Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 337 (2018) 155–172

www.elsevier.com/locate/entcs

Goals and Resource Constraints in CARMA

Paul Piho [1] Anastasis Georgoulas [2] Jane Hillston [3]

School of Informatics
University of Edinburgh
Edinburgh, UK

Abstract

CARMA is a recently developed, high-level quantitative modelling language developed for the design and analysis of collective adaptive systems. In the current CARMA language, agents within a system consist of a behaviour, captured as a process, and knowledge, represented as a store of attributes. In this paper we present the first steps to equipping agent specifications with more sophisticated forms of knowledge in terms of goals and targets, and demonstrate how these may be integrated into the modelling and analysis process. We illustrate the ideas with a simple example taken from the domain of swarm robotics.

Keywords: Stochastic modelling, collective adaptive systems, goals, control

1 Introduction

Examples of collective adaptive systems (CAS) are widespread in nature, ranging from the swarming behaviour of insects to patterns of epidemic spread in humans. These systems are characterised as consisting of a large number of simple entities or agents whose perception is limited to their own locality, but which nevertheless interact with their neighbours. Through these interactions complex emergent behaviour may be formed at the system level, often of a form which is difficult to predict from the simple behaviour of the individual agents.

The highly distributed and robust nature of these systems has meant that they have become a paradigm for the design of highly-distributed computer-based systems which are intended to operate without human intervention. In general it is difficult to predict the emergent behaviour and harder still to know a priori how to design the behaviour of individual agents in order to achieve a system level goal.

[1] Email: paul.piho@ed.ac.uk
[2] Email: anastasis.georgoulas@ed.ac.uk
[3] Email: jane.hillston@ed.ac.uk

https://doi.org/10.1016/j.entcs.2018.03.039

Thus modelling becomes a valuable tool in the design of collective adaptive systems, allowing possible configurations of agents to be evaluated with respect to global goals. Moreover, the highly-distributed nature of such systems means that they are often resource-constrained. For example individual entities may have only limited battery life. Therefore it is essential that models are capable to taking into account non-functional as well as functional requirements.

CARMA, *Collective Adaptive Resource-sharing Markovian Agents*, is a recently defined, high-level stochastic process algebra-based modelling language [10]. Previous work has focused on the use of CARMA to explore the potential behaviour of systems through Markovian simulation. When non-functional requirements and behavioural goals are considered, they must be expressed externally, for example as formulae in a suitable temporal logic [2,13,6]. However in the long-term we would like to be able to express goals within the models themselves and explore the use of control algorithms within the environment of a model. This would allow the behaviour of individual agents to be adapted in order to achieve the system goals, whilst respecting the resource constraints applicable to individual agents.

As a first step towards this long-term objective, in this paper we present an extension of the CARMA modelling language which supports the specification of resource constraints and system-level functional goals. This extension provides basic monitoring capabilities for CARMA, which can subsequently be used for reasoning about control policies. During a simulation, agents that violate resource constraints can be readily identified, allowing this aspect of possible behaviour to be explored and evaluated. At the system level, the specification of goals supports the identification of states that can be regarded as desirable targets within trajectories of system behaviour.

The rest of the paper is structured as follows. In Section 2 we present the CARMA process algebra. The description of the proposed extension for specifying individual-level resource constraints and global goals is given in Section 3. The extensions are illustrated through a swarm robot model in Section 4. In Section 5 we present the developed software tool for experimenting with the proposed changes. Finally, we conclude this paper in Section 6.

2 CARMA

CARMA is a new stochastic process algebra for the representation of systems developed in the CAS paradigm [10]. The language offers a rich set of communication primitives, and exploits *attributes*, captured in a *store* associated with each component, to enable attribute-based communication. For example, for many CAS systems the location is likely to be one of the attributes. Thus it is straightforward to model systems in which, for example, there is limited scope of communication, or interaction is restricted to co-located components, or where there is spatial heterogeneity in the behaviour of agents.

A CARMA system consists of a *collective* operating in an *environment*. The collective is a multiset of components that models the behaviour of a system; it is

used to describe a group of interacting *agents* that cooperate to achieve a given set of tasks. The environment models all those aspects which are intrinsic to the context where the agents are operating, i.e. the environment mediates agent interactions. This is one of the key features of CARMA. The environment is not a centralised controller but rather something more pervasive and diffusive — the physical context of the real system — which is abstracted as the environment, exercising influence and imposing constraints on the different agents in the system. Specifically the environment is responsible for setting the rates at which actions are performed, and probabilities of receiving a given message. For example, in a model of a smart transport system, the environment will determine the rate at which entities (buses, bikes, taxis etc) move through the city, which may also depend on the current time. This is an abstraction of the presence of other vehicles causing congestion which may impede the progress of the modelled entities to a greater or lesser extent at different times of the day. The role of the environment is also related to the spatially distributed nature of CAS — we expect that the location *where* an agent is will have an effect on *what* an agent can do.

A CARMA component captures an *agent* operating in the system. It consists of a process, that describes the agent's behaviour, and of a store, that models its *knowledge*. A store is a function which maps *attribute names* to *basic values*.

Processes located within a CARMA component interact with other components via the defined communication primitives. Specifically, CARMA supports both unicast and broadcast communication, and permits locally synchronous, but globally asynchronous communication. Distinct predicates (boolean expressions over attributes) associated with senders and potential receivers are used to filter possible interactions. Thus, a component can receive a message only when its store satisfies the target predicate. Similarly, a receiver also uses a *predicate* to identify accepted sources. An interaction will occur only when the sender satisfies the predicate used by the receiver, and the receiver satisfies the predicate used by the sender. The execution of communicating actions takes time, which is assumed to be an exponentially distributed random variable whose parameter is determined by the environment.

