Collective Adaptive Resource-sharing Markovian Agents (CARMA)

Jane Hillston

School of Informatics,
University of Edinburgh

8th April 2019

Outline

Outline

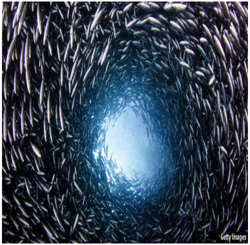
Collective Systems

We are surrounded by examples of collective systems:

Collective Systems

We are surrounded by examples of collective systems:

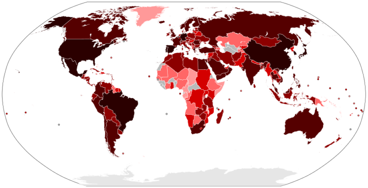
in the natural world ....



Collective Systems

We are surrounded by examples of collective systems:

.... and in the man-made world



Collective Systems

We are surrounded by examples of collective systems:

an informatic environment

Collective Systems

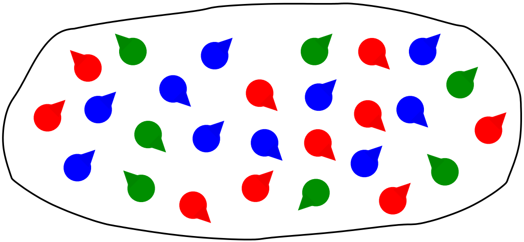
We are surrounded by examples of collective systems:

an informatic environment

Most of these systems are also adaptive to their environment

Collective Adaptive Systems

From a computer science perspective these systems can be viewed as being made up of a large number of interacting entities.

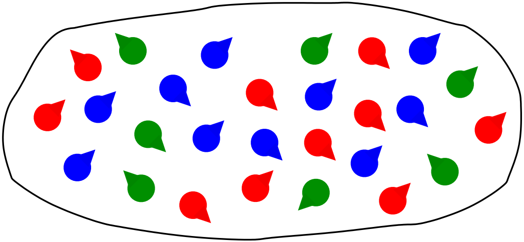


Each entity may have its own properties, objectives and actions.

At the system level these combine to create the collective behaviour.

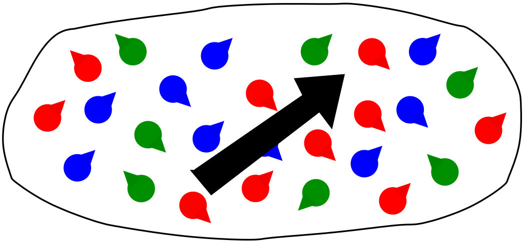
Collective Adaptive Systems

The behaviour of the system is thus dependent on the behaviour of the individual entities.



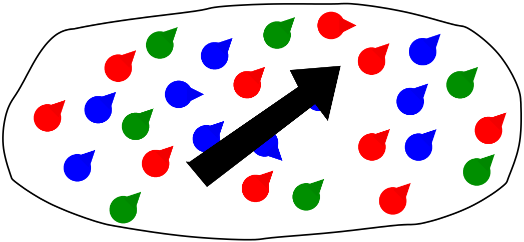
Collective Adaptive Systems

The behaviour of the system is thus dependent on the behaviour of the individual entities.



Collective Adaptive Systems

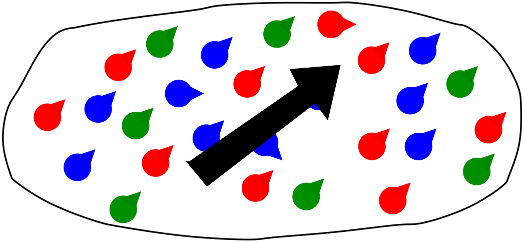
The behaviour of the system is thus dependent on the behaviour of the individual entities.



And the behaviour of the individuals will be influenced by the state of the overall system.

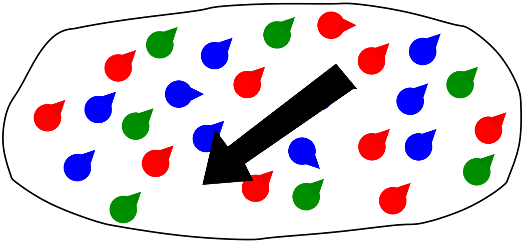
Collective Adaptive Systems

Such systems are often embedded in our environment and need to operate without centralised control or direction.



Collective Adaptive Systems

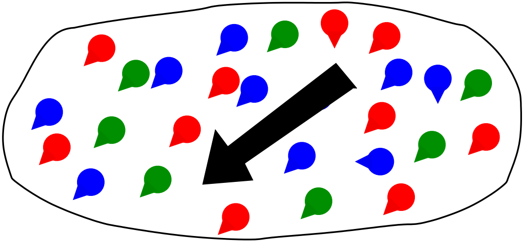
Such systems are often embedded in our environment and need to operate without centralised control or direction.



Moreover when conditions within the system change it may not be feasible to have human intervention to adjust behaviour appropriately.

Collective Adaptive Systems

Such systems are often embedded in our environment and need to operate without centralised control or direction.



Moreover when conditions within the system change it may not be feasible to have human intervention to adjust behaviour appropriately.

Thus systems must be able to autonomously adapt.

The Informatic Environment

Robin Milner coined the term of informatics environment, in which pervasive computing elements are embedded in the human environment, invisibly providing services and responding to requirements.

The Informatic Environment

Robin Milner coined the term of informatics environment, in which pervasive computing elements are embedded in the human environment, invisibly providing services and responding to requirements.

Such systems are now becoming the reality, and many form collective adaptive systems, in which large numbers of computing elements collaborate to meet the human need.

The Informatic Environment

Robin Milner coined the term of informatics environment, in which pervasive computing elements are embedded in the human environment, invisibly providing services and responding to requirements.

Such systems are now becoming the reality, and many form collective adaptive systems, in which large numbers of computing elements collaborate to meet the human need.

For instance, may examples of such systems can be found in components of Smart Cities, such as smart urban transport and smart grid electricity generation and storage.

Performance Modelling for Smart Cities

Capacity planning

- How many buses do I need to maintain service at peak time in a smart urban transport system?

Performance Modelling for Smart Cities

System Configuration

- What capacity do I need at bike stations to minimise the movement of bikes by truck?

Performance Modelling for Smart Cities

System Tuning

- What strategy can I use to maintain supply-demand balance within a smart electricity grid?

Quantitative Modelling

Markovian-based discrete event models have been applied to performance analysis of computer systems since the mid-1960s and communication systems since the early 20th century.

Quantitative Modelling

Markovian-based discrete event models have been applied to performance analysis of computer systems since the mid-1960s and communication systems since the early 20th century.

Performance Modelling

The size and complexity of real systems makes the direct construction of discrete state models costly and error-prone.

Performance Modelling

The size and complexity of real systems makes the direct construction of discrete state models costly and error-prone.

For the last three decades there has been substantial interest in applying formal modelling techniques enhanced with information about timing and probability.

Performance Modelling

The size and complexity of real systems makes the direct construction of discrete state models costly and error-prone.

For the last three decades there has been substantial interest in applying formal modelling techniques enhanced with information about timing and probability.

From these high-level system descriptions the underlying mathematical model (Continuous Time Markov Chain (CTMC)) can be automatically generated.

Performance Modelling

The size and complexity of real systems makes the direct construction of discrete state models costly and error-prone.

For the last three decades there has been substantial interest in applying formal modelling techniques enhanced with information about timing and probability.

From these high-level system descriptions the underlying mathematical model (Continuous Time Markov Chain (CTMC)) can be automatically generated.

In the previous lecture we saw that the Stochastic Process Algebra, PEPA has been designed for this purpose.

Challenges for modelling CAS

The compositional framework provided by stochastic process algebras are well suited to modelling collective behaviour but leave a number of challenges:

- Open-ness and richer forms of interaction
- The influence of space on behaviour
- Capturing adaptivity

Open-ness

SPAs such as PEPA are conservative meaning that agents are neither created nor destroyed during the operation of the system.

In CAS agents may enter and leave the system at random times, either through faults (either within the agent or within the communication network) or choice (e.g. disconnect from a peer-to-peer network).

Thus the communication structure needs to be robust to missing partners, e.g. non-blocking

Richer forms of interaction

If we consider real collective adaptive systems, especially those with emergent behaviour, they embody rich forms of interaction, often based on asynchronous communication.

