



HAL
open science

Modelling Ambulance Deployment with CarmaCARMA

Vashti Galpin

► **To cite this version:**

Vashti Galpin. Modelling Ambulance Deployment with CarmaCARMA. 18th International Conference on Coordination Languages and Models (COORDINATION), Jun 2016, Heraklion, Greece. pp.121-137, 10.1007/978-3-319-39519-7_8 . hal-01631713

HAL Id: hal-01631713

<https://inria.hal.science/hal-01631713v1>

Submitted on 9 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Modelling ambulance deployment with CARMA

Vashti Galpin

Laboratory for Foundations of Computer Science
School of Informatics, University of Edinburgh
Vashti.Galpin@ed.ac.uk

Abstract. CARMA is process-algebra influenced language for the quantitative modelling of collective adaptive systems which involve collaboration and coordination. These systems consist of multiple components that interact to achieve certain goals and that adapt to changes in the environment. As a case study for the application of CARMA, this paper presents an ambulance deployment system where ambulances go to medical incidents and either treat patients at the scene or transfer them to hospital. The Eclipse CARMA Plug-in is used to simulate the system, and demonstrate its behaviour in different circumstances.

1 Introduction

Creating formal dynamic models of systems that can be simulated and subjected to other forms of quantitative analysis is one way in which formal methods can be used in the development and evaluation of these systems. Frequently, it is not possible to experiment with the system itself, because of the cost or disruption involved. Hence development of models that can be used for experimentation is important. This paper applies a language that has been developed to model collective adaptive systems (CAS) to an existing ambulance deployment scenario.

CAS feature frequently in modern information systems. Multiple components interact (and sometimes compete) to achieve various outcomes. The components can be individual pieces of software or different physical devices, and such systems are often characterised by local information and local action which therefore, requires a notion of space. Coordination and collaboration are features of these models because of the components communicate to achieve their aims. The language CARMA and its associated software tool the Eclipse CARMA Plug-in have been developed for the quantitative modelling of CAS allowing for an understanding of both functional and nonfunctional properties of models [3, 5]. Important aspects of CARMA include attribute-based communication, in the sense that the possibility of a component taking part in an interaction depends on the current values in the store of the component, thus allowing for a rich representation of state. Both unicast and broadcast modes of communication are supported. Furthermore, CARMA allows the environment within which the model components interact to be defined separately from the components. The development of CARMA has been influenced by a number of previous process algebra including the Markovian process algebra PEPA [12], the location-focussed PALOMA [9] and SCEL [8] which uses attribute-based communication. Attribute-based communication is explored further in the process calculus *AbC* [2].

In this paper, the modelling and analysis of a particular system using CARMA is considered. Jagtenberg *et al* have proposed a new approach to ambulance deployment [14]. The general goal of such systems is to minimise the time it takes to respond to medical incidents by ensuring good base locations for ambulances together with a distribution of ambulances over bases that leads to fast response. Traditionally, deciding how to deploy ambulances across a region has been done statically, in the sense that once an ambulance has completed its current task, it returns to a predefined base, and moreover determining the best bases is done in advance of deployment. In the dynamic approach, depending on the locations of the other ambulances, an ambulance that is no longer busy can be requested to go to a specific location in a set of base locations to wait for its next task, thus allowing the system to adapt to the current circumstances. The ambulance system is modelled as a graph of locations with edges representing roads, annotated with information about how long it takes to traverse the edge, as shown in Figure 5. Locations may be cities, towns, road junctions or other points of interest. Each location has an incident probability, and some locations have ambulance bases or hospitals. There has been much research into different aspects of ambulance response time that use a graph to represent the road network. These have considered how many ambulances to use, the best locations for bases and how to distribute vehicles over bases [1, 4, 6, 10, 15, 16]. The specific system modelled here has been chosen because of its time-based performance evaluation aspects and straightforward heuristic function. The developed model could be modified to investigate other aspects of ambulance deployment.

This paper presents a CARMA model of such an ambulance system. As CARMA is a new modelling language, it is necessary to evaluate it by developing interesting and complex models, and the ambulance deployment system fulfils these requirements. First, the ambulance scenario is introduced after which details of CARMA are presented, followed by the ambulance model expressed in CARMA. Finally results of simulation of the model are presented and a discussion of further research relating to the model discussed.

2 An ambulance deployment scenario

This section describes the mathematical model that Jagtenberg *et al* [14] propose, together with their heuristic for best real-time redeployment of ambulances. This model considers a scenario where there is a fairly sparse network of roads between cities and towns (as shown in Figure 5), and hence is slightly more appropriate for a non-urban situation, where there are not many routes between each point of interest. An urban map can be transformed into a similar sparse network by focussing on major routes.

Let $N = (V, E)$ be a graph where $E \subseteq V \times V$. Four functions are associated with this graph.

- $r : E \rightarrow \mathbb{R}_{>0}$ describes the time it takes to traverse an edge (when using sirens and lights – this figure increases for travel without lights and sirens).
- $h : V \rightarrow \{0, 1\}$ defines the presence of a hospital at a vertex.
- $b : V \rightarrow \{0, 1\}$ defines whether a vertex is an ambulance base.
- $d : V \rightarrow [0, 1]$ with $\sum_{v \in V} d(v) = 1$, defines the distribution of incidences over the vertices of the graph.

There is also a set of ambulances A labelled $\{1, \dots, n\}$ and two functions $l : A \rightarrow V$ describing the current location of an ambulance and $w : A \rightarrow \{0, 1\}$ describing whether an ambulance is currently allocated to an incident. Furthermore, a function $s : A \rightarrow V$ which describes the home station of an ambulance in the static case. There are three rates that describe how long it takes for an ambulance to treat a patient at the scene, λ_p , how long it takes for an ambulance to load up a patient at the scene for transportation to hospital, λ_r and how long it takes to offload a patient at hospital, λ_d . It is assumed that patients are either treated at the scene or uploaded to be taken to hospital but not both. Furthermore, there is a probability m that determines whether an incident is severe, requiring the patient to be transported to hospital, or minor, meaning that the patient only needs to be treated at the scene. The operation of the system is now described.