More formally, we let SYS be the set of CARMA *systems* S defined by the following syntax:

$$S \ ::= \ N \ \mathbf{in} \ \mathcal{E}$$

where N is a collective and $\mathcal{E}$ is an environment. We let COL be the set of collectives N which are generated by the following grammar:

$$N ::= C \ \mid \ N \parallel N$$

A collective N is either a *component* C or the parallel composition of collectives $N_1 \parallel N_2$. The precise syntax of components is:

$$C ::= \mathbf{0} \ \mid \ (P, \gamma)$$

and we let COMP be the set of components C generated by this grammar. A component C can be either the *inactive component*, denoted by $\mathbf{0}$, or a term of the

form (P, γ), where P is a *process* and γ is a *store*. A store is a function which maps *attribute names* to *basic values*. We let:

- Attr be the set of *attribute names* a, a', a_1,..., b, b', b_1,...;
- Val be the set of *basic values* v, v', v_1,...;
- Γ be the set of *stores* $\gamma, \gamma_1, \gamma', \ldots$, i.e. functions from Attr to Val.

The behaviour of a component is specified via a process P. We let Proc be the set of Carma processes P, Q,... defined by the following grammar:

$$
\begin{array}{lll}
P, Q ::= \mathbf{nil} & & act ::= \alpha^\star[\pi_s]\langle\overrightarrow{e}\rangle\sigma \\
\quad \mid \; act.P & & \quad \mid \; \alpha[\pi_r]\langle\overrightarrow{e}\rangle\sigma \\
\quad \mid \; P + Q & & \quad \mid \; \alpha^\star[\pi_s](\overrightarrow{x})\sigma \\
\quad \mid \; P \mid Q & & \quad \mid \; \alpha[\pi_r](\overrightarrow{x})\sigma \\
\quad \mid \; [\pi]P & & e ::= a \mid \mathsf{my}.a \mid x \mid v \mid \mathsf{now} \mid \cdots \\
\quad \mid \; \mathbf{kill} & & \\
\quad \mid \; A \qquad (A \triangleq P) & & \pi_s, \pi_r, \pi ::= \top \mid \bot \mid e_1 \bowtie e_2 \mid \neg\pi \mid \pi \wedge \pi \mid \cdots
\end{array}
$$

The process specifications are fairly standard, with prefix, choice and parallel composition all with their usual meanings. A *predicate* π is used to indicate that the process is only active when the predicate is true. The distinguished process **kill** removes the enclosing component from the collective. In the action descriptions, the following notation is used:

- α is an *action type* in the set ActType;
- π_s and π_r are *predicates* that define filters on the acceptable communication partners;
- x is a *variable* in the set of variables Var;
- e is an expression in the set of expressions Exp [4];
- $\overrightarrow{\cdot}$ indicates a sequence of elements;
- σ is an *update*, i.e. a function from Γ to $Dist(\Gamma)$ in the set of *updates* Σ; where $Dist(\Gamma)$ is the set of probability distributions over Γ.

Formally, an environment consists of two elements: a *global store* γ_g, that models the overall state of the system, and an *evolution rule* ρ, which is a function that, depending on the global store and on the current state of the collective (i.e., on the configurations of each component in the collective), returns a tuple of functions $\varepsilon = \langle \mu_p, \mu_w, \mu_r, \mu_u \rangle$:

[4] The precise syntax of expressions e has been omitted for brevity. We only assume that expressions are built using the appropriate combinations of *values*, *attributes* (sometime prefixed with my), variables and the special term now. The latter is used to refer to the current time.

- $\mu_p : \Gamma \times \Gamma \times \text{Act} \to [0,1]$, $\mu_p(\gamma_s, \gamma_r, \alpha)$ expresses the probability that a component with store γ_r can receive a broadcast message from a component with store γ_s when α is executed;
- $\mu_w : \Gamma \times \Gamma \times \text{Act} \to [0,1]$, $\mu_w(\gamma_s, \gamma_r, \alpha)$ yields the weight that will be used to compute the probability that a component with store γ_r can receive a unicast message from a component with store γ_s when α is executed;
- $\mu_r : \Gamma \times \text{Act} \to \mathbb{R}_{\geq 0}$, $\mu_r(\gamma_s, \alpha)$ computes the execution rate of action α executed at a component with store γ_s;
- $\mu_u : \Gamma \times \text{Act} \to \Sigma \times \text{Col}$, $\mu_u(\gamma_s, \alpha)$ determines the updates on the environment (global store and collective) induced by the execution of action α at a component with store γ_s.

To extract observations from a model, a CARMA specification also contains a set of *measures*. Each measure is defined as:

$$\texttt{measure}\ m\_name[var_1 = range_1, \ldots, var_n = range_n] = expr;$$

Expression *expr* can be used to count or to compute statistics about attribute values of components operating in the system. These expressions are used to compute the minimum/maximum/average value of expression *expr* evaluated in the store of all the components satisfying boolean expression *guard*, respectively.

The formal semantics of CARMA gives rise to a continuous time Markov chain (CTMC). The state space of the system is represented as a finite, discrete set of states and the times of state transitions are governed by the rates given by the model description. The state space generated by CARMA models is usually too large to be analytically tractable and thus the models are analysed by simulating individual time trajectories.

The specification and analysis of CARMA models is supported by an Eclipse plug-in [1] and a model simulator. The plug-in implements an appropriate high-level language, named the CARMA *Specification Language*, that simplifies the creation of CARMA models by providing rich syntactic constructs inspired by main stream programming languages.

3 Capturing requirements

In our current work we consider two distinct forms of *requirements* or *constraints* that a CAS may be subject to. Firstly, the highly-distributed nature of CAS means that the individual components within the system may be constrained in their access to resources that they need in order to complete their tasks. Even in a system where components are required to work collaboratively they may nevertheless compete for resources. We can view this as a requirement for individual components to operate within given resource bounds or constraints. Conversely, at the global level, system requirements are often phrased in terms of goals or conditions on the global state space that must be satisfied, possibly within a given time bound.

In this section we consider how to extend the CARMA modelling language in

order to capture both forms of requirement in a natural way.