Richer forms of interaction

If we consider real collective adaptive systems, especially those with emergent behaviour, they embody rich forms of interaction, often based on asynchronous communication.

For example, pheromone trails left by social insects.

Richer forms of interaction

If we consider real collective adaptive systems, especially those with emergent behaviour, they embody rich forms of interaction, often based on asynchronous communication.

For example, pheromone trails left by social insects.

Languages like SCEL offer these richer communication patterns, with components which include a knowledge store which can be manipulated by other components and attribute-based communication.

R.De Nicola, G.Ferrari, M.Loreti, R.Pugliese. A Language-Based Approach to Autonomic Computing. FMCO 2011.

Richer forms of interaction

If we consider real collective adaptive systems, especially those with emergent behaviour, they embody rich forms of interaction, often based on asynchronous communication.

For example, pheromone trails left by social insects.

Languages like SCEL offer these richer communication patterns, with components which include a knowledge store which can be manipulated by other components and attribute-based communication.

R.De Nicola, G.Ferrari, M.Loreti, R.Pugliese. A Language-Based Approach to Autonomic Computing. FMCO 2011.

For quantitative modelling there is a tension between keeping state spaces tractable and capturing local knowledge in agents.

Modelling space

Location and movement play an important role within many CAS, e.g. smart cities.

Modelling space

Location and movement play an important role within many CAS, e.g. smart cities.

We can impose the effects of space by encoding it into the behaviour of the actions of components and distinguishing the same component in different location as distinct types, but this is modelling space implicitly.

Modelling space

Location and movement play an important role within many CAS, e.g. smart cities.

We can impose the effects of space by encoding it into the behaviour of the actions of components and distinguishing the same component in different location as distinct types, but this is modelling space implicitly.

It would be preferable to model space explicitly but this poses significant challenges both for model expression and model solution.

Capturing adaptivity

Existing process algebras tend to work with a fixed set of actions for each entity type.

Some stochastic process algebras allow the rate of activity to be dependent on the state of the system.

But for truly adaptive systems there should also be some way to identify the goal or objective of an entity in addition to its behaviour.

Outline

A new language for CAS

CARMA (Collective Adaptive Resource-sharing Markovian Agents), is a novel stochastic process algebra-style language which handles:

A new language for CAS

CARMA (Collective Adaptive Resource-sharing Markovian Agents), is a novel stochastic process algebra-style language which handles:

1. The behaviours of agents and their interactions;

A new language for CAS

CARMA (Collective Adaptive Resource-sharing Markovian Agents), is a novel stochastic process algebra-style language which handles:

1. The behaviours of agents and their interactions;
2. The global knowledge of the system and that of its agents;

A new language for CAS

CARMA (Collective Adaptive Resource-sharing Markovian Agents), is a novel stochastic process algebra-style language which handles:

1. The behaviours of agents and their interactions;
2. The global knowledge of the system and that of its agents;
3. The environment where agents operate...

A new language for CAS

CARMA (Collective Adaptive Resource-sharing Markovian Agents), is a novel stochastic process algebra-style language which handles:

1. The behaviours of agents and their interactions;
2. The global knowledge of the system and that of its agents;
3. The environment where agents operate...
   - taking into account open ended-ness and adaptation;

A new language for CAS

CARMA (Collective Adaptive Resource-sharing Markovian Agents), is a novel stochastic process algebra-style language which handles:

1. The behaviours of agents and their interactions;
2. The global knowledge of the system and that of its agents;
3. The environment where agents operate...
   - taking into account open ended-ness and adaptation;
   - taking into account resources, locations and visibility/reachability issues.

M.Loreti & J.Hillston, Modelling and Analysis of CAS with CARMA and its Tools. SFM 2016, LNCS 9700.

Interaction patterns in CAS

Typically, CAS exhibit two kinds of interaction pattern:

Interaction patterns in CAS

Typically, CAS exhibit two kinds of interaction pattern:

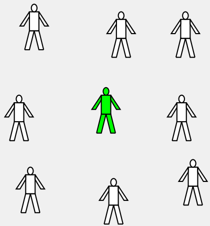
1. **Spreading**: one agent spreads relevant information to a given group of other agents

Interaction patterns in CAS

Typically, CAS exhibit two kinds of interaction pattern:

1. **Spreading**: one agent spreads relevant information to a given group of other agents

Spreading: 1-to-many

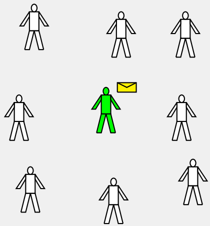


Interaction patterns in CAS

Typically, CAS exhibit two kinds of interaction pattern:

1. **Spreading**: one agent spreads relevant information to a given group of other agents

Spreading: 1-to-many

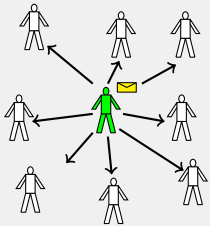


Interaction patterns in CAS

Typically, CAS exhibit two kinds of interaction pattern:

1. **Spreading**: one agent spreads relevant information to a given group of other agents

Spreading: 1-to-many

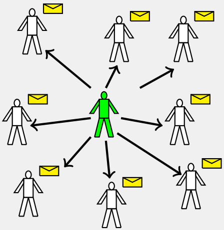


Interaction patterns in CAS

Typically, CAS exhibit two kinds of interaction pattern:

1. **Spreading**: one agent spreads relevant information to a given group of other agents

Spreading: 1-to-many

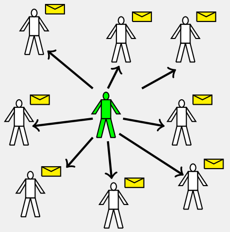


Interaction patterns in CAS

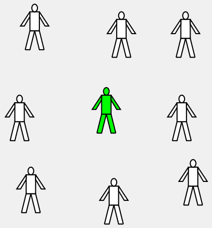
Typically, CAS exhibit two kinds of interaction pattern:

1. **Spreading**: one agent spreads relevant information to a given group of other agents
2. **Collecting**: one agent changes its behaviour according to data collected from one agent belonging to a given group of agents.

Spreading: 1-to-many



Collecting: 1-to-1

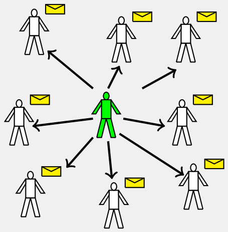


Interaction patterns in CAS

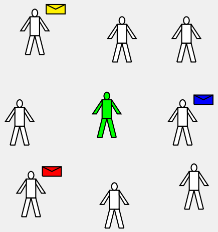
Typically, CAS exhibit two kinds of interaction pattern:

1. **Spreading**: one agent spreads relevant information to a given group of other agents
2. **Collecting**: one agent changes its behaviour according to data collected from one agent belonging to a given group of agents.

Spreading: 1-to-many



Collecting: 1-to-1

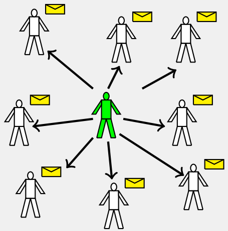


Interaction patterns in CAS

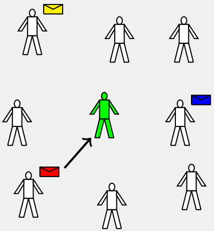
Typically, CAS exhibit two kinds of interaction pattern:

1. **Spreading**: one agent spreads relevant information to a given group of other agents
2. **Collecting**: one agent changes its behaviour according to data collected from one agent belonging to a given group of agents.

Spreading: 1-to-many



Collecting: 1-to-1

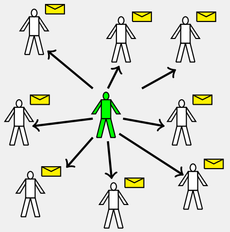


Interaction patterns in CAS

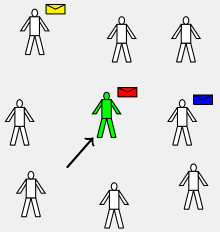
Typically, CAS exhibit two kinds of interaction pattern:

1. **Spreading**: one agent spreads relevant information to a given group of other agents
2. **Collecting**: one agent changes its behaviour according to data collected from one agent belonging to a given group of agents.