1. An incident occurs at a vertex based on the distribution defined by d and its level of severity is determined using the probability m .
2. An ambulance is identified to go to the incident location based on distance from the incident.
3. The ambulance uses the shortest route to get to the scene. Since distances are deterministic and unchanging, shortest routes can be determined in advance from the network, and hence are static in the model.
4. The ambulance treats the patient at the scene and then proceeds with item 7, or the ambulance uploads the patient to take them to hospital.
5. The ambulance uses the shortest route to the hospital using sirens and lights.
6. The ambulance drops the patient off at the hospital.
7. The ambulances uses the shortest route to go to a base but taking longer as it is not using sirens and lights. In the static case, this is the base defined by s . In the dynamic case, the base is determined by the heuristic.

Once an ambulance has been allocated to an incident, it must complete the journey to the incident, and to the hospital, if necessary, and cannot be diverted. However, once an ambulance has started to return to base, it can be immediately allocated to a new incident. An ambulance that is involved in items 2 to 6 is considered to be busy, otherwise it is idle. When it is idle, it is associated either with the base it has reached or the base to which it is travelling.

2.1 Evaluation of base heuristic

The reason to model such systems is to investigate the performance of different deployment approaches. The proportion of ambulances that do not reach the incident within a fixed time period (denoted T) after being allocated to that incident, is often used [14] and will be used here.

In the case of static deployment where each ambulance has a fixed base, simulation can be used to assess the best distribution of ambulances, in the planning stages of a system. Different variants of the maximum expected covering location problem (MEX-CLP) [6] have been used to tackle this task. The disadvantage of the static approach is that it ignores real-time information that can be used to provide better coverage [14].

Coverage describes the number of ambulances that can provide service at a specific location within the time limit. Increasing coverage means that there is a higher probability that an ambulance will be available if one is needed.

Dynamic deployment approaches can be divided into two main techniques. The first, based on lookup tables, requires dispatchers to steer the system to these optimal configurations. A better approach is based on real-time approximation. However, even using approximate dynamic programming and post-decision state is time-consuming and requires an expert to implement and choose base functions [14]. By contrast, the approach taken in [14] is moderately coarse-grained and needs little real-time information, and takes a marginal cover approach based on MEXCLP.

Consider a set of ambulances A with behaviour as described above. Also let $B = \{u \in V | b(u) = 1\}$ be the set of vertices that are bases. It is assumed that there is a probability q which is the same for all ambulances and represents the fraction of time that the ambulance is busy. It can be determined by dividing the load of the system by the number of ambulances [14]. The expected coverage at vertex v when v is in the range of k ambulances is $E_k(v) = d(v)(1 - q^k)$ and as shown in [6], the marginal coverage is $E_k(v) - E_{k-1}(v) = d(v)(1 - q)q^{k-1}$. This figure can be used to determine to which base to send an ambulance once it has completed its task, by finding $u \in B$ such that the increase in coverage is maximised, and hence the maximum coverage overall is obtained. Let $\rho(v, w)$ be the time taken for the shortest route between vertices v and w , calculated from the values of individual hops given by the function r . Let n_u represent the number of ambulances at $u \in B$, or moving towards $u \in B$ after completing their allocated task, and let $N = \{n_u | u \in B\}$ represent the current number of idle ambulances for each possible base location. The function p captures the heuristic for determining ambulance base and is defined by¹

$$p(N) = \arg \max_{w \in B} \sum_{v \in V} d(v)(1 - q)q^{k(v, w, N)-1} \cdot \mathbf{1}(\rho(w, v) \leq T) \quad \text{where}$$

$$k(v, w, N) = \sum_{u \in B} n_u \cdot \mathbf{1}(\rho(u, v) \leq T) + \mathbf{1}(\rho(w, v) \leq T).$$

Here $\mathbf{1}$ is the indicator function. This function can also be simplified to

$$p(N) = \arg \max_{w \in B} \sum_{v \in V} d(v)(1 - q)q^{c(v, N)-1} \cdot \mathbf{1}(\rho(w, v) \leq T) \quad \text{where}$$

$$c(v, N) = \sum_{u \in B} n_u \cdot \mathbf{1}(\rho(u, v) \leq T)$$

For each possible base of the ambulance that has just completed its task, the increased coverage is calculated for every location in the graph whenever addition of the base would increase coverage at that location, and summed. The function c counts the number of ambulances that are already in the range of each base (reachable within the limit of T time units) as defined by the set N . If it is the case that both $|A|$, the number of idle ambulances, and $|B|$, the number of bases, are small compared to the number of vertices $|V|$, then the algorithm is linear in $|V|$ [14].

¹ This function differs from that in [14] due to the additional term $\mathbf{1}(\rho(u, v) \leq T)$ in the first part of the definition. It does, however, match the algorithm that was used in that paper [13].

It is important to note that some aspects of the model are generic such as the ambulance behaviour but others are specific to the system under consideration such as the graph of locations, the functions relating to these locations, and the heuristic function p . This distinction will be used when deciding how to build the CARMA model. The next section introduces CARMA, after which the CARMA model of the system is presented.

3 CARMA

CARMA is a powerful language, influenced by process algebra, for describing systems consisting of different interacting components which allows for an explicit definition of environment. Its semantics are time-inhomogeneous continuous-time Markov chains [5] thus permitting both simulation and other analysis techniques. It is embodied in software in the CARMA Eclipse Plug-in tool which implements the basic language and supports features such as function definition, enumerated types, and measures which enable quantitative behaviour of a model to be calculated and recorded.