3.1 Individual constraints

We limit our attention to constraints on resource usage that are individual-based — examples include battery life, communication bandwidth and power consumption. In particular we point out that in some cases these are hard constraints which when violated cause the individual components to fail. In other cases we may want to consider constraints that when violated make the component unable to contribute to the current task — for example, the connection to the rest of the collective is lost. Moreover such individual-level constraints can also be used to model time-out behaviour or to induce component failures due to buffer overflows. We choose this perspective since it seems most in-keeping with the semi-autonomous nature of the components in a CARMA model.

Our objective is to support the monitoring of individual components with respect to the resource constraints. For example, this will allow us to easily keep track of failed components as well as simplify the creation of models by allowing for component failures to be defined implicitly through constraints. Thus we extend the CARMA language in order to specify constraints for individual components, leading to the definition:

$$C ::= \mathbf{0} \mid (P, \gamma, c(\gamma))$$

where, as previously, C is either the *null* process or defined as P along with the local store γ and $c(\gamma)$ denotes a set of constraints on the store attributes. The set $c(\gamma)$ consists of expressions defining constraints on the store variables given by the following grammar:

$$\begin{array}{lll} c(\gamma) & ::= e \mid e \wedge e & \\ e & ::= x \vartriangle x & \vartriangle \in \{<, >, \geq, \leq, =, \neq\} \\ x & ::= k \mid \gamma(a) & k \text{ is a constant, } \gamma(a) \text{ is the value of attribute } a \text{ in store } \gamma \end{array}$$

That is, component constraints are defined as conjunctions of linear inequalities and equalities on the evaluation of store attributes. Thus we assume that resource use is recorded as an attribute within the component.

The implementation of the CARMA specification language [1] allows definitions of attributes that take numerical values. As the underlying model is a finite state CTMC and the attributes are encoded in the state of the components, we require the set of possible attribute values to be denumerable — for example, a finite subset of real numbers or integers. Additionally, it is possible to define categorical attributes through specifying enumerated types, like *colour*, as a set of named values, for example *red*, *blue* and *green*. In the current implementation of the modelling tools, the values of enumerated types are not ordered and thus here we restrict ourselves to considering numerical attributes, where prescribing constraints in the form given above is possible.

The constraint specification needs to be incorporated into the formal semantics of the language. Consider the following CARMA process:

$$[\pi_1]\,\alpha^*[\pi_2]\langle x\rangle\sigma$$

The simulator for CARMA models proceeds as follows: (i) the predicate π_1 is evaluated to check whether the action is possible (ii) the filter predicate π_2 is evaluated to get all the possible receivers of the message x (iii) the evolution rule is evaluated in the context of all the actions in the system (iv) an action is sampled from the set of actions that can be performed. Supposing the action α^* is performed, we update the local store according to σ.

The two natural places for checking the constraints are when the predicate π_1 is checked or after the store update has been calculated. We select the former approach since it allows us to include the constraints in the set of guards for the actions and use the existing implementation with minimal modifications. In other words, in the example above, if we consider a constraint $c(\gamma)$, the process in effect becomes:

$$[\pi_1 \wedge c(\gamma)]\,\alpha^*[\pi_2]\langle x\rangle\sigma$$

Adopting this approach does not ensure that the constraint will not be violated in the evolution of the component. For an example, consider the store attribute $b = 2$ and the constraint $b \geq 0$. Suppose the performed action would reduce the constraint in the following way:

$$b \leftarrow b - 3$$

This means that the constraint is violated only after the action fires — this could be interpreted by the action firing despite lack of resources. Firing of such actions could be blocked by checking that the constraint will not be violated after the action is performed. This causes some technical difficulties as, in general, the update σ is probabilistic and returns a probability distribution over the possible updated store values. One of the options is to prescribe an additional set of constraints $c(\sigma(b))$ for the component such that possible outcomes of the update σ on b also have to satisfy the constraint on b. In the current work we opt for an alternative of leaving it up to the modeller to explicitly deal with such boundary cases in the most suitable way.

Thus in order to integrate the individual constraints into the operation of a CARMA model we give semantics to the constraint definitions in terms of guards on processes and their actions in the given CARMA model. We define translations from the set of syntactically defined CARMA components extended with constraints, denoted COMP^c, to the set of CARMA components COMP as given in Section 2. For example, the translation of the action prefix is given by the following:

$$\big(act.P, \gamma, c(\gamma)\big) \xrightarrow{translation} \big([c(\gamma)]\,act.[c(\gamma)]P + [\neg c(\gamma)]\,\mathbf{stop.nil}, \gamma\big)$$

This means that the components in COMP^c defined by the process $act.P$, store γ and set of constraints $c(\gamma)$ are translated to components in COMP. In particular, the constraints are included as guards on the process $act.P$ so that if constraints are violated the component will be unable to perform its usual actions. Instead,

if the constraints are not satisfied the component will only be able to perform the special action **stop** that turns the component into an inactive one defined by the **nil** process. We define the action **stop** as a broadcast action with no receivers and no message

$$\mathbf{stop} := \mathrm{stop}^{*}[\bot]\langle\rangle\{\}$$

Similarly, we give translations for the remaining elements of the process grammar that deal with choice, parallel composition and guards on processes.

$$\begin{aligned}
&\big(P,\gamma,c(\gamma)\big) && \xrightarrow{translation} && \big([c(\gamma)]\,P + [\neg c(\gamma)]\,\mathbf{stop.nil},\gamma\big)\\
&\big(P+Q,\gamma,c(\gamma)\big) && \xrightarrow{translation} && \big(([c(\gamma)]\,P + [c(\gamma)]\,Q) + [\neg c(\gamma)]\,\mathbf{stop.nil},\gamma\big)\\
&\big(P \parallel Q,\gamma,c(\gamma)\big) && \xrightarrow{translation} && \big(([c(\gamma)]\,P \parallel [c(\gamma)]\,Q) + [\neg c(\gamma)]\,\mathbf{stop.nil},\gamma\big)\\
&\big([\pi]P,\gamma,c(\gamma)\big) && \xrightarrow{translation} && \big([\pi \wedge c(\gamma)]\,P + [\neg c(\gamma)]\,\mathbf{stop.nil},\gamma\big)
\end{aligned}$$

