On Model-based Performance Analysis of Collective Adaptive Systems *

Maurizio Murgia, Riccardo Pinciroli, Catia Trubiani, and Emilio Tuosto

Gran Sasso Science Institute, Italy
{maurizio.murgia, riccardo.pinciroli, catia.trubiani, emilio.tuosto}@gssi.it

Abstract This paper fosters the analysis of performance properties of collective adaptive systems (CAS) since such properties are of paramount relevance practically in any application. We compare two recently proposed approaches: the first is based on generalised stochastic petri nets derived from the system specification; the second is based on queueing networks derived from suitable behavioural abstractions. We use a case study based on a scenario involving autonomous robots to discuss the relative merit of the approaches. Our experimental results assess a mean absolute percentage error lower than 4% when comparing model-based performance analysis results derived from two different quantitative abstractions for CAS.

1 Introduction

Increasingly *collective adaptive systems* (CAS) crop up in many application domains, spanning critical systems, smart cities, systems assisting humans during their working or daily live activities, etc. A paradigmatic example is the use of artificial autonomous agents in rescue contexts that may put operators lives at stake [3]. The components of these systems execute in a cyber-physical context and are supposed to exhibit an *adaptive* behaviour. This adaptation should be driven by the changes occurring in the components' operational environments as well as the changes in the local computational state of each component, "collectively taken". Also, the *global* behaviour of CAS should *emerge* from the *local* behaviour of its components. Let us explain this considering the coordination of a number of robots patrolling some premises to make sure that aid is promptly given to human operators in case of accidents.

A plausible local behaviour of each robot can be:

(1) to identify accidents,
(2) to assess the level of gravity of the situation (so to choose an appropriate course of action),
(3) to alert the rescue centre and nearby robots (so to e.g., divert traffic to let rescue vehicles reach the location of the accident more quickly), and

* Work partly funded by MIUR PRIN projects 2017FTXR7S *IT MATTERS* (Methods and Tools for Trustworthy Smart Systems) and 2017TWRCNB *SEDUCE* (Designing Spatially Distributed Cyber-Physical Systems under Uncertainty).

(4) to ascertain how to respond to alerts from other robots (e.g., if already involved in one accident or on a low battery, a robot may simply forward the alert to other nearby robots).

Note that robots' behaviour depends on the physical environment (tasks (1) to (3)) as well as their local computational state (task (4)).

A possible expected global behaviour is that robots try to maximise the patrolled area while trying to avoid remaining isolated and to minimise the battery consumption. It is worth remarking that the global behaviour is not typically formalised *explicitly*; it should rather emerge from combining the behaviour of the single components. For instance, when designing the algorithm for the roaming of robots one could assume that a robot does not move towards an area where there are already a certain number of robots.

This paper applies behavioural specifications to the quantitative analysis of CAS. Using a simple, yet representative, robots scenario inspired by the example above, we show how to use behavioural specifications to study non-functional properties of CAS (emergent) behaviour. This exercise is instrumental for our contribution, which is a study of the relation between two complementary approaches to the performance analysis of CAS recently proposed. More precisely, we compare the approach based on generalised stochastic petri nets proposed in [28] with the one based on behavioural specifications proposed in [16]. These approaches support two rather different methodologies for the quantitative modelling and analysis of CAS.

The main difference between these two approaches is the following. The former is based on the analysis proposed in [28] where the designer must directly come up with a performance model using generalised stochastic petri nets. In this sense this is a *model-based* methodology. Instead, for the latter approach [16], the designer does not have to directly develop a model for the quantitative analysis; such model —a queueing network— is indeed "compiled" from the behavioural specification of the CAS. Hence, this is a *language-based* methodology.

This paper aims to compare such methodologies and to study their relative merits. More precisely we address the following two research questions:

RQ1 To what extent the approaches in [16] and in [28] support performance-aware design of CAS?

RQ2 How do the features of the approaches in [16] and in [28] compare?

For this comparison we will use a robot scenario that will allow us to highlight the respective strengths and weaknesses of the methodologies. As we will see, our analysis suggests an hybrid combination hinging on both approaches.

Outline. Section 2 describes the scenario used in the rest of the paper. We will consider two different architectures (i.e., independent and collaborative) for this scenario. Section 3 provides the models based on the specification language in [16] for both the architectures. Section 4 shows the performance analysis based on the proposed models of Section 3. The comparison between the approach in [28] and the one illustrated in Section 3 is discussed in Section 5. Final comments, related, and future work are in Section 6.

2 A Robot Scenario

Our analysis is conducted on a scenario where robots have the task to transport some equipment necessary in an emergency from an initial zone to a target location. In order to reach the target location, robots need to pass through two doors, or take an alternative longer route. Robots take the alternative route only if they find a door closed. When this happens on the second door, it will take more time for robots to reach the destination than if the alternative route had been taken at the start of the journey. After the delivery, robots return to the initial zone trying to follow the reverse path and the same constraints apply.

It is commonly accepted that the performance of a cyber-physical system varies with changes in the physical environment. Moreover, as experimentally confirmed in [28], it is possible to measure the impact of architectural patterns and dynamic space changes on the performance of cyber-physical systems. This type of analysis suggests that in this domain it is useful to factor performance at design time. Following [28], we will consider two architectural scenarios:

Independent Robots do not cooperate with each other. In this architecture, robots simply detect the state of doors and behave as described above.

Collaborative Robots behave exactly as above on open doors; instead, on closed doors, they send a message to nearby robots before taking the alternative route. In this way, every robot that receives such message can directly follow the alternative route.