A CARMA model consists of a number of different elements. At the highest level, there is a collective that consists of different components that interact, together with an environment that contains information about the global state, as well as information about how components interact. Thus SYS is the set of CARMA *systems* S defined by

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

where N is a *collective* and \mathcal{E} is an *environment*. The set of collectives COL is defined by

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

A collective N is either a *component* C or the parallel composition of two collectives ($N \parallel N$). The syntax of components is

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

where $\mathbf{0}$ is the null component, P is a process that describes the behaviour of the component and γ is the store for the component. COMP is defined to be the set of components. A store maps from *attribute names* to *basic values* where

- ATTR is the set of *attribute names* $a, a', a_1, \dots, b, b', b_1, \dots$;
- VAL is the set of *basic values* v, v', v_1, \dots ; and
- Γ is the set of *stores* $\gamma, \gamma_1, \gamma', \dots$, are functions from ATTR to VAL.

PROC is the set of processes that define the behaviour of components and they are specified by

$$\begin{array}{l|l}
 P, Q ::= \mathbf{nil} & act ::= \alpha^*[\pi](\vec{e})\sigma \\
 \mid \mathbf{kill} & \mid \alpha[\pi](\vec{e})\sigma \\
 \mid act.P & \mid \alpha^*[\pi](\vec{x})\sigma \\
 \mid P + Q & \mid \alpha[\pi](\vec{x})\sigma \\
 \mid P \mid Q & \\
 \mid [\pi]P & e ::= a \mid \text{my}.a \mid x \mid v \mid \text{now} \mid \dots \\
 \mid A \quad (A \triangleq P) & \pi ::= \top \mid \perp \mid e_1 \bowtie e_2 \mid \neg\pi \mid \pi \wedge \pi \mid \dots
 \end{array}$$

In CARMA processes can have different prefixes relating to four types of actions which are *broadcast output* ($\alpha^*[\pi](\vec{e})\sigma$), *broadcast input* ($\alpha^*[\pi](\vec{x})\sigma$), *output* ($\alpha[\pi](\vec{e})\sigma$), and *input* ($\alpha[\pi](\vec{x})\sigma$), where

- α is an *action type* in the set of action type ACTTYPE;
- π is a *predicate*;
- x is a *variable* in the set of variables VAR;
- e is an *expression* in the set of expressions EXP;
- $\vec{\cdot}$ 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 .

A unicast communication involves two components where the sender and receiver attributes must satisfy any predicates in the prefixes, the expressions in the output prefix are assigned to the variables in the input prefix (and hence successful communication requires the two sequences are the same length), and updates are applied to both components to complete the interaction. Furthermore, there can be a probability describing whether the receiver does actually receive the communication. Unicast is blocking in that a sender cannot proceed until a receiver takes part in the interaction.

By contrast, broadcast is not blocking and the sender can proceed regardless of whether there are many, one or no suitable receivers. Again, the sender and potential receivers must satisfy the predicates in the prefixes, and additionally there is a probability that the receiver although suitable to take part in the interaction, does not receive it. Expressions from the output prefix are passed to variables in the input prefix and updates are applied to all participants on completion of their roles, in the same manner as unicast. For both unicast and broadcast, rates of actions and probabilities of receiving are defined in the environment part of the model.

Specific expressions of interest are now for the current simulation time and $my.a$ which refers to the value of the attribute a in the current component.

Apart from the four different prefix types, there is choice between two processes, the parallel composition of two processes, a guarded process which requires satisfaction of its predicate before it can perform an action and definition of constant processes. There are two distinct operators for termination. The operator **nil** represents the process that can perform no further actions and it can be placed in parallel with other processes. The operator **kill**, on the other hand, indicates termination of the whole component so that all processes in the component stop, and the component is transformed into **0**, the null component which can do nothing and has no store.

CARMA collectives interact within an environment \mathcal{E} . The environment describes the rules that regulate the system such as rates of interaction and probabilities that interaction may occur. It also contains global information. The environment has two elements: a *global store* γ_g , that records the value of global attributes, and an *evolution rule* ρ . This is a function which, depending on the *current time* (using *now*), on the global store and on the current state of the collective returns a tuple of functions $\varepsilon = \langle \mu_p, \mu_r, \mu_u \rangle$ known as the *evaluation context* where $ACT = ACTTYPE \cup \{\alpha^* | \alpha \in ACTTYPE\}$ and

- $\mu_p : \Gamma \times \Gamma \times \text{ACT} \rightarrow [0, 1]$, $\mu_p(\gamma_s, \gamma_r, \alpha)$ determines is the probability that a component with store γ_r can receive a message from a component with store γ_s when α is executed;
- $\mu_r : \Gamma \times \text{ACT} \rightarrow \mathbb{R}_{\geq 0}$, $\mu_r(\gamma, \alpha)$ determines the execution rate of action α executed at a component with store γ ;
- $\mu_u : \Gamma \times \text{ACT} \rightarrow \Sigma \times \text{COL}$, $\mu_u(\gamma, \alpha)$ determines the updates on the environment (global store and collective) induced by the execution of action α at a component with store γ . The execution of an action can modify the values of global variables and also add new components to the collective.

In each of the rules, the notation $\text{sender}.a$ is used to refer to the value of the attribute a in the store of the acting or sending component, and $\text{receiver}.a$ refers to the value of the attribute a in the store of the receiving component.

Operational semantics of CARMA specifications are defined in three stages using the following transition relations. For reasons of space, the rules are not presented here, but can be found in [5].

1. The relation \longrightarrow describes the behaviour of a single component.
2. The relation \longrightarrow builds on the first relation to describe the behaviour of collectives.
3. The relation \mapsto describes how CARMA systems evolve.

All relations are defined in the FUTS style [7] and are described using a triple (N, ℓ, \mathcal{N}) where the first element is a component, or a collective, or a system. The second element is a transition label. The third element is a function associating each component, collective, or system with a non-negative number. A non-zero value represents the rate of the exponential distribution characterising the time needed for the execution of the action represented by ℓ . The zero value is associated with unreachable terms. FUTS style semantics are used because it makes explicit an underlying (time-inhomogeneous) Action Labelled Markov Chain, which can be simulated with standard algorithms [11].