Thus, using this approach, individual-level constraints can be encoded in the existing CARMA framework. Implementing constraints in the CARMA specification language provides a clear and compact way of capturing when the behaviour of a component depends on some limited resource. The translations above are performed automatically by the modelling tool. This allows the modeller to focus on the behaviour and the constraints without having to explicitly deal with the consequences of the component violating the constraints. The proposed implementation allows different sets of constraints to be defined for each component type and for components to be initialised in a system with a single constraint or sets of constraints. This improves CARMA model re-usability as models under different individual constraints can be easily considered.

Figure 1 illustrates the extended syntax for the CARMA specification language. We aim to give an idea of the structure of the model, and so we have not given the definitions of particular constraints or full definitions of agents or their instantiations. In particular, specifications of the component's local store attributes, its process definitions as well as the full specification of the environment are omitted — omissions are denoted with "...".

The `component` block in Figure 1 for `Agent` is extended with the `constraints` block where constraints, in this case `constraint_1` and `constraint_2`, are defined. When initialising components of type `Agent` in the system `System` the keyword `following` is used to indicate which of the constraints the agents are using.

3.2 Global goals

In this section we shift our focus to global goals for systems. Global goals can represent the desired functional behaviour or global properties of the system. An example of the former would be specifying a target location that we want the components to navigate to. An example of a desired global property is the number of failed components being less than a given bound. Clearly these goals will have to be treated differently from the constraints as the system is not expected to start

```
component Agent {
    ...
    constraints {
        constraint_1 { ... };
        constraint_2 { ... }; }
}
system System {
    collective{
        new Agent( ... )@(x,y) following constraint_1;
        new Agent( ... )@(x',y') following constraint_2;
    }
    environment{ ... }
}
```

Fig. 1. Instantiating componènts with different constraints.

in a state that satisfies the prescribed goal. Thus while the individual constraints identify states to be avoided, the goals identify states to aim for.

We restrict ourselves to considering state-based goals. This means that the goals implicitly define a set of states of the model in which the desired behaviour is achieved. Both the target location and the failed components examples mentioned above are instances of state-based goals. Specifically, by prescribing the target location we say that we are interested in those states of the system in which the components have reached the target. Trajectory-based goals, in contrast, would describe the goals in terms of *trajectories*, i.e. traces or time-stamped sequences of states. This would be needed in order to consider timing properties of the modelled systems. For example, saying that the target location has to be reached within T time units would lead us to consider the trajectory of the system up to time T. However, in this paper we focus on state-based goals, and trajectory-based goals are left for further work.

In general state-based goals can be easily monitored: at each encountered state of the system we can decide whether the goal is satisfied or not. Thus, given a simulated trace, we can annotate it based on such goals. This can be done either on-line or off-line. Note that this does not change the behaviour of the system and thus the semantics of the model remain unchanged. In future work we would like to introduce more sophisticated ideas of control where the global goals are used to drive the behaviour of the system, through the evolution rule used in the environment of the model. However, in this paper we are using basic off-line monitoring based on global goals as a starting point.

The specification of global goals is as follows. First we define measures of interest as functions from state to $\mathbb{R}$, as outlined in Section 2. The goals are specified in a form similar to the individual-based constraints. In particular, each goal is expressed as a conjunction of inequalities and equalities on the defined measure. The grammar

for such goal definitions is the following:

$$
\begin{array}{ll}
g ::= u \mid u \wedge u & \\
u ::= y \vartriangle y & \vartriangle \in \{<, >, \geq, \leq, =, \neq\} \\
y ::= k \mid m(s) & k \text{ is a constant and } m(s) \text{ is the value of measure } m \text{ in state } s
\end{array}
$$

For the implementation we leverage already existing parts of the CARMA tools. In particular, the CARMA specification language already supports the definition of measures to extract observations and study the behaviour of the model. For example, the measure defined by

```
measure Stopped = #{Robot[Stop] | true};
```

would return a time-trajectory of the number of components named `Robot` that are in state `Stop`. For monitoring whether a goal is satisfied or not we are interested in boolean measures. Say we have a goal that the number of `Robot` components in state `Stop` state is less than five. This can be expressed in the following way:

```
measure StoppedLessThan5
        = bool2int( (#{Robot[Stop] | true } < 5));
```

where `bool2int` is a helper function that converts the boolean values to integers so that the measure is still a numerical value as required by the implemented model simulator.

Our simple initial implementation amounts to mapping the defined goals to measures like the `StoppedLessThan5` example given above. However, with the current restriction of the language and its implementation, measure values cannot be accessed in the environment and so they cannot be used in the evolution rule. In order to be able to use the measures directly in making control decisions the CARMA language will need to be modified in future work.

4 Case study

In this section we give a simple example of how the presented ideas of individual-level resource constraints and global goals can be used to analyse a model taken from the domain of swarm robotics.

In particular, we consider a model of a swarm of 10 robots navigating on a 4-by-4 grid. Robots have a choice between actions north and east. The action north corresponds to the movement in the y-axis of the plane and increments the robot's y coordinate by 1. Similarly, east increments the x coordinate. The robots are constrained to the 4-by-4 grid by disabling the actions that would take the robots out of the grid. This is done by setting a guard on the actions which checks whether the incremented location is still within the grid.