The approach proposed in [28] is new and it hinges on Generalised Stochastic Petri Nets (GSPN) [4] as suitable models of cyber-physical systems. In this paper we apply such approach by adopting (i) a different modelling language, hinging on behavioural specifications and (ii) relying on queueing networks [18] for performance analysis. The modelling language used here has been advocated in [16] for specifying global behaviour of CAS. As shown in [16], this modelling language has a natural connection with queueing networks, therefore enabling performance analysis of CAS.

3 A Behavioural Specification Model

The behavioural specifications in [16] are inspired by AbC, a calculus of attribute-based communication [1]. The key feature of AbC is an abstract mechanism of addressing partners of communications by letting the specification of many-to-many communication between dynamically formed groups of senders and receivers. Informally, components expose domain-specific *attributes* used to address senders and receivers of communications according to *predicate* on such attributes. For instance, the robots in the scenarios in Sections 1 and 2, may expose an attribute recording their physical position. This attribute can be used to specify communications among "nearby" robots through a suitable predicate so to determine the communication group as the set of robots satisfying such predicate. This mechanism is abstracted in [16] by *interactions* defined, in their most

general form, as

$$\mathsf{A}|\rho \xrightarrow{e \quad e'} \mathsf{B}|\rho' \tag{1}$$

where A and B are *role names*, ρ and ρ' are logical formulae, e is a tuple of expressions, and e' is a tuple of *patterns*, that is expressions possibly including variables. The intuitive meaning of the interaction in (1) is

> "*any* agent, say A, *satisfying* ρ generates an expression e for *any* agents satisfying ρ', dubbed B, provided that expression e' *matches* e." [16]

The conditions ρ and ρ' predicate over components' attributes. The payload of an output is a tuple of values e to be matched by receivers with the (tuple of) patterns e'; when e and e' match, the effect of the communication is that the variables in e' are instantiated with the corresponding values in e.

As said, a send operation targets components satisfying a given *predicate* on such attributes. For instance, if pos is the position attribute exposed by robots, the predicate

$$\rho \equiv \mathrm{abs}(\mathsf{self.pos} - \mathsf{pos}) < 5\mathrm{mt}$$

is satisfied by a receiving robot which is less than five meters away from the sending robot (i.e., the difference between the position self.pos of the receiver and the one pos of the sender is below five meters). Messages are disregarded if they do not satisfy ρ.

Role names A and B in (1) are pleonastic: they are used just for succinctness and may be omitted for instance writing $\rho \xrightarrow{e \quad e'} \mathsf{B}|\rho'$ or $\rho \xrightarrow{e \quad e'} \rho'$. Also, we abbreviate $\mathsf{A}|\rho$ with A when ρ is a tautology.

Interactions are the basic elements of an algebra of protocols [29] featuring iteration as well as non-deterministic and parallel composition. This algebra has an intuitive graphical presentation which we use here to avoid technicalities. In fact, we use *gates* to identify control points[1] of protocols:

- entry and exit points of loops are represented by ◇-gates,
- branching and merging points of a non-deterministic choice are represented by ◈-gates.

We remark that our behavioural model does not require fix in advance to the number of instances of agents. In fact, in our model:

- several agents can embody the same role at the same time; for instance, in our case study, an unspecified number of devices impersonate the robot role;
- instead of addressing senders and receivers by name, attribute-based communication by definition uses constraints to identify communication partners.

Therefore our model allows us to specify complex multiparty scenarios regardless the number of agents' instances.

The next sections give the architectures of our scenario in terms of the graphical notation sketched above.

[1] We do not consider forking and joining points of parallel composition (represented by ▯-gates) since this feature is not used in our case study.

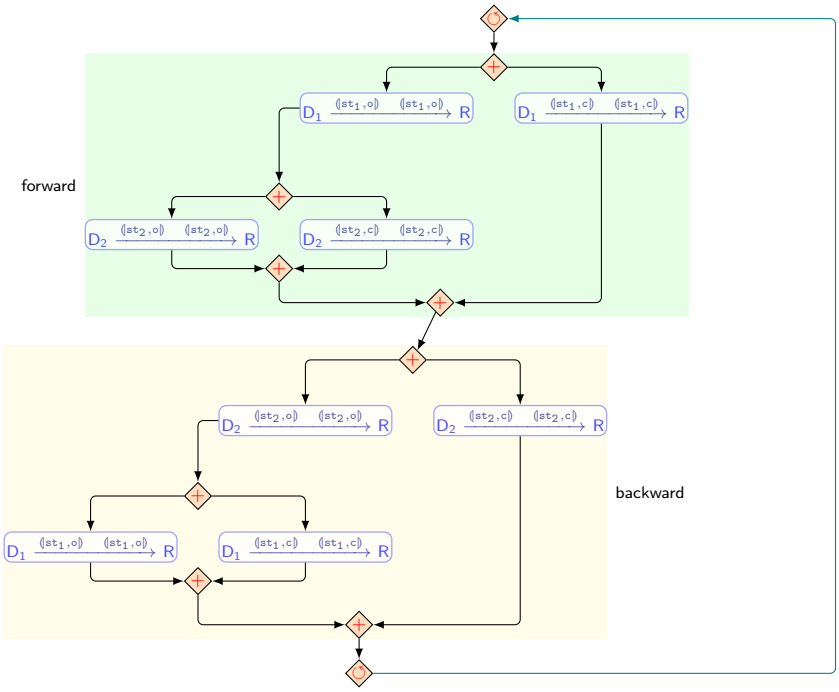

Figure 1. A model for the independent architecture

3.1 Independent architecture

Fig. 1 gives a possible model capturing the independent architecture described in Section 2 in the graphical notation of our specification language. The model consists of a loop whose body is made by the sequential composition of the behaviour for the forward and the backward journey of robots. Robots try to go through the first door and then through the second one on their forward journey and try the opposite on their backward journey.