4 Ambulance model

The CARMA model is presented in Figures 1, 2, 3 and 4. Each component consists of the attributes that form its local store, its behaviour defined by processes and its initial state. Additionally, the attributes of the global store are defined in Figure 4 together with the evolution rule functions over actions (one function for the rates, one for the probabilities and one for the updates of global attributes and additions to the collective). There are four actions with non-negligible rates, and the remainder are zero or fast (since CARMA does not currently support instantaneous actions). Figure 4 also includes the initial collective definition as the last item.

The symbols \top and \perp as used for true and false, respectively. A broadcast action of the form $\alpha^*[\perp]\langle \rangle$ is an action that cannot be received (because no component can satisfy the predicate false) and hence is local to the component that executes it, although it may also update the global store or add components to the collective.

As mentioned previously, some aspects of the model are generic and some are specific to the network of roads and places. These two concerns have been separated in

Store of Incident_Queue component:

inum number of incidents generated
rnum next incident to be dealt with

Behaviour of Incident_Queue component:

$IG \stackrel{\text{def}}{=} \text{incident}^*[\perp](\langle \rangle)\{inum \leftarrow inum + 1\}.IG$

$RN \stackrel{\text{def}}{=} \text{release}[\top](\langle rnum \rangle).RN'$

$RN' \stackrel{\text{def}}{=} \text{confirm}^*[\top](an, al)\{rnum \leftarrow rnum + 1\}.RN$

Initial state of Incident_Queue component: $IG \mid RN$

Store of Incident_Queue_Item component:

qnum number of incident
itime time of incident

Behaviour of Incident_Queue_Item component:

$IQI \stackrel{\text{def}}{=} \text{release}[my.qnum == n](n).IQI'$

$IQI' \stackrel{\text{def}}{=} \text{new_handler}^*[\perp](\langle \rangle).\text{kill}$

Initial state of Incident_Queue_Item component: IQI

Store of Incident_Handler component:

loc location of incident or hospital
anum id of ambulance assigned to incident
aloc current location of ambulance assigned to incident
dest current destination type and incident type
itime time of incident
atime arrival time at incident

Behaviour of Incident_Handler component:

$IH \stackrel{\text{def}}{=} \text{request_ambulance}^*[\top](\langle loc \rangle).IH_C$

$IH_C \stackrel{\text{def}}{=} \text{confirm}^*[an, al](\{anum \leftarrow an, aloc \leftarrow al\}).IH_I$

$IH_I \stackrel{\text{def}}{=} \text{makeroute}^*[\perp](\langle \rangle).IH_S$

$IH_S \stackrel{\text{def}}{=} \text{arrive}^*[an == my.anum](an).IH_T$

$IH_T \stackrel{\text{def}}{=} \text{timecheck}^*[\perp](\langle \rangle)\{atime \leftarrow now\}.IH_P$

$IH_P \stackrel{\text{def}}{=} \text{pickup}^*[an == my.anum](an)\{aloc \leftarrow my.loc, \\ loc \leftarrow \mathbf{HospitalLocation}(my.loc), dest \leftarrow ToHosp\}.IH_H + \\ \text{treat}^*[an == my.anum](an)\{aloc \leftarrow my.loc, dest \leftarrow ToBase\}.IH_F$

$IH_H \stackrel{\text{def}}{=} \text{makeroute}^*[\perp](\langle \rangle).IH_D$

$IH_D \stackrel{\text{def}}{=} \text{dropoff}^*[an == my.anum](an)\{aloc \leftarrow my.loc, dest \leftarrow ToBase\}.IH_F$

$IH_F \stackrel{\text{def}}{=} \text{tobase}^*[\perp](\langle \rangle).\text{kill}$

Initial state of Incident_Handler component: IH

Fig. 1. Incident queue, incident queue item and incident handler components

Store of Return_Handler component:

anum id of ambulance assigned
aloc current location of ambulance
dest current destination and incident type
loc location of base of ambulance

Behaviour of Return_Handler component:

$$RH \stackrel{\text{def}}{=} \text{tell_base}[\top](\langle loc \rangle).RH'$$

$$RH' \stackrel{\text{def}}{=} \text{makeroute}^*[\perp](\langle \rangle).RH'' + \text{kill_handler}^*[\text{my.anum} == \text{an}](\text{an}).\text{kill}$$

$$RH'' \stackrel{\text{def}}{=} \text{atbase}^*[\text{my.anum} == \text{an}](\text{an}).\text{kill} + \text{kill_handler}^*[\text{my.anum} == \text{an}](\text{an}).\text{kill}$$

Initial state of Return_Handler component: *RH*

Store of Closest_Idle_Ambulance component:

iloc location of incident
dloc location of idle ambulances
t timer variable for timeout

Behaviour of Closest_Idle_Ambulance component:

$$CIA \stackrel{\text{def}}{=} \text{request_ambulance}[\top](l)\{\text{iloc} \leftarrow l, \text{dloc} \leftarrow \mathbf{ClosestIdleLoc}(\text{iloc}, N), t \leftarrow \text{now}\}.CIA'$$

$$CIA' \stackrel{\text{def}}{=} \text{request}[\top](\langle \text{dloc} \rangle).CIA'' + \text{pause}^*[\perp](\langle \rangle)\{\text{dloc} \leftarrow \mathbf{ClosestIdleLoc}(\text{iloc}, N), t \leftarrow \text{now}\}.CIA'$$

$$CIA'' \stackrel{\text{def}}{=} \text{confirm}^*[\top](\text{an}, \text{al}).CIA$$

Initial state of Return_Handler component: *CIA*

Fig. 2. Return handler and idle ambulance components

the model. The components are generic, and functions (indicated in bold in the figures) embody the knowledge of the network². The CARMA Eclipse Plug-in supports function definitions, hence this separation is both possible and sensible, and also supports the design of a tool for ambulance modelling, as discussed in the further work section.

