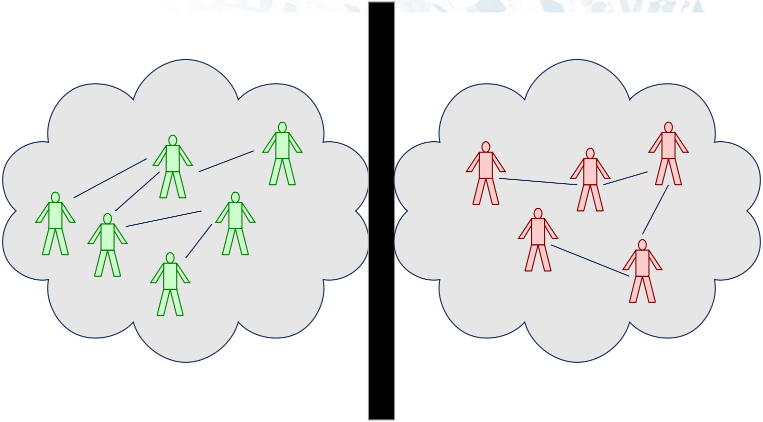
Ensembles as attributes

The interface of each components contains two distinguished attributes: ensemble and membership , to single out:

- the group of components with which the specific component wants to form an ensemble;
- the components from which it is willing to accept invitations to join in an ensemble.

Each ensemble has thus an initiator that can, however, change dynamically.

Static ensembles

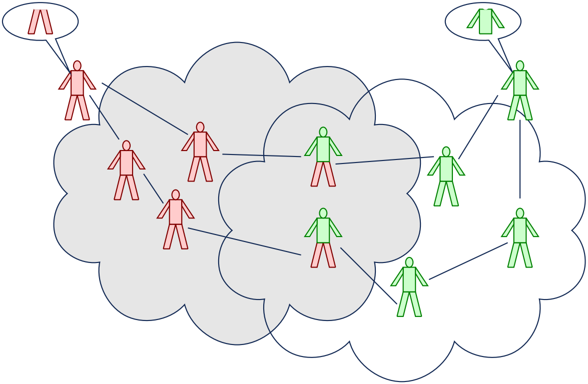


Drawback

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

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

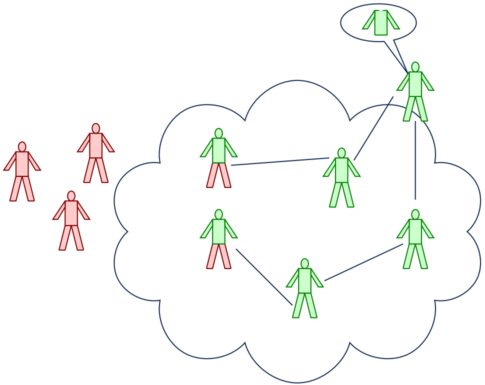
Dynamic ensemble



Drawback

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

Dynamic ensemble



Drawback

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

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

Running SCEL with jRESP

A Java-based run-time Environment for SCEL

jRESP - http://jresp.sourceforge.net/ - the runtime environment for the SCEL paradigm permits using SCEL constructs in Java programs

1. relies on heavy use of *recurrent patterns* to simplify the development of specific
   - knowledge (a single interface that contains basic methods to interact with knowledge)
   - policies (based on the pattern *composite* with policies structured as a stack)
   - ...
2. provides simulation module permitting to simulate SCEL programs and collect relevant data for analysis
3. is based on *open technologies* to support the integration with other tools/frameworks or with alternative implementations of SCEL

INSTITUTE FOR ADVANCED STUDIES LUCCA

Robotics scenario in SCEL

Robot Swarms

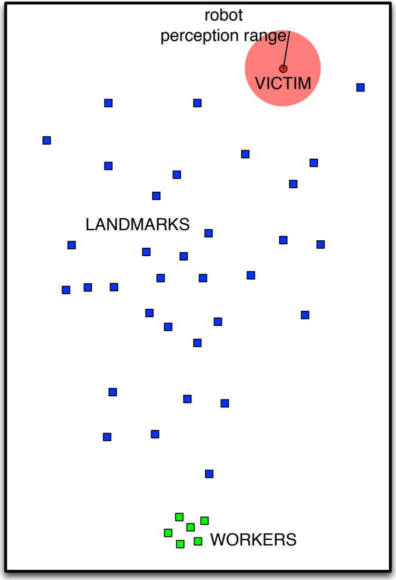
Robots of a swarm have to reach different target zones according to their assigned tasks (help other robots, reach a safe area, clear a minefield, etc.)
Robots:

- 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:

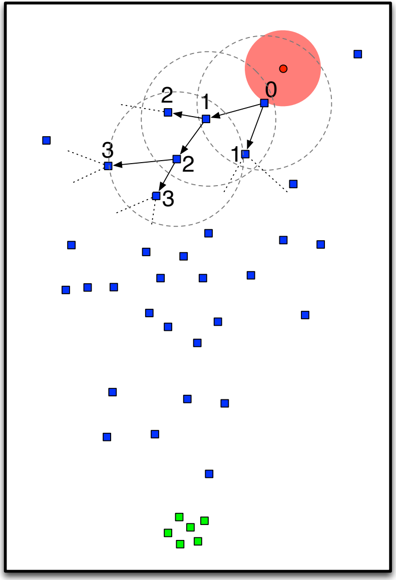
$$\mathcal{I}[\mathcal{K}_i, \Pi_i, P_i] \parallel \mathcal{J}[\mathcal{K}_j, \Pi_j, P_j] \ldots \mathcal{L}[\mathcal{K}_l, \Pi_l, P_l]$$

Victim rescuing robotics scenario



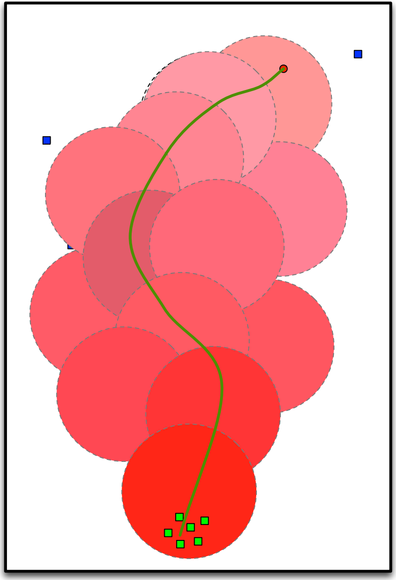
- Two kind of robots (landmarks and workers) and one victim to be rescued
- No obstacles (except room walls)
- Landmarks randomly walk until victim is found; they choose a new random direction when a wall is hit
- Workers initially motionless; they move only when signalled by landmarks

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 ...
4. ...until the news gets to the workers