The model in Fig. 1 is rather simplistic and we will refine it soon; we use it to introduce our graphical notation. Interactions among doors and robots do not involve value passing; for instance, robots detect the `st`atus of the first door when they pattern match on the tuples $(\!|\mathtt{st}_1, \mathtt{o}|\!)$ and $(\!|\mathtt{st}_1, \mathtt{c}|\!)$ for `o`pen and `c`losed doors respectively (an likewise for the second door). Robots detect the status of a door according to the format of the messages they intercept. For instance, on its forward journey a robot either pattern matches the tuple $(\!|\mathtt{st}_1, \mathtt{o}|\!)$ or the tuple $(\!|\mathtt{st}_1, \mathtt{c}|\!)$ from the first door. This choice is represented in Fig. 1 by ◇-gate immediately below the topmost ◇-gate. If the robot receives a $(\!|\mathtt{st}_1, \mathtt{c}|\!)$ tuple from the first door, it continues its journey on the alternative route after which it starts the backward journey. Otherwise, the robot approaches the second door and again goes through if $(\!|\mathtt{st}_2, \mathtt{o}|\!)$ is received otherwise takes the alternative

route. The behaviour on the return journey is similar modulo the order in which robots approach doors.

As said, this model is simplistic. Let us refine it. In Fig. 1, we used role names D_1, D_2, and R for simplicity. However, this is not very precise. In fact, we would like to express that robots detect the status of a door only when they are "close enough" to it. To capture this behaviour let us assume that robots and doors expose the attribute ID yielding their identity. Then, we can define the conditions

$$\rho_d(x) \equiv abs(\mathsf{self.pos} - \mathrm{pos}(x)) < d$$

where $\mathrm{pos}(x)$ is the position of the component with identifier x. Then we can replace in Fig. 1 the interactions $\mathsf{D}_1 \xrightarrow{\langle\!|\mathtt{st_1,o}|\!\rangle \quad \langle\!|\mathtt{st_1,o}|\!\rangle} \mathsf{R}$ with

$$\mathsf{ID} = \mathtt{d1} \xrightarrow{\langle\!|\mathtt{self.ID,o}|\!\rangle \quad \langle\!|x,\mathtt{o}|\!\rangle} \rho_{\mathtt{d}}(\mathsf{x}) \tag{2}$$

and similarly for the interactions $\mathsf{D}_1 \xrightarrow{\langle\!|\mathtt{st_1,c}|\!\rangle \quad \langle\!|\mathtt{st_1,c}|\!\rangle} \mathsf{R}$ and those involving D_2. Interaction (2) and the one for the closed status state that the door[2] with ID set to $\mathtt{d_1}$ emits a tuple with their identity and the status. These tuples are intercepted by components whose state satisfy $\rho_d(x)$ where x is the variable instantiated with the identity of the sender. Other components would simply disregard those messages.

3.2 Collaborative architecture

The collaborative architecture can be obtained by simply extending the independent one with the interactions among robots. A possible solution is given in Fig. 2 where for readability we only show the body of the loop and shorten $\langle\!|\mathtt{self.ID,o}|\!\rangle$ and $\langle\!|\mathtt{self.ID,c}|\!\rangle$ with o_{ID} and c_{ID} respectively, and $\langle\!|x,\mathtt{o}|\!\rangle$ and $\langle\!|x,\mathtt{c}|\!\rangle$ with $x_{\mathtt{o}}$ and $c_{\mathtt{o}}$ respectively.

As in the independent architecture, there are a forward and a backward phase. The only difference is that each time a robot detects a closed door, it will inform nearby robots that the door is closed. Once this communication is performed, the robot continues its journey on the alternative root. The fact that the adaptation is quite straightforward is due to the features offered by our modelling language. The attributes of components are indeed allowing us to just reuse the condition ρ_d also for coordinating inter-robots interactions.

There is however a crucial remark to be made. The behaviour of robots is to wait for three possible messages: the two sent by the door and one possibly coming from a robot which detected that the door was closed. In fact, there might be robots satisfying condition $\rho'(y,x)$ in Fig. 2, that is they are not close enough to the door but have a nearby robot, say r, aware that the door is closed. These robots should therefore be ready to receive the communication from robot r. Our model accounts for this type of robots but the graphical notion "hides"

[2] The fact that identifiers are unique is not built-in in our model; in principle there could be more doors with the same identifier.