These functions comprise of one to provide the distributions over location and type of incidents respectively, **IncidentLocation()**, **IncidentType()**; information about routes in the network, **RouteLength(.,.)**, **NextHop(.,.,.)** and **MoveTime(.,.,.)**; location of the closest hospital to an incident, **HospitalLocation(.)**; location of the closest idle ambulances to an incident location, **ClosestIdleLoc(.,N)**; and the base to which an ambulance should go, **GetBase(.,.,N)** which can be defined statically or dynamically. Note that both **ClosestIdleLoc** and **GetBase** take *N*, the set of the counts of idle ambulances at each base or on their way to each base as an argument. For the former, *N* is used to find the closest location to the incident where there are idle ambulances, so that a request can be sent to ambulances in that location. For the latter, the counts of idle ambulances are required to calculate *p*.

There are seven generic components in the model. The *Incident_Queue* generates *Incident_Queue_Items* using the action *incident** which has the side-effect of adding a new

² A different approach is to use components to embody the knowledge of the network and for the generic components to communicate with these components to obtain this information, but this leads to increase complexity of interaction. Alternatively, the environment could contain this knowledge.

Store of Ambulance component:

anum ambulance id
aloc current location of ambulance
abase current base of ambulance
idle whether ambulance is idle or not

Behaviour of Ambulance component:

$Idle \stackrel{\text{def}}{=} \text{request}[al == \text{my.aloc}](al)\{idle \leftarrow \perp\}.Respond$
 $Respond \stackrel{\text{def}}{=} \text{confirm}^*[\top](\langle anum, aloc \rangle).Busy$
 $Busy \stackrel{\text{def}}{=} \text{end_move}[an == \text{my.anum} \wedge \text{now} \geq t + d](an, al, t, d)\{aloc \leftarrow al\}.Busy +$
 $\text{arrive}^*[an == \text{my.anum}](an).AtScene +$
 $\text{drop_off}^*[an == \text{my.anum}](an)\{idle \leftarrow \top\}.AskBase$
 $AtScene \stackrel{\text{def}}{=} \text{pickup}^*[an == \text{my.anum}](an).Busy +$
 $\text{treat}^*[an == \text{my.anum}](an)\{idle \leftarrow \top\}.AskBase$
 $AskBase \stackrel{\text{def}}{=} \text{tell_base}[an == \text{my.anum}](an, ab)\{abase \leftarrow ab\}.GoToBase$
 $GoToBase \stackrel{\text{def}}{=} \text{end_move}[an == \text{my.anum} \wedge \text{now} \geq t + d](an, al, t, d)\{aloc \leftarrow al\}.GoToBase +$
 $\text{atbase}^*[an == \text{my.anum}](an).Idle +$
 $\text{request}[an == \text{my.anum}](an).CleanUp1$
 $CleanUp1 \stackrel{\text{def}}{=} \text{kill_handler}^*[\top](\langle anum \rangle).CleanUp2$
 $CleanUp2 \stackrel{\text{def}}{=} \text{kill_route}^*[\top](\langle anum \rangle).Respond$
Initial state of Ambulance component: *Idle*

Store of Route component:

anum number of ambulance
dest current destination type and incident type
start start of route
end end of route
nexts start of next hop
nexte end of next hop
h number of hops in route
i hop counter
t timer variable for deterministic movement

Behaviour of Route component:

$R \stackrel{\text{def}}{=} [i < h]\text{start_move}^*[\perp](\langle \rangle)\{i \leftarrow i + 1, t \leftarrow \text{now}\}.RC +$
 $[i = h \wedge (\text{my.dest} == \text{ToSevere} \vee \text{my.dest} == \text{ToMinor})]$
 $\text{arrive}^*[\top](\langle anum \rangle).RS +$
 $[i = h \wedge \text{my.dest} == \text{ToHosp}]\text{dropoff}^*[\top](\langle anum \rangle).\text{kill} +$
 $[i = h \wedge \text{my.dest} == \text{ToBase}]\text{atbase}^*[\top](\langle anum \rangle).\text{kill} +$
 $RS \stackrel{\text{def}}{=} [i = h \wedge \text{my.dest} == \text{ToSevere}]\text{pickup}^*[\top](\langle anum \rangle).\text{kill} +$
 $[i = h \wedge \text{my.dest} == \text{ToMinor}]\text{treat}^*[\top](\langle anum \rangle).\text{kill}$
 $RC \stackrel{\text{def}}{=} \text{end_move}[\top](\langle anum, nexte, t, \text{MoveTime}(nexts, nexte, dest) \rangle)$
 $\{nexts \leftarrow \text{my.nexte}, nexte \leftarrow \text{NextHop}(i, \text{my.start}, \text{my.end})\}.R$
 $KR = \text{kill_route}[\text{my.anum} == an](an).\text{kill}$

Fig. 3. Ambulance and route components

Constants:

T limit for response time
 $timeout$ time to wait for a response for a request

Measures:

N set containing the number of idle ambulances at each possible base

Global store:

$ontime$ number of ontime ambulances
 $late$ number of late ambulances

Evolution rule functions:

$$\begin{aligned} \mu_p(\gamma_s, \gamma_r, \alpha) &= 1 \\ \mu_r(\gamma_s, \alpha) &= \begin{cases} 1/r & \alpha = \text{incident}^* \quad (\text{where } r \text{ is the mean time between incidents}) \\ \lambda_p & \alpha = \text{pickup}^* \\ \lambda_t & \alpha = \text{treat}^* \\ \lambda_d & \alpha = \text{dropoff}^* \\ 0 & \alpha = \text{pause}^* \wedge \text{now} < \text{sender}.t + \text{timeout} \\ \lambda_{fast} & \text{otherwise} \end{cases} \\ \mu_{it}(\gamma_s, \alpha) &= \begin{cases} \{\text{ontime} \leftarrow \text{ontime} + 1\}, 0 & \alpha = \text{timecheck}^* \wedge \text{now} \leq \text{sender}.itime + T \\ \{\text{late} \leftarrow \text{late} + 1\}, 0 & \alpha = \text{timecheck}^* \wedge \text{now} > \text{sender}.itime + T \\ \{\}, (\text{Incident_Queue_Item}, \{qnum \leftarrow \text{sender}.inum, itime \leftarrow \text{now}\}) & \\ & \alpha = \text{incident}^* \\ \{\}, (\text{Incident_Handler}, \{loc \leftarrow \mathbf{IncidentLocation}(), dest \leftarrow \mathbf{IncidentType}(), \\ & \quad itime \leftarrow \text{sender}.itime\}) & \\ & \alpha = \text{new_handler}^* \\ \{\}, (\text{Return_Handler}, \{anum \leftarrow \text{sender}.anum, aloc \leftarrow \text{sender}.aloc, \\ & \quad dest \leftarrow \text{sender}.dest, \\ & \quad loc \leftarrow \mathbf{GetBase}(\text{sender}.anum, \text{sender}.aloc, N)\}) & \\ & \alpha = \text{tobase}^* \\ \{\}, (\text{Route}, \{anum \leftarrow \text{sender}.anum, dest \leftarrow \text{sender}.dest, \\ & \quad start \leftarrow \text{sender}.aloc, end \leftarrow \text{sender}.loc \\ & \quad nexts \leftarrow \text{sender}.aloc, nexte \leftarrow \mathbf{NextHop}(1, \text{sender}.aloc, \text{sender}.loc), \\ & \quad h \leftarrow \mathbf{RouteLength}(\text{sender}.aloc, \text{sender}.loc), \}) & \\ & \alpha = \text{makeroute}^* \\ \{\}, 0 & \text{otherwise} \end{cases} \end{aligned}$$

Collective:

$$\begin{aligned} EMS &\stackrel{\text{def}}{=} (\text{Incident_Queue}, \{inum \mapsto 0, rnum \mapsto 1\}) \parallel \\ &(\text{ClosestIdleAmbulance}, \{dloc \mapsto \text{nullLoc}, iloc \mapsto \text{nullLoc}, t \mapsto 0\}) \parallel \\ &(\text{Ambulance}, \{anum \mapsto 1, aloc \mapsto l_1, abase \mapsto l_1, idle \mapsto \top\}) \parallel \dots \parallel \\ &(\text{Ambulance}, \{anum \mapsto n, aloc \mapsto l_n, abase \mapsto l_n, idle \mapsto \top\}) \end{aligned}$$

Fig. 4. Constants, environment and collective

Incident_Queue_Item to the collective (as specified by the function μ_u appearing in the environment in Figure 4). Each item has a unique number, and in turn generates an *Incident_Handler* using the action `new_handler*`. The first action of the *Incident_Handler* is to request an ambulance from another component *Closest_Idle_Ambulance*. This is a separate component for clarity of structure. On receiving a request for a specific location, it calls the function to find the location with idle ambulances that is closest to that location, and take note of the current time. It then tries to communicate (via unicast) with any ambulance in that location. If that succeeds, then it, the *Incident_Handler* and the *Incident_Queue* receive a `confirm*` message from the ambulance that has responded. If there is no response, which is possible because there may be no idle ambulances, a timeout occurs. The timeout is defined by the rate for `pause*` in the environment in Figure 4 where if insufficient time has passed the rate is zero (otherwise, it is λ_{fast}). If the timeout happens, it calls the function again and send out another request to the location that the function returns.

The *Incident_Queue* interacts with each *Incident_Queue_Item* and its associated *Incident_Handler* to ensure that at most one *Incident_Handler* at a time is contacting the closest idle ambulance (because two incidents may have the same closest idle ambulance). The use of queue numbers ensures fairness in the sense that an *Incident_Handler* cannot be starved of access to an *Ambulance* by later incidents and their associated *Incident_Handlers*. A queue item cannot be released and hence cannot execute `new_handler*`, until the *Incident_Handler* associated with the previous *Incident_Queue_Item* and its interaction with *Closest_Idle_Ambulance* has successfully concluded negotiations with the closest idle ambulance via communication on the `confirm*` action.

Each *Incident_Handler* has an assigned ambulance and interacts with the ambulance and generates *Routes* to move the ambulance to the location of the incident and then to the nearest hospital, if required. After this a *Return_Handler* is created using `tobase*`. The two handlers are separate components because an ambulance cannot be diverted once it has been assigned to an incident but once it is returning to a base position, it can be called to a new incident, so *Return_Handler* has behaviour to allow it to remove itself from the collective if this happens, and it is no longer needed in the collective.

Once an *Incident_Handler* and an *Ambulance* have been matched, the ambulance number is used in all communication between the *Incident_Handler*, *Ambulance* and *Route* components to limit broadcast communication to these three components. The queue number could also have been used for these purposes but ambulance number is sufficient. Since broadcast is not blocking, the model must be constructed so that *Incident_Handler* and *Ambulance* are in a state to receive the message from *Route*.

The *Route* component works through the route hops. It is initialised with the start and end of the route, and after obtaining the length of route from **RouteLength**, it works through each hop of the route³ using the function **NextHop** until the end of the route when the appropriate action occurs depending on the destination type. For pickup and treatment at the scene, an `arrive*` action is required followed by a `timecheck*` action for the global count of late and on-time ambulances to be updated, based on the time the `timecheck*` action happens and the time the incident was generated.

³ For the calculation of the performance measure, this detailed level of movement is not necessary but if one wanted to create an animation from a simulation then this detail is required.