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

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: $GoToVictim$

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

Victim rescuing robotics scenario

LANDMARKS BEHAVIOUR: $VictimSeeker[DataForwarder[RandomWalk]]$

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

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

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

WORKERS BEHAVIOUR: $GoToVictim$

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

Victim rescuing robotics scenario

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

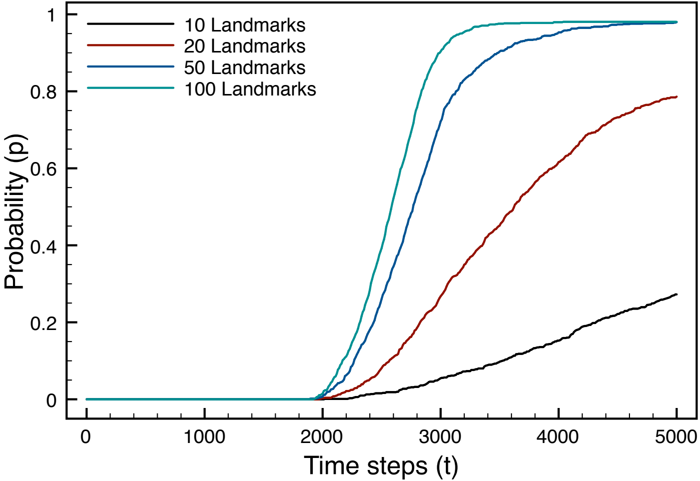
```
public class VictimSeeker extends Agent {
  private int robotId;

  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

DEMO: video. . .

Victim rescuing robotics scenario

Probability of rescuing the victim within a given time



Intermezzo


What people think about
during your conference talk
Hee hee! Animated arrows!
Typo on slide 14, line 3, centre-left. Noted.
Hey! That's what I'm working on!
(RANDOM HAPPY THOUGHTS)
I'm up next I'm up next I'm up next
I'm in the wrong session.
©The Upturned Microscope

Distilling a calculus from SCEL

Towards a Theory of CAS

We aim at developing a theoretical foundation of CAS, starting from their distinctive features, summarized as follows:

- CAS consist of large numbers of interacting components which exhibit complex behaviours depending on their attributes, objectives and actions.
- CAS components may enter or leave the collective at anytime and might have different (possibly conflicting) objectives and need to dynamically adapt to new requirements and contextual conditions.

AbC: A calculus with Attribute based Communication

We have defined *AbC*, a calculus inspired by SCEL and focusing on a minimal set of primitives that rely on attribute-based communication for systems interaction.

AbC at a glance

- Systems are represented as sets of parallel components, each of them equipped with a set of attributes whose values can be modified by internal actions.

AbC at a glance

- Systems are represented as sets of parallel components, each of them equipped with a set of attributes whose values can be modified by internal actions.
- Communication actions (send and receive) are decorated with predicates over attributes that partners have to satisfy to make the interaction possible.

AbC at a glance

- Systems are represented as sets of parallel components, each of them equipped with a set of attributes whose values can be modified by internal actions.
- Communication actions (send and receive) are decorated with predicates over attributes that partners have to satisfy to make the interaction possible.
- Communication takes place in an implicit multicast fashion, and communication partners are selected by relying on predicates over the attributes exposed in their interfaces.

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

AbC at a glance

- Systems are represented as sets of parallel components, each of them equipped with a set of attributes whose values can be modified by internal actions.
- Communication actions (send and receive) are decorated with predicates over attributes that partners have to satisfy to make the interaction possible.
- Communication takes place in an implicit multicast fashion, and communication partners are selected by relying on predicates over the attributes exposed in their interfaces.
- Components are unaware of the existence of each other and they receive messages only if they satisfy senders requirements.