By construction of the model each robot starts at the location $(0, 0)$ and moves to $(3, 3)$ by taking one of the possible paths. If we set the rates of the actions to be equal, in this case $r = 1$, then each of the paths is equally likely given no

additional restrictions. The structure of this basic model is illustrated in Figure 2a. Note that in this model there are no interactions between components. Such a model is enough to demonstrate the individual constraints and global goals while being very fast to simulate. Later work will consider more complicated systems that include communication between components. The resource that we consider within this model is the battery life of the robots. The robots are given a local store attribute, denoted b, that keeps track of their battery levels. In particular, we let the attribute value 10 denote the full battery and instantiate each robot in the system with $b = 10$.

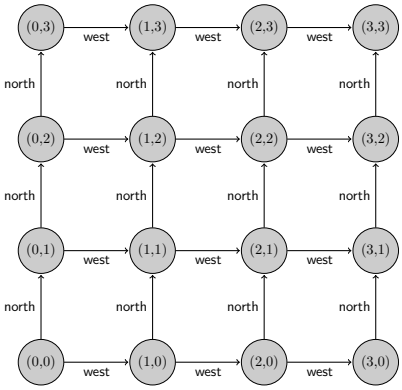
We model battery depletion by setting probabilistic store updates for the actions that move the robot, and we assume that some moves are more costly than others. That is, we define two sets of locations. The first set corresponds to locations where robots operate under normal energy consumption, coloured grey across Figure 2, while the second corresponds to locations where robots have to spend more energy, coloured red on the same figures. In particular, each action performed at a grey location reduces the battery level by 0 or 1 with equal probability. On the other hand, at red locations the battery level is reduced by 4 or 5 with equal probability. We then set the resource constraint $b > 0$ for each robot. The constraint is interpreted in the obvious way: for the robot to be functional its battery has to be not empty.

Each robot needs to perform exactly 6 actions to move from $(0, 0)$ to $(3, 3)$. However, as the battery life is a consideration, some of the possible paths may consume too much battery causing the robot to fail before reaching $(3, 3)$. For example the expected state of the battery indicator for a robot at $(3, 3)$ given it has taken a path involving two red locations is $10 - (4 \times 0.5 + 2 \times 4.5) = -1$. Thus, we are expecting the robots that take a path through two red nodes to violate the constraints.

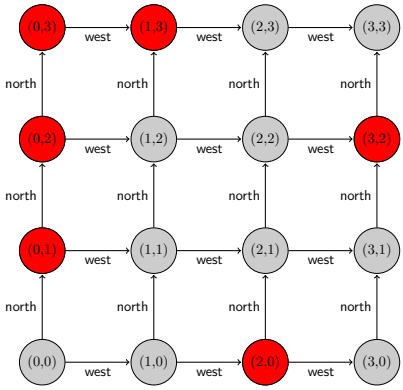
The left-hand side of Figures 3, 4, and 5 plot the expected numbers of robots at $(3, 3)$ and stopped robots against time. The trajectories are obtained by simulating the created CARMA model 5000 times with each trajectory consisting of 100 samples. The mean of trajectories is plotted with 95% confidence intervals. Due to the high number of simulation runs these intervals around the means values are very narrow.

In the scenarios with both red and grey locations, the expected number of robots that run out of battery before reaching $(3, 3)$ is high in all cases — by taking a random path many of the robots will visit two red locations. Thus we have modelled a simple system under individual-level constraints — in this case battery life.

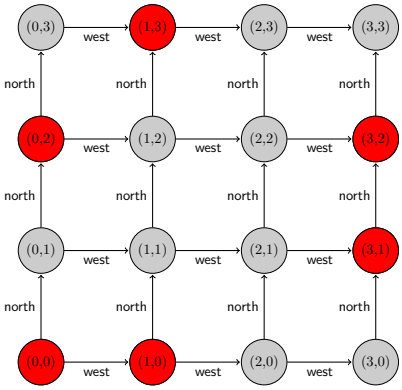
The next step is to exemplify the use of global goals. As explained above, global goals can either be functional targets for the system or some desired global properties. In this case, we are interested in the swarm of robots making it to the location $(3, 3)$. We can specify the number of robots that need to reach the target for the system to successfully fulfil its intended function. Suppose in this case we require five or more robots to reach the location $(3, 3)$ for the goal to be satisfied. Conversely we could also prescribe that no more than five robots can fail to make it to $(3, 3)$, which is equivalent since all the robots that do not fail will reach $(3, 3)$



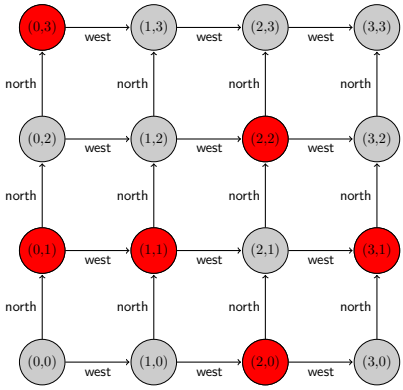
(a) Spatial structure of the robot swarm model with no constraints.



(b) Spatial configuration for the first experiment.



(c) Spatial configuration for the second experiment.



(d) Spatial configuration for the third experiment.

Fig. 2. Spatial structures for the robot swarm models. Red nodes denote the locations where actions require more energy.

by construction. Thus, the global measure of interest is either

```
measure Stopped = #{Robot[nil] | true}
```

giving the number of Robot components that have violated their individual constraints and ended up in a state where they are incapable of performing additional actions, or alternatively,

```
measure RobotsAtTarget
        = #{Robot[ReadyToMove] | my.loc == [3,3]}
```

which gives the number of Robot components that are in location $(3, 3)$ and in the state ReadyToMove indicating a state where they are ready to perform another move action.

