Modeling and Analysis of Collective Adaptive Systems with CARMA and its Tools

Jane Hillston
LFCS, University of Edinburgh

Michele Loreti
Università di Firenze

22nd June 2016

Outline

1. Introduction
   - Collective Adaptive Systems
   - Quantitative Analysis
2. Modelling CAS
3. CARMA
   - The CARMA Modelling Language
4. CARMA Operational semantics
5. CASL:CARMA Specification Language
6. CARMA Eclipse plug-in
7. CARMA and its tools in Action

Outline

1. Introduction
   - Collective Adaptive Systems
   - Quantitative Analysis
2. Modelling CAS
3. CARMA
   - The CARMA Modelling Language
4. CARMA Operational semantics
5. CASL:CARMA Specification Language
6. CARMA Eclipse plug-in
7. CARMA and its tools in Action

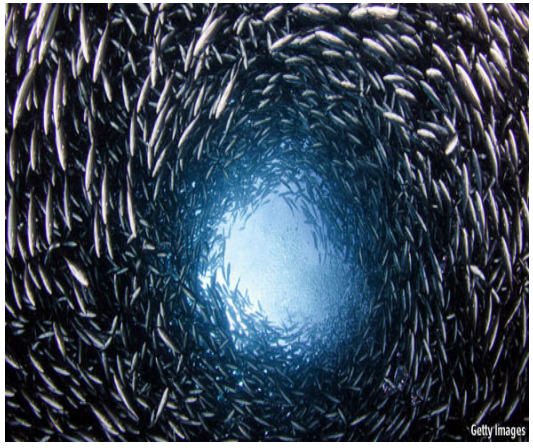
Collective Systems

We are surrounded by examples of collective systems:

Collective Systems

We are surrounded by examples of collective systems:

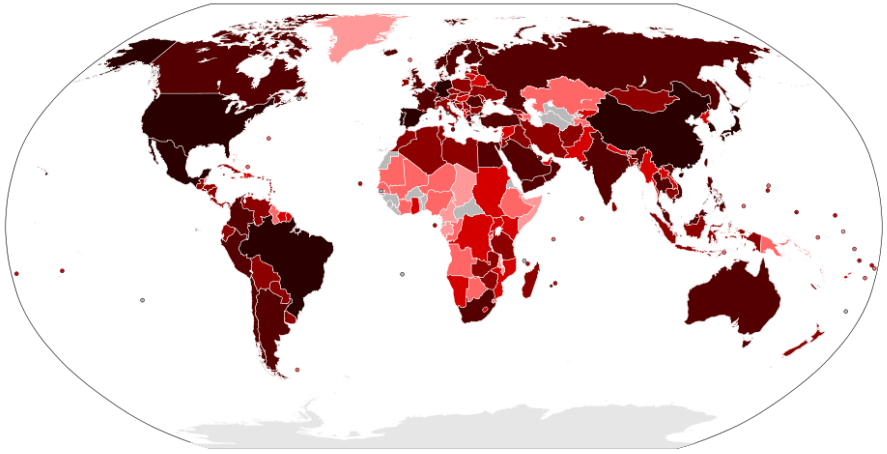
in the natural world ....



Collective Systems

We are surrounded by examples of collective systems:

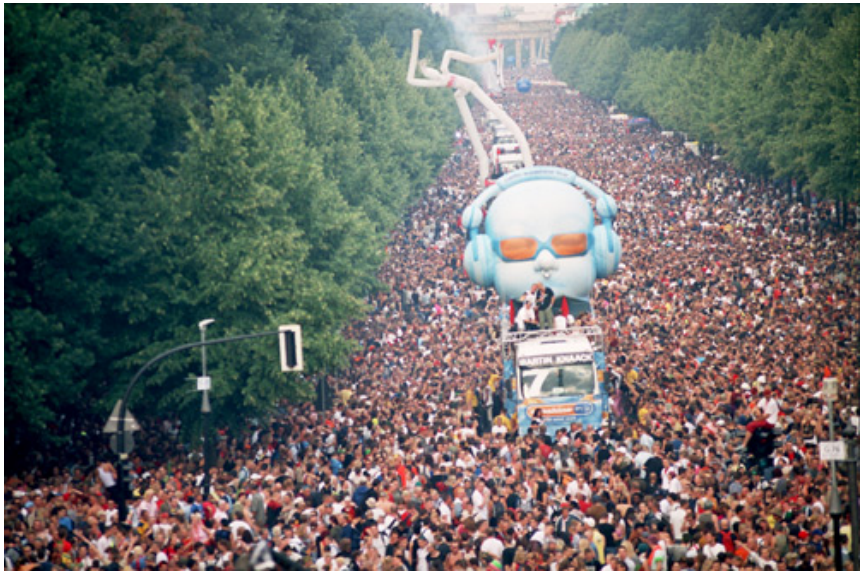
.... and in the man-made world

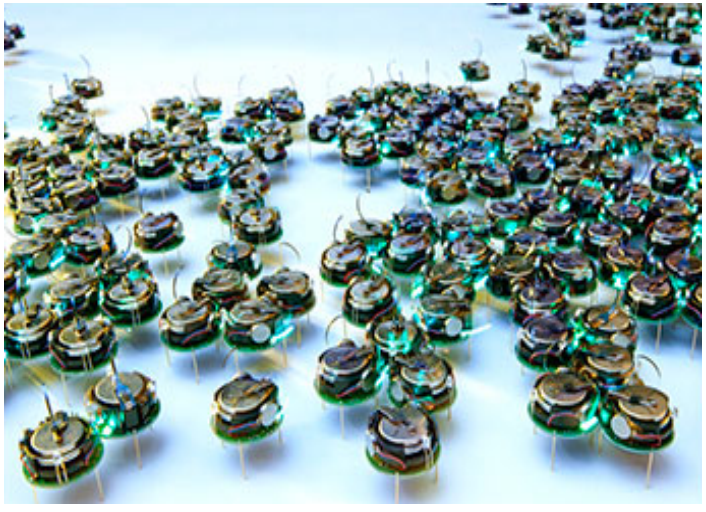


Collective Systems

We are surrounded by examples of collective systems:

.... and in the man-made world

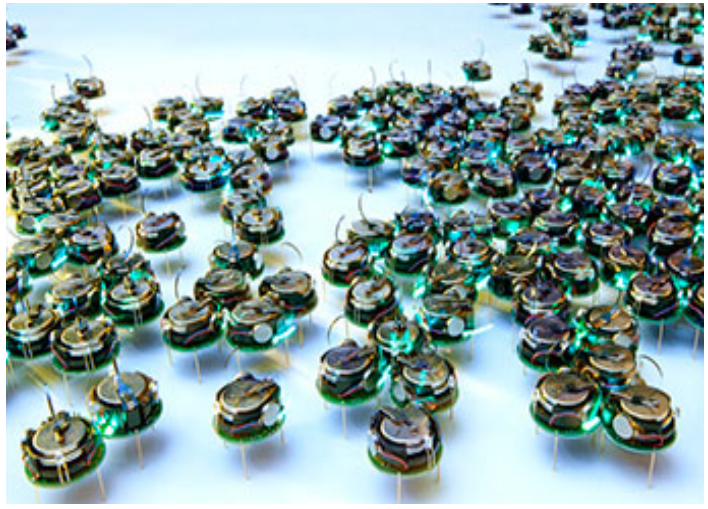




Collective Systems

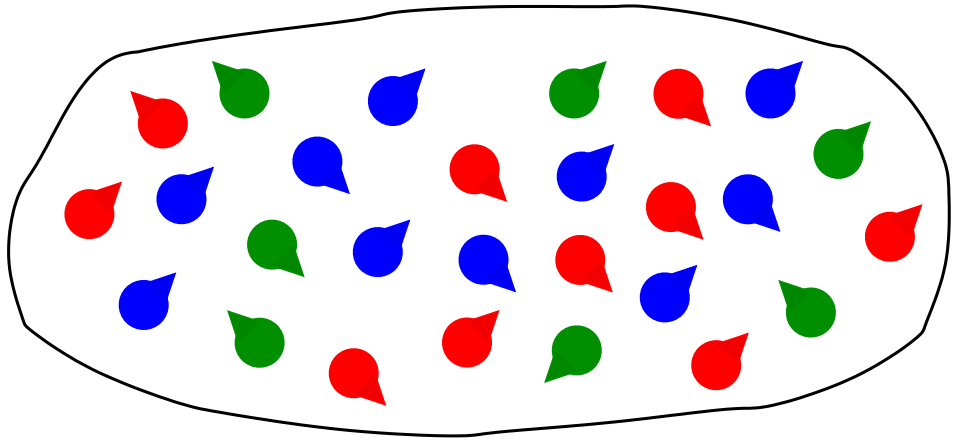
We are surrounded by examples of collective systems:

.... and in the man-made world



Most of these systems are also adaptive to their environment

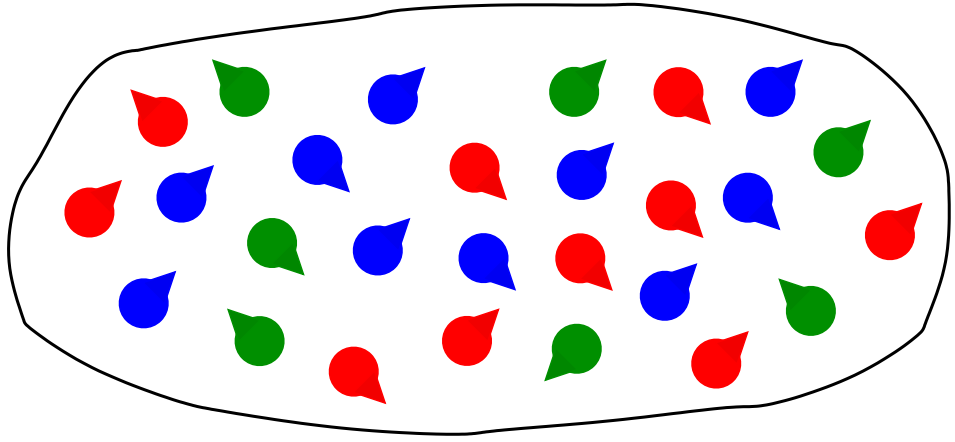
From a computer science perspective these systems can be viewed as being made up of a large number of interacting entities.



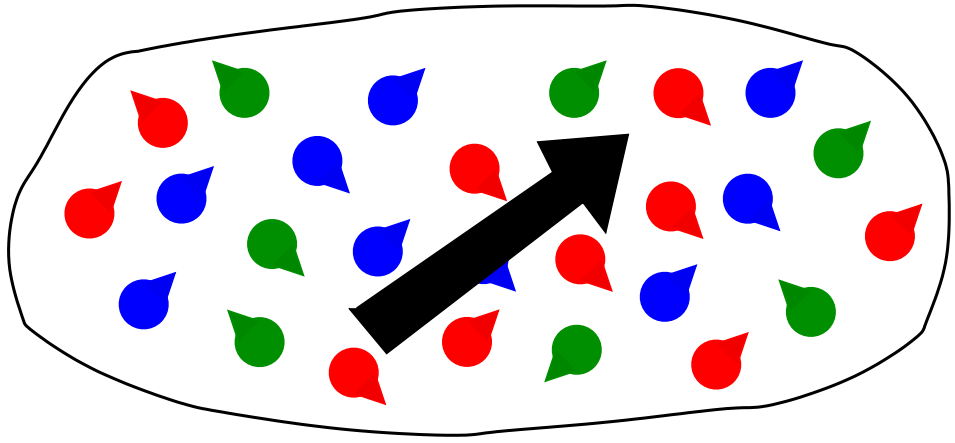
Each entity may have its own properties, objectives and actions.

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

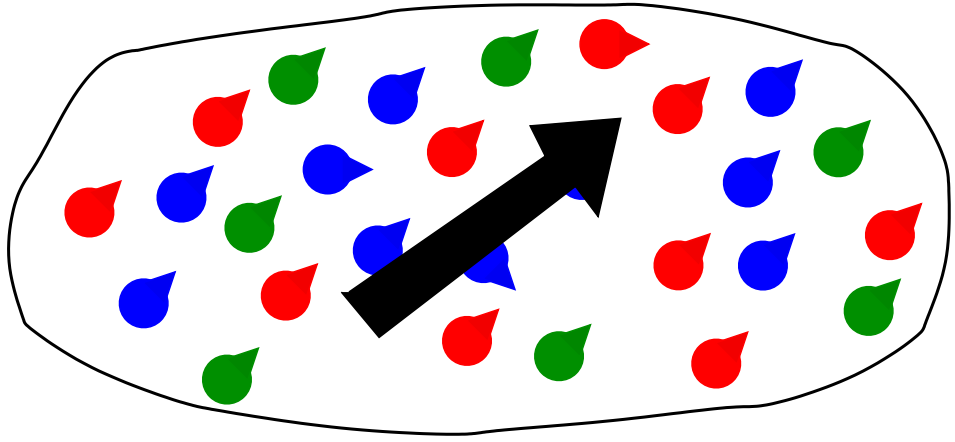
The behaviour of the system is thus dependent on the behaviour of the individual entities.



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

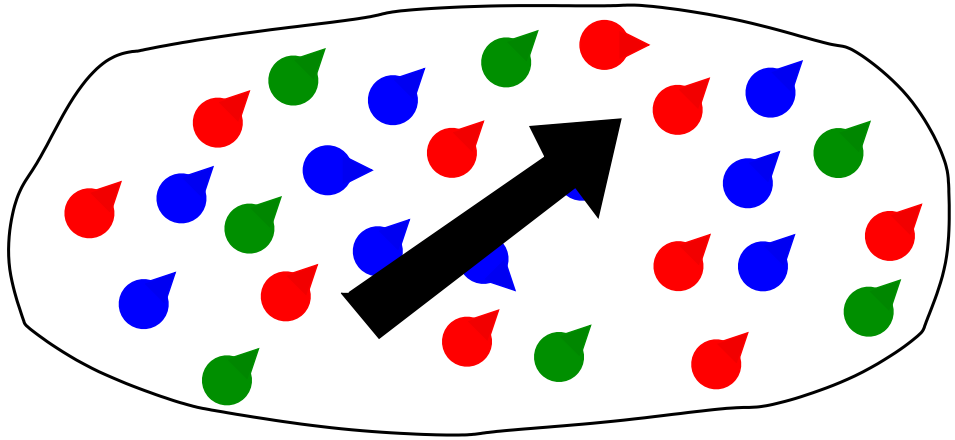


The behaviour of the system is thus dependent on the behaviour of the individual entities.

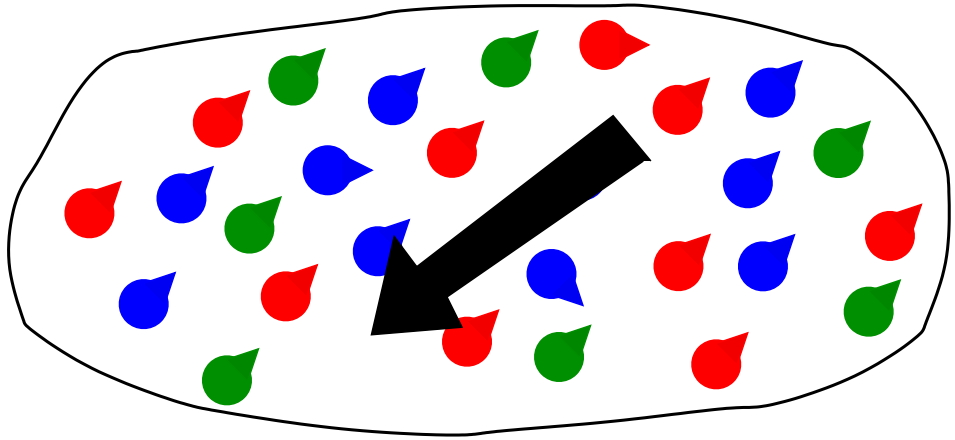


And the behaviour of the individuals will be influenced by the state of the overall system.

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

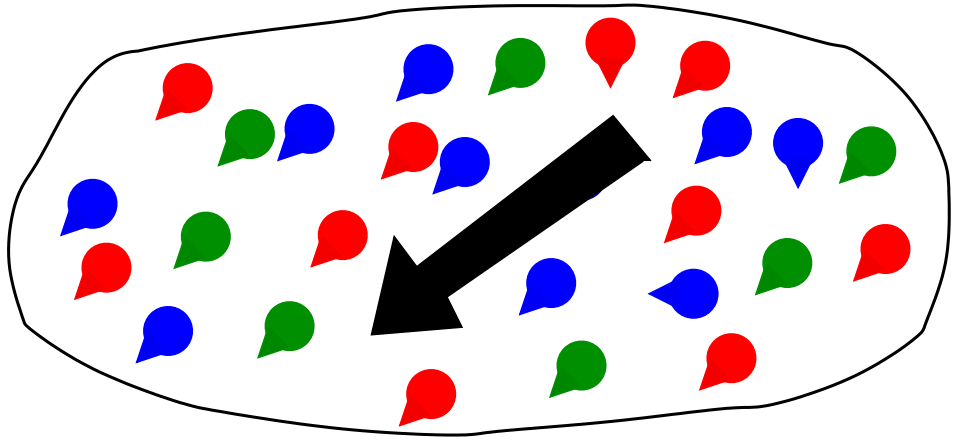


Such systems are often embedded in our environment and need to operate without centralised control or direction.



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

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 — pervasive computing elements embedded in the human environment, invisibly providing services and responding to requirements.

The Informatic Environment

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

The Informatic Environment

Robin Milner coined the term of informatics environment — pervasive computing elements 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 — pervasive computing elements 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.

Quantitative Modelling

Performance modelling aims to construct models of the dynamic behaviour of systems in order to support the fair and efficient sharing of resources.

Quantitative Modelling

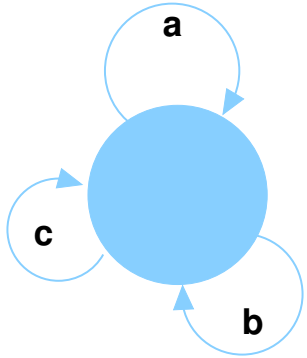
Performance modelling aims to construct models of the dynamic behaviour of systems in order to support the fair and efficient sharing of resources.

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

Quantitative Modelling

Performance modelling aims to construct models of the dynamic behaviour of systems in order to support the fair and efficient sharing of resources.

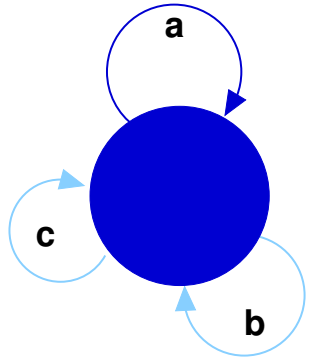
Markovian-based discrete event models have been applied to computer systems since the mid-1960s and communication systems since the early 20th century.



Quantitative Modelling

Performance modelling aims to construct models of the dynamic behaviour of systems in order to support the fair and efficient sharing of resources.

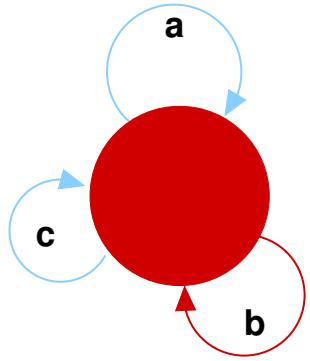
Markovian-based discrete event models have been applied to computer systems since the mid-1960s and communication systems since the early 20th century.



Quantitative Modelling

Performance modelling aims to construct models of the dynamic behaviour of systems in order to support the fair and efficient sharing of resources.