Spreading: 1-to-many

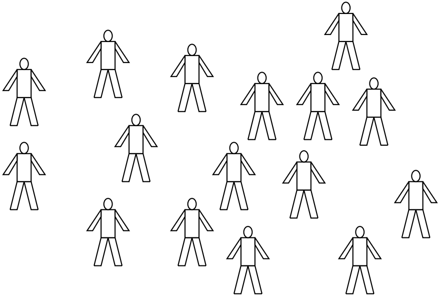


Collecting: 1-to-1



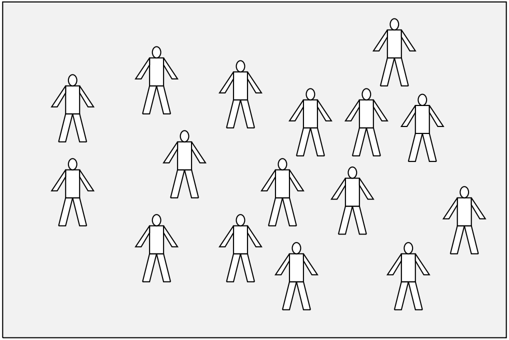
CAS: CARMA perspective

Collective



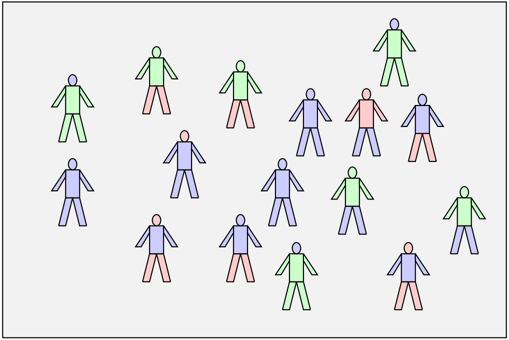
CAS: CARMA perspective

Collective **Environment**



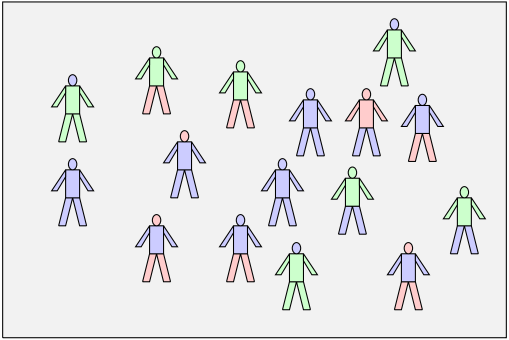
CAS: CARMA perspective

Collective **Environment** **Attributes**



CAS: CARMA perspective

Collective **Environment** **Attributes**



Processes are referenced via their attributes!

Collective Adaptive Resource-sharing Markovian Agents

A CARMA system consists of

Collective Adaptive Resource-sharing Markovian Agents

A CARMA system consists of

- a collective (N)...

Collective Adaptive Resource-sharing Markovian Agents

A CARMA system consists of

- a collective (N)...
- ...operating in an environment ($\mathcal{E}$).

Collective Adaptive Resource-sharing Markovian Agents

A CARMA system consists of

- a collective (N)...
- ...operating in an environment ($\mathcal{E}$).

Collective...

- is composed by a set of components, i.e. the Markovian agents that concur and cooperate to achieve a set of given tasks
- models the behavioural part of a system

Collective Adaptive Resource-sharing Markovian Agents

A CARMA system consists of

- a collective (N)...
- ...operating in an environment ($\mathcal{E}$).

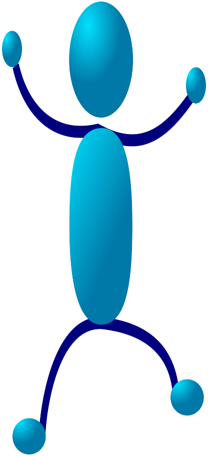
Collective...

- is composed by a set of components, i.e. the Markovian agents that concur and cooperate to achieve a set of given tasks
- models the behavioural part of a system

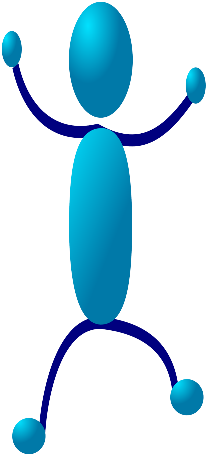
Environment...

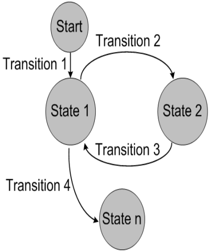
- models the rules intrinsic to the context where agents operate;
- mediates and regulates agent interactions.

Agents in CARMA

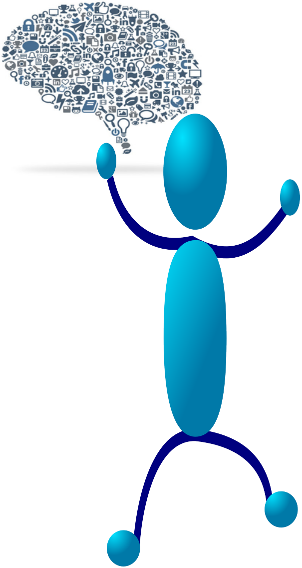


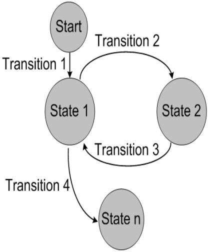
Agents in CARMA



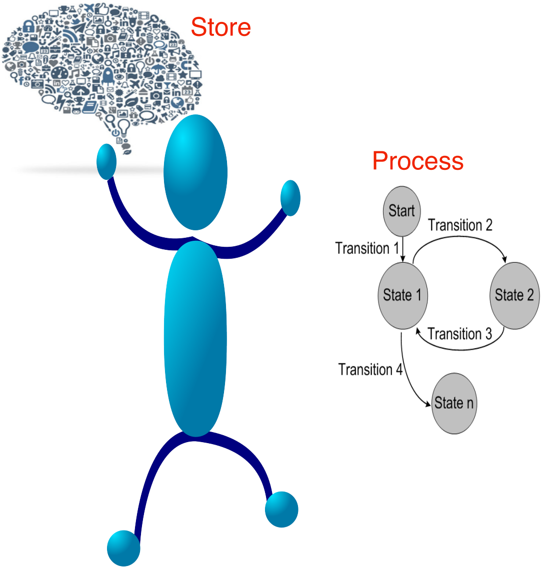


Agents in CARMA

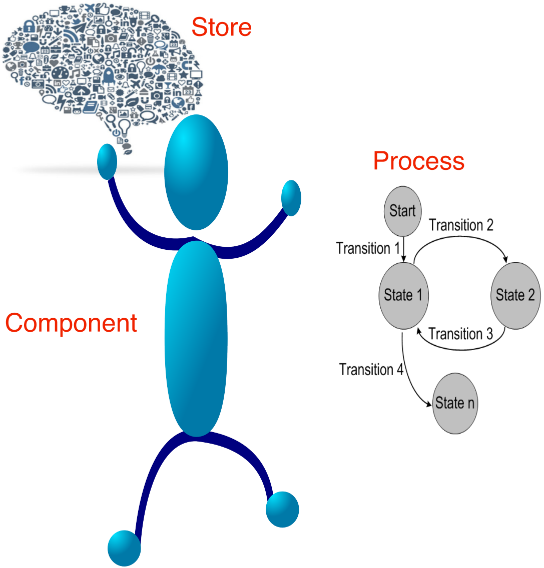




Agents in CARMA



Components


Store
Process
Start
Transition 2
Transition 1
State 1
State 2
Transition 3
Component
Transition 4
State n

Components

Agents in CARMA are defined as components C of the form (P, γ) where...

- P is a process, representing agent behaviour;
- γ is a store, modelling agent knowledge.

Components

Agents in CARMA are defined as components C of the form (P, γ) where...

- P is a process, representing agent behaviour;
- γ is a store, modelling agent knowledge.