INSTITUTE FOR ADVANCED STUDIES LUCCA

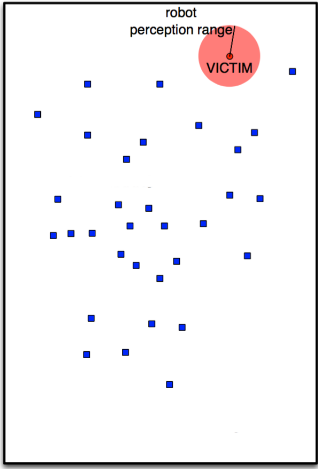
AbC at a glance

- Systems are represented as sets of parallel components, each of them equipped with a set of attributes whose values can be modified by internal actions.
- Communication actions (send and receive) are decorated with predicates over attributes that partners have to satisfy to make the interaction possible.
- Communication takes place in an implicit multicast fashion, and communication partners are selected by relying on predicates over the attributes exposed in their interfaces.
- Components are unaware of the existence of each other and they receive messages only if they satisfy senders requirements.
- Components can offer different views of themselves and can communicate with different partners according to different criteria.

AbC at a glance

- Systems are represented as sets of parallel components, each of them equipped with a set of attributes whose values can be modified by internal actions.
- Communication actions (send and receive) are decorated with predicates over attributes that partners have to satisfy to make the interaction possible.
- Communication takes place in an implicit multicast fashion, and communication partners are selected by relying on predicates over the attributes exposed in their interfaces.
- Components are unaware of the existence of each other and they receive messages only if they satisfy senders requirements.
- Components can offer different views of themselves and can communicate with different partners according to different criteria.
- Semantics for output actions is non-blocking while input actions are blocking in that they can only take place through synchronization with an available sent message.

AbC through a running example



- A swarm of robots is spread throughout a disaster area with the goal of locating victims to rescue.
- Robots have rôles modelled via functional behaviours that can be changed via appropriate adaptation mechanisms.
- Initially all robots are explorers; a robot that finds a victim becomes a rescuer and sends info about the victim to nearby explorers; to form ensembles.
- An explorer that receives information about a victim changes its rôle into helper and joins the rescuers ensemble.
- The rescuing procedure starts when the ensemble is complete.

Some of the attributes (e.g. battery level) are the projection of the robot internal state controlled via sensors and actuators.

AbC Components

$$\text{(Components)}\ C ::= \Gamma : P \quad | \quad C_1 \| C_2 \quad | \quad \nu x C$$

- Single component $\Gamma : P$ – Γ denotes sets of attributes and P processes
- Parallel composition $\_\|\_$ – of components
- Name restriction νx (to delimit the scope of name x) – in $C_1 \| (\nu x) C_2$, name x is invisible from within C_1

Running example (step 1/5)

- Each robot is modeled as an *AbC* component ($Robot_i$) of the following form ($\Gamma_i : P_R$).
- Robots execute in parallel and collaborate.

$$Robot_1 \| \ldots \| Robot_n$$

AbC Processes

$$P ::= 0 \quad | \quad Act.P \quad | \quad \texttt{new(x)}P \quad | \quad \langle\Pi\rangle P \quad | \quad P_1 + P_2 \quad | \quad P_1|P_2 \quad | \; K$$

- $\texttt{new(x)}P$ – Process name restriction
- $\langle\Pi\rangle P$ – blocks P until the evaluation of Π under the local environment becomes true (awareness operator).
- Act – communication and attribute update actions

Running example (step 2/5)

P_R running on a robot has the following form:

$$P_R \triangleq (\langle\Pi\rangle a_1.P_1 + a_2.P_2)|P_3$$

- When Π evaluates to true (e.g., victim detection), the process performs action a_1 and continues as P_1;
- Otherwise P_R performs a_2 to continue as P_2 (help rescuing a victim).

INSTITUTE FOR ADVANCED STUDIES LUCCA

Example Cont.

AbC Actions

$$\text{Act} ::= \Pi(\tilde{x}) \quad | \quad (\tilde{E})@\Pi \vdash_{\{s\}} \quad | \quad [a := E]$$

- $\Pi(\tilde{x})$ – receive from any component satisfying Π;
- $(\tilde{E})@\Pi \vdash_{\{s\}}$ – send to components satisfying Π while exposing only the attributes in set s;
- $[a := E]$: – updates the value of a with the result of evaluating E.

Example Cont.

AbC Actions

$$\text{Act} ::= \Pi(\tilde{x}) \quad | \quad (\tilde{E})@\Pi \vdash_{\{s\}} \quad | \quad [a := E]$$

- $\Pi(\tilde{x})$ – receive from any component satisfying Π;
- $(\tilde{E})@\Pi \vdash_{\{s\}}$ – send to components satisfying Π while exposing only the attributes in set s;
- $[a := E]$: – updates the value of a with the result of evaluating E.

Running example (step 3/5)

- By specifying Π, a_1, and a_2, P_R becomes:

$$P_R \triangleq (\langle \texttt{this}.victimPerceived = \text{tt}\rangle\ [\texttt{this}.state := stop].P_1 \quad + \quad (\texttt{this}.id, qry)@(role = rescuer \vee role = helping)\vdash_{\{role\}}.P_2\ )\ |\ P_3$$

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

Example Cont.

AbC Actions

$$\text{Act} ::= \Pi(\tilde{x}) \quad | \quad (\tilde{E})@\Pi \vdash_{\{s\}} \quad | \quad [a := E]$$

- $\Pi(\tilde{x})$ – receive from any component satisfying Π;
- $(\tilde{E})@\Pi \vdash_{\{s\}}$ – send to components satisfying Π while exposing only the attributes in set s;
- $[a := E]$: – updates the value of a with the result of evaluating E.

Running example (step 3/5)

- By specifying Π, a_1, and a_2, P_R becomes:

$$P_R \triangleq (\langle \mathtt{this}.victimPerceived = \mathrm{tt}\rangle\ [\mathtt{this}.state := stop].P_1 \quad + \quad (\mathtt{this}.id, qry)@(role = rescuer \vee role = helping)\vdash_{\{role\}}.P_2\ )\ |\ P_3$$

We are dwelling whether to use $\Pi(\tilde{x})(\sigma)$ with $\sigma = [a_1 \mapsto E_1, \ldots, a_n \mapsto E_n]$ as input action to atomically update the local environment of the receiver.

AbC Calculus

$$
\begin{array}{lll}
\text{(Components)} & C ::= \Gamma : P \quad | \quad C_1 \| C_2 \quad | \quad \nu x C \\
\text{(Processes)} & P ::= \\
\quad \text{(Inaction)} & \quad 0 \\
\quad \text{(Input)} & \quad | \quad \Pi(\tilde{x}).P \\
\quad \text{(Output)} & \quad | \quad (\tilde{E})@\Pi \vdash_{\{s\}} .P \\
\quad \text{(Update)} & \quad | \quad [a := E].P \\
\quad \text{(New)} & \quad | \quad \mathtt{new(x)}\, P \\
\quad \text{(Match)} & \quad | \quad \langle \Pi \rangle P \\
\quad \text{(Choice)} & \quad | \quad P_1 + P_2 \\
\quad \text{(Par)} & \quad | \quad P_1 | P_2 \\
\quad \text{(Call)} & \quad | \quad K \\
\text{(Predicates)} & \Pi ::= \mathsf{tt} \quad | \quad \mathsf{ff} \quad | \quad E_1 \bowtie E_2 \quad | \quad \Pi_1 \wedge \Pi_2 \quad | \quad \ldots \\
\text{(Data)} & E ::= v \quad | \quad x \quad | \quad a \quad | \quad this.a \quad | \quad \ldots
\end{array}
$$

Operational Semantics

Transitions Labels

- we use the λ-label to range over broadcast, input, update and internal labels respectively

$$\lambda \in \{\nu\tilde{x}\overline{\Gamma{:}(\tilde{v})@\Pi}, \quad \Gamma{:}(\tilde{v})@\Pi, \quad [a := v], \quad \tau\}$$

- we use the α-label to range over all λ-labels plus the input-discarding label as follows:

$$\alpha \in \lambda \cup \{\widetilde{\Gamma{:}(\tilde{v})@\Pi}\}$$