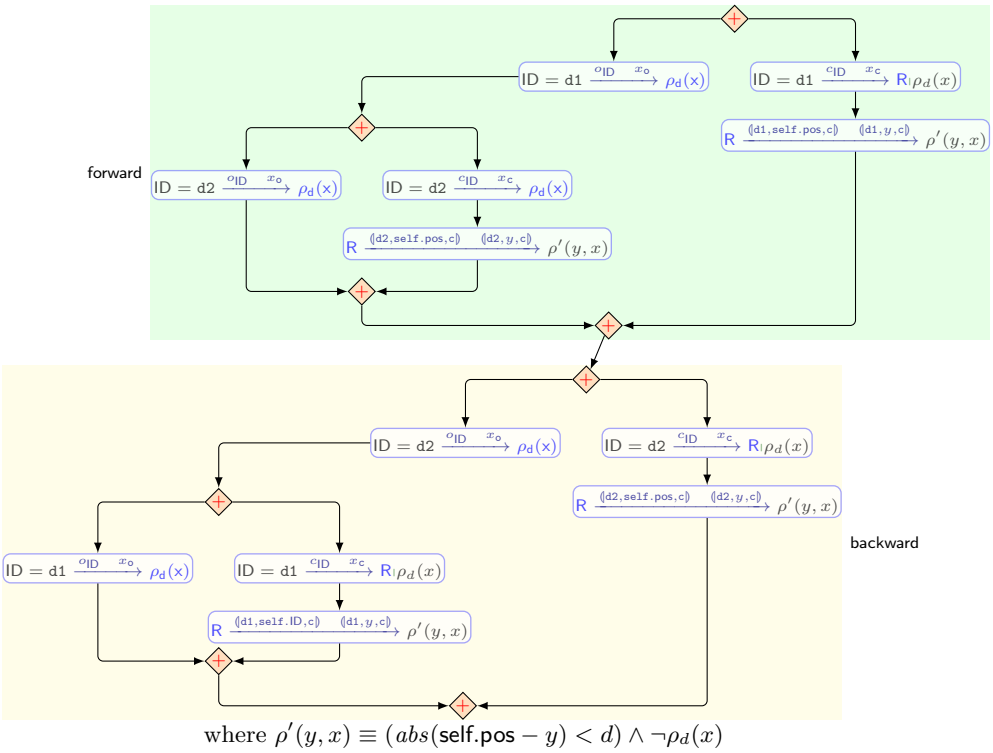

Figure 2. A model for the collaborative architecture

this since there are only two possibilities on branching ◈-gates. As we will see in Section 4, this is a key observation for our performance analysis.

4 Quantitative Analysis

In [16] we relate our modelling language to Queueing Networks (QNs) [19], a widely used mathematical model to study waiting lines of systems represented as a network of queues [11,14]. Two main elements of QNs are *customers* (i.e., jobs, requests, or transactions) that need to be served by *service centres* (i.e., system resources). When a service centre is busy serving a customer, other jobs to be processed by the same resource wait in a queue for their turn. Also, QNs feature *routers* to dispatch customers to different centres and *delay stations* in order to model e.g., lags in the processing or "internal" computations not requiring further system resources. In [16], we provided some rules to automatically get QN performance models from a behavioural specification. The basic idea is to transform (i) an interaction into a service centre and (ii) a non-deterministic choice into a router. We will apply this construction to our robot scenario and its architectures.

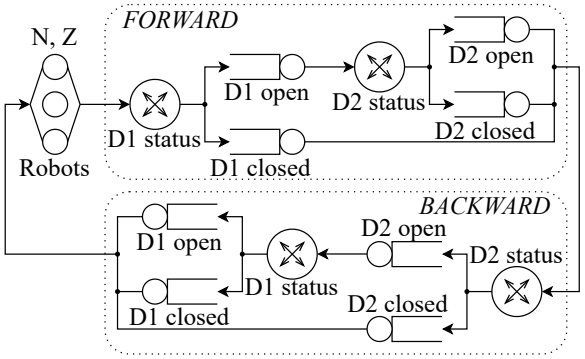

Figure 3. QN of the independent architecture

We build on our recent experience [28] on using GSPN [4] performance models. A GSPN consists of places (represented as circles), tokens (represented as dots), transitions (represented as rectangles), and arcs that connect places to transitions and vice-versa. A token is removed from a place (and possibly added to another one) every time a transition fires. A transition is *enabled* when all the input places contain a number of tokens larger or equal to a pre-defined multiplicity (if not expressed it is equal to 1). There are two types of transitions in GSPN: *immediate transitions* and *timed transitions*. The former are graphically represented as thin black rectangles and fire when enabled, no timing is associated to the transition. The latter are graphically represented as thick white rectangles and fire following a randomly distributed time (in this paper we use exponential distributions), meaning that the transition implies some timing.

The analysis conducted in [28] shows that it is possible to measure the impact of architectural patterns and dynamic space changes on the performance of cyber-physical systems.

In the following we are interested in studying the applicability of GSPN in CAS and compare this approach to the one based on QNs. Both GSPN and QNs are analysed using JSIMgraph, i.e., the simulator of Java Modelling Tools (JMT) [7]. JSIMgraph discards the initial transient system behaviour and automatically stops when the desired confidence interval (i.e., the probability that the sample data lie within it, set to 99% for our experiments) is observed for all performance indices under analysis.

4.1 Independent architecture

To address RQ1, we start by describing the approaches defined in [16] and in [28] to support performance-aware design of CAS. Let us focus on the independent architecture first.

The application of the rules introduced in [16] to the behavioural specification of Fig. 1 yields the QN depicted in Fig. 3. Specifically, the first and second non-

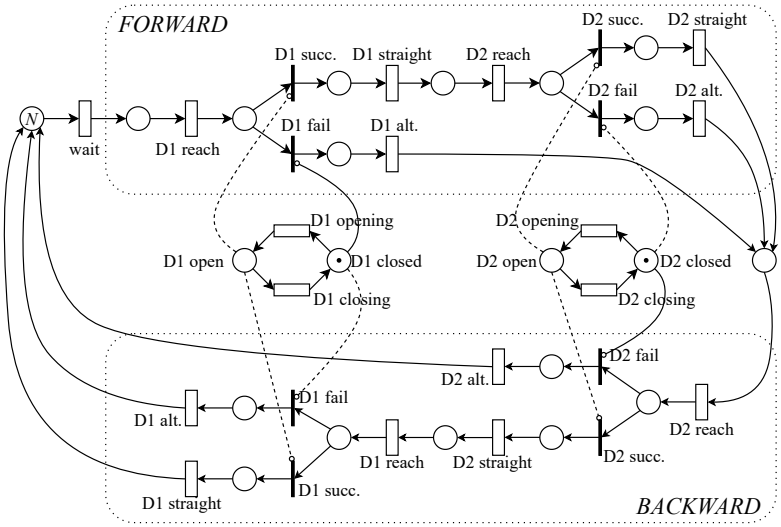

Figure 4. GSPN of the independent architecture

deterministic choices in the forward box of Fig. 1 become respectively the `D1 status` and `D2 status` router in the *FORWARD* box of Fig. 3. Interactions on the left (resp.right) branch of each choice in the forward box of Fig. 1 become the `D1 open` and `D2 open` (resp. `D1 closed` and `D2 closed`) service centres in the *FORWARD* box of Fig. 3. Similarly, the *BACKWARD* box in Fig. 3 is derived from the backward box in Fig. 1. A delay centre (i.e., `Robots` Fig. 3) represents the number of robots in the system and their *think time* is also added to the QN model as suggested in [16].