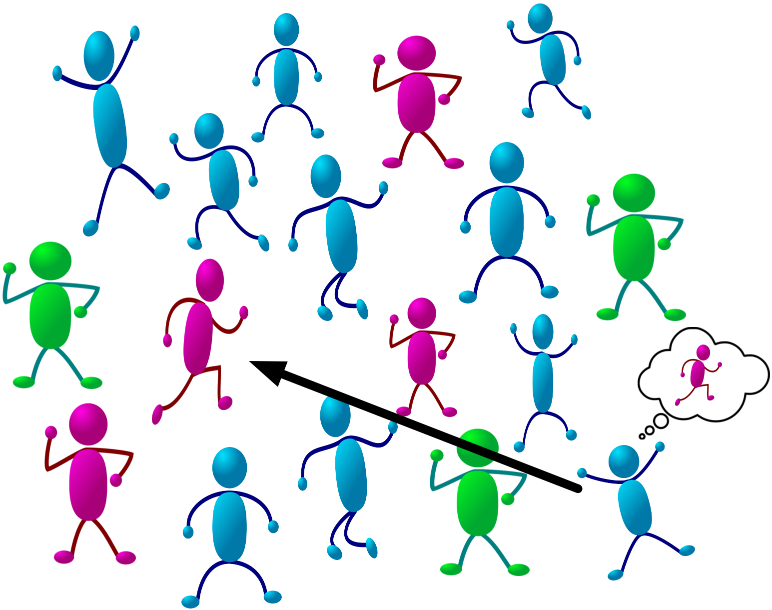
The participants of an interaction are identified via predicates...

- the counterpart of a communication is selected according its properties

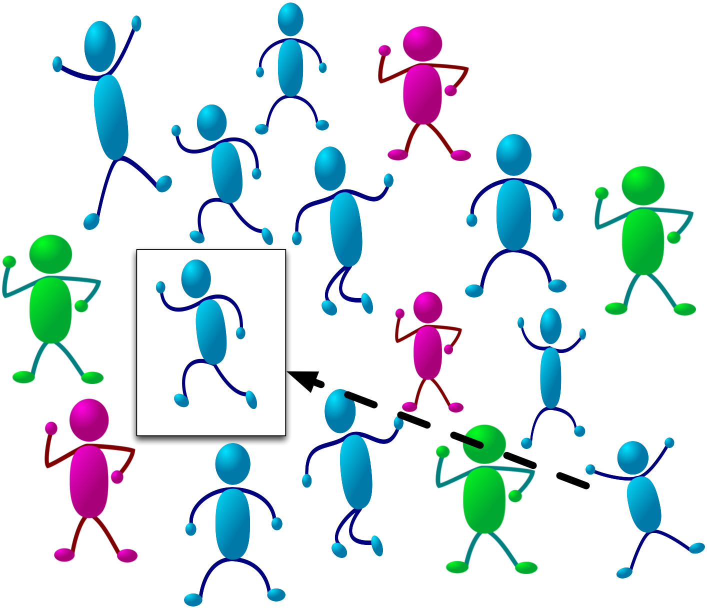
Collective

Unicast communication

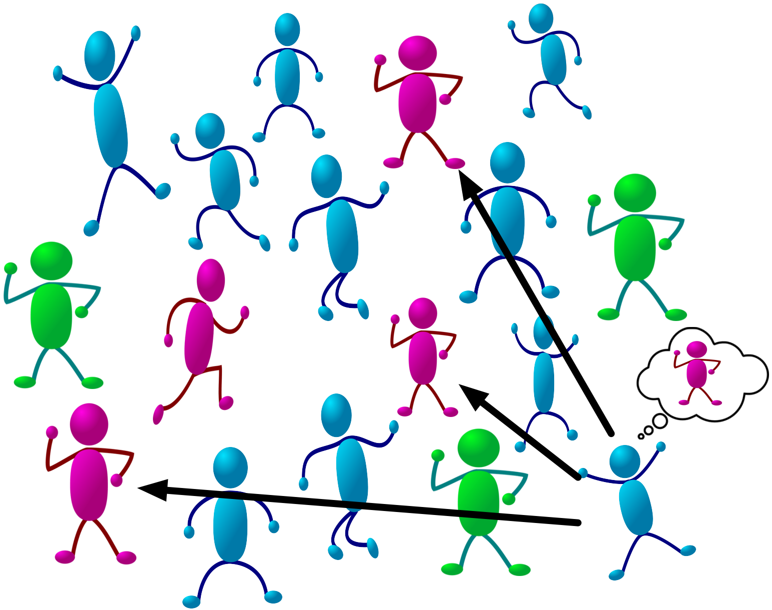
Unicast communication



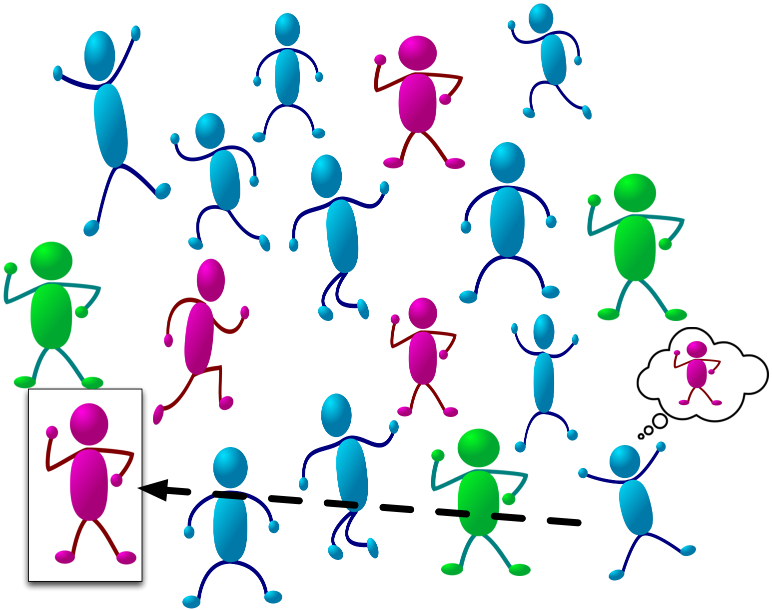
Unicast communication



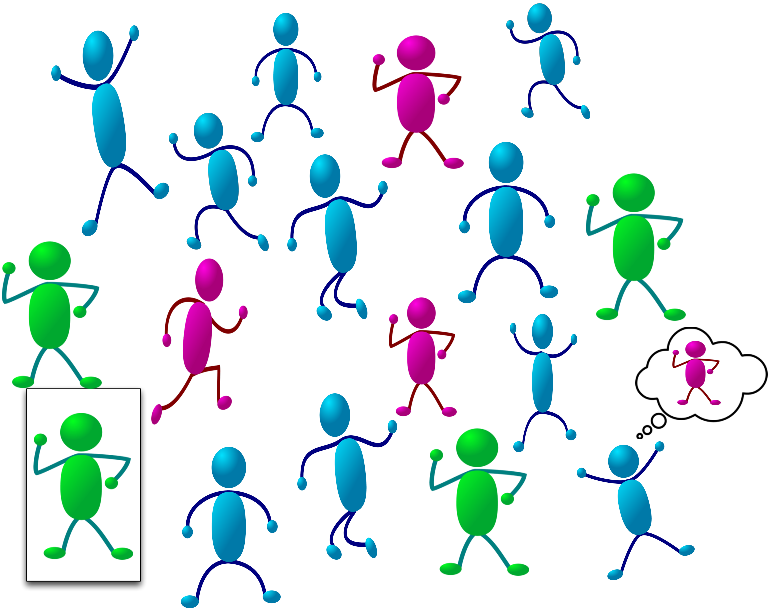
Unicast communication



Unicast communication



Unicast communication



Interaction primitives

Processes interact via attribute based communications. . .

Interaction primitives

Processes interact via attribute based communications. . .

- **Broadcast output**: a message is sent to all the components satisfying a predicate π;

Interaction primitives

Processes interact via attribute based communications. . .

- **Broadcast output**: a message is sent to all the components satisfying a predicate π;
- **Broadcast input**: a process is willing to receive a broadcast message from a component satisfying a predicate π;

Interaction primitives

Processes interact via attribute based communications. . .

- **Broadcast output**: a message is sent to all the components satisfying a predicate π;
- **Broadcast input**: a process is willing to receive a broadcast message from a component satisfying a predicate π;
- **Unicast output**: a message is sent to one of the components satisfying a predicate π;

Interaction primitives

Processes interact via attribute based communications...

- **Broadcast output**: a message is sent to all the components satisfying a predicate π;
- **Broadcast input**: a process is willing to receive a broadcast message from a component satisfying a predicate π;
- **Unicast output**: a message is sent to one of the components satisfying a predicate π;
- **Unicast input**: a process is willing to receive a message from a component satisfying a predicate π.

Interaction primitives

Processes interact via attribute based communications...

- **Broadcast output**: a message is sent to all the components satisfying a predicate π;
- **Broadcast input**: a process is willing to receive a broadcast message from a component satisfying a predicate π;
- **Unicast output**: a message is sent to one of the components satisfying a predicate π;
- **Unicast input**: a process is willing to receive a message from a component satisfying a predicate π.