Operational Semantics

Processes and Systems Semantics

AbC is equipped with a two levels labelled semantics.

1. the behaviour of processes is modelled by the transition relation
   $\mapsto \ \subseteq Proc \ \times \ PLAB \ \times \ Proc$
2. the behaviour of component is modelled by the transition relation:
   $\rightarrow \subseteq Comp \ \times \ CLAB \ \times \ Comp$

where

- *Proc* stands for Processes and *Comp* stands for a Components,
- *PLAB* stands stands for
  $\{\nu\tilde{x}\overline{\Gamma{:}(\tilde{v})@\Pi}, \quad \Gamma{:}(\tilde{v})@\Pi, \quad [a := v], \quad \tau, \ \widetilde{\Gamma{:}(\tilde{v})@\Pi}\}$
- *CLAB* stands for $\{\nu\tilde{x}\overline{\Gamma{:}(\tilde{v})@\Pi}, \ \Gamma{:}(\tilde{v})@\Pi, \ \tau\}$

Semantics of Processes (excerpt)

$$\textbf{(Brd)}\ \frac{[\![\tilde{E}]\!]_\Gamma = \tilde{v} \qquad [\![\Pi_1]\!]_\Gamma = \Pi}{(\tilde{E})@\Pi_1 \vdash_s .P \xrightarrow{\overline{\Gamma|_s:(\tilde{v})@\Pi}}_\Gamma P}$$

$$\Gamma|_s = \begin{cases} \Gamma(a) & \texttt{if } a \in s \\ \bot & \texttt{otherwise} \end{cases}$$

$$\textbf{(Rcv)}\ \frac{[\![\Pi_1[\tilde{v}/\tilde{x}]]\!]_\Gamma = \Pi_1' \qquad (\Gamma' \models \Pi_1')}{\Pi_1(\tilde{x}).P \xrightarrow{\Gamma':(\tilde{v})@\Pi_2}_\Gamma P[\tilde{v}/\tilde{x}]}$$

Running example (step 4/5)

- P_R resides within a robot with $\Gamma(id) = 1$
- Some possible evolutions where $\Gamma' = \Gamma_1|_{\{role\}}$ are:

$$P_R \xrightarrow{[\texttt{this}.state{:=}stop]}_{\Gamma_1} P_1|P_3$$

$$P_R \xrightarrow{\overline{\Gamma':(1,\ qry)@(role{=}rescuer\ \vee\ role{=}helping)}}_{\Gamma_1} P_2|P_3$$

Semantics of Processes (excerpt)

Discarding Label

$$(\textbf{FBrd})\ (\tilde{E})@\Pi_1 \vdash_s .P \xrightarrow{\Gamma' : \widetilde{(\tilde{v})@\Pi_2}}_{\Gamma} (\tilde{E})@\Pi_1 \vdash_s .P$$

$$(\textbf{FSum})\ \frac{P_1 \xrightarrow{\Gamma' : \widetilde{(\tilde{v})@\Pi}}_{\Gamma} P_1 \qquad P_2 \xrightarrow{\Gamma' : \widetilde{(\tilde{v})@\Pi}}_{\Gamma} P_2}{P_1 + P_2 \xrightarrow{\Gamma' : \widetilde{(\tilde{v})@\Pi}}_{\Gamma} P_1 + P_2}$$

- Rules like (**FBrd**) models the non-blocking nature of the broadcast;
- Rules like (**FSum**)), are instead used to control internal non-determinism as side-effect.

Running example (step 4/6)

- P_R resides within a robot with explorer role.
- P_R can discard unwanted broadcasts.

$$P_R \xrightarrow{\Gamma_2' : (info)@\widetilde{(role = explorer)}}_{\Gamma_1} P_R$$

From Processes to Components (excerpt)

$$(\textbf{C-Brd})\frac{P \xmapsto{\overline{\Gamma' : (\tilde{v})@\Pi}}_{\Gamma} P'}{\Gamma : P \xrightarrow{\overline{\Gamma' : (\tilde{v})@\Pi}} \Gamma : P'} \qquad (\textbf{C-Rcv})\frac{P \xmapsto{\Gamma' : (\tilde{v})@\Pi}_{\Gamma} P' \quad (\Gamma \models \Pi)}{\Gamma : P \xrightarrow{\Gamma' : (\tilde{v})@\Pi} \Gamma : P'}$$

$$(\textbf{Com})\frac{C_1 \xrightarrow{\nu\tilde{x}\overline{\Gamma : (\tilde{v})@\Pi}} C_1' \quad C_2 \xrightarrow{\Gamma : (\tilde{v})@\Pi} C_2' \qquad \Pi \not\simeq \text{ff}}{C_1 \parallel C_2 \xrightarrow{\nu\tilde{x}\overline{\Gamma : (\tilde{v})@\Pi}} C_1' \parallel C_2'} \; \tilde{x} \cap fn(C_2) = \emptyset$$

Running example (step 5/5): Further specifying P_2 in P_R

$$\begin{aligned} Query \triangleq \; & (\texttt{this}.id,\; qry)@(role = rescuer \lor role = helper) \vdash_{\{role\}} . \\ & (\quad ((role = rescuer \lor role = helper) \land x = ack) \\ & \qquad\qquad (victim_{pos},\; x).P_2' \\ & \quad + \\ & \quad Query \;) \end{aligned}$$

Running example (step 5/5): Cont.