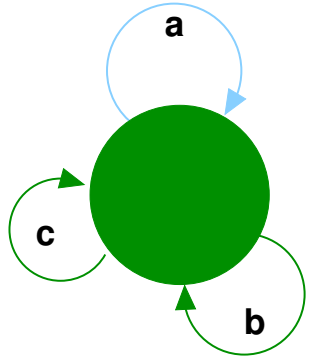
Markovian-based discrete event models have been applied to computer systems since the mid-1960s and communication systems since the early 20th century.



Quantitative Modelling

Performance modelling aims to construct models of the dynamic behaviour of systems in order to support the fair and efficient sharing of resources.

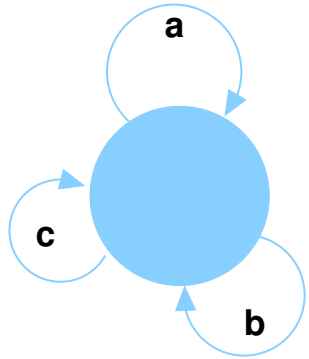
Markovian-based discrete event models have been applied to computer systems since the mid-1960s and communication systems since the early 20th century.



Quantitative Modelling

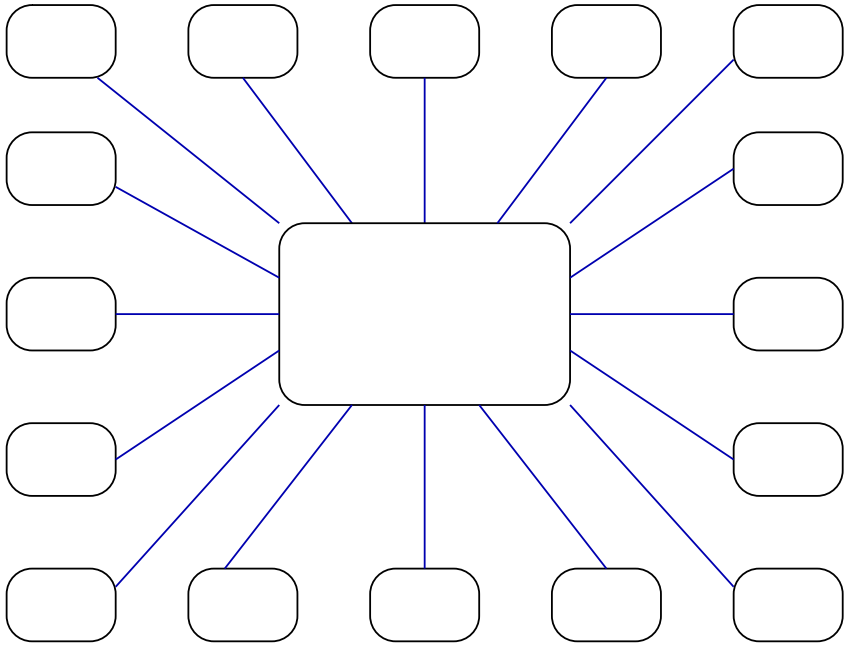
Performance modelling aims to construct models of the dynamic behaviour of systems in order to support the fair and efficient sharing of resources.

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



Various formalisms have been designed for capturing such behaviour.

Performance Modelling: Motivation



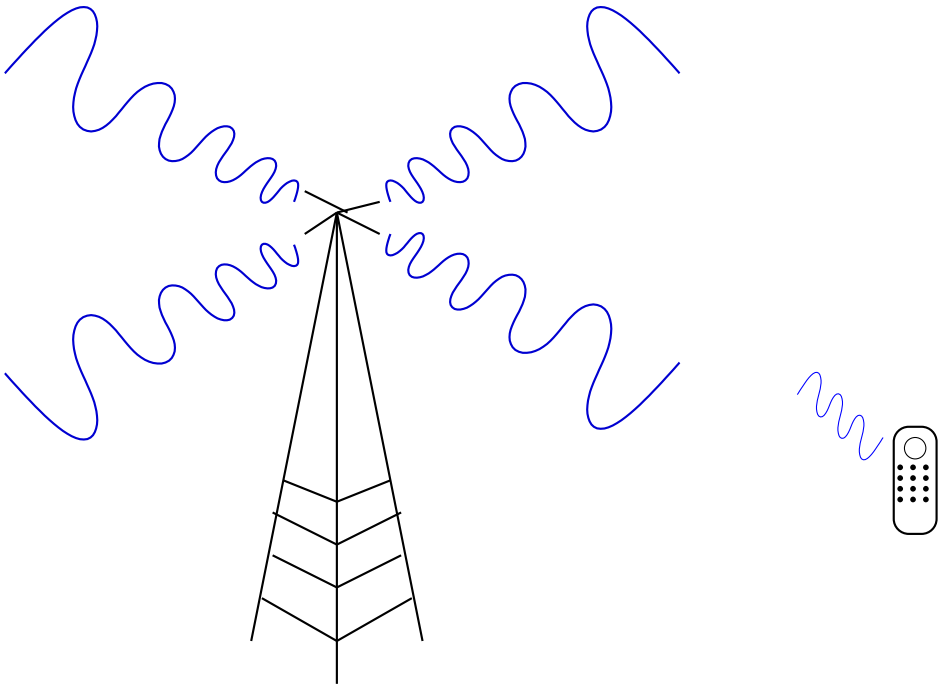
Capacity planning

- How many clients can the existing server support and maintain reasonable response times?

Performance Modelling: Motivation

Capacity planning

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



Mobile Telephone Antenna

System Configuration

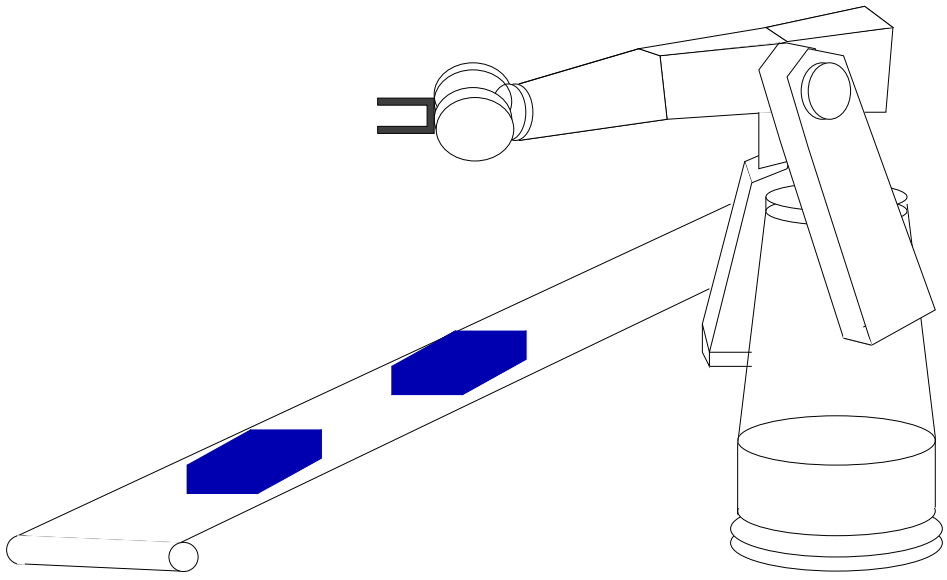
- How many frequencies do you need to keep blocking probabilities low?

Performance Modelling: Motivation

System Configuration

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

Performance Modelling: Motivation



System Tuning

- What speed of conveyor belt will minimize robot idle time and maximize throughput whilst avoiding lost widgets?

Performance Modelling: Motivation

System Tuning

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

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.

Primary examples include:

- Stochastic Petri Nets and
- Stochastic Process Algebras.

Stochastic Process Algebra

- Models are constructed from components which engage in activities.

Stochastic Process Algebra

- Models are constructed from components which engage in activities.
- Activities have a name and a rate.

Stochastic Process Algebra

- Models are constructed from components which engage in activities.
- Activities have a name and a rate.
- The rate defines an exponential distribution which means that the duration of an activity is a random variable.

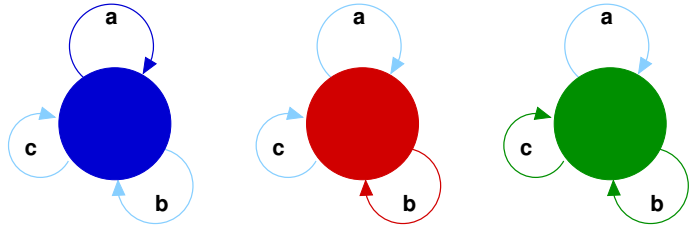
Stochastic Process Algebra

- Models are constructed from components which engage in activities.
- Activities have a name and a rate.
- The rate defines an exponential distribution which means that the duration of an activity is a random variable.
- A small set of language constructs determine how the model will evolve.

Stochastic Process Algebra

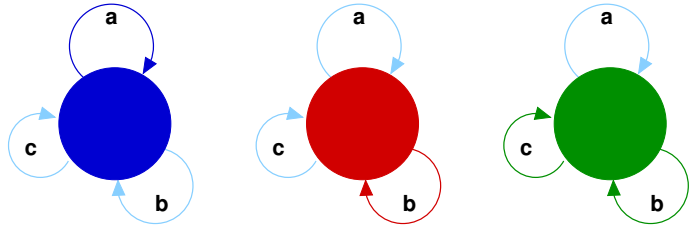
- Models are constructed from components which engage in activities.
- Activities have a name and a rate.
- The rate defines an exponential distribution which means that the duration of an activity is a random variable.
- A small set of language constructs determine how the model will evolve.
- The language is used to generate a CTMC for performance modelling (via the semantics).

Solving discrete state models

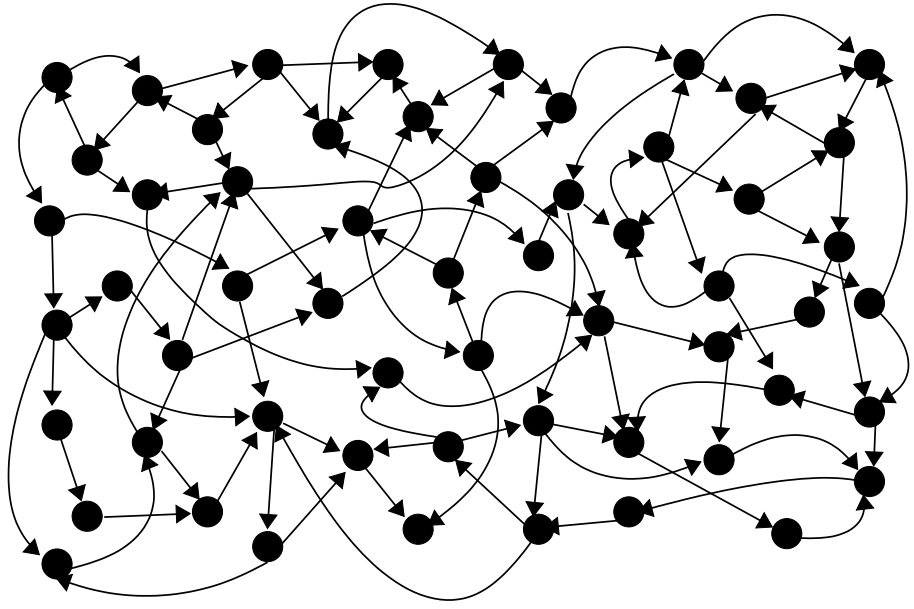


Using the semantics a SPA model is mapped to a CTMC with global states determined by the local states of all the participating components.

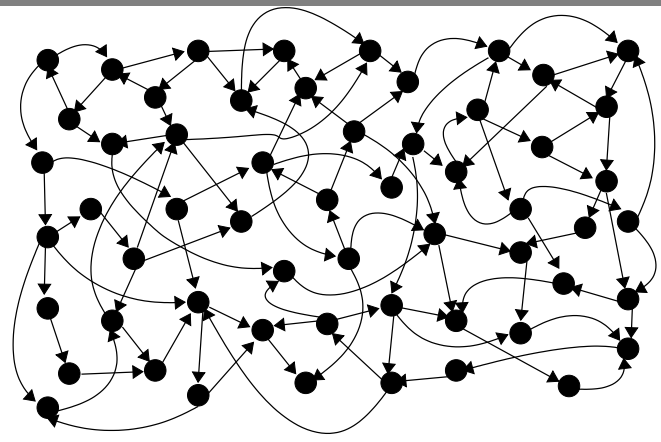
Solving discrete state models



Using the semantics a SPA model is mapped to a CTMC with global states determined by the local states of all the participating components.

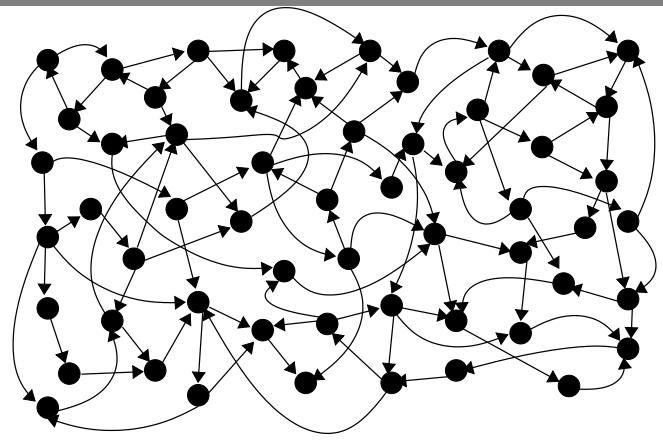


Solving discrete state models



When the size of the state space is not too large they are amenable to numerical solution (linear algebra) to determine a steady state or transient probability distribution.

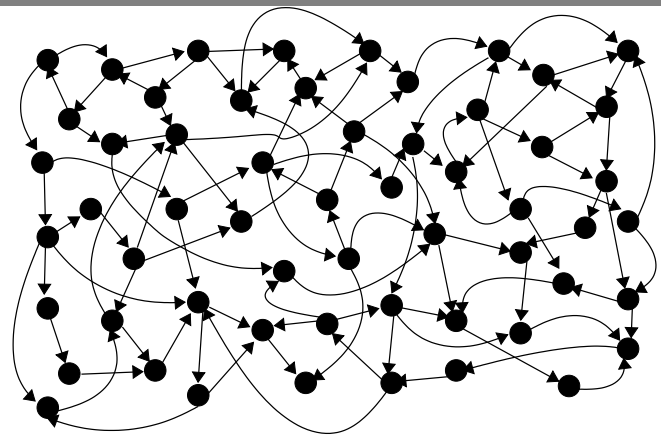
Solving discrete state models



When the size of the state space is not too large they are amenable to numerical solution (linear algebra) to determine a steady state or transient probability distribution.

$$Q = \begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,N} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,N} \\ \vdots & \vdots & & \vdots \\ q_{N,1} & q_{N,2} & \cdots & q_{N,N} \end{pmatrix}$$

Solving discrete state models



When the size of the state space is not too large they are amenable to numerical solution (linear algebra) to determine a steady state or transient probability distribution.

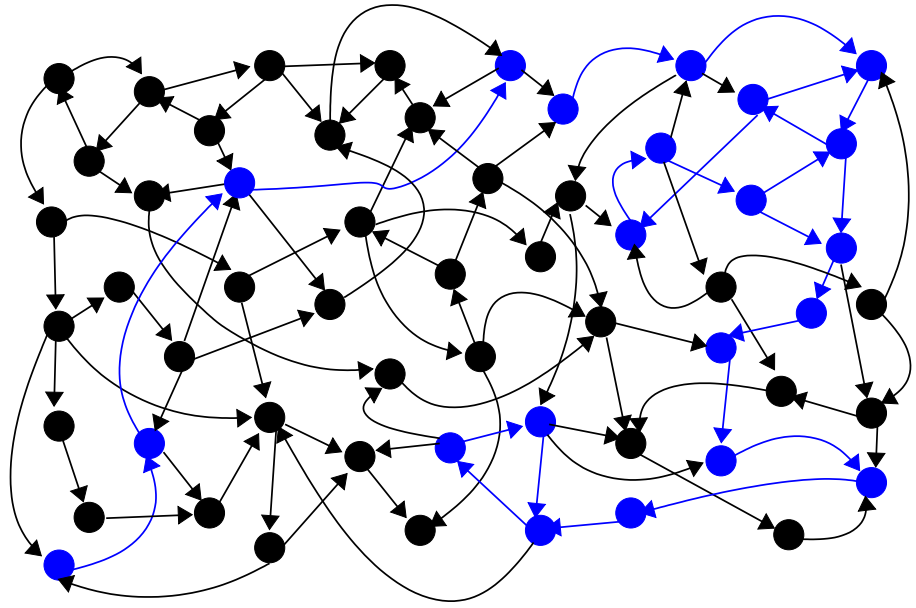
$$Q = \begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,N} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,N} \\ \vdots & \vdots & & \vdots \\ q_{N,1} & q_{N,2} & \cdots & q_{N,N} \end{pmatrix}$$

$$\pi(t) = (\pi_1(t), \pi_2(t), \ldots, \pi_N(t))$$

$$\pi(\infty)Q = 0$$

Solving discrete state models

Alternatively they may be studied using stochastic simulation. Each run generates a single trajectory through the state space. Many runs are needed in order to obtain average behaviours.



State space explosion

As the size of the state space becomes large it becomes infeasible to carry out numerical solution and extremely time-consuming to conduct stochastic simulation.

State space explosion

As the size of the state space becomes large it becomes infeasible to carry out numerical solution and extremely time-consuming to conduct stochastic simulation.

As we have seen in a previous lecture, when the population sizes become large we can make a mean-field approximation, approximating the average trajectory of the CTMC by a set of ordinary differential equations.

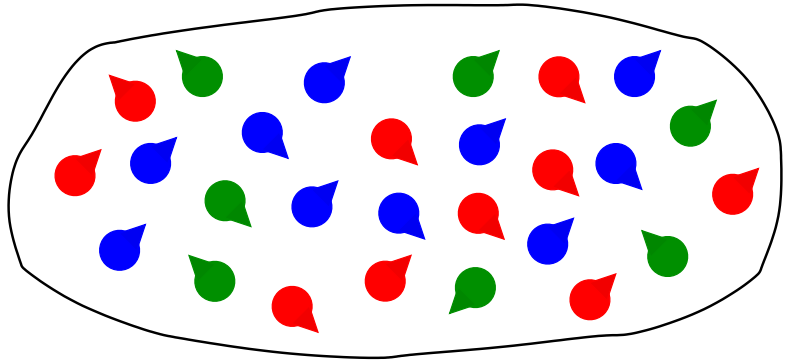
M.Tribastone, S.Gilmore and J.Hillston. Scalable Differential Analysis of Process Algebra Models. IEEE TSE 2012.

The Fluid Approximation Alternative

We can shift attention to the populations rather than the individual entities, and then consider the average behaviour within a population.

The Fluid Approximation Alternative

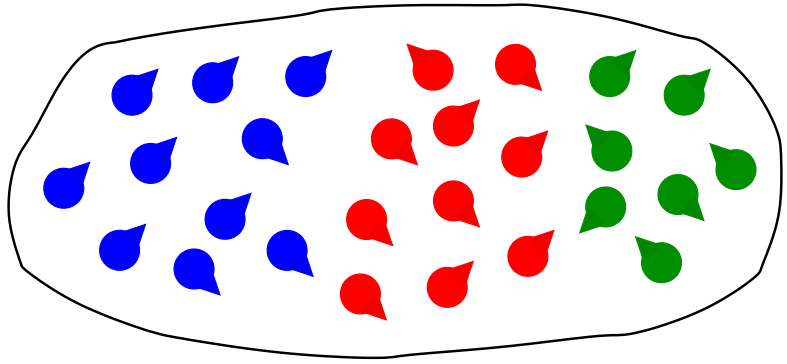
We can shift attention to the populations rather than the individual entities, and then consider the average behaviour within a population.