The execution of an action takes an exponentially distributed time; the rate of each action is determined by the environment.

Interaction primitives
Syntax

act	$::=$	$\alpha^{\star}[\pi]\langle\overrightarrow{e}\rangle\sigma$	Broadcast output
	$\mid$	$\alpha^{\star}[\pi](\overrightarrow{x})\sigma$	Broadcast input
	$\mid$	$\alpha[\pi]\langle\overrightarrow{e}\rangle\sigma$	Unicast output
	$\mid$	$\alpha[\pi](\overrightarrow{x})\sigma$	Unicast input

Interaction primitives

Syntax

act	$::=$	$\alpha^{\star}[\pi]\langle\overrightarrow{e}\rangle\sigma$	Broadcast output
	$\mid$	$\alpha^{\star}[\pi](\overrightarrow{x})\sigma$	Broadcast input
	$\mid$	$\alpha[\pi]\langle\overrightarrow{e}\rangle\sigma$	Unicast output
	$\mid$	$\alpha[\pi](\overrightarrow{x})\sigma$	Unicast input

- α is an action type;

Interaction primitives

Syntax

$$
\begin{array}{rcll}
act & ::= & \alpha^{\star}[\pi]\langle\overrightarrow{e}\rangle\sigma & \text{Broadcast output}\\
& | & \alpha^{\star}[\pi](\overrightarrow{x})\sigma & \text{Broadcast input}\\
& | & \alpha[\pi]\langle\overrightarrow{e}\rangle\sigma & \text{Unicast output}\\
& | & \alpha[\pi](\overrightarrow{x})\sigma & \text{Unicast input}
\end{array}
$$

- α is an action type;
- π is a predicate;

Interaction primitives

Syntax

act	$::=$	$\alpha^{\star}[\pi]\langle\overrightarrow{e}\rangle\sigma$	Broadcast output
	$\mid$	$\alpha^{\star}[\pi](\overrightarrow{x})\sigma$	Broadcast input
	$\mid$	$\alpha[\pi]\langle\overrightarrow{e}\rangle\sigma$	Unicast output
	$\mid$	$\alpha[\pi](\overrightarrow{x})\sigma$	Unicast input

- α is an action type;
- π is a predicate;
- σ is the effect of the action on the store.

Updating the store

After the execution of an action, a process can update the component store:

- σ denotes a function mapping each γ to a probability distribution over possible stores.

Updating the store

After the execution of an action, a process can update the component store:

- σ denotes a function mapping each γ to a probability distribution over possible stores.

$$\mathsf{move}^{\star}[\pi]\langle v\rangle\{x := x + U(-1,+1)\}$$

Updating the store

After the execution of an action, a process can update the component store:

- σ denotes a function mapping each γ to a probability distribution over possible stores.

$$\text{move}^{\star}[\pi]\langle v\rangle\{x := x + U(-1,+1)\}$$

Remark:

- Processes running in the same component can implicitly interact via the local store;
- Updates are instantaneous.

More on synchronisation

Predicates regulating broadcast/unicast inputs can refer also to the received values.

More on synchronisation

Predicates regulating broadcast/unicast inputs can refer also to the received values.

Example:

A value greater than 0 is expected from a component with a *trust_level* less than 3:

$$\alpha^{\star}[(x > 0) \wedge (\mathit{trust\_level} < 3)](x)\sigma.P$$

Examples of interactions...

Broadcast synchronisation:

$$(\text{stop}^{\star}[\text{bl} < 5\%]\langle v\rangle\sigma_1.P \;, \{role = \text{“}master\text{”}\}) \parallel$$
$$(\text{stop}^{\star}[\text{role} = \text{“}master\text{”}](x)\sigma_2 \;.Q_1 \;, \{\text{bl} = 4\%\}) \parallel$$
$$(\text{stop}^{\star}[\text{role} = \text{“}super\text{”}](x)\sigma_3.Q_2 \;, \{\text{bl} = 2\%\}) \parallel$$
$$(\text{stop}^{\star}[\top](x)\sigma_4.Q_3 \;, \{\text{bl} = 2\%\})$$

Examples of interactions. . .

Broadcast synchronisation:

$$
\begin{array}{l}
(\ \mathsf{stop}^{\star}[\mathsf{bl} < 5\%]\langle v\rangle\sigma_1.P\ ,\{role = \text{"}master\text{"}\})\ \|\\
\quad (\ \mathsf{stop}^{\star}[\mathsf{role} = \text{"}master\text{"}](x)\sigma_2\ .Q_1\ ,\{\mathsf{bl} = 4\%\})\ \|\\
\quad\quad (\ \mathsf{stop}^{\star}[\mathsf{role} = \text{"}super\text{"}](x)\sigma_3.Q_2\ ,\{\mathsf{bl} = 2\%\})\ \|\\
\quad\quad\quad (\ \mathsf{stop}^{\star}[\top](x)\sigma_4.Q_3\ ,\{\mathsf{bl} = 2\%\})
\end{array}
$$

Examples of interactions. . .

Broadcast synchronisation:

$$
\begin{array}{l}
(\ \mathsf{stop}^{\star}[\mathsf{bl} < 5\%]\langle v\rangle\sigma_1.P\ ,\{role = \text{``}master\text{''}\})\ \|\\
\quad (\ \mathsf{stop}^{\star}[\mathsf{role} = \text{``}master\text{''}](x)\sigma_2\ .Q_1\ ,\{\mathsf{bl} = 4\%\})\ \|\\
\qquad (\ \mathsf{stop}^{\star}[\mathsf{role} = \text{``}super\text{''}](x)\sigma_3.Q_2\ ,\{\mathsf{bl} = 2\%\})\ \|\\
\qquad\qquad (\ \mathsf{stop}^{\star}[\top](x)\sigma_4.Q_3\ ,\{\mathsf{bl} = 2\%\})
\end{array}
$$

Examples of interactions. . .

Broadcast synchronisation:

$$(\text{stop}^\star[\text{bl} < 5\%]\langle v\rangle\sigma_1.P \;, \{role = \text{“}master\text{''}\}) \parallel$$
$$(\text{stop}^\star[\text{role} = \text{“}master\text{''}](x)\sigma_2 \;.Q_1 \;, \{\text{bl} = 4\%\}) \parallel$$
$$(\text{stop}^\star[\text{role} = \text{“}super\text{''}](x)\sigma_3.Q_2 \;, \{\text{bl} = 2\%\}) \parallel$$
$$(\text{stop}^\star[\top](x)\sigma_4.Q_3 \;, \{\text{bl} = 2\%\})$$

$$\Downarrow$$

$$(P, \sigma_1(\{role = \text{“}master\text{''}\})) \parallel$$
$$(Q_1[v/x], \sigma_2(\{\text{bl} = 4\%\})) \parallel$$
$$(\text{stop}^\star[\text{role} = \text{“}super\text{''}](x)\sigma_3.Q_2, \{\text{bl} = 2\%\}) \parallel$$
$$(Q_3[v/x], \sigma_4(\{\text{bl} = 2\%\}))$$

Examples of interactions. . .

Broadcast synchronisation:

$$(\mathrm{stop}^{\star}[\mathrm{bl} < 5\%]\langle v\rangle\sigma_1.P, \{role = \text{“}master\text{”}\}) \parallel$$
$$(\mathrm{stop}^{\star}[\mathrm{role} = \text{“}master\text{”}](x)\sigma_2.Q_1, \{\mathrm{bl} = 45\%\}) \parallel$$
$$(\mathrm{stop}^{\star}[\mathrm{role} = \text{“}super\text{”}](x)\sigma_3.Q_2, \{\mathrm{bl} = 2\%\}) \parallel$$
$$(\mathrm{stop}^{\star}[\top](x)\sigma_4.Q_3, \{\mathrm{bl} = 25\%\})$$

Examples of interactions...

Broadcast synchronisation:

$$(\mathsf{stop}^\star[\mathsf{bl} < 5\%]\langle v\rangle\sigma_1.P, \{role = \text{“}master\text{”}\}) \parallel$$
$$(\mathsf{stop}^\star[\mathsf{role} = \text{“}master\text{”}](x)\sigma_2.Q_1, \{\mathsf{bl} = 45\%\}) \parallel$$
$$(\mathsf{stop}^\star[\mathsf{role} = \text{“}super\text{”}](x)\sigma_3.Q_2, \{\mathsf{bl} = 2\%\}) \parallel$$
$$(\mathsf{stop}^\star[\top](x)\sigma_4.Q_3, \{\mathsf{bl} = 25\%\})$$

Examples of interactions...