- Assume $Robot_2$ is "*rescuer*", $Robot_3$ is "*helper*", and all others are explorers.
- $Robot_3$ received victim information from $Robot_2$ and now is in charge.
- $Robot_1$ sent a msg containing its identity "*this.id*" and "*qry*" request and $Robot_3$ caught it. Now by using rule (**C-Brd**), $Robot_3$ sends the victim position "$< 3, 4 >$" and "*ack*" back to $Robot_1$ as follows:

$$\Gamma_3 : P_{R_3} \xrightarrow{\overline{\Gamma:(<3,4>,\ ack)@(id=1)}} \Gamma_3 : P'_{R_3} \qquad \text{where } \Gamma = \Gamma_3|_{\{role\}}.$$

- $Robot_1$ applies rule (**C-Rcv**) to receive victim information and generates this transition.

$$\Gamma_1 : P_{R_1} \xrightarrow{\Gamma:(<3,4>,\ ack)@(id=1)} \Gamma_1 : P'_2[< 3, 4 >/victim_{pos},\ ack/x]$$

Running example (step 5/5): Cont.

- Robots can perform the above transitions since

  $\Gamma_1 \models (id = 1)$ and $\Gamma \models ((role = rescuer \vee role = helper) \wedge x = ack)$.

  Other robots discard the broadcast.

- Now the overall system evolves by applying rule (**Com**) as follows:

$$S \xrightarrow{\overline{\Gamma:(<3,4>,\ ack)@(id=1)}} \Gamma_1 : P'_2[<3,4>/victim_{pos},\ ack/x] \;\|\; \Gamma_2 : P_{R_2} \| \Gamma_3 : P'_{R_3} \| \ \ldots \| \Gamma_n : P_{R_n}$$

Behavioural Theory for *AbC*

Some Notations

- $\Rightarrow$ denotes $\xrightarrow{\tau}^*$
- $\stackrel{\gamma}{\Rightarrow}$ denotes $\Rightarrow\xrightarrow{\gamma}\Rightarrow$ if $(\gamma \neq \tau)$
- $\stackrel{\hat{\gamma}}{\Rightarrow}$ denotes $\Rightarrow$ if $(\gamma = \tau)$ and $\stackrel{\gamma}{\Rightarrow}$ otherwise.
- $\rightharpoonup$ denotes $\{\xrightarrow{\gamma} \mid \gamma$ is an output or $\gamma = \tau\}$
- $\rightharpoonup^*$ denotes$(\rightharpoonup)^*$

AbC Contexts

A context $\mathcal{C}[\bullet]$ is a component term with a hole, denoted by $[\bullet]$ and *AbC* contexts are generated by the following grammar:

$$\mathcal{C}[\bullet] ::= [\bullet] \quad | \quad [\bullet]\|C \quad | \quad C\|[\bullet] \quad | \quad \nu x[\bullet]$$

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

Barbed Congruence

Observable Barbs

Let $C\downarrow_\Pi$ mean that component C can broadcast a message with a predicate Π (i.e., $C \xrightarrow{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}}$ where $[\![\Pi]\!] \neq \text{ff}$). We write $C \Downarrow_\Pi$ if $C \rightharpoonup^* C' \downarrow_\Pi$.

Weak Reduction Barbed Congruence Relations

A Weak Reduction Barbed Relation is a symmetric relation $\mathcal{R}$ over the set of *AbC*-components which is barb-preserving, reduction-closed, and context-closed.

Barbed Bisimilarity

Two components are weakly reduction barbed congruent, written $C_1 \cong C_2$, if $(C_1, C_2) \in \mathcal{R}$ for some weak reduction barbed congruent relation $\mathcal{R}$.
The strong reduction congruence "$\simeq$" is obtained in a similar way by replacing $\Downarrow$ with $\downarrow$ and $\rightharpoonup^*$ with $\rightharpoonup$.

Bisimulation for *AbC* Components

Weak Labelled Bisimulation

A symmetric binary relation $\mathcal{R}$ over the set of *AbC*-components is a weak bisimulation if for every action γ, whenever $(C_1, C_2) \in \mathcal{R}$ and

- γ is of the form τ, $\Gamma{:}(\tilde{v})@\Pi$, or $(\nu\tilde{x}\overline{\Gamma{:}(\tilde{v})@\Pi}$ with $[\![\Pi]\!] \neq \mathrm{ff})$, it holds that $C_1 \xrightarrow{\gamma} C_1'$ implies $C_2 \overset{\hat{\gamma}}{\Rightarrow} C_2'$ and $(C_1', C_2') \in \mathcal{R}$

Bisimilarity

Two components C_1 and C_2 are weak bisimilar, written $C_1 \approx C_2$ if there exists a weak bisimulation $\mathcal{R}$ relating them.