Ceasing to distinguish between instances of components we form an aggregation or counting abstraction to reduce the state space.

The Fluid Approximation Alternative

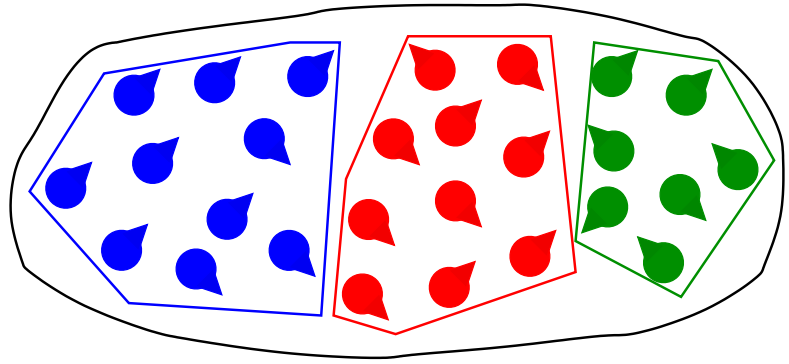
We can shift attention to the populations rather than the individual entities, and then consider the average behaviour within a population.



Ceasing to distinguish between instances of components we form an aggregation or counting abstraction to reduce the state space.

The Fluid Approximation Alternative

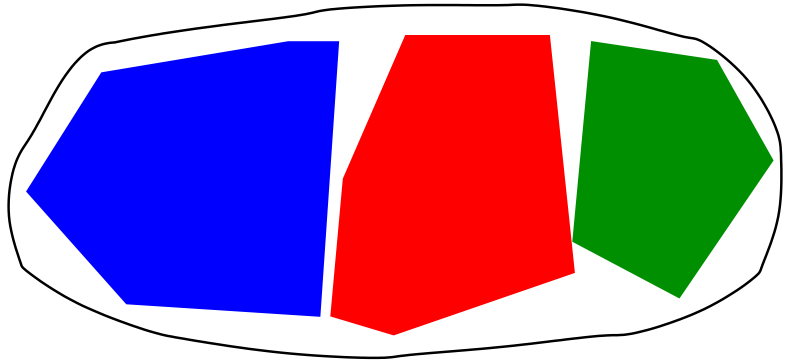
We can shift attention to the populations rather than the individual entities, and then consider the average behaviour within a population.



Ceasing to distinguish between instances of components we form an aggregation or counting abstraction to reduce the state space.
We now disregard the identity of components.

The Fluid Approximation Alternative

We can shift attention to the populations rather than the individual entities, and then consider the average behaviour within a population.



Ceasing to distinguish between instances of components we form an aggregation or counting abstraction to reduce the state space.
We now disregard the identity of components.

Even better reductions can be achieved when we no longer regard the components as individuals.

Outline

1 Introduction
- Collective Adaptive Systems
- Quantitative Analysis

2 Modelling CAS

3 CARMA
- The CARMA Modelling Language

4 CARMA Operational semantics

5 CASL:CARMA Specification Language

6 CARMA Eclipse plug-in

7 CARMA and its tools in Action

Challenges for modelling CAS

Work over the last twenty years on stochastic process algebra provides a solid basic framework for modelling CAS but there remain a number of challenges:

- Richer forms of interaction
- The influence of space on behaviour
- Capturing adaptivity

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.

But languages designed for other purposes typically contain too much detail to be used as the basis of quantitative modelling and analysis.

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 is preferable to model space explicitly although this poses significant challenges both for model expression and model solution.

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 is preferable to model space explicitly although this poses significant challenges both for model expression and model solution.

There is a tension with scalable analysis which is often based on an implicit assumption that all components are co-located.

Capturing adaptivity

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

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.

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

1 Introduction
- Collective Adaptive Systems
- Quantitative Analysis

2 Modelling CAS

3 CARMA
- The CARMA Modelling Language

4 CARMA Operational semantics

5 CASL:CARMA Specification Language

6 CARMA Eclipse plug-in

7 CARMA and its tools in Action

A new language for CAS

quanticol
www.quanticol.eu

The QUANTICOL project seeks to develop a coherent, integrated set of linguistic primitives, methods and tools to build systems that can operate in open-ended, unpredictable environments.

A new language for CAS

quanticol
www.quanticol.eu

The QUANTICOL project seeks to develop a coherent, integrated set of linguistic primitives, methods and tools to build systems that can operate in open-ended, unpredictable environments.

A new language for CAS

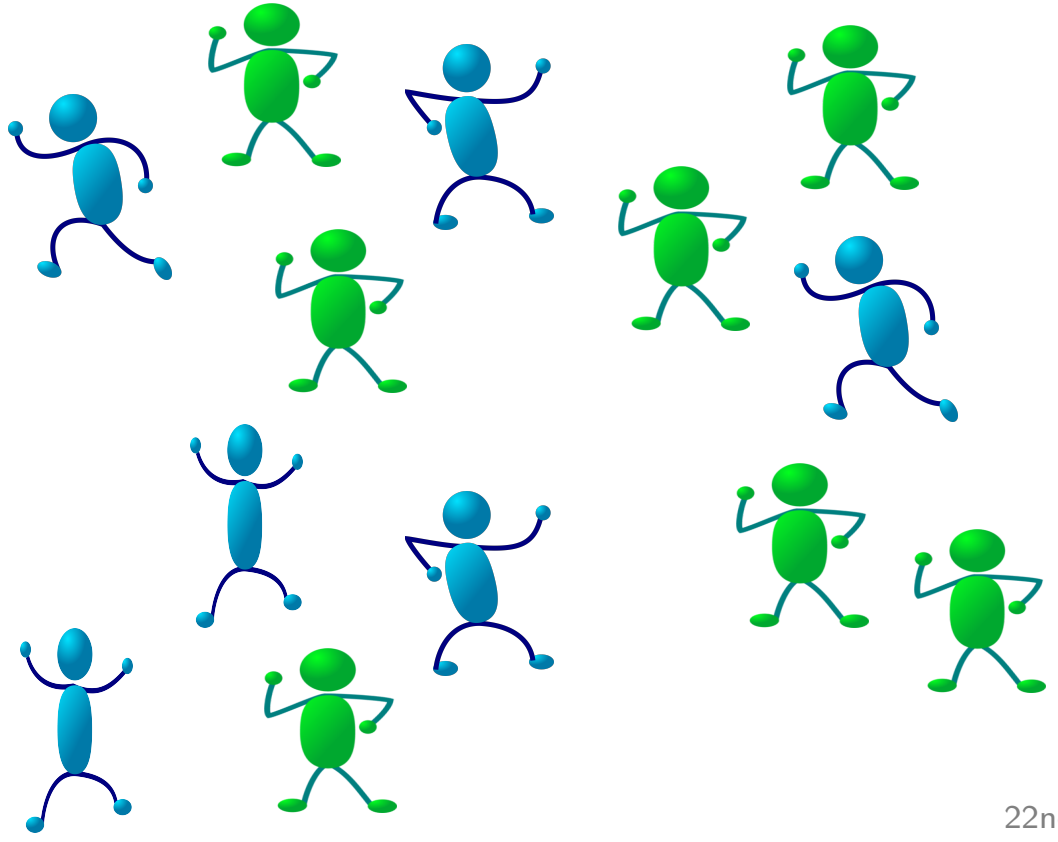
quanticol
www.quanticol.eu

The QUANTICOL project seeks to develop a coherent, integrated set of linguistic primitives, methods and tools to build systems that can operate in open-ended, unpredictable environments.

A new language for CAS

quanticol
www.quanticol.eu

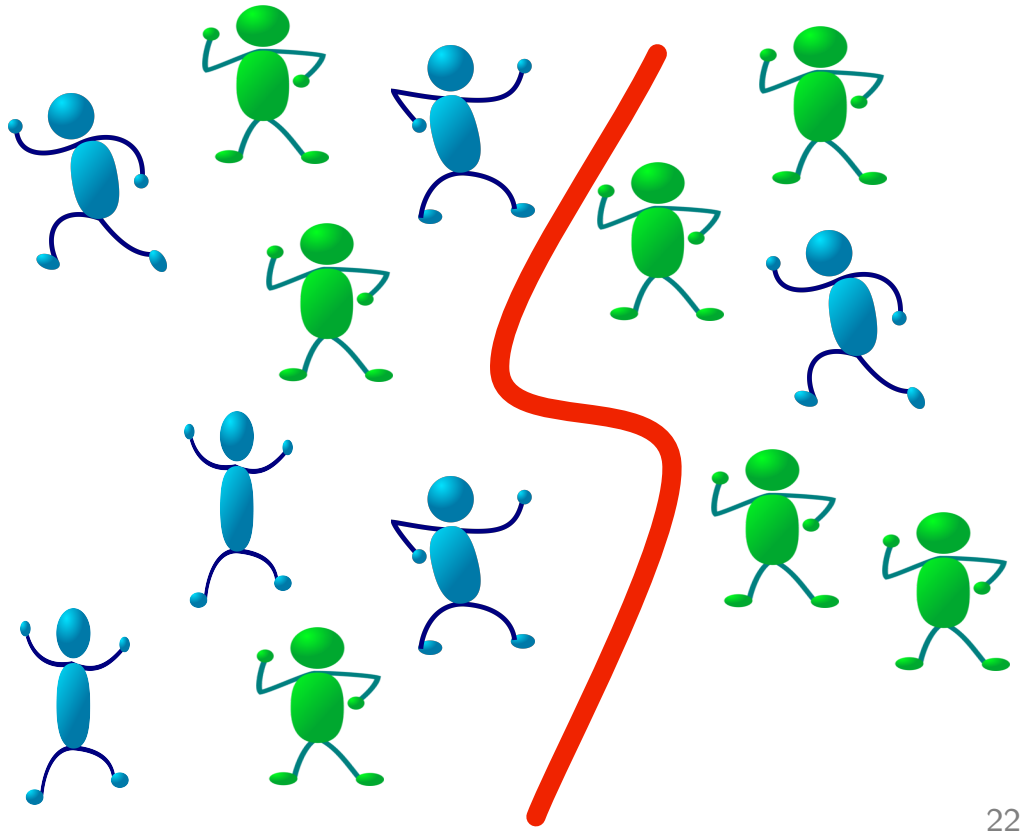
The QUANTICOL project seeks to develop a coherent, integrated set of linguistic primitives, methods and tools to build systems that can operate in open-ended, unpredictable environments.



A new language for CAS

quanticol
www.quanticol.eu

The QUANTICOL project seeks to develop a coherent, integrated set of linguistic primitives, methods and tools to build systems that can operate in open-ended, unpredictable environments.



CARMA

A key element of the QUANTICOL framework is the language, CARMA (Collective Adaptive Resource-sharing Markovian Agents), which handles:

CARMA

A key element of the QUANTICOL framework is the language, CARMA (Collective Adaptive Resource-sharing Markovian Agents), which handles:

1. The behaviours of agents and their interactions;

CARMA

A key element of the QUANTICOL framework is the language, CARMA (Collective Adaptive Resource-sharing Markovian Agents), which handles:

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

CARMA

A key element of the QUANTICOL framework is the language, CARMA (Collective Adaptive Resource-sharing Markovian Agents), 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...

CARMA

A key element of the QUANTICOL framework is the language, CARMA (Collective Adaptive Resource-sharing Markovian Agents), 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;

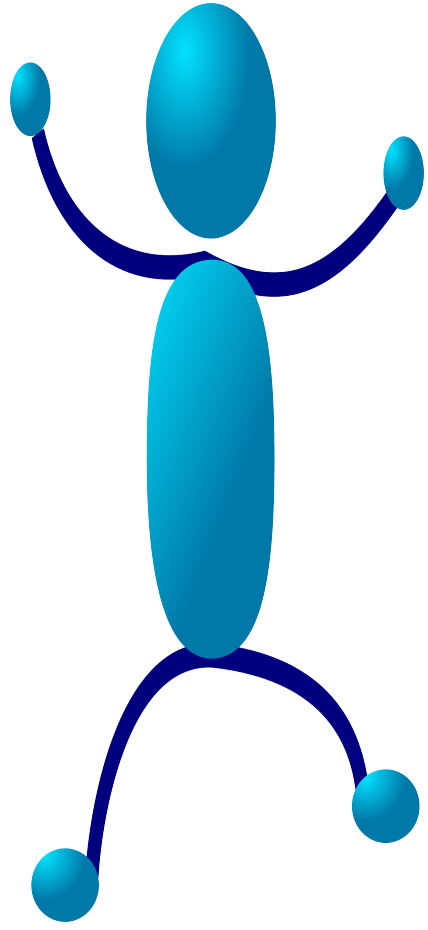
CARMA

A key element of the QUANTICOL framework is the language, CARMA (Collective Adaptive Resource-sharing Markovian Agents), which handles:

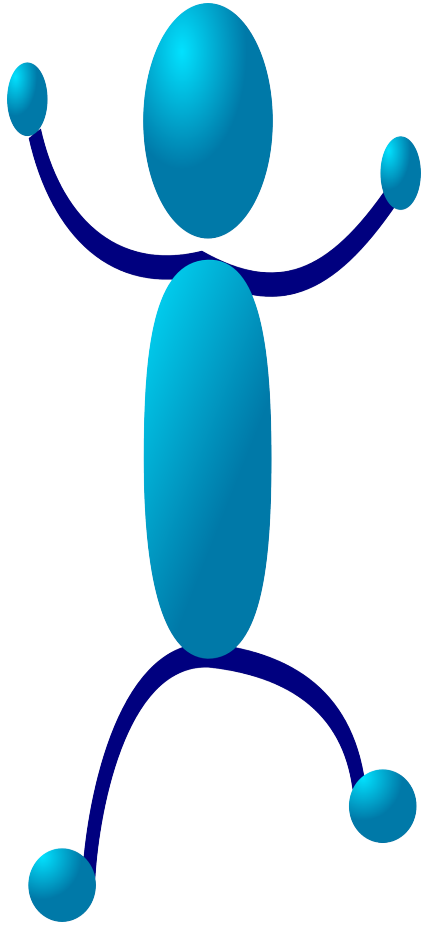
1. The behaviours of agents and their interactions;
2. The global knowledge of the system and that of its agents;
3. The environment where agents operate...
   - taking into account open ended-ness and adaptation;
   - taking into account resources, locations and visibility/reachability issues.

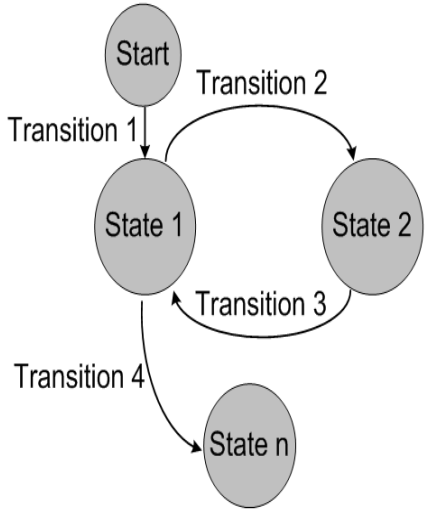
M.Loreti et al. CARMA: Collective Adaptive Resource-sharing Markovian Agents. QAPL 2015.

Agents in CARMA

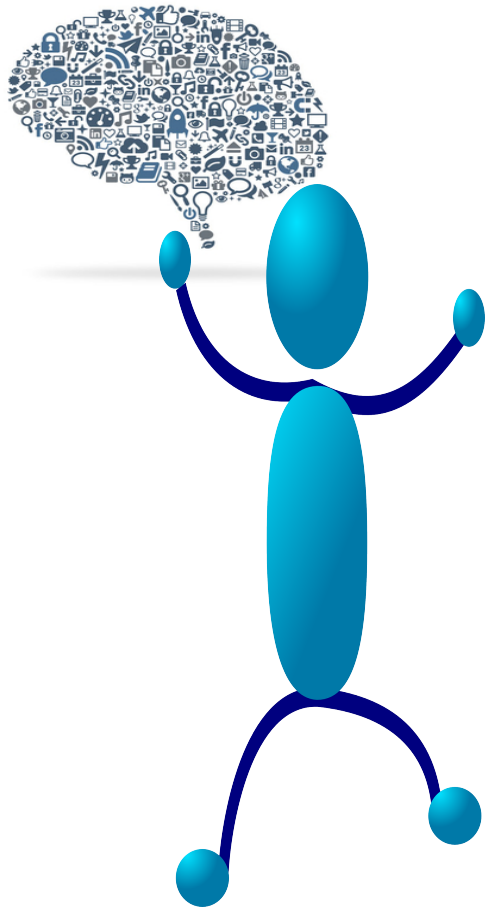


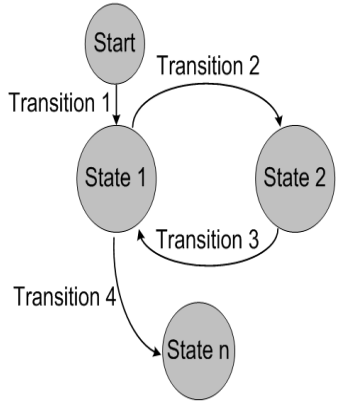
Agents in CARMA



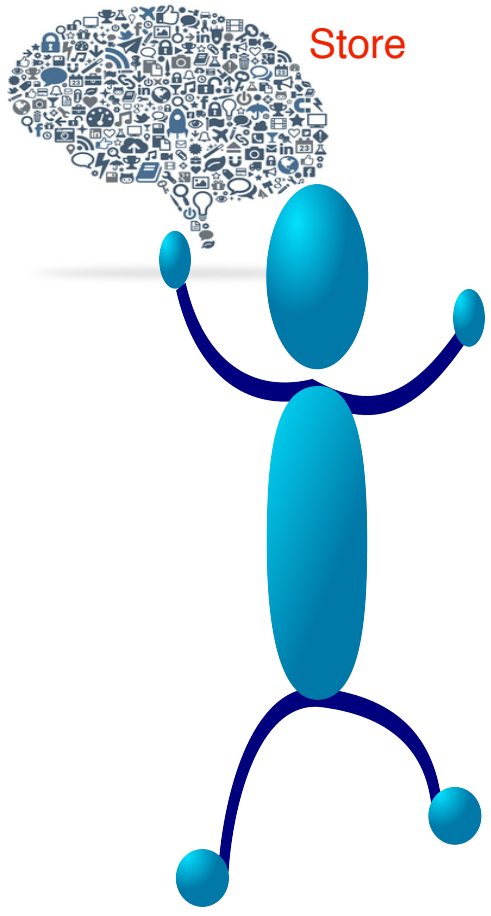


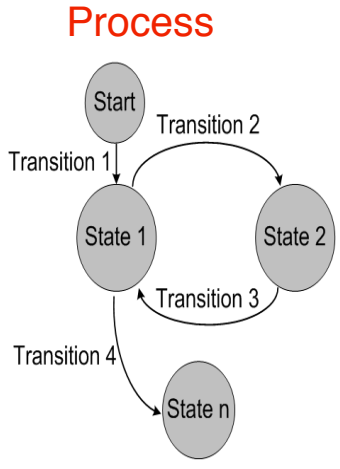
Agents in CARMA



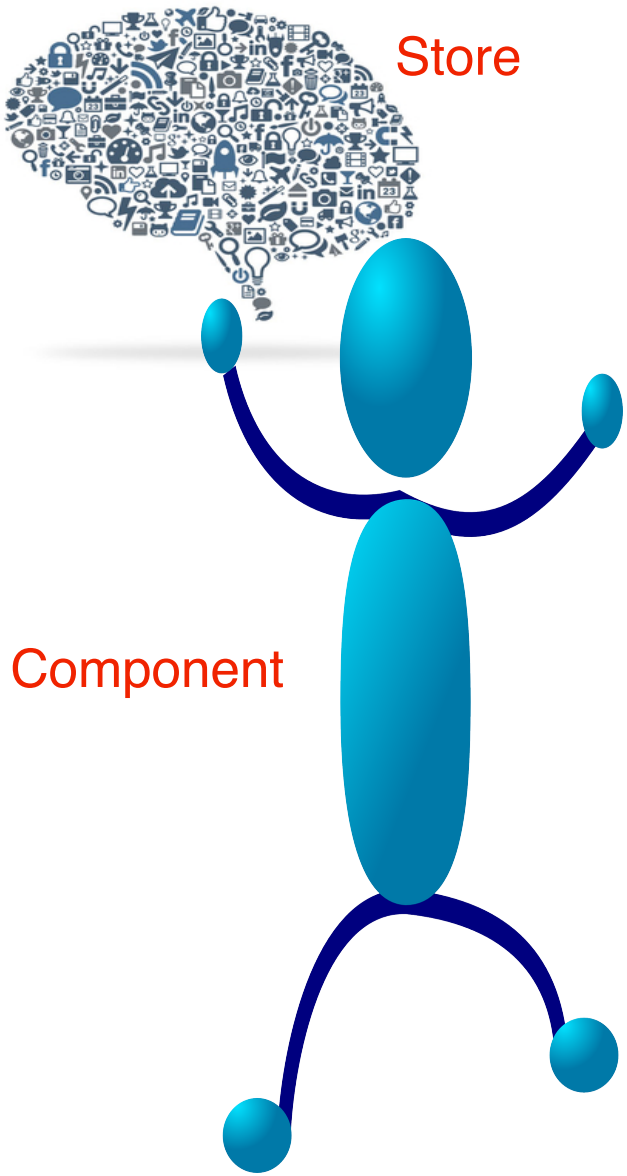


Agents in CARMA

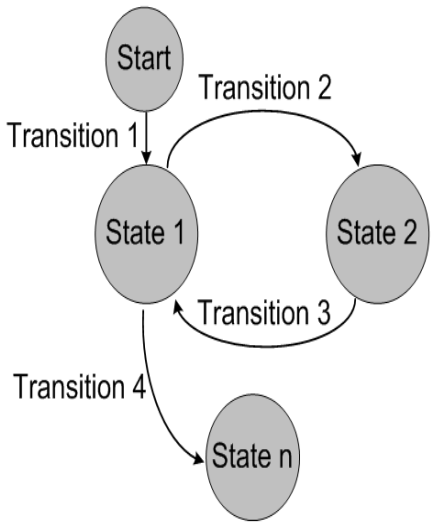




Components



Process



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...