λ_p	1/12	rate of pickup* action	q	0.45	busy fraction
λ_t	1/12	rate of treat* action	m	1	proportion of serious incidents
λ_d	1/15	rate of dropoff* action	r	25	mean time between incidents
λ_{fast}	100	rate of fast actions	$timeout$	1	timeout period in minutes

Table 1. Parameters for model

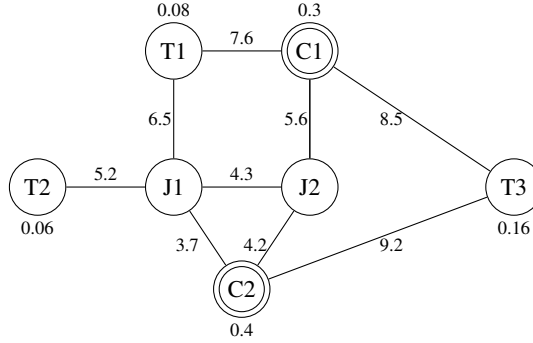


Fig. 5. Network configuration for experiments

The *Ambulance* and *Route* components assume that travel times are deterministic rather than stochastic (although it is straightforward to modify the model to use stochastic durations). The *Route* component performs a local `start_move*` action and notes the time of the action. The ambulance component responds to an `end_move` action once sufficient time has elapsed. Since *Route* components can also be generated by *Return_Handlers*, they can remove themselves from the collective on receipt of a kill command from the associated *Ambulance* when it receives an incident request during going to a base location.

After a simulation completes, the performance measure of the proportion of late ambulances can be calculated from the two global variables *ontime* and *late*. The model describes a system where ambulances behave in the manner described by the seven points in Section 2. The next section considers results of experiments with this model.

5 Results

To explore the behaviour of the model and the heuristic, the network shown in Figure 5 is used (which by contrast with [14] is a compact network rather than long and narrow), and is somewhat simpler than a real scenario. The number annotating the edges of the network gives the time in minutes that it takes to traverse the edge at the faster speed (with sirens and lights on). Locations with hospitals are indicated by double circles at vertices. The number annotating a vertex is the proportion of incidents at that location.

The parameters chosen for the model are given in Table 1. The parameter T which is used as the limit in the calculation of the late rate varies across experiments. The busy fraction q which is used in the calculation of the heuristic function π is estimated by simulation, for the given network with three ambulances to be 0.65.

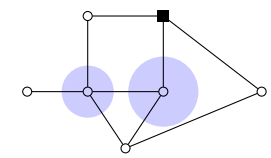
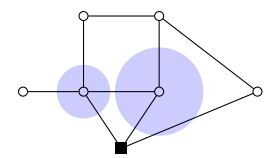
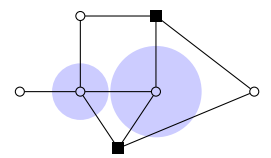
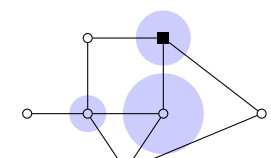
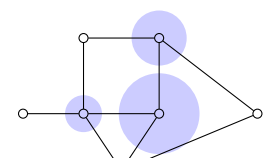
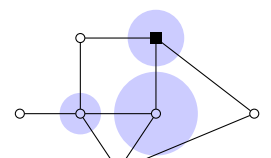
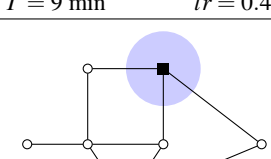
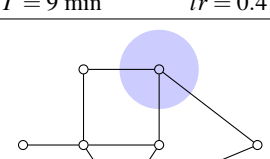
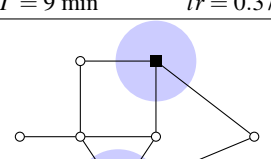
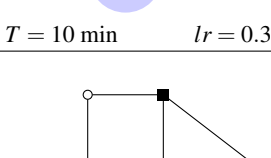
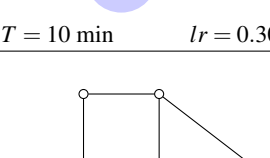
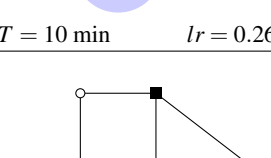
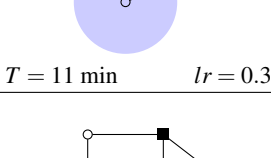
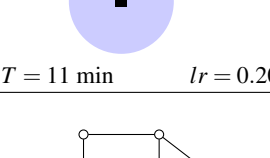
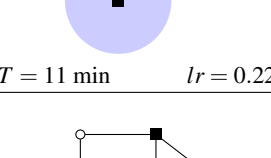
Hospital at C1	Hospital at C2	Hospital at C1 and C2
 <p>$T = 7 \text{ min}$ $lr = 0.53$</p>	 <p>$T = 7 \text{ min}$ $lr = 0.46$</p>	 <p>$T = 7 \text{ min}$ $lr = 0.46$</p>
 <p>$T = 9 \text{ min}$ $lr = 0.42$</p>	 <p>$T = 9 \text{ min}$ $lr = 0.41$</p>	 <p>$T = 9 \text{ min}$ $lr = 0.37$</p>
 <p>$T = 10 \text{ min}$ $lr = 0.30$</p>	 <p>$T = 10 \text{ min}$ $lr = 0.30$</p>	 <p>$T = 10 \text{ min}$ $lr = 0.26$</p>
 <p>$T = 11 \text{ min}$ $lr = 0.35$</p>	 <p>$T = 11 \text{ min}$ $lr = 0.20$</p>	 <p>$T = 11 \text{ min}$ $lr = 0.22$</p>
 <p>$T = 13 \text{ min}$ $lr = 0.44$</p>	 <p>$T = 13 \text{ min}$ $lr = 0.35$</p>	 <p>$T = 13 \text{ min}$ $lr = 0.35$</p>

Fig. 6. Idle ambulances and proportions of late arrivals for time limits and hospital locations

The experiments explore how the late rate varies for different values of the time limit, and furthermore, they consider how the hospital location can affect the late rate, and are illustrated in Figure 6. The square nodes indicate the hospital locations in each case. Each combination of time limit and hospital location was simulated for 500 runs over 20 hours of simulated time. The shaded circles indicate at which locations the idle ambulances were based, and all locations were considered as possible base locations. The area of the circle represents the proportion of simulations that idle ambulances are at a location (or on their way to that location as a base) at the time point of 1200 minutes (20 hours). Since there are three ambulances, fewer than three circles indicate that multiple ambulances are idle at a location.