Broadcast synchronisation:

$$(\mathsf{stop}^{\star}[\mathsf{bl} < 5\%]\langle v\rangle\sigma_1.P, \{role = \text{``}master\text{''}\}) \parallel$$
$$(\mathsf{stop}^{\star}[\mathsf{role} = \text{``}master\text{''}](x)\sigma_2.Q_1, \{\mathsf{bl} = 45\%\}) \parallel$$
$$(\mathsf{stop}^{\star}[\mathsf{role} = \text{``}super\text{''}](x)\sigma_3.Q_2, \{\mathsf{bl} = 2\%\}) \parallel$$
$$(\mathsf{stop}^{\star}[\top](x)\sigma_4.Q_3, \{\mathsf{bl} = 25\%\})$$

$$\Downarrow$$

$$(P, \sigma_1(\{role = \text{``}master\text{''}\})) \parallel$$
$$(\mathsf{stop}^{\star}[\mathsf{role} = \text{``}master\text{''}](x)\sigma_2.Q_1, \{\mathsf{bl} = 45\%\}) \parallel$$
$$(\mathsf{stop}^{\star}[\mathsf{role} = \text{``}super\text{''}](x)\sigma_3.Q_2, \{\mathsf{bl} = 2\%\}) \parallel$$
$$(\mathsf{stop}^{\star}[\top](x)\sigma_4.Q_3, \{\mathsf{bl} = 25\%\})$$

Examples of interactions...

Unicast synchronisation:

$$(\mathsf{stop}[\mathsf{bl} < 5\%]\langle\bullet\rangle\sigma_1.P, \{role = \text{``}master\text{''}\}) \parallel$$
$$(\mathsf{stop}[\mathsf{role} = \text{``}master\text{''}](x)\sigma_2.Q_1, \{\mathsf{bl} = 4\%\}) \parallel$$
$$(\mathsf{stop}[\mathsf{role} = \text{``}super\text{''}](x)\sigma_3.Q_2, \{\mathsf{bl} = 2\%\}) \parallel$$
$$(\mathsf{stop}[\top](x)\sigma_4.Q_3, \{\mathsf{bl} = 2\%\})$$

Examples of interactions...

Unicast synchronisation:

$$
\begin{aligned}
&(\mathsf{stop}[\mathsf{bl} < 5\%]\langle\bullet\rangle\sigma_1.P, \{role = \text{``}master\text{''}\}) \parallel \\
&\quad (\mathsf{stop}[\mathsf{role} = \text{``}master\text{''}](x)\sigma_2.Q_1, \{\mathsf{bl} = 4\%\}) \parallel \\
&\qquad (\mathsf{stop}[\mathsf{role} = \text{``}super\text{''}](x)\sigma_3.Q_2, \{\mathsf{bl} = 2\%\}) \parallel \\
&\qquad\qquad (\mathsf{stop}[\top](x)\sigma_4.Q_3, \{\mathsf{bl} = 2\%\})
\end{aligned}
$$

Examples of interactions...

Unicast synchronisation:

$$(\mathsf{stop}[\mathsf{bl} < 5\%]\langle\bullet\rangle\sigma_1.P, \{role = \text{“}master\text{”}\}) \parallel$$
$$(\mathsf{stop}[\mathsf{role} = \text{“}master\text{”}](x)\sigma_2.Q_1, \{\mathsf{bl} = 4\%\}) \parallel$$
$$(\mathsf{stop}[\mathsf{role} = \text{“}super\text{”}](x)\sigma_3.Q_2, \{\mathsf{bl} = 2\%\}) \parallel$$
$$(\mathsf{stop}[\top](x)\sigma_4.Q_3, \{\mathsf{bl} = 2\%\})$$

Examples of interactions...

Unicast synchronisation:

$$(\mathsf{stop}[\mathsf{bl} < 5\%]\langle\bullet\rangle\sigma_1.P, \{role = \text{``}master\text{''}\}) \parallel$$
$$(\mathsf{stop}[\mathsf{role} = \text{``}master\text{''}](x)\sigma_2.Q_1, \{\mathsf{bl} = 4\%\}) \parallel$$
$$(\mathsf{stop}[\mathsf{role} = \text{``}super\text{''}](x)\sigma_3.Q_2, \{\mathsf{bl} = 2\%\}) \parallel$$
$$(\mathsf{stop}[\top](x)\sigma_4.Q_3, \{\mathsf{bl} = 2\%\})$$

$$\Downarrow$$

$$(P, \sigma_1(\{role = \text{``}master\text{''}\})) \parallel$$
$$(\mathsf{stop}[\mathsf{role} = \text{``}master\text{''}](x)\sigma_2.Q_1, \{\mathsf{bl} = 4\%\}) \parallel$$
$$(\mathsf{stop}[\mathsf{role} = \text{``}super\text{''}](x)\sigma_3.Q_2, \{\mathsf{bl} = 2\%\}) \parallel$$
$$(Q_3, \sigma_4(\{\mathsf{bl} = 2\%\}))$$

Modelling the environment

Interactions between components can be affected by the environment:

- a wall can inhibit wireless interactions;
- two components are too distant to interact;
- ...

Modelling the environment

Interactions between components can be affected by the environment:

- a wall can inhibit wireless interactions;
- two components are too distant to interact;
- . . .

The environment. . .

- is used to model the intrinsic rules that govern the physical context;

Modelling the environment

Interactions between components can be affected by the environment:

- a wall can inhibit wireless interactions;
- two components are too distant to interact;
- ...

The environment...

- is used to model the intrinsic rules that govern the physical context;
- consists of a pair (γ, ρ):

Modelling the environment

Interactions between components can be affected by the environment:

- a wall can inhibit wireless interactions;
- two components are too distant to interact;
- ...

The environment...

- is used to model the intrinsic rules that govern the physical context;
- consists of a pair (γ, ρ):
  - a global store γ, that captures knowledge at the system level;

Modelling the environment

Interactions between components can be affected by the environment:

- a wall can inhibit wireless interactions;
- two components are too distant to interact;
- ...

The environment...

- is used to model the intrinsic rules that govern the physical context;
- consists of a pair (γ, ρ):
  - a global store γ, that captures knowledge at the system level;
  - an evolution rule ρ that regulates component interactions (receiving probabilities, action rates,...).

Example: Smart Taxi System

System description:

- We consider a set of taxis operating in a city, providing service to users;
- Both taxis and users are modelled as components.
- The city is subdivided into a number of patches arranged in a grid over the geography of the city.
- The users arrive randomly in different patches, at a rate that depends on the specific time of day.
- After arrival, a user makes a call for a taxi and then waits in that patch until they successfully engage a taxi and move to another randomly chosen patch.
- Unengaged taxis move about the city, influenced by the calls made by users.

J.Hillston and M.Loreti. Specification and analysis of open-ended systems with CARMA. In LNCS 9068, 2015.

Taxis and Users: stores

Both kinds of component use the local store to publish the relevant data that will be used to represent the state of the agent.

Taxis and Users: stores

Both kinds of component use the local store to publish the relevant data that will be used to represent the state of the agent.

Taxis

- *loc*: identifies current taxi location;
- *occupancy*: ranging in $\{0, 1\}$ describes if a taxi is free ($occupancy = 0$) or engaged ($occupancy = 1$);
- *dest*: if occupied, this attribute indicates the destination of the taxi journey.

Taxis and Users: stores

Both kinds of component use the local store to publish the relevant data that will be used to represent the state of the agent.

Taxis

- *loc*: identifies current taxi location;
- *occupancy*: ranging in $\{0, 1\}$ describes if a taxi is free ($occupancy = 0$) or engaged ($occupancy = 1$);
- *dest*: if occupied, this attribute indicates the destination of the taxi journey.

Users

- *loc*: identifies user location;
- *dest*: indicates user destination.