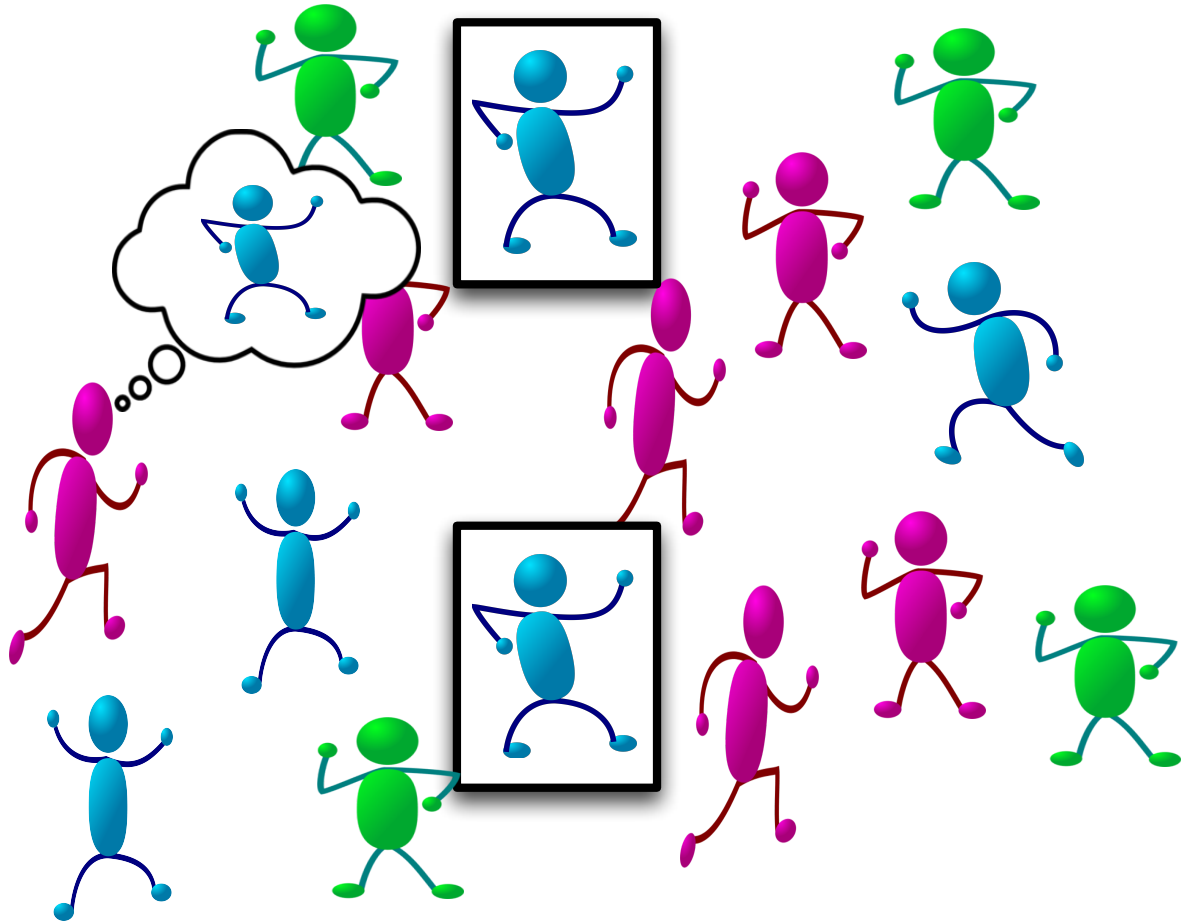
- P is a process, representing agent behaviour;
- γ is a store, modelling agent knowledge.

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

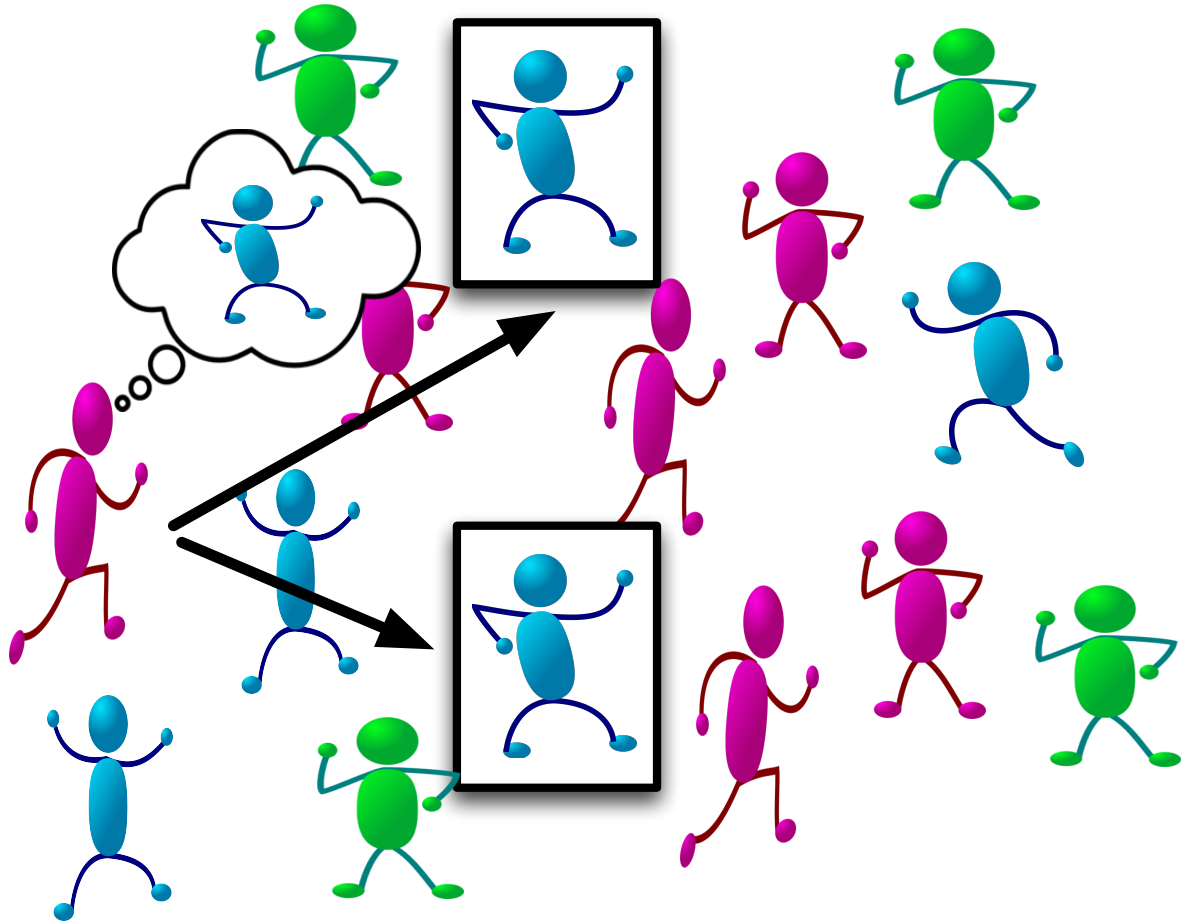
- the counterpart of a communication is selected according its properties

Attribute-based communication

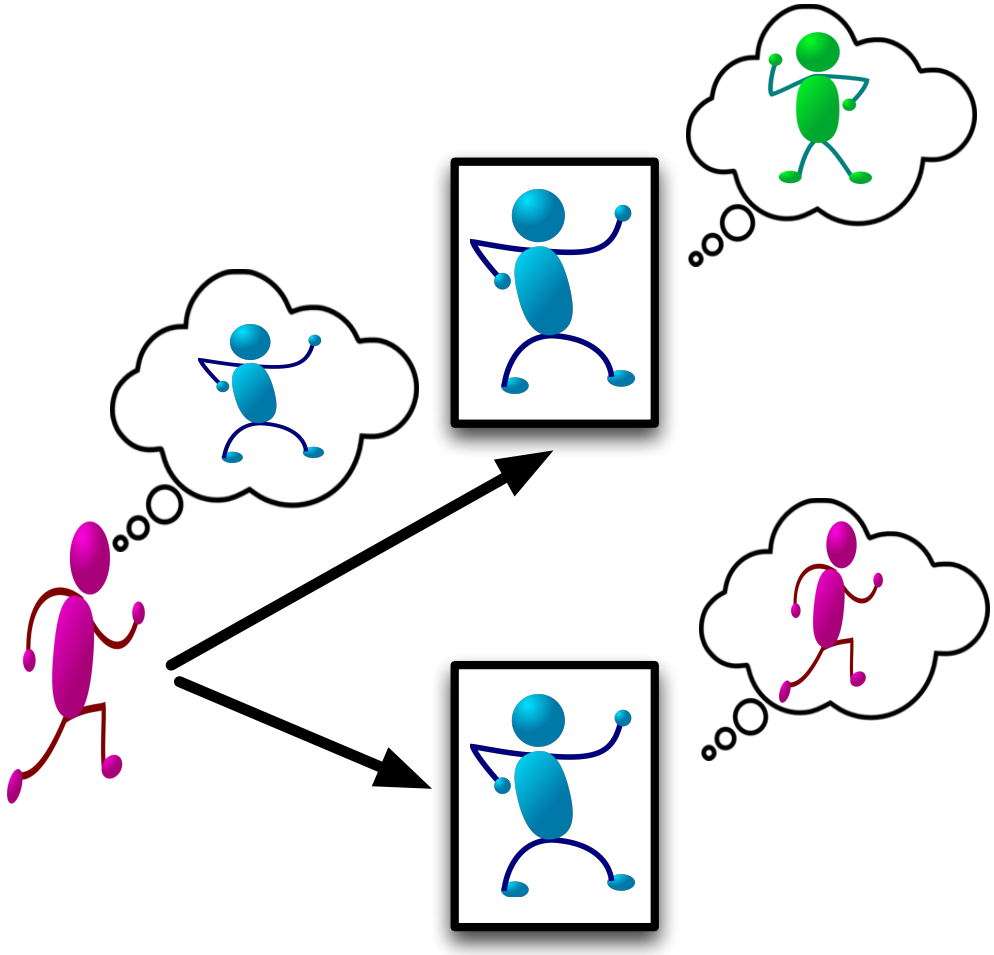
Attribute-based communication



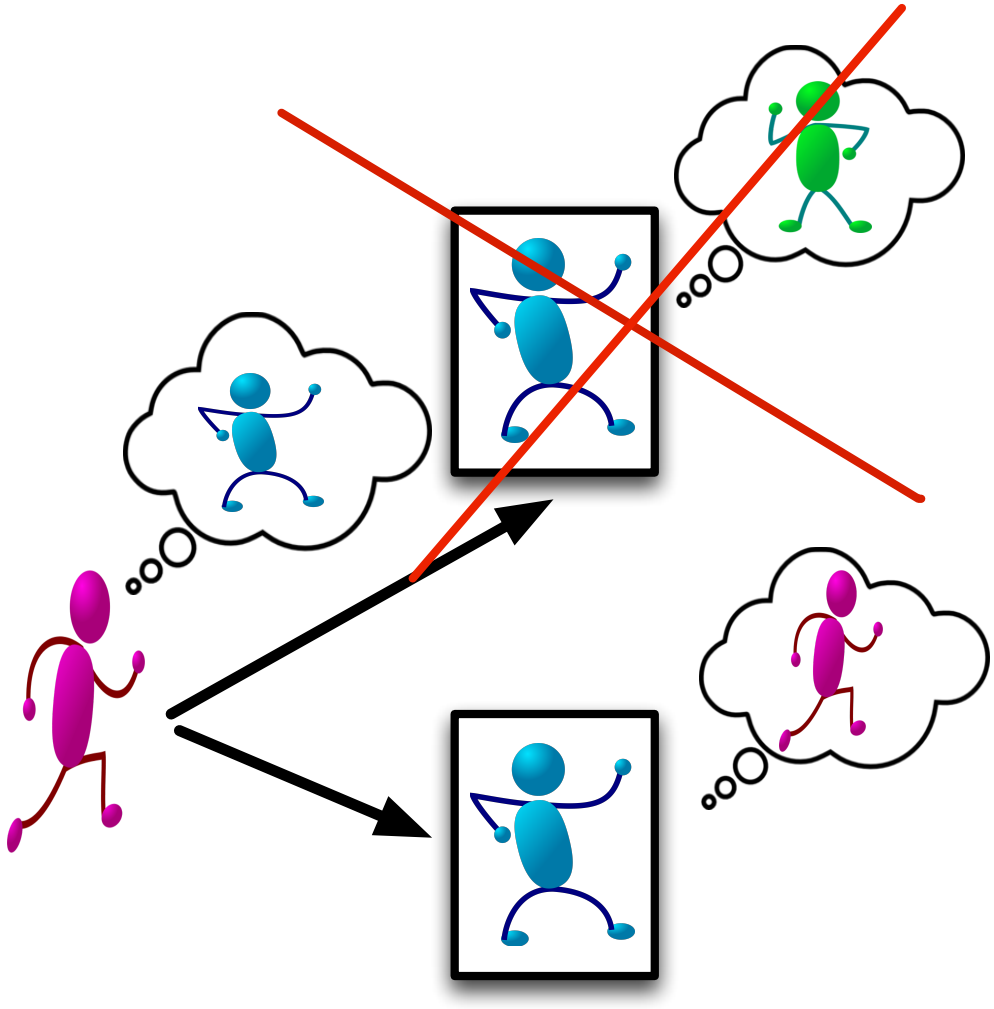
Attribute-based communication



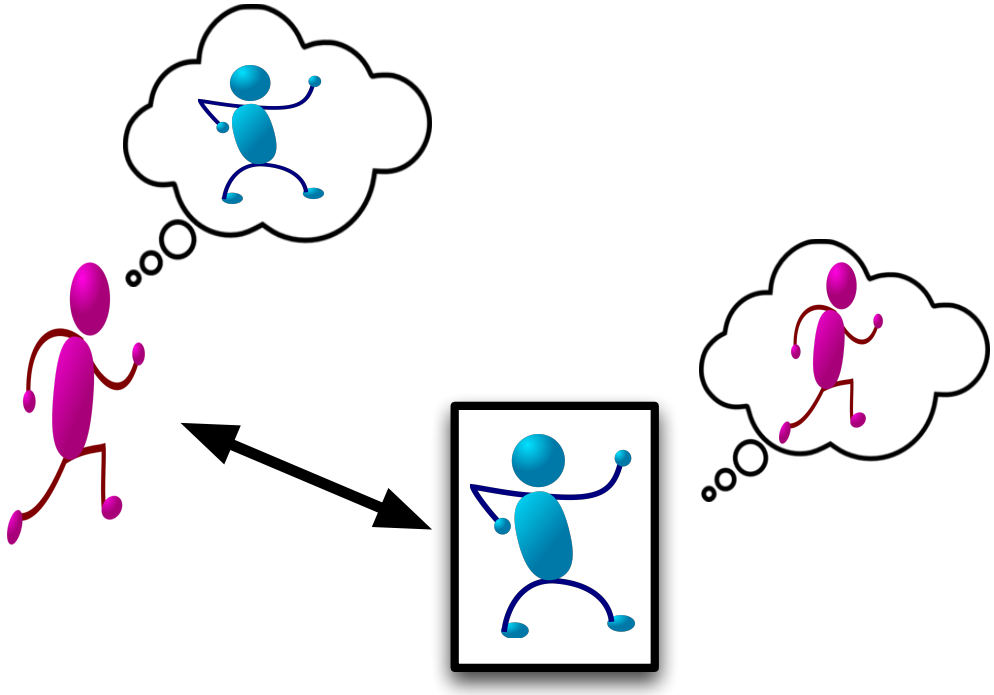
Attribute-based communication



Attribute-based communication



Attribute-based 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.