The global goal can thus be defined as Stopped ≤ 5. Deriving a measure to record whether the system satisfies this goal was described in Section 3.2. Note

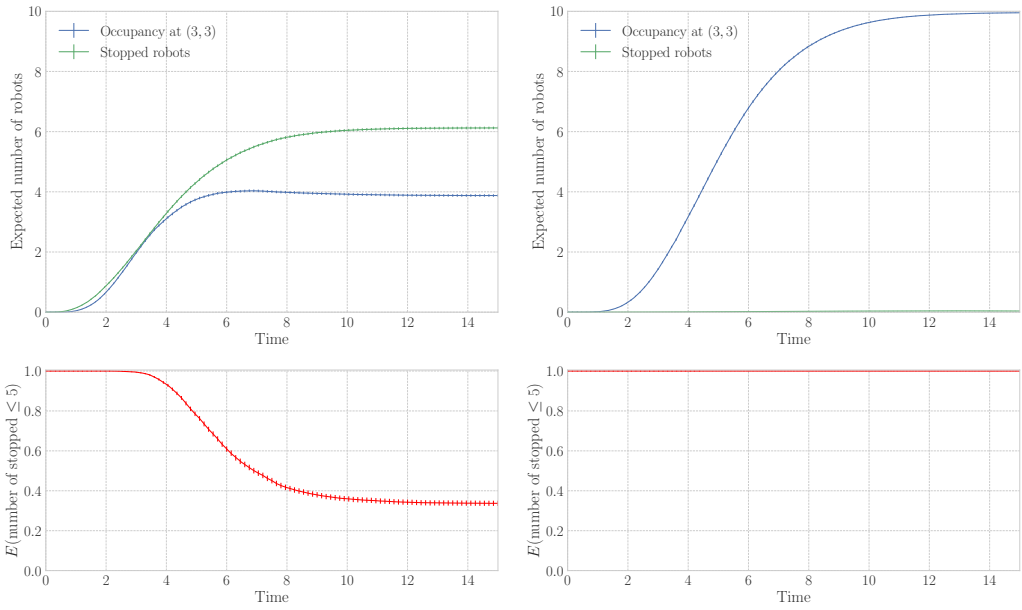


Fig. 3. Simulation results for uncontrolled (left) and controlled (right) system with spatial configuration given by Figure 2b.

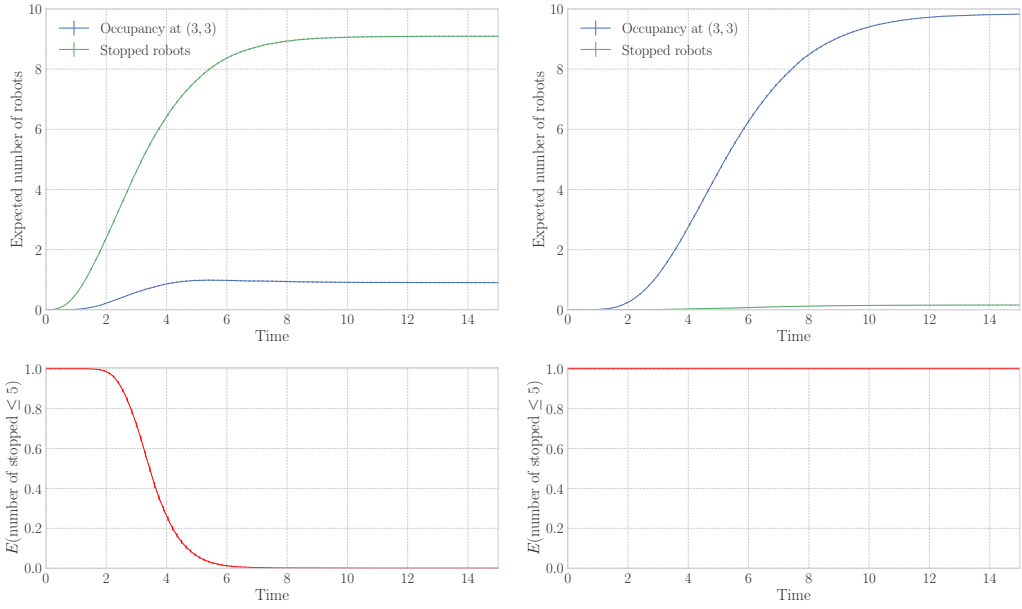


Fig. 4. Simulation results for uncontrolled (left) and controlled (right) system with spatial configuration given by Figure 2c.

that for a single simulation run the result would be a binary function of time. On the other hand for the 5000 simulation runs the output of the CARMA simulation tool is an aggregated measure corresponding to the proportion of trajectories for

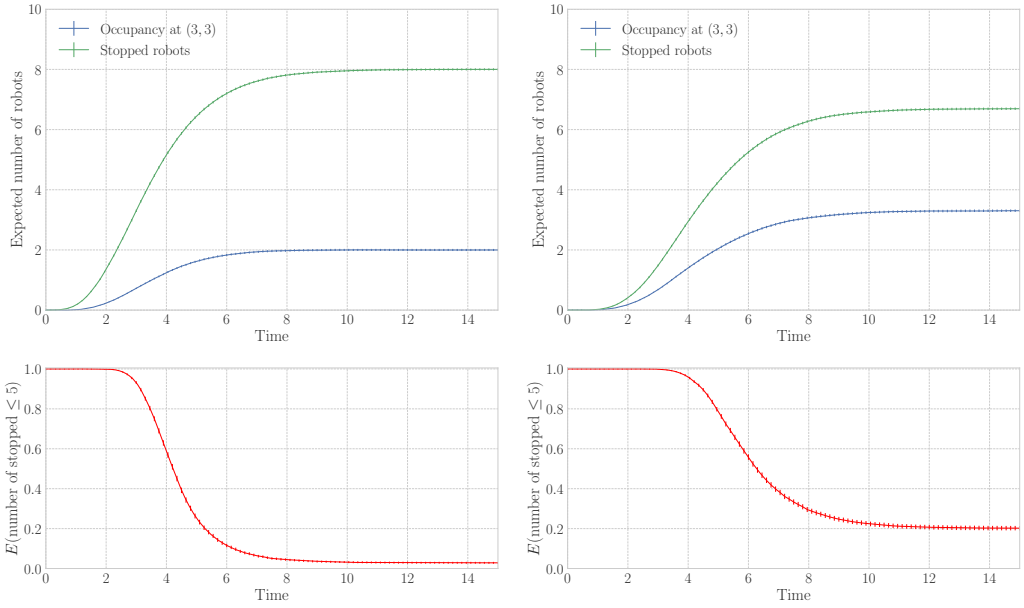


Fig. 5. Simulation results for uncontrolled (left) and controlled (right) system with spatial configuration given by Figure 2d.

which the property holds. We can interpret this as the expected probability with which the goal is satisfied by the system. The time evolutions of the measure corresponding to the goal is given on the left-hand side of Figures 3, 4, and 5, for three distinct configurations of the grid. As expected, in the case of uncontrolled systems, the probability of satisfying the goal is low.