User processes

Users

process *User* =

$$Wait : \mathrm{call}^{\star}[\top]\langle \mathrm{my.loc.}x, \mathrm{my.loc.}y\rangle.Wait$$

$$+$$

$$\mathrm{take}[\mathrm{loc.}x == \mathrm{my.loc.}x \ \wedge\ \mathrm{loc.}y == \mathrm{my.loc.}y]$$

$$\langle \mathrm{my.dest.}x, \mathrm{my.dest.}y\rangle.\mathbf{kill}$$

endprocess

Taxi processes

Taxis

$$
\begin{aligned}
&\text{process } \mathit{Taxi} = \\
&\qquad F : \text{call}^{\star}[(\text{my.loc.x} \neq \text{posx}) \;\wedge\; \text{my.loc.y} \neq \text{posy}](\mathit{posx}, \mathit{posy}) \\
&\qquad\qquad\qquad\qquad \{\text{dest} := [x := \mathit{posx}, y := \mathit{posy}]\}.G \\
&\qquad\quad + \\
&\qquad\quad \text{take}[\top](\mathit{posx}, \mathit{posy}) \\
&\qquad\qquad\qquad \{\text{dest} := [x := \mathit{posx}, y := \mathit{posy}], \text{occupancy} := 1\}.G \\
&\qquad G : \text{move}^{\star}[\bot]\langle\circ\rangle \\
&\qquad\qquad \{\text{loc} := \text{dest}, \text{dest} := [x := 3, y := 3], \text{occupancy} := 0\}.F \\
&\text{endprocess}
\end{aligned}
$$

Modelling arrivals

The Arrivals process has a single attribute loc.

Arrivals process for users

$$
\begin{aligned}
&\text{process } \mathit{Arrivals} = \\
&\qquad A : \text{arrival}^{\star}[\bot]\langle\circ\rangle.A \\
&\text{endprocess}
\end{aligned}
$$

Modelling arrivals

The Arrivals process has a single attribute loc.

Arrivals process for users

$$\begin{array}{l}\text{process } \mathit{Arrivals} = \\ \qquad A : \text{arrival}^{\star}[\bot]\langle\circ\rangle.A \\ \text{endprocess}\end{array}$$

This process is executed in a separate component where attribute loc indicates the location where the user arrives.

Modelling arrivals

The Arrivals process has a single attribute loc.

Arrivals process for users

$$\begin{array}{l} \text{process } \mathit{Arrivals} = \\ \qquad A : \mathsf{arrival}^{\star}[\bot]\langle\circ\rangle.A \\ \text{endprocess} \end{array}$$

This process is executed in a separate component where attribute loc indicates the location where the user arrives.

The precise role of this process will be clear when the environment is described.

The environment

It is assumed that all actions in CARMA take some time complete and that this duration is governed by an exponential distribution.

The environment

It is assumed that all actions in CARMA take some time complete and that this duration is governed by an exponential distribution.

However the action descriptions do not include any information about the timing (unlike many other stochastic process algebras).

The environment

It is assumed that all actions in CARMA take some time complete and that this duration is governed by an exponential distribution.

However the action descriptions do not include any information about the timing (unlike many other stochastic process algebras).

We also do not assume perfect communication, i.e. there may be a probability that an interaction will fail to complete even between components with appropriately match attributes.

The environment

It is assumed that all actions in CARMA take some time complete and that this duration is governed by an exponential distribution.

However the action descriptions do not include any information about the timing (unlike many other stochastic process algebras).

We also do not assume perfect communication, i.e. there may be a probability that an interaction will fail to complete even between components with appropriately match attributes.

The environment manages these aspects of system behaviour, and others in the evolution rule.

The evolution rule ρ

ρ is a function, dependent on current time, the global store and the current state of the collective, returns a tuple of functions $\varepsilon = \langle \mu_p, \mu_w, \mu_r, \mu_u \rangle$ known as the evaluation context

- $\mu_p(\gamma_s, \gamma_r, \alpha)$: the probability that a component with store γ_r can receive a broadcast message α from a component with store γ_s;
- $\mu_w(\gamma_s, \gamma_r, \alpha)$: the weight to be used to compute the probability that a component with store γ_r can receive a unicast message α from a component with store γ_s;
- $\mu_r(\gamma_s, \alpha)$ computes the execution rate of action α executed at a component with store γ_s;
- $\mu_u(\gamma_s, \alpha)$ determines the updates to the environment (global store and collective) induced by the execution of action α at a component with store γ_s.

Evolution rule: μ_p

Defining the probabilities of broadcast actions

$$\begin{array}{l} \mathsf{prob}\{ \\ \qquad \top, \mathsf{call}^\star : \mathsf{global.p}_{\mathsf{lost}} \\ \qquad \mathsf{default} \;\; 1 \\ \} \end{array}$$

- call* can be missed with a probability p_{lost} defined in the global store.
- All the other interactions occur with probability 1.

Evolution rule: μ_w

Defining the weights of unicast actions

$$
\begin{array}{l}
\text{prob}\{ \\
\qquad \top, \text{take} : \mathit{Takeprob}(\mathit{real}(\#\{\mathit{Taxi}[F] \mid \\
\qquad\qquad\qquad (\text{my.loc.x} == \text{sender.loc.x}) \wedge \\
\qquad\qquad\qquad (\text{my.loc.y} == \text{sender.loc.y})\})); \\
\}
\end{array}
$$

- Each taxi receives a user request (take) with a weight that depends on the number of taxis in the patch.

Evolution rule: μ_r

Defining the rates of actions

$$
\begin{array}{l}
\mathsf{rate}\{ \\
\quad \top, \mathsf{take} : \mathsf{global.r_t} \\
\quad \top, \mathsf{call}^\star : \mathsf{global.r_c} \\
\quad \top, \mathsf{move}^\star : \mathit{Mtime}(\mathsf{now}, \mathsf{sender.loc}, \mathsf{sender.dest}, 6) \\
\quad \top, \mathsf{arrival}^\star : \mathit{Atime}(\mathsf{now}, \mathsf{sender.loc}, 1) \\
\quad \mathsf{default} \;\; 0 \\
\}
\end{array}
$$

While take and call have constant rates, the rates of the actions move and arrival are functions that depend on time, reflecting shifting traffic patterns within the city over the course of a day.

Evolution rule: μ_u

In the taxi example, the arrival of a new user is achieved via the update rule:

Update rule

$$\begin{array}{l}\text{update}\{\\ \qquad \top, \text{arrival}^{\star} : \textbf{new } \text{User}(\text{sender.loc}, \text{DestLoc}(\text{now}, \text{sender.loc}), \text{Wait})\\ \}\end{array}$$

Measures

To extract data from a system, a CARMA specification also contains a set of measures.

Measures

To extract data from a system, a CARMA specification also contains a set of measures.

The number of waiting users at a location

$$\textbf{measure}\ \mathrm{WaitingUser}_{00}[i := 0] = \#\{\mathrm{User}[\mathrm{Wait}] \mid \mathrm{my.loc.x} == 0\ \wedge\ \mathrm{my.loc.y} == 0\};$$

Measures

To extract data from a system, a CARMA specification also contains a set of measures.

The number of waiting users at a location

measure $\mathrm{WaitingUser}_{00}[i := 0] = \#\{\mathrm{User}[\mathrm{Wait}] \mid \mathrm{my.loc.x} == 0 \ \wedge \ \mathrm{my.loc.y} == 0\};$

The number of taxis relocating

measure $\mathrm{Taxi\_Relocating}[i := 1] = \#\{\mathrm{Taxi}[G] \mid \mathrm{my.occupancy} == 0\};$

Two Scenarios

We consider a grid of 3×3 patches, i.e., a set of locations (i, j) where $0 \leq i, j \leq 2$, and two different scenarios:

Scenario 1: Users arrive in all the patches at the same rate;

Scenario 2: At the beginning users arrive with a higher probability to the patches at the border of the grid; subsequently, users arrive with higher probability in the centre of the grid.

Two Scenarios

We consider a grid of 3×3 patches, i.e., a set of locations (i, j) where $0 \leq i, j \leq 2$, and two different scenarios:

Scenario 1: Users arrive in all the patches at the same rate;

Scenario 2: At the beginning users arrive with a higher probability to the patches at the border of the grid; subsequently, users arrive with higher probability in the centre of the grid.

These are investigated by placing the same collective in different environments.

Smart Taxi System Collective

collective {
  new : Arrival(0 : 2, 0 : 2);
  new Taxi(0 : 2, 0 : 2, 3, 3, 0, F);
}

Quantitative Analysis

The semantics of CARMA gives rise to a Continuous Time Markov Chain (CTMC).

This can be analysed by

- by numerical analysis of the CTMC for small systems;
- by stochastic simulation of the CTMC;
- by fluid approximation of the CTMC under certain restrictions (particularly on the environment).