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

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

- α is an action type;

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

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

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

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

Simple example

$$
\begin{aligned}
\mathit{Lecturer} &= \mathit{teach}^{\star}[\mathit{awake} = \mathit{true}]\langle \mathit{fact} \rangle \{\} \ldots \\
&+ \mathit{coffee}^{\star}[\mathit{true}]\langle \cdot \rangle \{\mathit{boring} := \mathit{false}\} \ldots
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Student} &= \mathit{teach}^{\star}[\mathit{boring} = \mathit{false}](\mathit{fact})\{\mathit{know} := \mathit{know} + \mathit{fact}\} \ldots \\
&+ \mathit{coffee}^{\star}[\mathit{true}](\cdot)\{\mathit{awake} := \mathit{true}\} \ldots
\end{aligned}
$$

Simple example

$$
\begin{aligned}
\mathit{Lecturer} &= \mathit{teach}^{\star}[\mathit{awake} = \mathit{true}]\langle \mathit{fact} \rangle \{\} \ldots \\
&+ \mathit{coffee}^{\star}[\mathit{true}]\langle \cdot \rangle \{\mathit{boring} := \mathit{false}\} \ldots \\
\\
\mathit{Student} &= \mathit{teach}^{\star}[\mathit{boring} = \mathit{false}](\mathit{fact})\{\mathit{know} := \mathit{know} + \mathit{fact}\} \ldots \\
&+ \mathit{coffee}^{\star}[\mathit{true}](\cdot)\{\mathit{awake} := \mathit{true}\} \ldots
\end{aligned}
$$

Simple example

$$
\begin{aligned}
\mathit{Lecturer} &= \mathit{teach}^{\star}[\mathit{awake} = \mathit{true}]\langle \mathit{fact} \rangle \{\} \ldots \\
&+ \mathit{coffee}^{\star}[\mathit{true}]\langle \cdot \rangle \{\mathit{boring} := \mathit{false}\} \ldots
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Student} &= \mathit{teach}^{\star}[\mathit{boring} = \mathit{false}](\mathit{fact})\{\mathit{know} := \mathit{know} + \mathit{fact}\} \ldots \\
&+ \mathit{coffee}^{\star}[\mathit{true}](\cdot)\{\mathit{awake} := \mathit{true}\} \ldots
\end{aligned}
$$

Simple example

$$
\begin{aligned}
\mathit{Lecturer} &= \mathit{teach}^{\star}[\mathit{awake} = \mathit{true}]\langle \mathit{fact} \rangle\{\} \ldots \\
&+ \mathit{coffee}^{\star}[\mathit{true}]\langle \cdot \rangle\{\mathit{boring} := \mathit{false}\} \ldots
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Student} &= \mathit{teach}^{\star}[\mathit{boring} = \mathit{false}](\mathit{fact})\{\mathit{know} := \mathit{know} + \mathit{fact}\} \ldots \\
&+ \mathit{coffee}^{\star}[\mathit{true}](\cdot)\{\mathit{awake} := \mathit{true}\} \ldots
\end{aligned}
$$

Simple example

$$
\begin{aligned}
\mathit{Lecturer} &= \mathit{teach}^{\star}[\mathit{awake} = \mathit{true}]\langle \mathit{fact} \rangle \{\} \ldots \\
&+ \mathit{coffee}^{\star}[\mathit{true}]\langle \cdot \rangle \{\mathit{boring} := \mathit{false}\} \ldots
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Student} &= \mathit{teach}^{\star}[\mathit{boring} = \mathit{false}](\mathit{fact})\{\mathit{know} := \mathit{know} + \mathit{fact}\} \ldots \\
&+ \mathit{coffee}^{\star}[\mathit{true}](\cdot)\{\mathit{awake} := \mathit{true}\} \ldots
\end{aligned}
$$

Short (5 minute) Break!

Interaction patterns in CAS

Typically, CAS exhibit two kinds of interaction pattern:

Interaction patterns in CAS

Typically, CAS exhibit two kinds of interaction pattern:

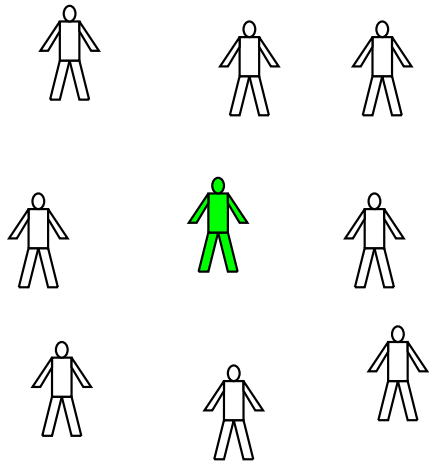
1. **Spreading**: one agent spreads relevant information to a given group of other agents

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

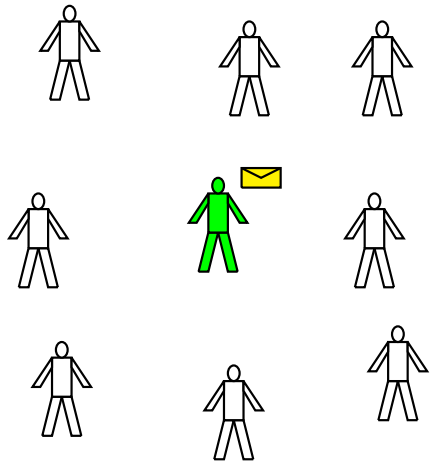


Interaction patterns in CAS

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

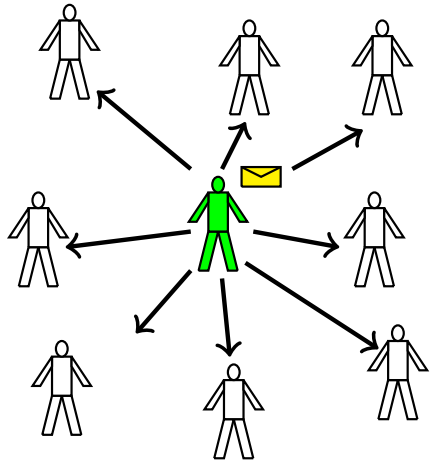


Interaction patterns in CAS

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

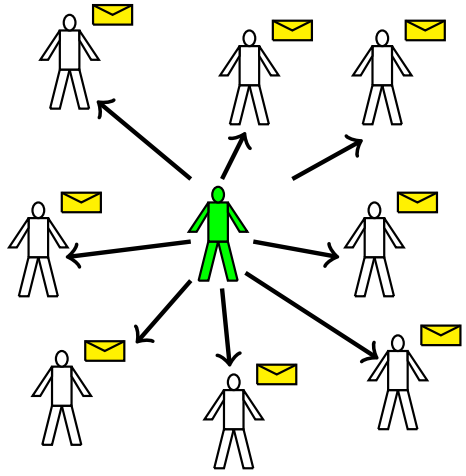


Interaction patterns in CAS

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

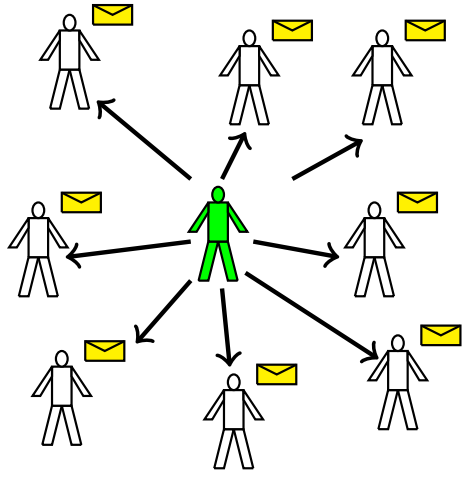


Interaction patterns in CAS

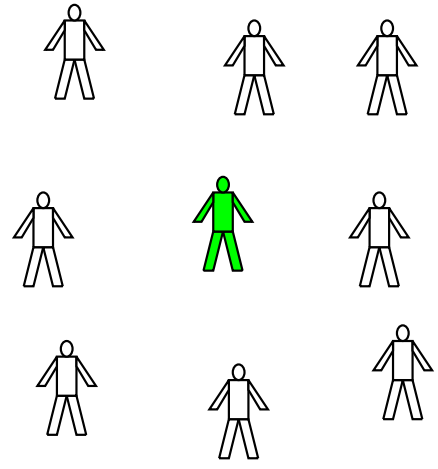
Typically, CAS exhibit two kinds of interaction pattern:

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

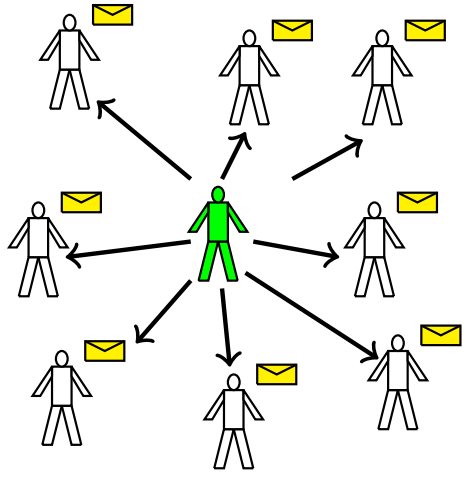


Interaction patterns in CAS

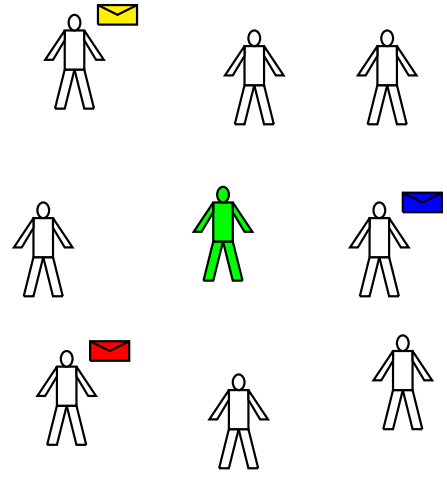
Typically, CAS exhibit two kinds of interaction pattern:

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

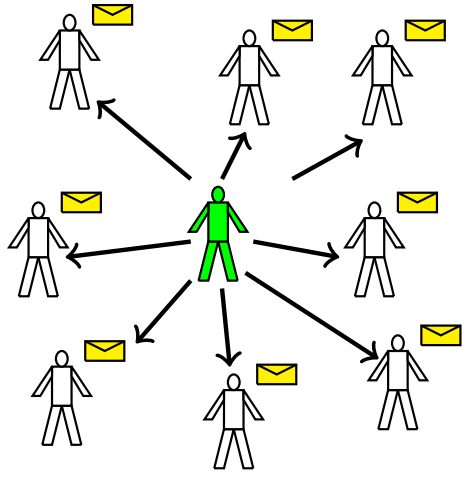


Interaction patterns in CAS

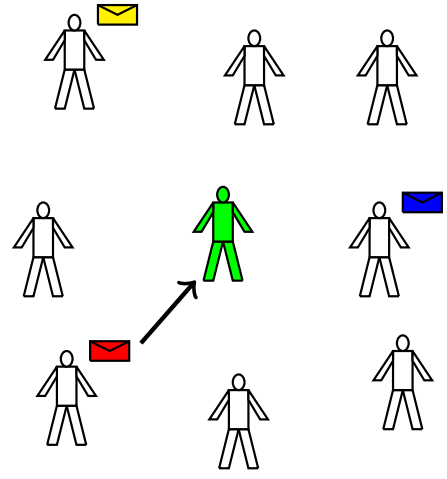
Typically, CAS exhibit two kinds of interaction pattern:

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

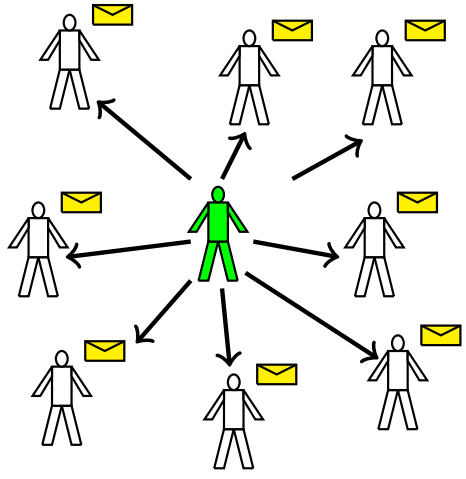


Interaction patterns in CAS

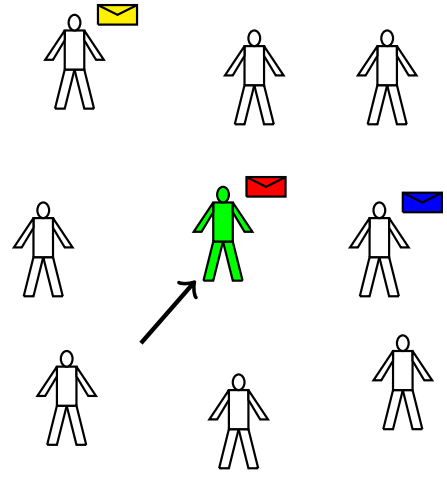
Typically, CAS exhibit two kinds of interaction pattern:

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.

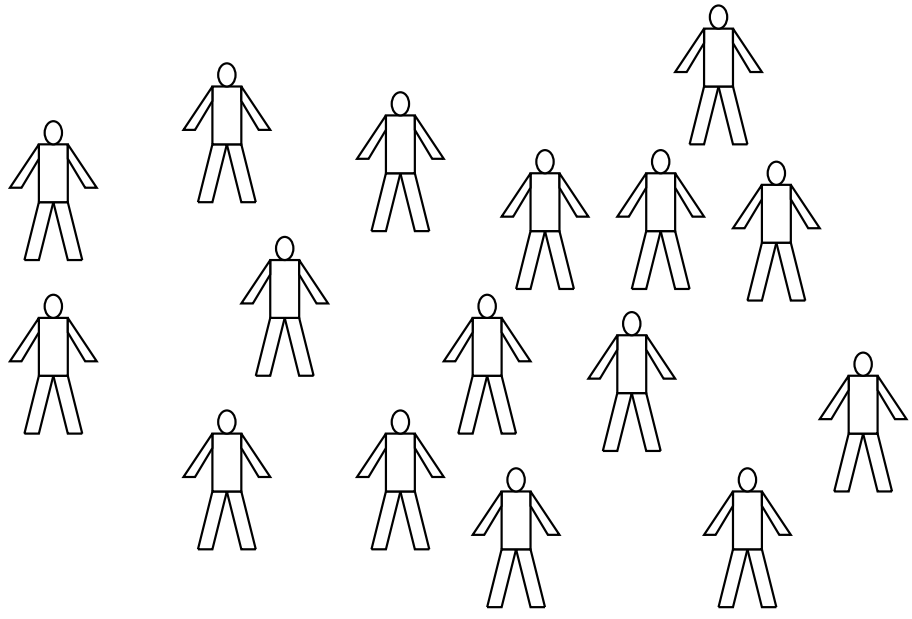
Spreading: 1-to-many



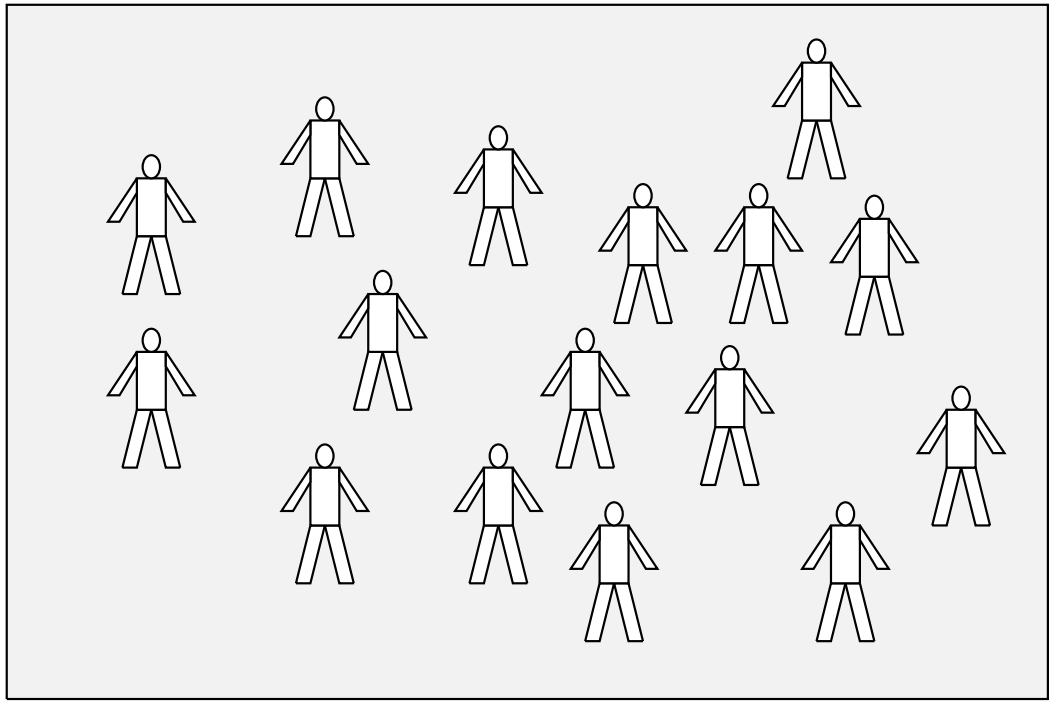
Collecting: 1-to-1



Collective

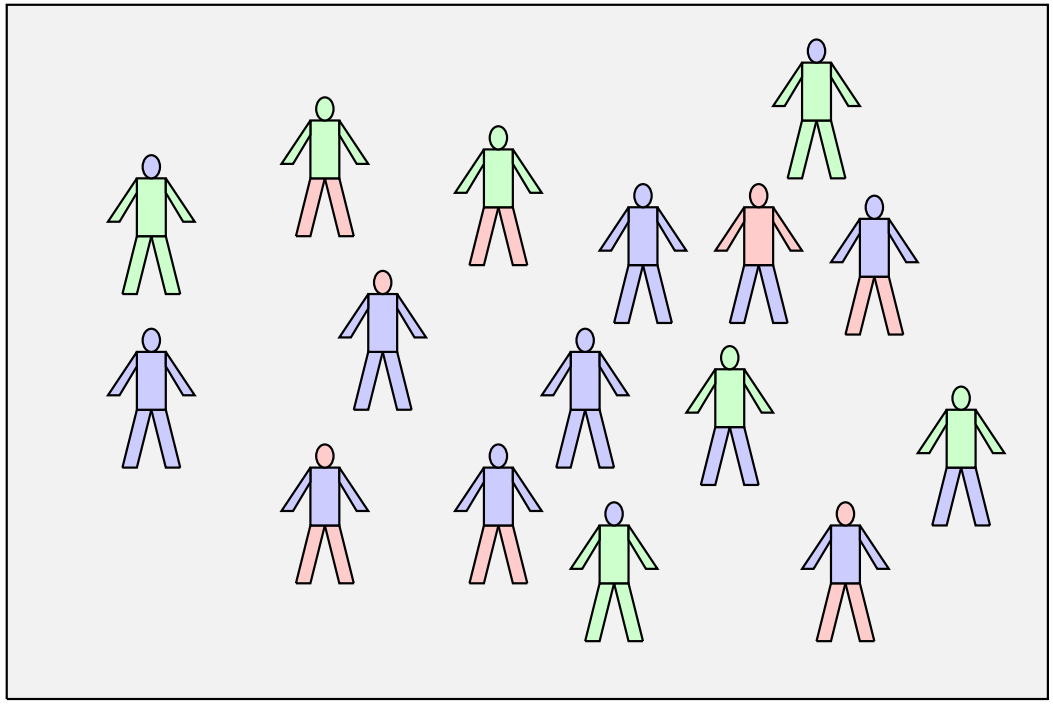


Collective **Environment**



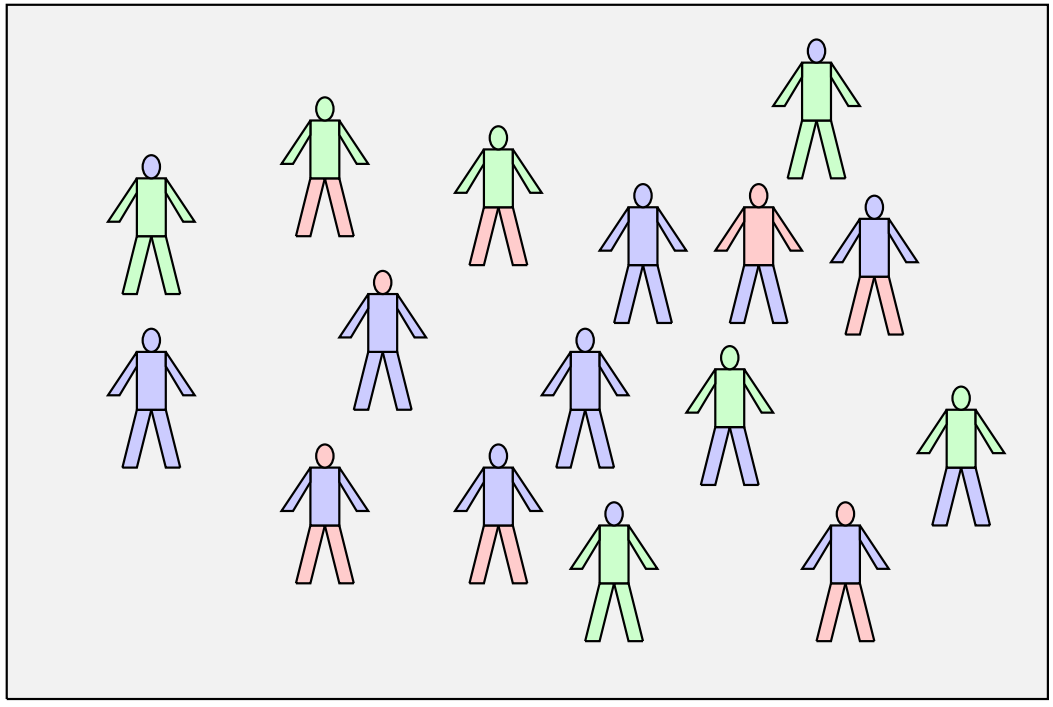
CAS: CARMA perspective

Collective **Environment** **Attributes**



CAS: CARMA perspective

Collective **Environment** **Attributes**



Processes are referenced via their attributes!