The results show that the heuristic does not appear to have monotonic behaviour since an increased time limit can lead to a different location with a worse late rate, and this requires further investigation. The heuristic does not use the hospital location but obviously distance from hospital back to base will impact availability, and hence late rate. The lowest late rates occur when there are two hospitals at the two cities, and an ambulance goes to the closest hospital. This experiment shows how the late rates would be affected if it was necessary to close one of the hospitals. However, for some time limits, the presence of two hospitals has little effect on the late rate when compared with just one hospital at the second city, where for others it makes a significant difference. The fact that hospital location can affect the proportion of late arrivals suggests a role for the hospital location in the heuristic function.

6 Conclusions and further work

This paper has demonstrated how the language CARMA can be successfully used to model a system involving coordination where communication is often complex and multiway, and various components interaction to achieve goals. In this case, this is identification of an ambulance to go to an incident, movement of the ambulance to the incident using a generic component that draws on functions to give the specifics of a route, movement to hospital where required, then choice of a base to return to on completion and movement to that base. Because of the use of generic functions together with generic components, the model can be made applicable to any road network simply by substituting the appropriate functions.

Further research relating to this case study include further exploration of the parameter state space, in particular to understand the nature of the heuristic function as mean time between incidents change, as well as modifying the heuristic to take into account the current location of idle ambulances as well as their bases. Different performance measures could also be investigated that consider not just a single deadline but how late the ambulance is, in the case of late arrival at the scene. Clearly, many variations can be made to the ambulance model, including for example, switching to an ambulance closer to the incident if one becomes available when another is already on the way; investigating movement between bases as an incident occurs; and the use of different time limits depending on the severity of the incident. At the modelling level, comparison of the use of CARMA with other formalisms and an assessment of its strengths and weaknesses is important.

As often is the case in modelling for performance assessment, the users who are interested in the measure often do not have the skills to work with the modelling language directly. A ongoing project is to develop a graphical front-end for general modelling of this ambulance scenario. The final goal is software which allows a user to graphically create a road network as shown in Figure 5 with appropriate annotations, after which it will automatically generate a CARMA model consisting of the seven generic components, the model parameters and the functions to implement the network. The model can then be simulated in the CARMA Eclipse Plug-in. An additional step would be to take the output of a single simulation and use it to create an animation over the network, to give users an insight to what is happening during the simulation. This gives more information about the model over and above the performance measure.

Acknowledgements This work is supported by the EU project QUANTICOL, 600708. The author thanks Jane Hillston and Yehia Abd Alrahman for their useful comments.

References

1. Alanis, R., Ingolfsson, A., Kolfal, B.: A Markov chain model for an EMS system with repositioning. *Production and Operations Management* 22, 216–231 (2013)
2. Alrahman, Y.A., De Nicola, R., Loreti, M., Tiezzi, F., Vigo, R.: A calculus for attribute-based communication. In: *Proceedings of SAC 2015*. pp. 1840–1845 (2015)
3. Bortolussi, L., De Nicola, R., Galpin, V., Gilmore, S., Hillston, J., Latella, D., Loreti, M., Massink, M.: CARMA: collective adaptive resource-sharing markovian agents. In: *Proceedings of QAPL 2015. EPTCS*, vol. 194, pp. 16–31 (2015)
4. Church, R., Velle, C.: The maximal covering location problem. *Papers of the Regional Science Association* 32, 101–118 (1974)
5. Ciancia, V., De Nicola, R., Hillston, J., Latella, D., Loreti, M., Massink, M.: CAS-SCEL semantics and implementation. QUANTICOL Deliverable D4.2 (2015)
6. Daskin, M.: A maximum expected covering location model: Formulation, properties and heuristic solution. *Transportation Science* 17, 48–70 (1983)
7. De Nicola, R., Latella, D., Loreti, M., Massink, M.: A uniform definition of stochastic process calculi. *ACM Computing Surveys* 46, 5 (2013)
8. De Nicola, R., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: The SCEL language. *ACM TAAS* 9, 7:1–7:29 (2014)
9. Feng, C., Hillston, J.: PALOMA: A process algebra for located Markovian agents. In: *Proceedings of QEST 2014*. pp. 265–280. LNCS 8657, Springer (2014)
10. Gendreau, M., Laporte, G., Semet, F.: A dynamic model and parallel tabu search heuristic for real-time ambulance relocation. *Parallel Computing* 27, 1641–1653 (2001)
11. Gillespie, D.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics* 22, 403–434 (1976)
12. Hillston, J.: A compositional approach to performance modelling. CUP (1996)
13. Jagtenberg, C.: Personal communication (2016)
14. Jagtenberg, C., Bhulai, S., van der Mei, R.: An efficient heuristic for real-time ambulance redeployment. *Operations Research for Health Care* 4, 27–35 (2015)
15. Maxwell, M., Henderson, S., Topaloglu, H.: Tuning approximate dynamic programming policies for ambulance redeployment via direct search. *Stochastic Systems* 3, 322–361 (2013)
16. Toregas, C., Swain, R., ReVelle, C., Bergman, L.: The location of emergency service facilities. *Operations Research* 19, 1363–1373 (1971)