We assess the usefulness of the QN derived from the behavioural specification and rules defined in [16] by comparing it with the GSPN introduced in [28], and depicted in Fig. 4. Initially, there are N robots waiting for task assignment. When transition `wait` fires after an exponentially distributed time (i.e., when a task is assigned to a robot), a token is moved to the next place. This represents the fact that the robot starts moving towards the first door `D1 reach`. After some time, the robot reaches the door. If the door is closed (`D1 fail`) the robot has to take the alternative (and longer) route (`D1 alt.`); otherwise the robot goes through the first door (`D1 succ.`) and continues its journey towards the second door (`D1 straight`). After some time the robot approaches the second door (`D2 reach`). The status of each door is controlled by two places (e.g., `D1 open` and `D1 closed` for the first door) and two transitions. The door is initially closed (i.e., a token is in the `D1 closed` place). When the enabled transition fires, the token is removed from the `D1 closed` place and added to the `D1 open` place. Hence, the door stays open until the new enabled transition fires and the door status changes again.

Table 1. Numerical values used for GSPN and QN models of independent and collaborative architectures. *Direction* indicates `Forward` (F) or `Backward` (B), the parameter and its value are used. $S_{D* \ open}$ (QN) is obtained by summing $S_{D* \ reach}$ and $S_{D* \ straight}$ (GSPN). All timing parameters are in second.

GSPN			QN		
Parameter	Direction	Value	Parameter	Direction	Value
N		100	N		100
S_{wait}		10	Z		10
$S_{D* \ closing} + S_{D* \ opening}$		60	–	–	–
$S_{D* \ reach}$	F / B	5	$S_{D* \ open}$	F / B	10
$S_{D* \ straight}$	F / B	5			
$S_{D1 \ alt.}$	F	45	$S_{D1 \ closed}$	F	45
$S_{D2 \ alt.}$	F	60	$S_{D2 \ closed}$	F	60
$S_{D1 \ alt.}$	B	60	$S_{D1 \ closed}$	B	60
$S_{D2 \ alt.}$	B	45	$S_{D2 \ closed}$	B	45
$S_{D* \ follow}$	F / B	46	$S_{D* \ msg.}$	F / B	46
$S_{D* \ send}$	F / B	1	$S_{D* \ send}$	F / B	1

The two models are parameterized with values from the literature [32] as shown in Table 1. In our model the system response time is the time spent by each robot to complete a task and go back to the initial room.

To answer RQ2, we estimate the response time using both models against the probability that each door is open. The results of this analysis are given in the left histogram of Fig. 5 together with the confidence interval. Notice that extreme cases of 0.01 and 0.99 probabilities are reported instead of 0 and 1 since the latter ones are not probabilistic by definition, these values would imply doors are always either closed or open. We point out that our experimental results show high agreement in the performance predictions. The QN derived from the behavioural specification predicts the system response time with values similar to those predicted by the GSPN. As expected, the shortest response time of the system is observed when there is a high probability that both doors are open and robots can almost always take the fastest route. The longest system response time is observed for $0.2 \leq$ *Pr(Door is Open)* ≤ 0.3, when the probability of finding the first door open and the second one closed (or the second door open and the first on the backward journey) is higher. In this case, robots spend a longer time taking the alternative route, see Table 1. The QN predictions are further assessed in the right histogram of Fig. 5 via mean absolute percentage error (MAPE) calculated as

$$\text{MAPE}[\%] = \frac{|R_{GSPN} - R_{QN}|}{R_{GSPN}} \cdot 100$$

where R_{GSPN} and R_{QN} are the system response time estimated using GSPN and QN, respectively. The value of MAPE is always smaller than 4%, an excellent result when estimating the system response time [27].

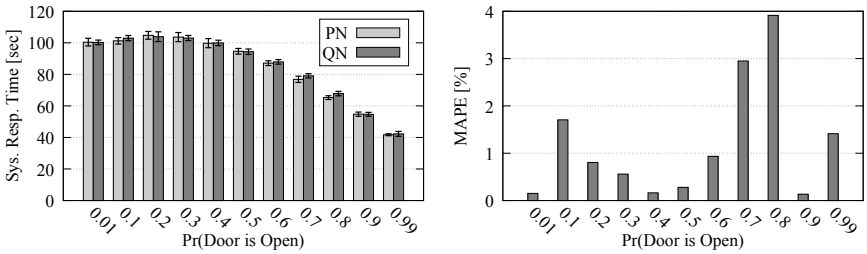

Figure 5. Independent architecture – System response time (left) and MAPE (right) vs. Probability door is open