A CARMA system consists of

CARMA

A CARMA system consists of

- a collective (N)...

CARMA

A CARMA system consists of

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

CARMA

A CARMA system consists of

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

Collective...

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

CARMA

A CARMA system consists of

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

Collective...

- is composed by a set of components, i.e. the Markovian agents that compete and/or 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.

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

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

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

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

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:

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

Examples of interactions...

Broadcast synchronisation:

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

Examples of interactions...

Broadcast synchronisation:

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

Examples of interactions...

Broadcast synchronisation:

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

$$\Downarrow$$

$$
\begin{array}{l}
(P, \sigma_1(\{\mathit{role} = \text{“}\mathit{master}\text{”}\})) \parallel \\
\quad (Q_1[v/x], \sigma_2(\{\mathsf{bl} = 4\%\})) \parallel \\
\quad\quad (\mathsf{stop}^{\star}[\mathsf{role} = \text{“}\mathit{super}\text{”}](x)\sigma_3.Q_2, \{\mathsf{bl} = 2\%\}) \parallel \\
\quad\quad\quad\quad (Q_3[v/x], \sigma_4(\{\mathsf{bl} = 2\%\}))
\end{array}
$$

Examples of interactions...

Broadcast synchronisation:

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

Examples of interactions...

Broadcast synchronisation:

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

Examples of interactions...

Broadcast synchronisation:

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

$$\Downarrow$$

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

Examples of interactions...

Unicast synchronisation:

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

Examples of interactions...

Unicast synchronisation:

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

Examples of interactions...

Unicast synchronisation:

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

Examples of interactions...

Unicast synchronisation:

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

$$\Downarrow$$

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

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 models the overall state of the system;

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 models the overall state of the system;
  - an evolution rule ρ that regulates component interactions (receiving probabilities, action rates,...).

The evolution rule

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

The evolution rule

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 to do include any information about the timing (unlike many other stochastic process algebras).

The evolution rule

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 to do 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 evolution rule

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 to do 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 γ

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 on the environment (global store and collective) induced by the execution of action α at a component with store γ_s.

Outline

1 Introduction
- Collective Adaptive Systems
- Quantitative Analysis

2 Modelling CAS

3 CARMA
- The CARMA Modelling Language

4 CARMA Operational semantics

5 CASL:CARMA Specification Language

6 CARMA Eclipse plug-in

7 CARMA and its tools in Action

Operational Semantics of PA

Behaviour of process algebras is classically represented via transition relations, e.g. $\alpha.P \xrightarrow{\alpha} P$.

Operational Semantics of PA

Behaviour of process algebras is classically represented via transition relations, e.g. $\alpha.P \xrightarrow{\alpha} P$.

These relations, defined following a Plotkin-style, are used to infer possible computations of a process.

$$\frac{\textit{premise}}{\textit{conclusion}}\ \textbf{Rule}$$

Operational Semantics of PA

Behaviour of process algebras is classically represented via transition relations, e.g. $\alpha.P \xrightarrow{\alpha} P$.

These relations, defined following a Plotkin-style, are used to infer possible computations of a process.

$$\frac{\textit{premise}}{\textit{conclusion}}\ \textbf{Rule}$$

Note that, due to nondeterminism, starting from the same process, different evolutions can be inferred.

Operational Semantics of PA

Behaviour of process algebras is classically represented via transition relations, e.g. $\alpha.P \xrightarrow{\alpha} P$.

These relations, defined following a Plotkin-style, are used to infer possible computations of a process.

$$\frac{\textit{premise}}{\textit{conclusion}}\ \textbf{Rule}$$

Note that, due to nondeterminism, starting from the same process, different evolutions can be inferred.

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}\ \textbf{Choice1} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}\ \textbf{Choice2}$$

However, in Stochastic Process Algebras, like CARMA, there is not any form of nonterminism...

Operational Semantics of SPA

However, in Stochastic Process Algebras, like CARMA, there is not any form of nonterminism...

...while the selection of possible next state is governed by a probability distribution.

However, in Stochastic Process Algebras, like CARMA, there is not any form of nonterminism...

...while the selection of possible next state is governed by a probability distribution.

Standard compositional approaches are cumbersome and may fail when rich SPA are considered (e.g., when the multiplicity of transitions is important).

Operational Semantics of SPA

The operational semantics of CARMA is defined in the FuTS style.

Operational Semantics of SPA

The operational semantics of CARMA is defined in the FuTS style.

In FuTS the behaviour of a term is described using a function that, given a term and a transition label, yields a function associating each component, collective, or system with a non-negative number.

Operational Semantics of SPA

The operational semantics of CARMA is defined in the FuTS style.

In FuTS the behaviour of a term is described using a function that, given a term and a transition label, yields a function associating each component, collective, or system with a non-negative number.

The meaning of this value depends on the context:

- the rate of the exponential distribution characterising the time needed for the execution of an action;

Operational Semantics of SPA

The operational semantics of CARMA is defined in the FuTS style.

In FuTS the behaviour of a term is described using a function that, given a term and a transition label, yields a function associating each component, collective, or system with a non-negative number.

The meaning of this value depends on the context:

- the rate of the exponential distribution characterising the time needed for the execution of an action;
- the probability of receiving a given broadcast message;

Operational Semantics of SPA

The operational semantics of CARMA is defined in the FuTS style.

In FuTS the behaviour of a term is described using a function that, given a term and a transition label, yields a function associating each component, collective, or system with a non-negative number.

The meaning of this value depends on the context:

- the rate of the exponential distribution characterising the time needed for the execution of an action;
- the probability of receiving a given broadcast message;
- the weight used to compute the probability that a given component is selected for the synchronisation.

The operational semantics of CARMA specifications is defined in terms of three functions that compute the possible next states of a component, a collective and a system:

The operational semantics of CARMA specifications is defined in terms of three functions that compute the possible next states of a component, a collective and a system:

1. the function $\mathbb{C}$ that describes the behaviour of a single component;

The operational semantics of CARMA specifications is defined in terms of three functions that compute the possible next states of a component, a collective and a system:

1. the function $\mathbb{C}$ that describes the behaviour of a single component;
2. the function $\mathbb{N}_{\varepsilon}$ builds on $\mathbb{C}$ to describe the behaviour of collectives;

The operational semantics of CARMA specifications is defined in terms of three functions that compute the possible next states of a component, a collective and a system:

1. the function $\mathbb{C}$ that describes the behaviour of a single component;
2. the function $\mathbb{N}_{\varepsilon}$ builds on $\mathbb{C}$ to describe the behaviour of collectives;
3. the function $\mathbb{S}_t$ that shows how CARMA systems evolve.

The operational semantics of CARMA specifications is defined in terms of three functions that compute the possible next states of a component, a collective and a system:

1. the function $\mathbb{C}$ that describes the behaviour of a single component;
2. the function $\mathbb{N}_{\varepsilon}$ builds on $\mathbb{C}$ to describe the behaviour of collectives;
3. the function $\mathbb{S}_t$ that shows how CARMA systems evolve.

In all cases the value zero is associated with unreachable terms.

Component Semantics

The behaviour of a single component is defined by a function

$$\mathbb{C} : \textsc{Comp} \times \textsc{Lab} \to [\textsc{Comp} \to \mathbb{R}_{\geq 0}]$$

Component Semantics

The behaviour of a single component is defined by a function

$$\mathbb{C} : \textsc{Comp} \times \textsc{Lab} \to [\textsc{Comp} \to \mathbb{R}_{\geq 0}]$$

where $\textsc{Lab}$ denotes the set of transition labels ℓ:

$$\begin{array}{rcll} \ell & ::= & \alpha^\star[\pi_s]\langle\overrightarrow{v}\rangle, \gamma & \text{Broadcast Output} \\ & | & \alpha^\star[\pi_s](\overrightarrow{v}), \gamma & \text{Broadcast Input} \\ & | & \alpha[\pi_s]\langle\overrightarrow{v}\rangle, \gamma & \text{Unicast Output} \\ & | & \alpha[\pi_s](\overrightarrow{v}), \gamma & \text{Unicast Input} \end{array}$$

Component Semantics

The behaviour of a single component is defined by a function

$$\mathbb{C} : \textsc{Comp} \times \textsc{Lab} \to [\textsc{Comp} \to \mathbb{R}_{\geq 0}]$$

where $\textsc{Lab}$ denotes the set of transition labels ℓ:

$$\begin{array}{rcll} \ell & ::= & \alpha^\star[\pi_s]\langle\overrightarrow{v}\rangle, \gamma & \text{Broadcast Output} \\ & | & \alpha^\star[\pi_s](\overrightarrow{v}), \gamma & \text{Broadcast Input} \\ & | & \alpha[\pi_s]\langle\overrightarrow{v}\rangle, \gamma & \text{Unicast Output} \\ & | & \alpha[\pi_s](\overrightarrow{v}), \gamma & \text{Unicast Input} \end{array}$$

If $\mathbb{C}[C, \ell] = \mathscr{C}$ and $\mathscr{C}(C') = p$ then C evolves to C' with a weight p when ℓ is executed.

$$\frac{}{\mathbb{C}[(\mathbf{nil},\gamma),\ell] = \emptyset}\ \textbf{Nil} \qquad \frac{}{\mathbb{C}[\mathbf{0},\ell] = \emptyset}\ \textbf{Zero}$$

$$\frac{[\![\pi_s]\!]_\gamma = \pi_s' \quad [\![\overrightarrow{e}]\!]_\gamma = \overrightarrow{v} \quad \mathbf{p} = \sigma(\gamma)}{\mathbb{C}[(\alpha^\star[\pi_s]\langle\overrightarrow{e}\rangle\sigma.P,\gamma),\alpha^\star[\pi_s']\langle\overrightarrow{v}\rangle,\gamma] = (P,\mathbf{p})}\ \textbf{B-Out}$$

$$\frac{\gamma_r \models \pi_s \quad \gamma_s \models \pi_r[\overrightarrow{v}/\overrightarrow{x}] \quad \mathbf{p} = \sigma[\overrightarrow{v}/\overrightarrow{x}](\gamma_2)}{\mathbb{C}[(\alpha^\star[\pi_r](\overrightarrow{x})\sigma.P,\gamma_r),\alpha^\star[\pi_s](\overrightarrow{v}),\gamma_s] = (P[\overrightarrow{v}/\overrightarrow{x}],\mathbf{p})}\ \textbf{B-In}$$

$$\frac{\mathbb{C}[(P,\gamma),\ell] = \mathscr{C}_1 \quad \mathbb{C}[(Q,\gamma),\ell] = \mathscr{C}_2}{\mathbb{C}[(P+Q,\gamma),\ell] = \mathscr{C}_1 \oplus \mathscr{C}_2}\ \textbf{Plus}$$

$$\frac{\mathbb{C}[(P,\gamma),\ell] = \mathscr{C}_1 \quad \mathbb{C}[(Q,\gamma),\ell] = \mathscr{C}_2}{\mathbb{C}[(P|Q,\gamma),\ell] = \mathscr{C}_1|Q \oplus P|\mathscr{C}_2}\ \textbf{Par}$$

Semantics of Collectives

The operational semantics of a collective is defined via the function

$$\mathbb{N}_{\varepsilon} : \textsc{Col} \times \textsc{Lab}_{I} \to [\textsc{Col} \to \mathbb{R}_{\geq 0}]$$

defining how a collective reacts when a broadcast/unicast message is received.

Semantics of Collectives

The operational semantics of a collective is defined via the function

$$\mathbb{N}_{\varepsilon} : \textsc{Col} \times \textsc{Lab}_{I} \to [\textsc{Col} \to \mathbb{R}_{\geq 0}]$$

defining how a collective reacts when a broadcast/unicast message is received.

$\textsc{Lab}_{I}$ denotes the subset of $\textsc{Lab}$ with only input labels:

$$\begin{array}{rcll} \ell & ::= & \alpha^{\star}[\pi_s](\overrightarrow{v}), \gamma & \text{Broadcast Input} \\ & | & \alpha[\pi_s](\overrightarrow{v}), \gamma & \text{Unicast Input} \end{array}$$

$$\frac{}{\mathbb{N}_{\varepsilon}[\mathbf{0},\ell] = \emptyset}\ \textbf{Zero}$$

$$\frac{\mathbb{C}[(P,\gamma),\alpha^{\star}[\pi_s](\overrightarrow{v}),\gamma] = \mathscr{N} \quad \mathscr{N} \neq \emptyset \quad \varepsilon = \langle \mu_p, \mu_w, \mu_r, \mu_u \rangle}{\mathbb{N}_{\varepsilon}[(P,\gamma),\alpha^{\star}[\pi_s](\overrightarrow{v}),\gamma] = \frac{\mu_p(\gamma,\alpha^{\star})}{\oplus\mathscr{N}} \cdot \mathscr{N} + [(P,\gamma) \mapsto (1-\mu_p(\gamma,\alpha^{\star})]}\ \textbf{Comp-B-In}$$

$$\frac{\mathbb{N}_{\varepsilon}[N_1,\alpha^{\star}[\pi_s](\overrightarrow{v}),\gamma] = \mathscr{N}_1 \quad \mathbb{N}_{\varepsilon}[N_2,\alpha^{\star}[\pi_s](\overrightarrow{v}),\gamma] = \mathscr{N}_2}{\mathbb{N}_{\varepsilon}[N_1 \parallel N_2,\alpha^{\star}[\pi_s](\overrightarrow{v}),\gamma] = \mathscr{N}_1 \parallel \mathscr{N}_2}\ \textbf{B-In-Sync}$$

Semantics of Systems

The operational semantics of systems is defined via the function

$$\mathbb{S}_t : \textsc{Sys} \times \textsc{Labs} \to [\textsc{Sys} \to \mathbb{R}_{\geq 0}]$$

Semantics of Systems

The operational semantics of systems is defined via the function

$$\mathbb{S}_t : \textsc{Sys} \times \textsc{Labs} \to [\textsc{Sys} \to \mathbb{R}_{\geq 0}]$$

This function only considers synchronisation labels $\textsc{Labs}$:

$$\begin{array}{lcll} \ell & ::= & \alpha^\star[\pi_s]\langle \overrightarrow{v} \rangle, \gamma & \text{Broadcast Output} \\ & | & \tau[\alpha[\pi_s]\langle \overrightarrow{v} \rangle, \gamma] & \text{Unicast Synchronization} \end{array}$$

$$\frac{\rho(t,\gamma_g,N)=\varepsilon=\langle\mu_p,\mu_w,\mu_r,\mu_u\rangle \quad \mu_u(\gamma_g,\alpha^\star)=(\sigma,N') \quad \sum_{C\in N} N(C)\cdot \mathsf{bSync}(C,N-C,\alpha^\star[\pi_s]\langle\overrightarrow{v}\rangle,\gamma)=\mathscr{N}}{\mathbb{S}_t[N \textbf{ in } (\gamma_g,\rho),\alpha^\star[\pi_s]\langle\overrightarrow{v}\rangle,\gamma]=\mathscr{N} \parallel N' \textbf{ in } (\sigma(\gamma_g),\rho)} \textbf{ Sys-B}$$

where

$$\frac{\varepsilon=\langle\mu_p,\mu_w,\mu_r,\mu_u\rangle \quad \mathbb{C}[C,\alpha^\star[\pi_s]\langle\overrightarrow{v}\rangle,\gamma]=\mathscr{C} \quad \mathbb{N}_\varepsilon[N,\alpha^\star[\pi_s](\overrightarrow{v}),\gamma]=\mathscr{N}}{\mathsf{bSync}_\varepsilon(C,N,\alpha^\star[\pi_s]\langle\overrightarrow{v}\rangle,\gamma) \quad = \quad \mu_r(\gamma,\alpha^\star[\pi_s]\langle\overrightarrow{v}\rangle,\gamma)\cdot\mathscr{C} \parallel \mathscr{N}}$$

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).

Outline

1 Introduction
- Collective Adaptive Systems
- Quantitative Analysis

2 Modelling CAS

3 CARMA
- The CARMA Modelling Language

4 CARMA Operational semantics

5 CASL:CARMA Specification Language

6 CARMA Eclipse plug-in

7 CARMA and its tools in Action

Recap. . .

Recap...

CARMA is equipped with linguistic constructs specifically developed for modelling and programming systems that can operate in open-ended and unpredictable environments.

Recap...

CARMA is equipped with linguistic constructs specifically developed for modelling and programming systems that can operate in open-ended and unpredictable environments.

However, CARMA is a SPA:

- concise syntax;
- parametric with respect to the used *expressions* and *data types*.

Recap...

CARMA is equipped with linguistic constructs specifically developed for modelling and programming systems that can operate in open-ended and unpredictable environments.

However, CARMA is a SPA:

- concise syntax;
- parametric with respect to the used *expressions* and *data types*.

To facilitate the use of CARMA in the specification/analysis process of CAS we developed:

- a *specifiation language*;
- an Eclipse plug-in as a container for CARMA tools.

A running example...

Bike Sharing System...

We want to use CARMA to model a bike sharing system where:

- bikes are made available in a number of stations that are placed in various areas of a city;
- Users that plan to use a bike for a short trip
  - can pick up a bike at a suitable origin station
  - return it to any other station close to their planned destination.
- we assume that the city is partitioned in homogeneous zones...
  - and that all the *stations* in the same zone can be equivalently used by any user in that zone.

Each CARMA specification, also named CARMA *model*, provides definitions for:

- structured *data types*;
- *constants* and *functions*;
- prototypes of *components*;
- *collective* of components;
- *systems* composed by collective and environment;
- *measures*, that identify the relevant data to *retrieve* during simulation runs.

CASL: Data types

Basic data types

- `bool`, for booleans;
- `int`, for integers;
- `real`, for real values.

Basic data types

- `bool`, for booleans;
- `int`, for integers;
- `real`, for real values.

Collections

- Sets: `{` $exp_1, \ldots, exp_n$ `}`
- Arrays: `[:` $exp_1, \ldots, exp_n$ `:]`

Basic data types

- `bool`, for booleans;
- `int`, for integers;
- `real`, for real values.

Collections

- Sets: `{` $exp_1, \ldots, exp_n$ `}`
- Arrays: `[:` $exp_1, \ldots, exp_n$ `:]`

Custom data types

- Enumerations: `enum` *name* `=` $elem_1, \ldots, elem_n$`;`
- Records: `record` *name* `= [` $type_1$ $field_1, \ldots,$ $type_n$ $field_n$`];`

CASL: Expressions

In CASL syntax of *expressions* includes:

- standard arithmetic and logical operations ($+, -, \ldots$)
- common functions (log,sin,...)
- conditional expression `(`*exp*1`?`*exp*2`:`*exp*3`)`
- special value `now`, indicating the current time
- ...