In the case of Figure 3 we point out that the expected goal satisfaction measure ends up being far from the extremes 1 and 0. This means that the stochastic behaviour of the system gives rise to enough variance to cause high uncertainty in the goal satisfaction. Note that in Figure 3 the measure corresponding to the number of robots at location $(3, 3)$ contradicts expectations by not being monotonically increasing. This is a modelling artefact which relates to the discussion in Section 3.1 and is due to the constraints being checked after the action has been performed. In particular, some robots will reach the target location $(3, 3)$ but immediately move into the failed state. In this example, we have chosen to treat them as failed robots.

We can now study how the expectation of satisfying the goal changes when a simple control policy is applied. In particular, suppose a robot can sense whether it is moving to a red or a grey location. With this information one can give a control policy: if the robot has a choice between a red location and a grey location then it will pick the grey. Again, the measure trajectories resulting from simulating the controlled system for three different spatial configurations are given on the right-hand side of Figures 3, 4, and 5. We have plotted the expected number of robots at $(3, 3)$, stopped robots and the expected probability of the goal being satisfied.

In the case of spatial configurations given in Figures 2b and 2c with the cor-

responding simulation results in Figures 3 and 4 we see that all the robots are expected to reach the location $(3, 3)$ with high probability. This is indicated by the expected occupancy measure at location $(3, 3)$ growing to 10 and the expected number of stopped robots remaining very low. Note in these cases the simple control policy is enough to make the robots take the paths that feature at most one red location. Thus, the worst case scenario where the robots reach the location $(3, 3)$ with battery level 0 happens with a low probability of $\left(\frac{1}{2}\right)^6$ for spatial configuration in Figure 2b and probability of $\left(\frac{1}{2}\right)^5$ for configuration in Figure 2c. This makes any single robot stopping unlikely and hence we expect the goal to be satisfied with very high probability.

In the case of spatial configuration given in Figures 2d and simulation trajectories in Figure 5 we see that, although improved very slightly, most of the robots are expected to fail to reach $(3, 3)$ even under the given control policy. This is explained by the placement of red locations. All of the robots are first steered to the location $(1, 0)$ from where the locations $(1, 1)$ and $(2, 0)$ are equally probable. All of the robots that choose to go to $(2, 0)$ are likely to fail to get to $(3, 3)$ as all paths from $(2, 0)$ require going though another red location. From $(1, 1)$ robots again have two equally probable destinations — in particular, $(1, 2)$ and $(2, 1)$. The robots taking the path through $(2, 1)$ are again likely to fail while only the robots going through $(1, 2)$ would arrive at the destination with high probability.

5 Translation tool

We have developed a software tool to experiment with the proposed changes to the Carma language. The implementation includes an extension to the parsing machinery, to support the new syntax, but does not change the simulation framework. Instead, it transforms a model so that it only uses the standard Carma syntax, and can thus be analysed and simulated with the existing software.

In the Carma specification language, a model includes one or more variants called *scenarios* (denoted with the `system` keyword in Figure 1), which describe different initialisations of the model and environment. In the extended syntax, each scenario also specifies which individual constraints and global goals should be considered. This allows different variants to be easily created by choosing from the full set of constraints and goals defined.

The tool takes as input a Carma model and a scenario name, and, by examining the syntax tree generated by the parser, constructs a new Carma model which reflects the specified constraints and goals. The resulting model differs from the original in three ways. Firstly, it only includes the chosen scenario, with the rest being discarded. Secondly, the component definitions are modified appropriately by adding new guards and processes, following the mapping in Section 3.1. Thirdly, each goal is replaced by a measure, whose value at any given time is 1 if the goal is satisfied or 0 if it is not. The transformation process is described in pseudocode in Algorithm 1.

This prototype implementation allows for easy experimentation with the new

language features. While simple, it is a first step towards integration of the changes proposed here. We envision that a future release of the CARMA tools will include not only support for the syntax, but also modifications to the simulator to fully and natively account for the presence of goals and constraints, with a view to allowing goal-based control of execution.

Algorithm 1 Sketch of the transformation performed by the implementation

```
Retrieve all individual constraints for the given scenario
for each component do
    Create the conjunction of all the constraints
    Add guards and stop process as in Section 3.1
end for
for each global goal do
    Create a measure expressing whether the goal is satisfied as in Section 3.2
end for
Remove the goals and constraints from the scenario definition
Remove other scenarios from the model
```

6 Conclusion

Stochastic process algebras have been shown to be a useful modelling paradigm for analysis of emergent phenomena in CAS [3,12,11,5]. The work in this paper is motivated by studying emergent behaviour in the case of highly-distributed computer-based systems like robot swarms and wireless sensor networks. In particular, we are interested in applying a process algebraic framework to guide the design of control policies for such systems.

Winfield *et al.* [14] is an example of an early work on applying formal methods to study of CAS and in particular robot swarms. The paper presented a robot swarm model where actions of individuals were given in terms of linear temporal logic formulae. The aim was to use automatic theorem proving methods to prove or disprove whether the given emergent property, also expressed in linear temporal logic, holds. A more closely related approach, applying formal methods and modelling to the design of CAS and specifically swarm robotics, was taken by Brambilla *et al.* [4] and Konur *et al.* [8]. Both papers dealt with model checking properties of a population-based swarm model. Finally, there has been recent work on the use of formal models of robot swarms in conjunction with logic-based specifications of desired properties for automatic synthesis of control policies [7,9].