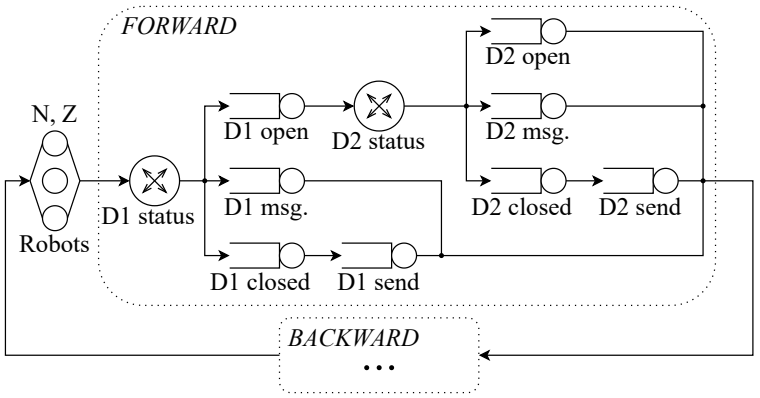

Figure 6. QN of the collaborative architecture

4.2 Collaborative architecture

We now repeat the previous exercise focusing on the collaborative architecture.

The application of the rules defined in [16] yields the QN from the behavioural specification of the collaborative scenario depicted in Fig. 6. In this case, routers have three branches. As discussed in Section 3.2, besides finding an open or a closed door, robots can also receive a message from their peers when a door is closed. This is modelled by `D1 msg.` and `D2 msg.` service centres. Now, robots can take the alternative route without spending extra time checking the door status. Moreover, the robot finding the closed door has to communicate its finding to other robots (`D1 send` and `D2 send`).

We now compare the performance measures on the QN in Fig. 6 with the ones obtained using the GSPN presented in [28] and depicted in Fig. 7. The two models are parameterized with numerical values reported in Table 1 except for probabilities used in the QN routers (i.e., `D1 status` and `D2 status`). Since the probability of receiving a message from a peer depends on other characteristics of the system (e.g., the probability that doors are open, robots' position, and their speed), there is not an easy way to set router probabilities. To fairly compare

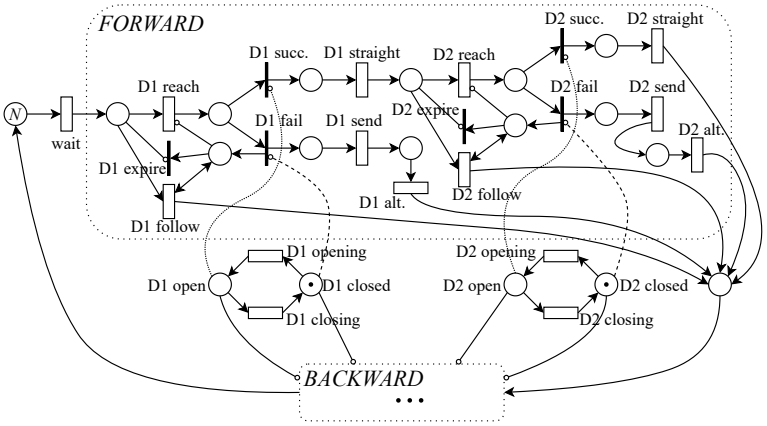

Figure 7. GSPN of the collaborative architecture

QN results to GSPN ones, it is necessary to first analyse the GSPN model of the collaborative architecture given in Fig. 7. From this analysis we infer the probability values (i.e., for receiving a message from a peer under certain system circumstances) that are needed to properly parametrise the QN model.

We now reconsider RQ2 for the collaborative architecture. Similarly to the independent architecture, we estimate the system response time using both models against the probability of doors being open. The response times of the system obtained with GSPN and QN are shown in Fig. 8. Also for the collaborative

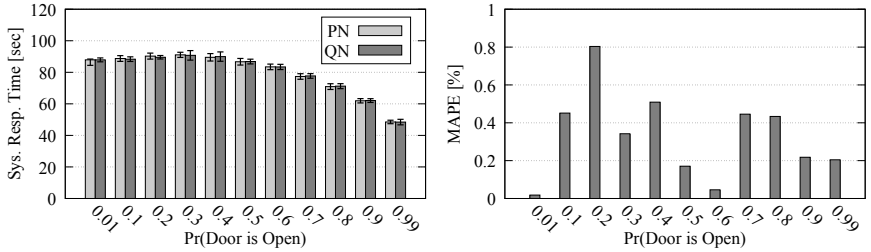

Figure 8. Collaborative architecture – System response time (left) and MAPE (right) vs. Probability door is open

architecture, the QN parameterized as previously described allows us to obtain results that are close to those obtained with the GSPN. The response time of collaborative systems is generally shorter than the one observed for independent architectures since robots share knowledge about the state of the environment. The only exception to this observation is for *Pr(Door is Open)* $\geq$ 0.9; in this

case, a robot finding a closed door affects negatively the performance of its peers. Indeed, other robots take the alternative route after receiving the message. However, there is a high probability that the door will be open again when they will approach it. In this case, the value of MAPE is even smaller (i.e., less than 1%, see the right histogram of Fig. 8) than the one observed for the independent architecture. This is due to the router probabilities being directly derived from the GSPN used for comparison.

5 Discussion

Answering to our research questions, we can state that both QNs and GSPN are suitable for the performance analysis of CAS. Our experience shows that there is a trade-off between simplicity and expressiveness in the use of these models (RQ1). An interesting outcome of our simulations is that the two different model-based performance predictions match, the error is never larger than 4% denoting high agreement between the proposed performance abstractions (RQ2). Our experimental results confirm the expectation on the analysed scenarios. For instance, the system response time is minimised in case of doors open with a high probability. Collaboration among robots pays off, the collaborative architecture shows shorter response times since robots are informed before reaching doors and gain from promptly taking alternative paths.