A limited set of expression has to be used when specific analysis tools (*fluid semantics*) are used.

CASL: Operators

[Add here a table with a list of relevant operators!]

A Carma specification can also contain *constants* and *functions* declarations having the following syntax:

```
const name = expression;

fun type name( type_1 arg_1 , ... , type_k arg_k ) {
  ...
}
```

Variable declaration, assignment and return:

```
type var = exp;
var = exp;
return exp;
```

If-then-else:

```
if (exp) { ... } else { ... }
```

Iterators:

```
for (var=exp1 ; exp2 ; exp3 ){ ... }
for var in exp { ... }
```

[ADD HERE TWO EXAMPLES OF FUNCTIONS]

CASL: Component Prototypes

A *component prototype* defines the general structure of a component:

$$
\begin{array}{l}
\textbf{component}\ \mathit{name}(\ \mathit{type}_1\ \ \mathit{arg}_1\ ,\ldots,\ \mathit{type}_n\ \ \mathit{arg}_n\ )\ \{ \\
\quad \textbf{store}\ \{\ \cdots \\
\quad\quad \textbf{attrib}\ \ \mathit{aname}_i\ :=\ \mathit{expression}_i\ ;\ \ \cdots \\
\quad \} \\
\quad \textbf{behaviour}\ \{\ \cdots \\
\quad\quad \mathit{proc}_i\ =\ \mathit{pdef}_i\ ;\ \ \cdots \\
\quad \} \\
\quad \textbf{init}\ \ \{\ \ P_1 \,|\, \cdots \,|\, P_w\ \ \} \\
\}
\end{array}
$$

Two kinds of components, one for each of the two groups of agents involved in our BSS, can be considered:

- *parking stations*;
- *users*.

PS attributes:

- zone: indicates where the station is located;
- capacity: the number of slots installed in the station;
- available: the number of available bikes.

User attributes:

- zone: current user location;
- dest: user destination.

```
component Station( int zone , int capacity , int
    available ) {
  store {
    zone = zone;
    available = available;
    capacity = capacity;
  }
  ...
}
component User( int zone , int dest ) {
  store {
    zone = zone;
    dest = dest;
  }
  ...
}
```

Behaviour...

The block `behaviour` is used to define the component behaviour...

... it consists of a sequence of *process definitions*

$$
\begin{array}{l}
\textbf{behaviour}\ \{ \\
\quad proc_1 = pdef_1\,; \\
\quad \ldots \\
\quad proc_n = pdef_n\,; \\
\}
\end{array}
$$

... that associate each *process name* with alternative actions.

Output actions:

$$[\ \mathit{guard}\ ]\ \texttt{act}\ [\ \mathit{pred}\ ] < \mathit{exp}_1\ ,\ \ldots\ ,\ \mathit{exp}_n > \{$$
$$\quad \mathit{attr}_1 = \mathit{exp}_1\,;$$
$$\quad \ldots$$
$$\quad \mathit{attr}_n = \mathit{exp}_n\,;$$
$$\}$$

Input actions:

$$[\ \mathit{guard}\ ]\ \texttt{act}\ [\ \mathit{pred}\ ]\ (\ x_1\ ,\ \ldots\ ,\ x_n\ )\ \{$$
$$\quad \mathit{attr}_1 = \mathit{exp}_1\,;$$
$$\quad \ldots$$
$$\quad \mathit{attr}_n = \mathit{exp}_n\,;$$
$$\}$$

Two processes are defined at `Station` component that model the procedures to *get* and *returning* a bike:

```
G = [my.available>0] get<> { my.available := my.
    available-1 s}.G;

R = [my.available<my.capacity] ret<>{ my.available :=
     my.available+1  }.R;
```

Procedures *get* and *returning* are modelled via *unicast output* over `get` and `ret`:

- `get` is enabled when there are bikes available (`my.available>0`);
- `get` is enabled when there are available slots (`my.available<my.capacity`).

Each user can be in three different states. . .

Each user can be in three different states...

... `P`, denoting a *pedestrian*:

```
P = get[ my.loc == loc ]().B;
```

Each user can be in three different states...

... `P`, denoting a *pedestrian*:

```
P = get[ my.loc == loc ]().B;
```

... a *pedestrian* executes unicast input `get` to collect a bike from a station located in his/her current zone (`my.zone == zone`) and then becomes a *bikers*:

```
B = move*[ false ]<>{ my.loc := my.dest }.W;
```

Each user can be in three different states...

... `P`, denoting a *pedestrian*:

```
P = get[ my.loc == loc ]().B;
```

... a *pedestrian* executes unicast input `get` to collect a bike from a station located in his/her current zone (`my.zone == zone`) and then becomes a *bikers*:

```
B = move*[ false ]<>{ my.loc := my.dest }.W;
```

... in that state a user *move* to the final destination and then *waits* for a slot:

```
W = ret[ my.loc == loc ]().kill;
```

Each user can be in three different states...

... `P`, denoting a *pedestrian*:

```
P = get[ my.loc == loc ]().B;
```

... a *pedestrian* executes unicast input `get` to collect a bike from a station located in his/her current zone (`my.zone == zone`) and then becomes a *bikers*:

```
B = move*[ false ]<>{ my.loc := my.dest }.W;
```

... in that state a user *move* to the final destination and then *waits* for a slot: SPONTANEOUS ACTION!

```
W = ret[ my.loc == loc ]().kill;
```

Each user can be in three different states...

... `P`, denoting a *pedestrian*:

```
P = get[ my.loc == loc ]().B;
```

... a *pedestrian* executes unicast input `get` to collect a bike from a station located in his/her current zone (`my.zone == zone`) and then becomes a *bikers*:

```
B = move*[ false ]<>{ my.loc := my.dest }.W;
```

... in that state a user *move* to the final destination and then *waits* for a slot: SPONTANEOUS ACTION!

```
W = ret[ my.loc == loc ]().kill;
```

... when a slot in the same zone is found, the user *disappear*.

Example: Station


```
component Station( int zone , int capacity , int
    available ) {
  store {
      zone = zone;
      available = available;
      capacity = capacity;
    }
    behaviour {
    G = [my.available >0] get<> {
        my.available := my.available -1
    }.G;
    R = [my.available <my.capacity] ret<>{
        my.available := my.available+1
    }.R;
  }
  init { G|R }
}
```

Example: User


```
component User( int loc , int dest ) {
  store {
    loc = loc;
      dest = dest;
    }
    behaviour {
      P = get[ my.loc == loc ]().B;
      B = move*[ false ]<>{ my.loc := my.dest }.W;
      W = ret[ my.loc == loc ]().kill;
    }
    init { P }
}
```

Example: Arrival

To model the of new users, another component is considered in our model:

```
component Arrival( int loc ) {
  store {
    attrib loc := loc;
  }
  behaviour {
    A = arrival*[false]<>.A;
  }
  init {
    A
  }
}
```

Example: Arrival

To model the of new users, another component is considered in our model:

```
component Arrival( int loc ) {
  store {
    attrib loc := loc;
  }
  behaviour {
    A = arrival*[false]<>.A;
  }
  init {
    A
  }
}
```

Above `zone` indicates the zone indicates the location where users arrive.

Block `collective` can be used to define groups of components:

collective *name* ($type_1$ var_1 , ... , $type_n$ var_n) {
  ...
}

Block `collective` can be used to define groups of components:

```
collective name ( type_1 var_1 , ... , type_n var_n ) {
  ...
}
```

BSS Collective:

```
collective bssCollective( int zones , int n ) {
  for ( i ; i<zones ; 1 ) {
    new Station( i , C, A )< n >;
  }
  new Arrival( i );
}
```

A system definition consists of two blocks, namely `collective` and `environment`:

```
system name {
  collective collective
  environment { ···
  }
}
```

A system definition consists of two blocks, namely `collective` and `environment`:

```
system name {
  collective collective
  environment { ···
  }
}
```

BSS System:

```
system Scenario {
  collective bssCollective( 10 , 10 , 10 )
  environment { ···
  }
}
```

Syntax of an block `environment` is the following:

```
environment {
  store { ··· }
  prob { ··· }
  weight { ··· }
  rate { ··· }
  update { ··· }
}
```

In our scenario use a global attribute to count the amount of *active users* in the system:

```
store {
  attrib users := 0;
}
```

```
weight {
  get {
    return ReceivingProb(#{User[P]|my.loc==sender.loc
        })
  }
  ret {
    return ReceivingProb(#{User[W]|my.loc==sender.loc
        });
  }
  default {
    return 1;
  }
}
```

```
rate{
   get { return get_rate; }
   ret { return ret_rate; }
   move* { return move_rate; }
   arrival* { (global.users<TOTAL_USERS?arrival_rate
       :0.0); }
}
```

```
update {
  arrival* {
    users := global.users+1;
    new User( sender.zone , U[0:ZONES-1] );
  }
  ret {
    users := global.users-1;
  }
}
```

To extract observations from a model, a CARMA specification also contains a set of *measures*:

measure *m_name*(*type*$_1$ *var*$_1$, ... , *type*$_1$ *var*$_n$) = *expr* ;

```
measure AverageBikes( int z ) =
    avg{ my.available | my.zone == z };
measure MinBikes( int z ) =
    min{ my.available | my.zone == z };
measure MaxBikes( int z ) =
    max{ my.available | my.zone == z };
```

Outline

1 Introduction
- Collective Adaptive Systems
- Quantitative Analysis

2 Modelling CAS

3 CARMA
- The CARMA Modelling Language

4 CARMA Operational semantics

5 CASL:CARMA Specification Language

6 CARMA Eclipse plug-in

7 CARMA and its tools in Action

CARMA Eclipse Plug-In provide tools for specification and quantitative analysis of CARMA models.

The CARMA Eclipse Plug-in

CARMA Eclipse Plug-In provide tools for specification and quantitative analysis of CARMA models.

Frontend

CARMA Eclipse Plug-In provide tools for specification and quantitative analysis of CARMA models.

Frontend

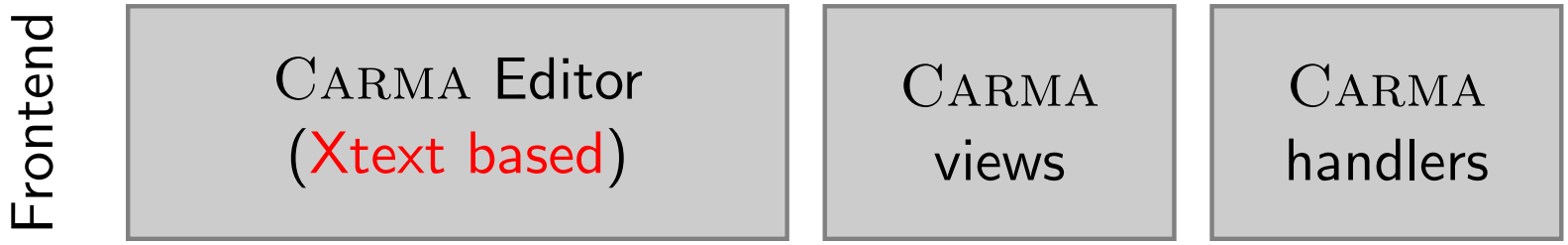
CARMA Editor
(Xtext based)

The CARMA Eclipse Plug-in

CARMA Eclipse Plug-In provide tools for specification and quantitative analysis of CARMA models.

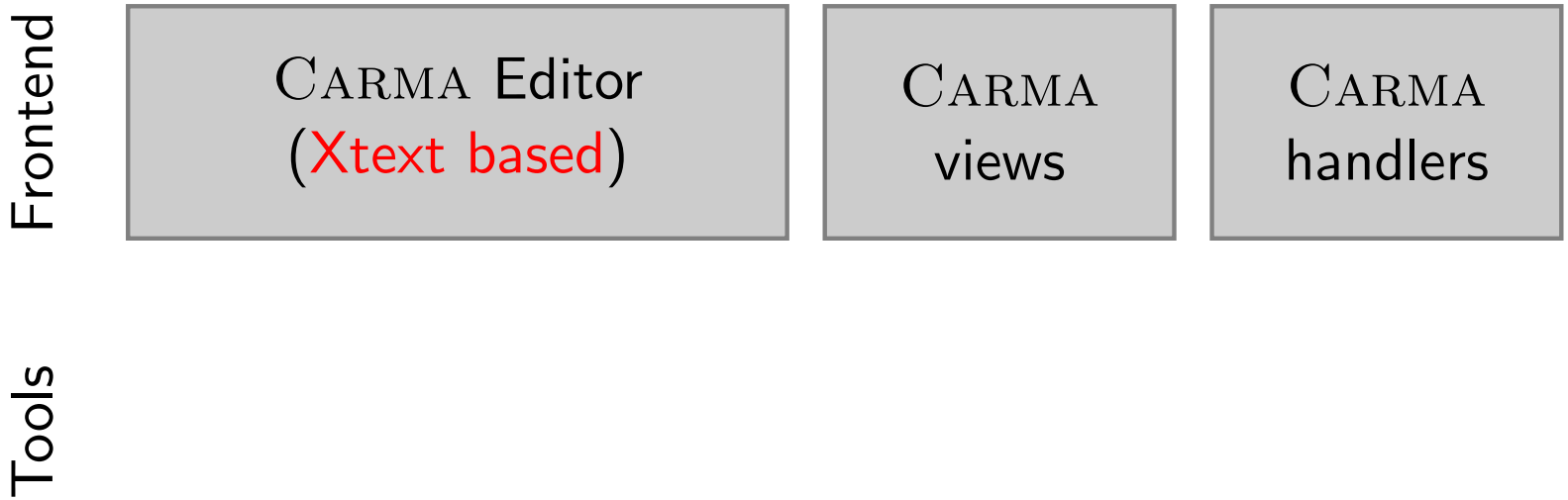
The CARMA Eclipse Plug-in

CARMA Eclipse Plug-In provide tools for specification and quantitative analysis of CARMA models.



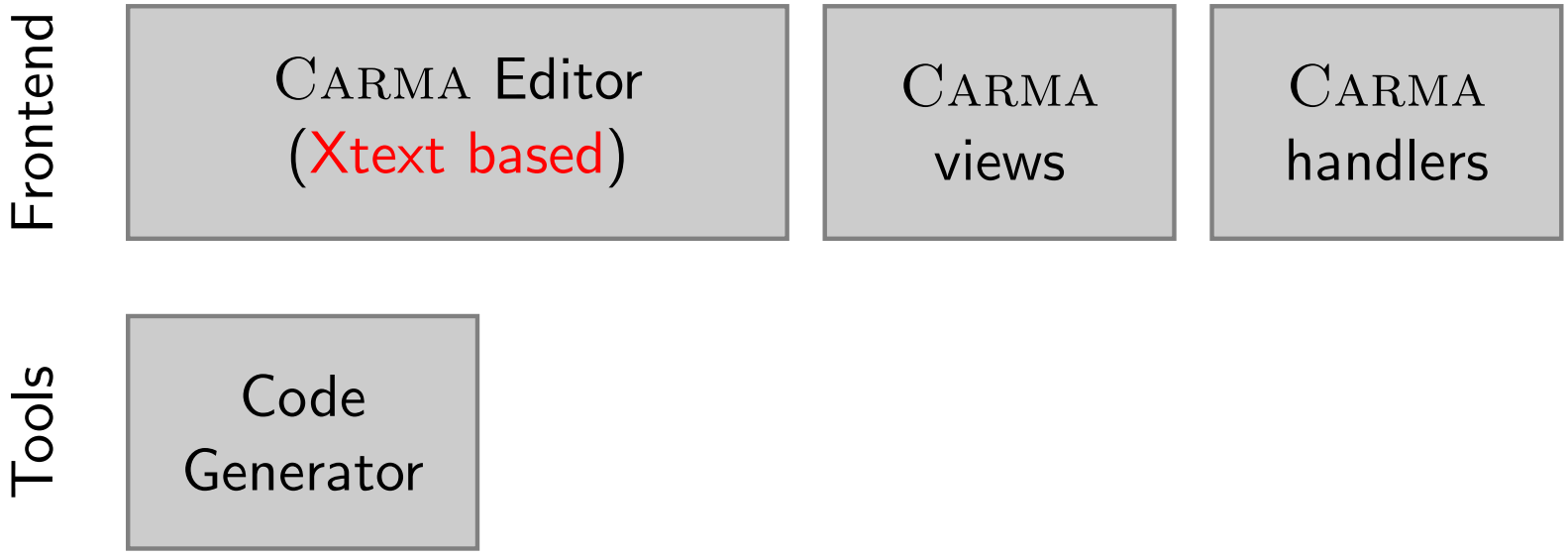
The CARMA Eclipse Plug-in

CARMA Eclipse Plug-In provide tools for specification and quantitative analysis of CARMA models.



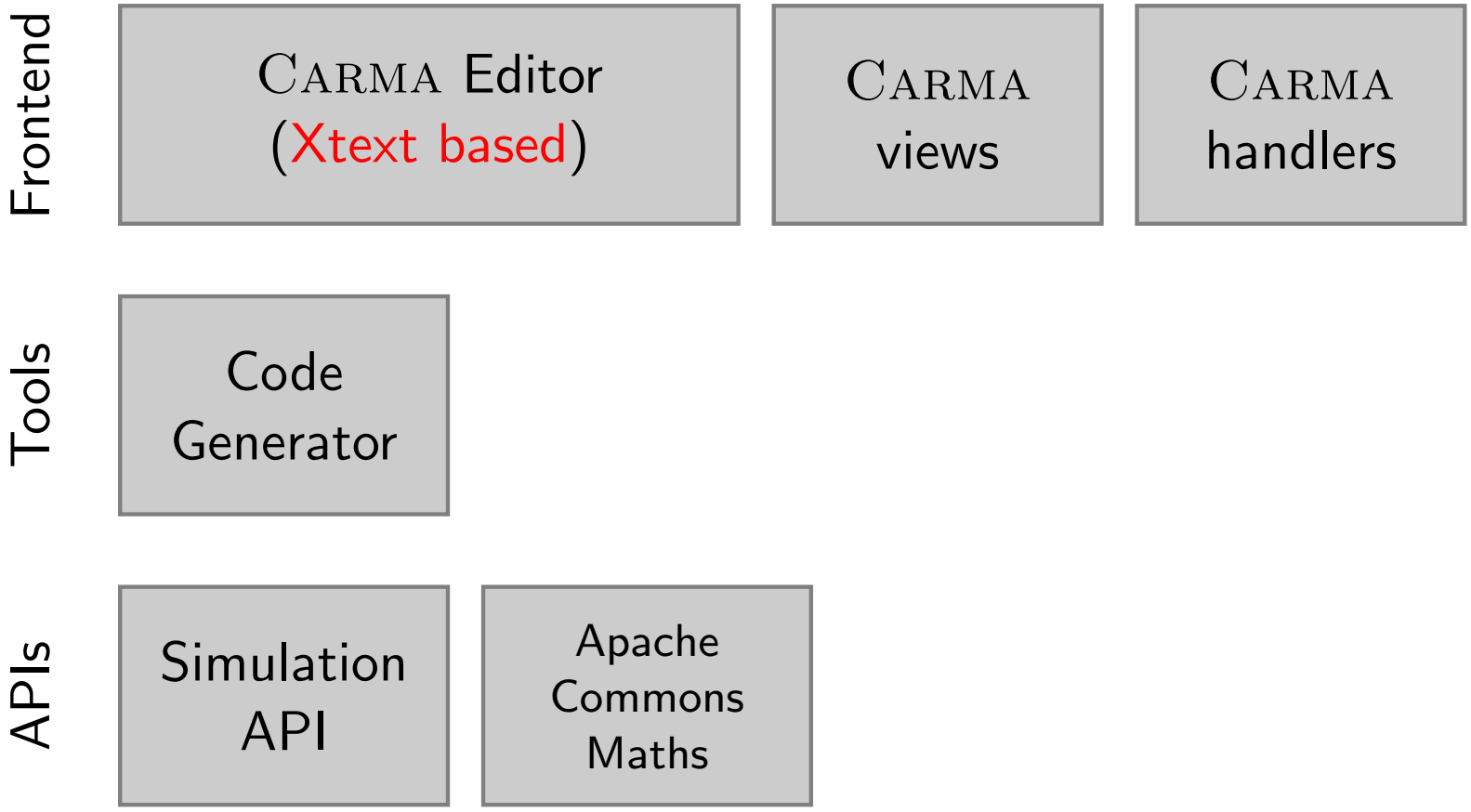
The CARMA Eclipse Plug-in

CARMA Eclipse Plug-In provide tools for specification and quantitative analysis of CARMA models.