In this paper we presented an extension of the CARMA process algebra and its accompanying modelling tools. Firstly, we noted that CAS often feature agents that are resource-constrained — for example, by their battery life. We extended the CARMA language to accommodate prescribing individual-level constraints which when violated cause the given component to fail. The second extension was motivated by noting that the desired behaviour of CAS is often expressed in terms of the system-level behaviour rather than that of individual components. We proposed an

extension of the CARMA modelling language for specifying desired system properties and functional goals in terms of global measures of the system. The extension provides basic monitoring capabilities that provide a valuable platform for later work on reasoning about, and devising, control policies.

An example from the domain of swarm robotics was presented to demonstrate the use of resource constraints and global goals in the analysis of CAS. In particular, a model of a robot swarm was given a simple control policy and compared to the uncontrolled case. In both cases simulated trajectories of the system were monitored for satisfaction of the prescribed global goal.

As further work we will consider ways in which goal monitoring can be used to adapt the components, through the operation of the evolution rule in the environment, in order to steer the system towards satisfying the prescribed goals.

Acknowledgement

This work was supported by grant EP/L01503X/1 for the University of Edinburgh School of Informatics Centre for Doctoral Training in Pervasive Parallelism (http://pervasiveparallelism.inf.ed.ac.uk/) from the EPSRC.

References

[1] *CARMA Eclipse Plugin — QUANTICOL Toolset for the analysis of Collective Adaptive Systems.* URL http://quanticol.sourceforge.net/?page_id=27

[2] Aziz, A., K. Sanwal, V. Singhal and R. Brayton, *Model-checking continuous-time markov chains*, ACM Trans. Comput. Logic **1** (2000), pp. 162–170.
URL http://doi.acm.org/10.1145/343369.343402

[3] Bortolussi, L., D. Latella and M. Massink, *Stochastic process algebra and stability analysis of collective systems*, in: *Coordination Models and Languages, 15th International Conference, COORDINATION 2013, Held as Part of the 8th International Federated Conference on Distributed Computing Techniques, DisCoTec 2013, Florence, Italy, June 3-5, 2013. Proceedings*, 2013, pp. 1–15.
URL https://doi.org/10.1007/978-3-642-38493-6_1

[4] Brambilla, M., A. Brutschy, M. Dorigo and M. Birattari, *Property-driven design for robot swarms: A design method based on prescriptive modeling and model checking*, ACM Trans. Auton. Adapt. Syst. **9** (2014), pp. 17:1–17:28.
URL http://doi.acm.org/10.1145/2700318

[5] Coronato, A., V. D. Florio, M. Bakhouya and G. D. M. Serugendo, *Formal modeling of socio-technical collective adaptive systems*, in: *Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2012, Lyon, France, September 10-14, 2012*, 2012, pp. 187–192.
URL https://doi.org/10.1109/SASOW.2012.40

[6] Haghighi, I., A. Jones, Z. Kong, E. Bartocci, R. Grosu and C. Belta, *Spatel: a novel spatial-temporal logic and its applications to networked systems*, in: *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015*, 2015, pp. 189–198.
URL http://doi.acm.org/10.1145/2728606.2728633

[7] Haghighi, I., S. Sadraddini and C. Belta, *Robotic swarm control from spatio-temporal specifications*, in: *55th IEEE Conference on Decision and Control, CDC 2016, Las Vegas, NV, USA, December 12-14, 2016*, 2016, pp. 5708–5713.
URL https://doi.org/10.1109/CDC.2016.7799146

[8] Konur, S., C. Dixon and M. Fisher, *Analysing robot swarm behaviour via probabilistic model checking*, Robotics and Autonomous Systems **60** (2012), pp. 199–213.
URL https://doi.org/10.1016/j.robot.2011.10.005

[9] Lopes, Y. K., S. M. Trenkwalder, A. B. Leal, T. J. Dodd and R. Groß, *Supervisory control theory applied to swarm robotics*, Swarm Intelligence **10** (2016), pp. 65–97.
URL https://doi.org/10.1007/s11721-016-0119-0

[10] Loreti, M. and J. Hillston, *Modelling and analysis of collective adaptive systems with CARMA and its tools*, in: *Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems - 16th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2016, Bertinoro, Italy, June 20-24, 2016, Advanced Lectures*, 2016, pp. 83–119.
URL https://doi.org/10.1007/978-3-319-34096-8_4

[11] Massink, M., M. Brambilla, D. Latella, M. Dorigo and M. Birattari, *On the use of Bio-PEPA for modelling and analysing collective behaviours in swarm robotics*, Swarm Intelligence **7** (2013), pp. 201–228.
URL http://link.springer.com/10.1007/s11721-013-0079-6

[12] Massink, M., D. Latella, A. Bracciali and J. Hillston, *Modelling non-linear crowd dynamics in bio-pepa*, in: *Fundamental Approaches to Software Engineering - 14th International Conference, FASE 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, 2011, pp. 96–110.
URL https://doi.org/10.1007/978-3-642-19811-3_8

[13] Nenzi, L. and L. Bortolussi, *Specifying and monitoring properties of stochastic spatio-temporal systems in signal temporal logic*, in: *8th International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS 2014, Bratislava, Slovakia, December 9-11, 2014*, 2014.
URL https://doi.org/10.4108/icst.valuetools.2014.258183

[14] Winfield, A. F., J. Sa, M.-C. Fernndez-Gago, C. Dixon and M. Fisher, *On formal specification of emergent behaviours in swarm robotic systems*, International Journal of Advanced Robotic Systems **2** (2005), p. 39.
URL http://dx.doi.org/10.5772/5769