The two modelling approaches offer different advantages which we discuss herafter. A main advantage of QNs is that they are conceptually simple: performance analysis is based on the probabilities assigned to observable events (e.g., door open). Moreover, QNs can be automatically derived from our behavioural specification of the system. A key observation is that our behavioural specification models introduce a clear separation of concerns: the modelling of the system is orthogonal to its performance analysis that is done by using the derived QNs; hence one just needs to fix the probabilities for the observable events. However, this comes with a cost: it is not usually easy to determine such probabilities.

Instead, the modelling with GSPN does not require to directly specify probabilities, a clear advantage over QNs. Indeed, with GSPN one has to simply select a suitable time distribution: this is therefore a more reliable approach compared to QNs. Besides, GSPN allow for controlling events with a same process; for instance, if a door is open in a direction, it must also be open in the other direction; this cannot be modelled using probabilities only. Overall, GSPN requires more expertise on building the performance model, but its parameterisation includes timing values only, hence they may also be used for monitoring (see e.g., [25]).

However, GSPN are more "rigid" than QNs because certain characteristics of the system are hardwired in the model itself. For instance, changing the number of doors robots have to traverse would require a more complex performance model. Moreover, this kind of generalisation will make the size of the model much larger, which can severely affect the performance of the analysis as the state space grows exponentially [4]. This is not the case for QNs derived from

our behavioural specification because they permit to easily abstract away from the number of system components. On the other hand, GSPN allow to easily model other types of sophisticate coordination policies. For instance, in the collaborative architecture it is easy to let the robot first noticing the closed door wait for all nearby robots to take the alternative route before continuing its journey. This is not simple to model with our behavioural specification language or with QNs.

6 Conclusions, Related & Future Work

We proposed two approaches for the performance analysis of CAS. Our experimental observations are conducted on a simple case study of autonomous robots for which we consider two architectures. Our first approach is based on behavioural abstraction and QNs while the second is based on GSPN. Finally, we compare the two approaches by exploiting the models of the two architectures and observe a high level of agreement on model-based performance predictions.
Behavioural Abstractions. Coreographic models have been applied to Cyber-Phisical Systems [21,22], IoT [20] and robotics [24]. These papers focus on verification of correctness properties (e.g., deadlock freedom and session fidelity) and are not concerned with quantitative aspects or performance analysis. Some works in the literature exploits behavioural abstraction for cost analysis of message passing systems. Cost-aware session types [10] are multiparty session types annotated with size types and local cost, and can estimate upper-bounds of the execution time of participants of a protocol. Temporal session types [12] extend session types with temporal primitives that can be exploited to reason about quantitative properties like the response time. A parallel line of research studies timed session types [5,8,9], that is session types annotated with timed constraint in the style of timed automata [2]. They have been used for verification of timed distributed programs by means of static type checking [8,9] and runtime monitoring [6,25]. Despite the presence of timed constraints, which makes timed session types appealing for performance analysis, they have never been applied in such setting. Session types have been extended with probabilities [15] for verification of concurrent probabilistic programs, which is potentially useful for the CAS analysis. A common limitation of these approaches is that they do not easily permit to define the number of agents' instances embodying a specific role in the system specification. Our behavioural model instead allows it, as explained in the final remark of Section 3, hence it is suitable for performance analysis that indeed requires such a system workload information.
Quantitative Abstractions. Rigorous engineering of collective adaptive systems calls for quantitative approaches that drive the design and management of coordination actions, as recently advocated in [13]. Verification tools for CAS formation are surveyed in [17] and analysis techniques are considered still immature to deal with possible changes in decision-making, hence quantitative approaches that keep track of behavioural alternatives and their impact on system performance are of high relevance for CAS. Ordinary differential equations (ODEs) have

been proposed in [30] as quantitative abstractions for CAS. The use of ODEs allows one to express large-scale systems that are analyzed through reaction network bisimulations. A limitation of the approach is that results are not reliable when the population of agents is small. A quantitative analysis of CAS is also pursued in [23] where the focus is on investigating the probabilistic behaviour of agents, and a specific language (i.e., CARMA) is introduced along with a simulation environment that provides quantitative information on CAS, however the scalability is limited and alternative (quantitative) semantics are claimed to be desirable to speed up the analysis. Performance characteristics of CAS are tackled in [31] where the goal is to select optimal (from a performance perspective) implementations of collective behaviour while preserving system functionalities and resiliency properties, however various implementations are required and the switch of identified alternatives cannot be executed at runtime. More recently, in [26] a design pattern, i.e., self-organising coordination regions, is proposed to partition system devices into regions and enable internal coordination activities. This supports our investigation of independent and collaborative architectures since their optimality relies on the physical space, as emerged by our quantitative analysis on the probability of doors being open/closed. As opposed to these approaches, we aim to automatically derive quantitative abstractions from the behavioural specification of CAS, thus simplifying the process to get performance indicators of interest.

Future work. We plan to consider more complex application scenarios and investigate generalisations of our model-based performance analysis. In particular, we are interested to explore the possibility to automatically derive the structure of GSPN from our behavioural specifications. We conjecture that this could allow us to overcome some of the drawbacks of GSPN while avoiding the need to determine probabilities.

An interesting research direction to explore is the performance analysis of CAS in presence of dependencies among input parameters. For instance, in our scenario one could think of a synchronised behaviour of doors so that they change state in a coordinated way. This implies that the probability for a robot to find the second door open (after it crossed the first one) depends on its speed and the time when the robot went through the first door.