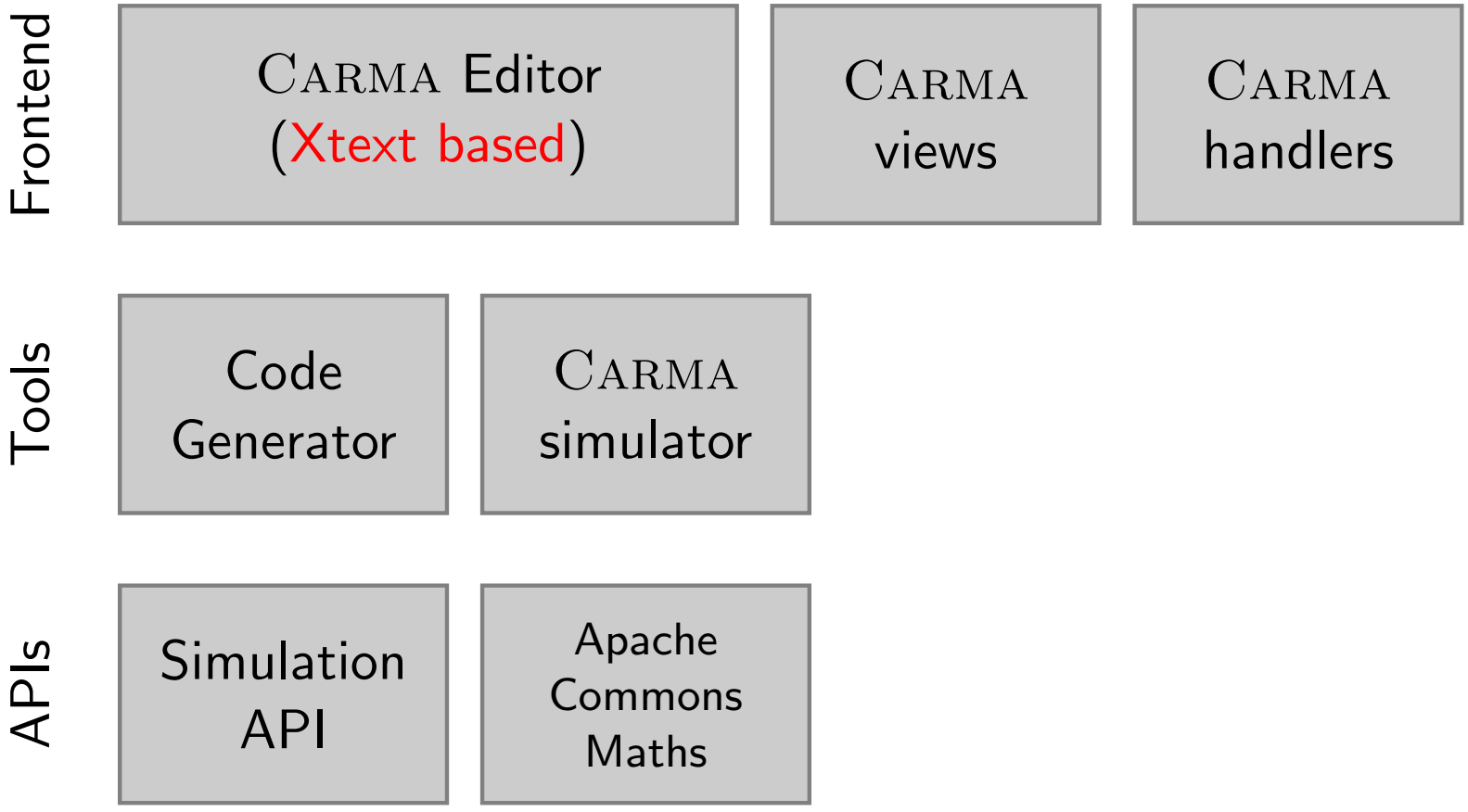
The CARMA Eclipse Plug-in

CARMA Eclipse Plug-In provide tools for specification and quantitative analysis of CARMA models.



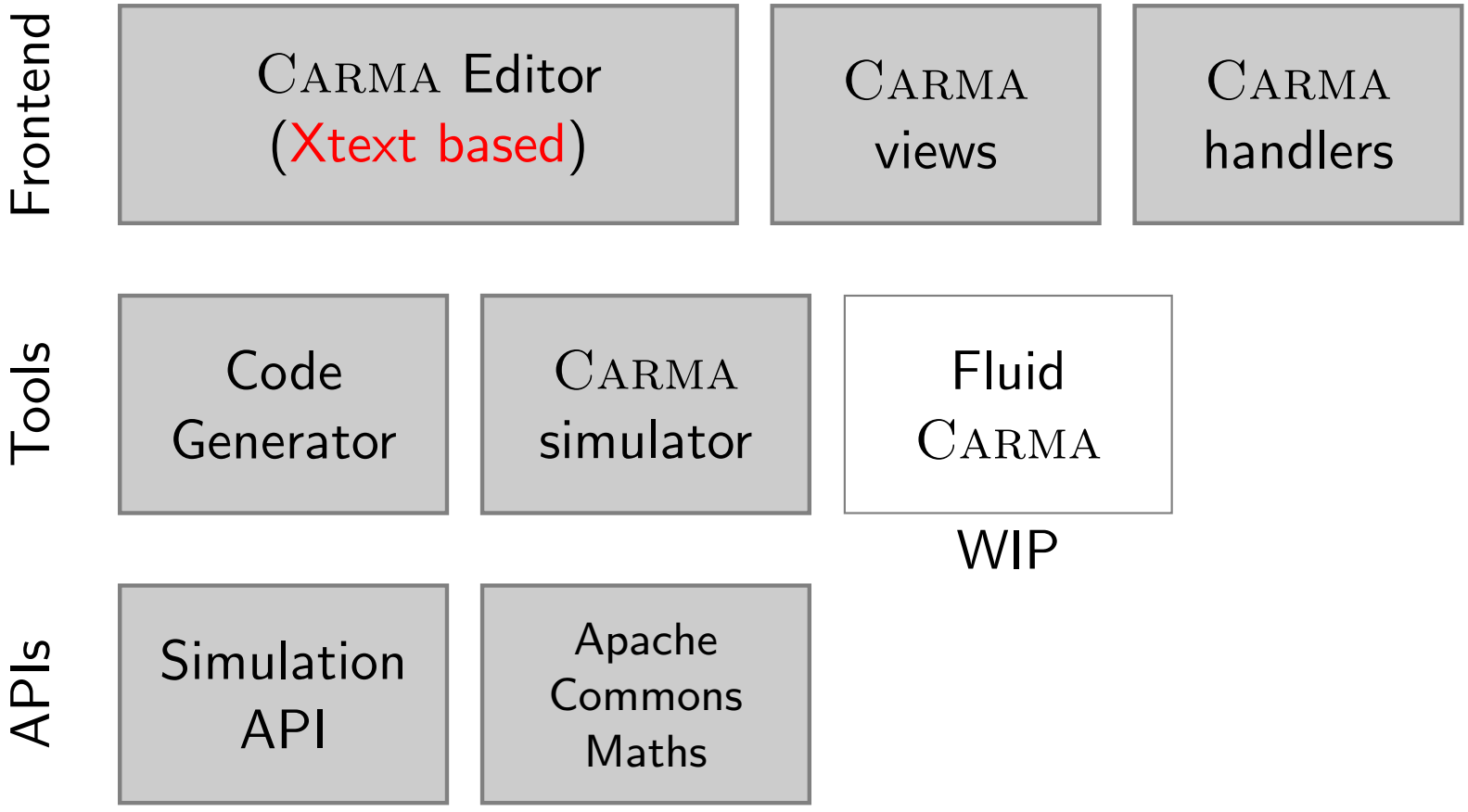
The CARMA Eclipse Plug-in

CARMA Eclipse Plug-In provide tools for specification and quantitative analysis of CARMA models.



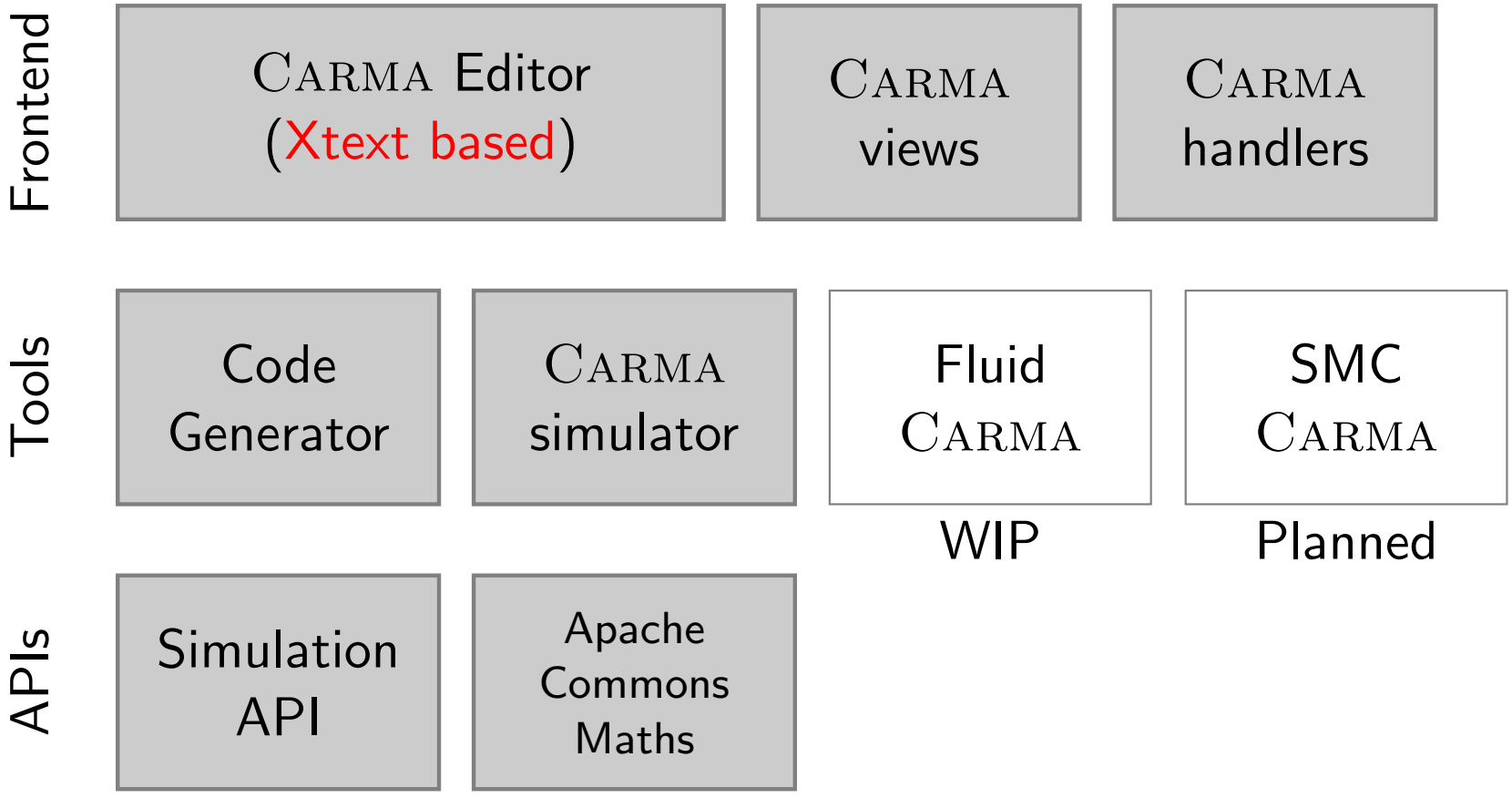
The CARMA Eclipse Plug-in

CARMA Eclipse Plug-In provide tools for specification and quantitative analysis of CARMA models.

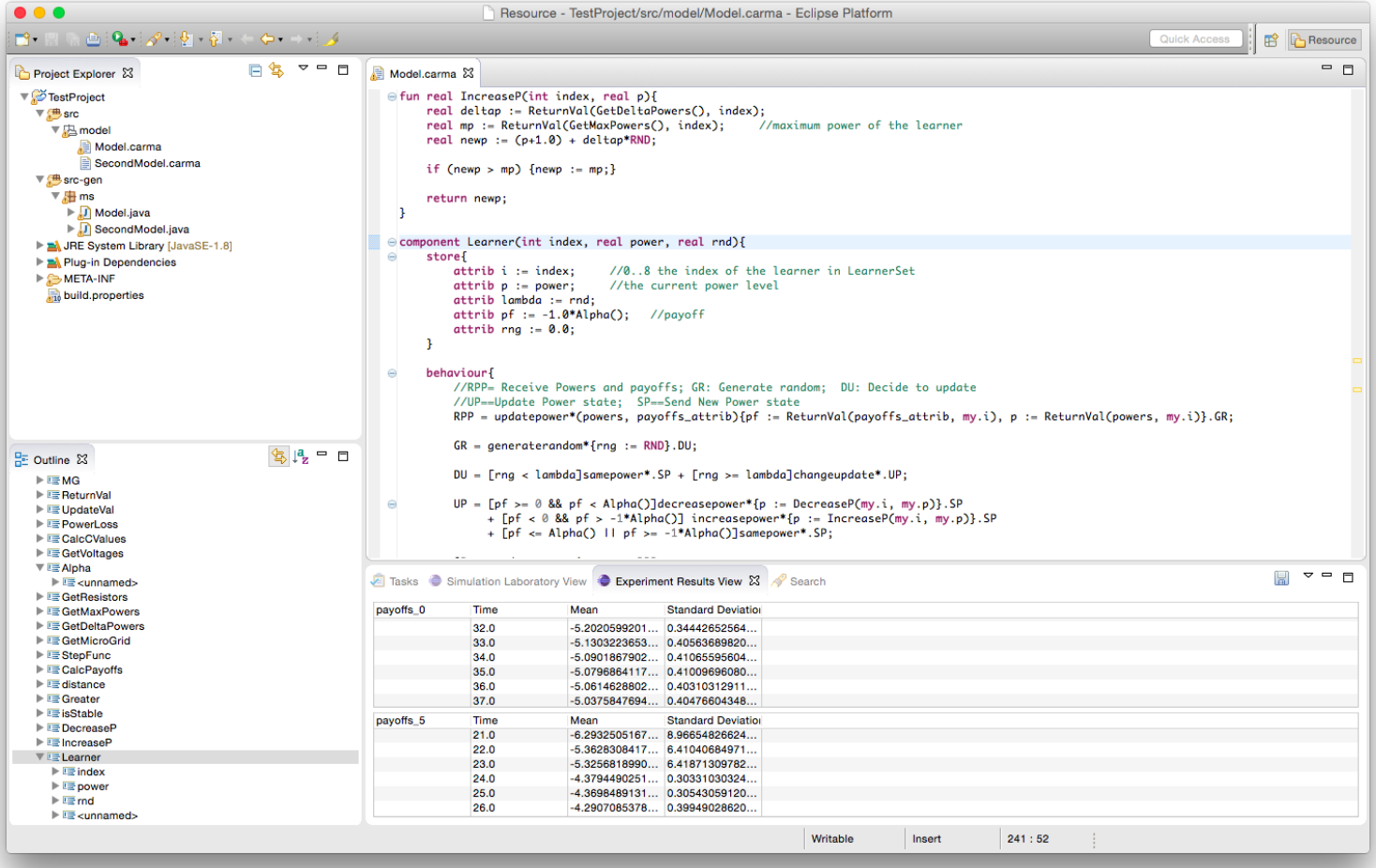


The CARMA Eclipse Plug-in

CARMA Eclipse Plug-In provide tools for specification and quantitative analysis of CARMA models.



LIVE DEMO


Resource - TestProject/src/model/Model.carma - Eclipse Platform
Project Explorer
TestProject
src
model
Model.carma
SecondModel.carma
src-gen
ms
Model.java
SecondModel.java
JRE System Library [JavaSE-1.8]
Plug-in Dependencies
META-INF
build.properties
Outline
MG
ReturnVal
UpdateVal
PowerLoss
CalcCValues
GetVoltages
Alpha
<unnamed>
GetResistors
GetMaxPowers
GetDeltaPowers
GetMicroGrid
StepFunc
CalcPayoffs
distance
Greater
isStable
DecreaseP
IncreaseP
Learner
index
power
rnd
<unnamed>
Model.carma
fun real IncreaseP(int index, real p){
real deltap := ReturnVal(GetDeltaPowers(), index);
real mp := ReturnVal(GetMaxPowers(), index); //maximum power of the learner
real newp := (p+1.0) + deltap*RND;
if (newp > mp) {newp := mp;}
return newp;
}
component Learner(int index, real power, real rnd){
store{
attrib i := index; //0..8 the index of the learner in LearnerSet
attrib p := power; //the current power level
attrib lambda := rnd;
attrib pf := -1.0*Alpha(); //payoff
attrib rng := 0.0;
}
behaviour{
//RPP= Receive Powers and payoffs; GR: Generate random; DU: Decide to update
//UP==Update Power state; SP==Send New Power state
RPP = updatepower*(powers, payoffs_attrib){pf := ReturnVal(payoffs_attrib, my.i), p := ReturnVal(powers, my.i)}.GR;
GR = generaterandom*{rng := RND}.DU;
DU = [rng < lambda]samepower*.SP + [rng >= lambda]changeupdate*.UP;
UP = [pf >= 0 && pf < Alpha()]decreasepower*{p := DecreaseP(my.i, my.p)}.SP
+ [pf < 0 && pf > -1*Alpha()] increasepower*{p := IncreaseP(my.i, my.p)}.SP
+ [pf <= Alpha() || pf >= -1*Alpha()]samepower*.SP;
Tasks
Simulation Laboratory View
Experiment Results View
Search
payoffs_0
Time
Mean
Standard Deviatio
32.0
-5.2020599201...
0.34442652564...
33.0
-5.1303223653...
0.40563689820...
34.0
-5.0901867902...
0.41065595604...
35.0
-5.0796864117...
0.41009696080...
36.0
-5.0614628802...
0.40310312911...
37.0
-5.0375847694...
0.40476604348...
payoffs_5
Time
Mean
Standard Deviatio
21.0
-6.2932505167...
8.96654826624...
22.0
-5.3628308417...
6.41040684971...
23.0
-5.3256818990...
6.41871309782...
24.0
-4.3794490251...
0.30331030324...
25.0
-4.3698489131...
0.30543059120...
26.0
-4.2907085378...
0.39949028620...
Writable
Insert
241 : 52

Outline

1 Introduction
- Collective Adaptive Systems
- Quantitative Analysis

2 Modelling CAS

3 CARMA
- The CARMA Modelling Language

4 CARMA Operational semantics

5 CASL:CARMA Specification Language

6 CARMA Eclipse plug-in

7 CARMA and its tools in Action