Strong bisimilarity, "$\sim$", is defined in a similar way by replacing $\Rightarrow$ with $\rightarrow$.

Bisimilarity and Barbed Congruence do coincide

$C_1 \cong C_2$ if and only if $C_1 \approx C_2$.

Encoding other communication paradigms

A number of alternative communication paradigms such as:

- Explicit Message Passing
- Group based Communications
- Publish-Subscribe

can be easily modelled by relying on *AbC primitives*

Encoding the $b\pi$-calculus

A $b\pi$-calculus process P is rendered as an AbC component $\Gamma{:}P$ where $\Gamma = \emptyset$.

Possible problem

Impossibility of specifying the channel along which the exchange has to happen instantaneously.

Way out

Send the communication channel as a part of the transmitted values and the receiver checks its compatibility.

$$\llparenthesis\ \bar{a}x.P\ \rrparenthesis \triangleq (a, x)@(a = a) \vdash_{\{\}} .\llparenthesis\ P\ \rrparenthesis$$

$$\llparenthesis\ a(x).P\ \rrparenthesis \triangleq \Pi(y, x).\llparenthesis\ P\ \rrparenthesis \quad \texttt{with} \quad \Pi = (y = a) \quad \texttt{and} \quad y \notin n(\llparenthesis\ P\ \rrparenthesis)$$

Group based communication

Group-based interaction

- A group name is encoded as an attribute in *AbC*.
- The constructs for joining or leaving a given group can be encoded as attribute updates.
- ...

$$\Gamma_1 : (msg)@(group = b) \vdash_{\{group\}}$$
$$\|$$
$$\Gamma_2 : (group = a)(x)$$
$$\|$$
$$\vdots$$
$$\|$$
$$\Gamma_7 : (group = a)(x) \mid [\texttt{this}.group := b]$$

Let $\Gamma_1(group) = a$, $\Gamma_2(group) = b$, $\Gamma_7(group) = c$

Publish-Subscribe

Publish-Subscribe interaction is a simple special case of attribute-based communication:

- A Publisher sends tagged messages for all subscribers by exposing from his environment only the current topic.
- Subscribers check compatibility of messages according to their subscriptions.

$$\Gamma_1 : (msg)@(\mathrm{tt}) \vdash_{\{topic\}} \;\; \|$$
$$\Gamma_2 : (topic = \texttt{this}.subscription)(x) \;\|$$
$$\vdots$$
$$\Gamma_n : (topic = \texttt{this}.subscription)(x) \;\|$$

Observation

Dynamic updates of attributes and the possibility of controlling their visibility give *AbC* great flexibility and expressive power.

Ongoing & Future Work

We have concentrated on modelling behaviours of components and their interactions. We are currently tackling other research items.

- working on interaction policies for SCEL 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
- developping quantitative variants of SCEL and AbC to support components in taking decisions (e.g. via probabilistic model checking).
- Considering alternative semantics and behavioural equivalences for *AbC*
- Studying the impact of bisimulation (algebraic laws, axioms, proof techniques, . . .)

Many thanks for your time.

Questions?

IMT INSTITUTE FOR ADVANCED STUDIES LUCCA

Breaking News

EATCS FELLOWS – CALL FOR NOMINATIONS FOR 2016

- Fellows are expected to be *model citizens* of the TCS community, helping to develop the standing of TCS beyond the frontiers of the community.
- INSTRUCTIONS:
  - All nominees and nominators must be EATCS Members
  - Submit by December 31 of the current year for Fellow consideration by email to the EATCS Secretary (secretary@eatcs.org).
  - The EATCS Fellows-Selection Committee
    - Rocco De Nicola (IMT Lucca, Italy, chair)
    - Paul Goldberg (Oxford, United Kingdom)
    - Anca Muscholl (Bordeaux, France)
    - Dorothea Wagner (Karlsruhe, Germany)
    - Roger Wattenhofer (ETH Zurich, Switzerland)