References

1. Y. Abd Alrahman, R. De Nicola, and M. Loreti. A calculus for collective-adaptive systems and its behavioural theory. *Inf. Comput.*, 268:104457, 2019.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
3. L. Apvrille, T. Tanzi, and J.-L. Dugelay. Autonomous drones for assisting rescue services within the context of natural disasters. In *URSI General Assembly and Scientific Symposium (URSI GASS)*, pages 1–4, 2014.
4. G. Balbo and G. Ciardo. On Petri Nets in Performance and Reliability Evaluation of Discrete Event Dynamic Systems. In *Carl Adam Petri: Ideas, Personality, Impact*, pages 173–185. Springer, 2019.

5. M. Bartoletti, T. Cimoli, and M. Murgia. Timed session types. *Log. Methods Comput. Sci.*, 13(4), 2017.
6. M. Bartoletti, T. Cimoli, M. Murgia, A. S. Podda, and L. Pompianu. A contract-oriented middleware. In *International Conference on Formal Aspects of Component Software (FACS)*, volume 9539, pages 86–104. Springer, 2015.
7. M. Bertoli, G. Casale, and G. Serazzi. JMT: performance engineering tools for system modeling. *SIGMETRICS Perform. Evaluation Rev.*, 36(4):10–15, 2009.
8. L. Bocchi, M. Murgia, V. T. Vasconcelos, and N. Yoshida. Asynchronous timed session types - from duality to time-sensitive processes. In *Programming Languages and Systems (ESOP)*, volume 11423, pages 583–610. Springer, 2019.
9. L. Bocchi, W. Yang, and N. Yoshida. Timed multiparty session types. In *International Conference on Concurrency Theory (CONCUR)*, volume 8704, pages 419–434. Springer, 2014.
10. D. Castro-Perez and N. Yoshida. CAMP: cost-aware multiparty session protocols. *Proc. ACM Program. Lang.*, 4(OOPSLA):155:1–155:30, 2020.
11. D. Cerotti, M. Gribaudo, P. Piazzolla, R. Pinciroli, and G. Serazzi. Multi-class queuing networks models for energy optimization. In *International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)*. EAI, 2014.
12. A. Das, J. Hoffmann, and F. Pfenning. Parallel complexity analysis with temporal session types. *Proc. ACM Program. Lang.*, 2(ICFP):91:1–91:30, 2018.
13. R. De Nicola, S. Jähnichen, and M. Wirsing. Rigorous engineering of collective adaptive systems. *International Journal on Software Tools for Technology Transfer*, 22(4):389–397, 2020.
14. M. Gribaudo, R. Pinciroli, and K. S. Trivedi. Epistemic uncertainty propagation in power models. *Electronic Notes in Theoretical Computer Science*, 337:67–86, 2018.
15. O. Inverso, H. C. Melgratti, L. Padovani, C. Trubiani, and E. Tuosto. Probabilistic analysis of binary sessions. In *International Conference on Concurrency Theory (CONCUR)*, volume 171 of *LIPIcs*, pages 14:1–14:21, 2020.
16. O. Inverso, C. Trubiani, and E. Tuosto. Abstractions for collective adaptive systems. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, LNCS. Springer, 2020.
17. M. H. Johari, S. N. A. Jawaddi, and A. Ismail. Survey on formation verification for ensembling collective adaptive system. *Advances in Data Computing, Communication and Security*, pages 219–228, 2022.
18. E. Lazowska, J. Zahorjan, G. Scott Graham, and K. Sevcik. *Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., Englewood Cliffs, 1984.
19. E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance - Computer System Analysis Using Queueing Network Models*. Prentice Hall, 1984.
20. L. Lopes and F. Martins. A safe-by-design programming language for wireless sensor networks. *J. Syst. Archit.*, 63:16–32, 2016.
21. H. López, F. Nielson, and H. Nielson. Enforcing availability in failure-aware communicating systems. In *International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE)*, volume 9688, pages 195–211. Springer, 2016.
22. H. A. López and K. Heussen. Choreographing cyber-physical distributed control systems for the energy sector. In *SAC*, pages 437–443. ACM, 2017.

23. M. Loreti and J. Hillston. Modelling and analysis of collective adaptive systems with carma and its tools. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 83–119. Springer, 2016.
24. R. Majumdar, N. Yoshida, and D. Zufferey. Multiparty motion coordination: from choreographies to robotics programs. *Proc. ACM Program. Lang.*, 4(OOPSLA):134:1–134:30, 2020.
25. R. Neykova, L. Bocchi, and N. Yoshida. Timed runtime monitoring for multiparty conversations. *Formal Aspects Comput.*, 29(5):877–910, 2017.
26. D. Pianini, R. Casadei, M. Viroli, and A. Natali. Partitioned integration and coordination via the self-organising coordination regions pattern. *Future Generation Computer Systems*, 114:44–68, 2021.
27. R. Pinciroli, C. U. Smith, and C. Trubiani. Qn-based modeling and analysis of software performance antipatterns for cyber-physical systems. In *International Conference on Performance Engineering (ICPE)*, pages 93–104. ACM, 2021.
28. R. Pinciroli and C. Trubiani. Model-based performance analysis for architecting cyber-physical dynamic spaces. In *International Conference on Software Architecture (ICSA)*, pages 104–114, 2021.
29. E. Tuosto and R. Guanciale. Semantics of global view of choreographies. *J. Log. Algebr. Meth. Program.*, 95:17–40, 2018.
30. A. Vandin and M. Tribastone. Quantitative abstractions for collective adaptive systems. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 202–232. Springer, 2016.
31. M. Viroli, G. Audrito, J. Beal, F. Damiani, and D. Pianini. Engineering resilient collective adaptive systems by self-stabilisation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 28(2):1–28, 2018.
32. F. Weidinger, N. Boysen, and D. Briskorn. Storage assignment with rack-moving mobile robots in KIVA warehouses. *Transp. Sci.*, 52(6):1479–1495, 2018.