Quantitative Analysis

The semantics of CARMA gives rise to a Continuous Time Markov Chain (CTMC).

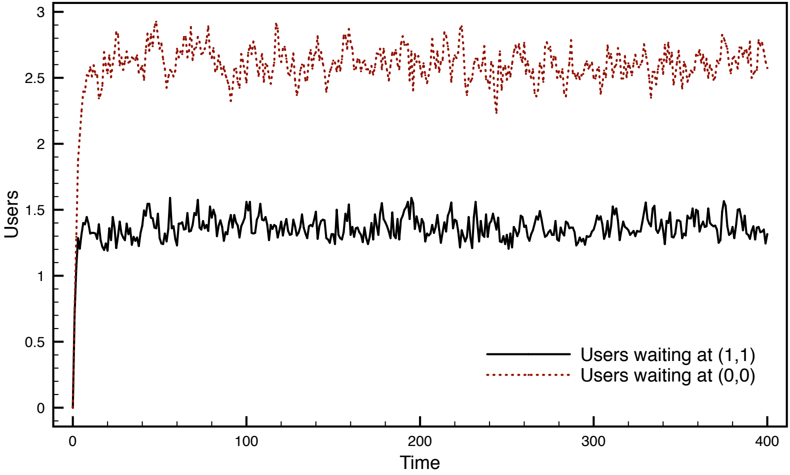
This can be analysed by

- by numerical analysis of the CTMC for small systems;
- by stochastic simulation of the CTMC;
- by fluid approximation of the CTMC under certain restrictions (particularly on the environment).

Here we show the results of stochastic simulation.

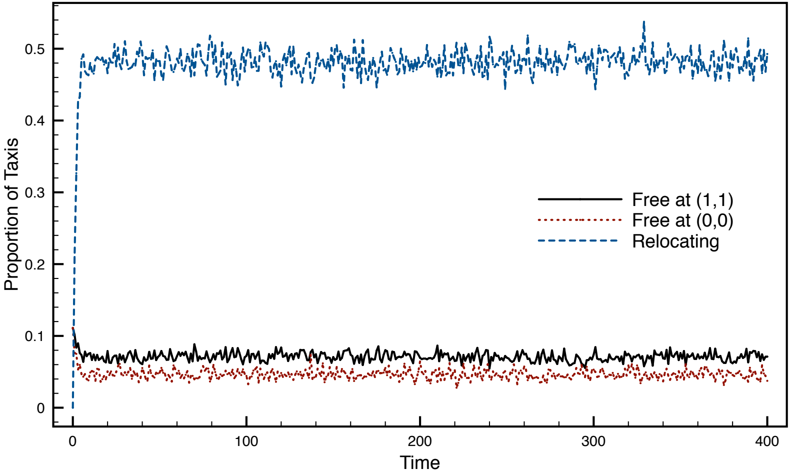
Scenario 1 results

Average number of users waiting at $(1, 1)$ and $(0, 0)$



Scenario 1 results

Proportion of free taxis at $(1,1)$ and $(0,0)$ and in transit

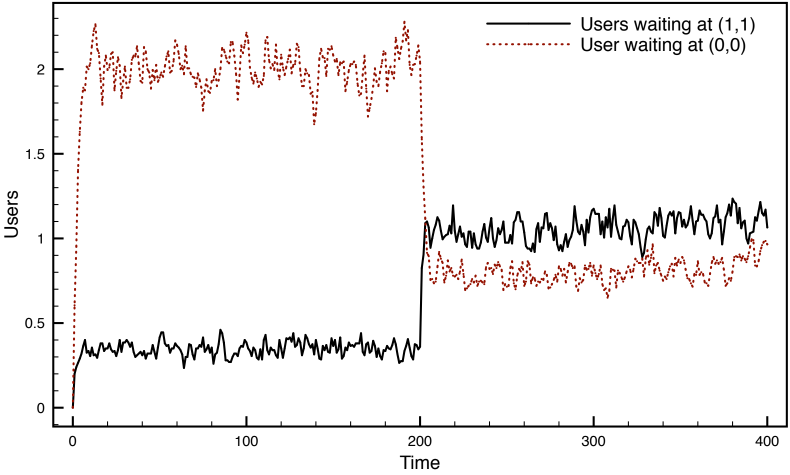


Comments: Scenario 1

- In Scenario 1 after an initial startup period, around 2.5 users are waiting for a taxi in the peripheral location while only 1.5 users are waiting for a taxi in location $(1, 1)$.
- In this scenario a larger fraction of users are delivered to location $(1, 1)$ so soon a larger fraction of taxis are available to collect users at the centre.
- A large fraction of taxis (around 50%) are continually moving between the different patches.

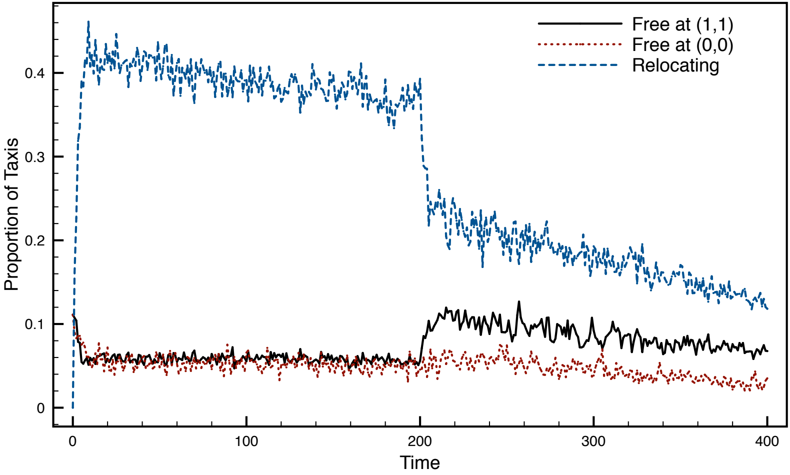
Scenario 2 results

Average number of users waiting at $(1, 1)$ and $(0, 0)$



Scenario 2 results

Proportion of free taxis at $(1, 1)$ and $(0, 0)$ and in transit



Comments: Scenario 2

- In Scenario 2 the location of new arrivals depends on the current time:
  - $[0, 200)$: 3/4 of users arrive on the border and only 1/4 in the centre;
  - $[200, 400)$: 1/4 of users arrive on the border and 3/4 in the centre.
- Results in the first phase are similar to Scenario 1.
- After time 200, the number of users waiting for a taxi in the border decreases below 1 whilst the average number waiting for a taxi in the centre increases to just over 1 and the fraction of taxis continually moving is reduced to 20%.

Outline

Concluding remarks

- Collective Systems are an interesting and challenging class of systems to design and construct.

Concluding remarks

- Collective Systems are an interesting and challenging class of systems to design and construct.
- Their role within infrastructure, such as within smart cities, make it essential that quantitive aspects of behaviour is taken into consideration, as well as functional correctness.

Concluding remarks

- Collective Systems are an interesting and challenging class of systems to design and construct.
- Their role within infrastructure, such as within smart cities, make it essential that quantitive aspects of behaviour is taken into consideration, as well as functional correctness.
- The complexity of these systems poses challenges both for model construction and model analysis.

Concluding remarks

- Collective Systems are an interesting and challenging class of systems to design and construct.
- Their role within infrastructure, such as within smart cities, make it essential that quantitive aspects of behaviour is taken into consideration, as well as functional correctness.
- The complexity of these systems poses challenges both for model construction and model analysis.
- CARMA aims to address many of these challenges, supporting rich forms of interaction, using attributes to capture explicit locations and the environment to allow adaptivity.

Concluding remarks

- Collective Systems are an interesting and challenging class of systems to design and construct.
- Their role within infrastructure, such as within smart cities, make it essential that quantitive aspects of behaviour is taken into consideration, as well as functional correctness.
- The complexity of these systems poses challenges both for model construction and model analysis.
- CARMA aims to address many of these challenges, supporting rich forms of interaction, using attributes to capture explicit locations and the environment to allow adaptivity.
- Fluid approximation based analysis offers hope for scalable quantitative analysis techniques, but this is yet to included in the tool.

Thanks!

Thanks!

Thanks to my collaborators and colleagues on the QUANTICOL project, especially Michele Loreti.

This work has been funded by the CEC through the FET-Proactive QUANTICOL project

quanticol

www.quanticol.eu