Computing
https://doi.org/10.1007/s00607-020-00861-2

SPECIAL ISSUE ARTICLE

Modelling resilient collaborative multi-agent systems

Inna Vistbakka[1] · **Elena Troubitsyna**[2]

Received: 22 February 2020 / Accepted: 19 October 2020

Abstract
Multi-agent systems constitute a wide class of decentralised systems. Their functions are usually carried out by collaborative activities of agents. To ensure resilience of multi-agent systems, we should endow them with a capability to dynamically reconfigure. Usually, as a result of reconfiguration, the existing relationships between agents are changed and new collaborations are established. This is a complex and error-prone process, which can be facilitated by the use of formal reasoning and automated verification. In this paper, we propose a generic resilience-explicit formalisation of the main concepts of multi-agent systems. Based on it, we introduce corresponding specification and refinement patterns in Event-B. Our patterns facilitate modelling behaviour of resilient multi-agent systems in a rigorous systematic way and verification of their properties. We demonstrate the application of the proposed approach by a case study—a smart warehouse system.

Keywords Multi-agent systems · Resilience · Formal modelling · Event-B · Refinement

1 Introduction

Multi-agent systems constitute a large class of decentralised systems [9,23]. There are many examples of multi-agent systems from different domains including robotics, health care, manufacturing etc. Despite differences in the application domains and correspondingly, the nature of their agents, all multi-agent systems rely on agent collaboration to deliver their functions [15]. Moreover, the systems are expected to cope with the unforeseen changes in their operating environment as well as internal

✉ Inna Vistbakka
inna.vistbakka@gmail.com

Elena Troubitsyna
elenatro@kth.se

1 Åbo Akademi University, Turku, Finland

2 KTH – Royal Institute of Technology, Stockholm, Sweden

Published online: 19 November 2020

failures. Hence, an important requirement imposed on the multi-agent systems is resilience—an ability to deliver services in a dependable way despite the changes [19].

To achieve resilience, the systems should be able to recognise the changes and adapt to them. One of the main mechanisms to achieve resilience is dynamic reconfiguration [13,16]. In this paper, we propose a generic formalisation of the concept of dynamic reconfiguration of multi-agent systems and define the corresponding specification patterns for resilience-explicit modelling of multi-agent systems in Event-B. Our formalisation introduces the notion of agent capability that dynamically changes according to the internal system state and the changes in the operating environment. The functional behaviour of the system is structured using the concept of goals [18]. Based on their capabilities, the agents can establish collaborations and perform their activities in a cooperative way to achieve the required goals. We define the logical relationships between the main generic concepts and then map them into Event-B framework.

Event-B [1] is a state-based modelling framework for formal specification and proof-based verification of distributed and reactive systems. The framework is supported by the Rodin platform [2] that provides us with an integrated environment for modelling and verification. Event-B supports correct-by-construction development paradigm, which enables a derivation of a system specification in a number of correctness-preserving refinement steps. In this paper, we rely on our generic formalisation to define the modelling patterns required for specification and verification of dynamic reconfiguration. Our patterns facilitate verification of correctness of complex dynamically-changing agent collaboration and interactions during dynamic reconfiguration. We demonstrate an application of the proposed approach by a case study—a development of smart warehouse system.

In this paper, we use Event-B refinement to unfold system architecture in a stepwise way. Refinement allows us to incrementally introduce resilience mechanisms at different levels of system architecture. By formally specifying agent capabilities and their collaborations, we systematically derive the specifications of both system-level and local reconfiguration mechanisms required to achieve resilience. Our reasoning about resilience at different levels of abstraction facilitates verifying that they allow the system to achieve its goal.

We believe that the proposed approach facilitates the development of complex multi-agent systems by formalising the main concepts of dynamic reconfiguration mechanism and demonstrating how to develop resilient multi-agent systems in a systematic and rigorous way.

2 Resilient multi-agent system

Multi-agent systems belong to a large class of distributed systems composed of asynchronously communicating heterogeneous components. In our work, we focus on studying a behaviour of multi-agent systems that should function autonomously, i.e., without human intervention, for the extended periods of time [13,16]. Usually, these are different kinds of robotic systems that can be deployed, e.g., in hazardous or unac-

cessible areas [11]. Autonomy and resilience require from a multi-agent system a capability to monitor and adapt its behaviour in response to the external and internal conditions. Typically, adaptability is achieved via dynamic reconfiguration.

A system configuration is a specific arrangement of the elements (components) that compose the system [24]. A configuration can be defined by relationships and dependencies between system elements that are established according to the missions (or functions) of the system. Dynamic reconfiguration implies that the system is capable of changing its configuration, i.e., evolve from one configuration to another. As a result of reconfiguration, some components might be replaced or removed from the system, while new components being introduced. Consequently, this leads to changing interdependencies between components and probably, also their interactions.

The purpose of reconfiguration is to ensure that the system remains operational and dependable, i.e., achieve resilience [24]. Since the components of the system—the agents—should perform some functions in a collaborative way, dynamic reconfiguration might have unforeseen effect on agent's relationships and interactions. It is clear that resilience plays an important role in the design of multi-agent systems and hence, should be addressed explicitly while reasoning about relationships between the system components and their interactions, as we demonstrate next.

2.1 Resilience-explicit modelling of multi-agent interactions

In this section, we present a formalisation of the key concepts of resilient multi-agent systems and resilience-explicit reasoning about collaborative multi-agent system. We focus on formalising the notions of agents, their attributes as well as agent relationships and interactions. The formalisation facilitates an analysis of logical connections between agents and the conditions under which agent interactions result in a correct execution of a cooperative activity. The established dynamic relationships between the agents allow us to reason about resilience of complex agent interactions.

Agents are autonomous heterogeneous components that asynchronously communicate with each other. Each agent has a certain functionality within a system and contributes to achieving system goals. Goals are the functional and non-functional objectives of a system [18]. Goals constitute suitable basics for reasoning about the system behaviour and its resilience. Resilience can be seen as a property that allows the system to progress towards achieving its functional goals despite changes in the internal and external operating conditions.

The goal-oriented framework provides us with a suitable basis for reasoning about reconfigurable autonomous systems. We formulate reconfigurability as an ability of agents to redistribute their responsibilities and restore or compensate their capabilities to ensure goal reachability. Next we discuss how the notions of goals, agents, agent capabilities and agent interactions can be used to reason about behaviour of an autonomous resilient multi-agents system.

2.2 Main concepts of multi-agent systems

We assume that there is a number of main (global) *goals* defined for the system. Let G be a set of functional and non-functional goals that system should achieve. Goals can be decomposed into a subset of corresponding subgoals and organised hierarchically. In general, the goals at the same level of hierarchy are considered to be independent. They might have a conflict on some system resource required for their accomplishment. Such a conflict can be resolved by explicitly modelling the state of the resource and locking and unlocking by the corresponding goal. Since the focus of this work is on agent interactions and resilience, for brevity, we assume that goals are on the same level and not conflicting with each other.

The system consists of a number of *agents* (components, in general). Let A be a set containing all possible system agents. We also define agent classes. Each system agent belongs to a particular agent class. These classes represent a partitioning of the system agents into different groups according to their functional capabilities. In general, there can be many agent classes $A_i, i \in 1..n$, such that $\forall\, i \in 1..n, A_i \subseteq A$, $A_1 \cup \ldots \cup A_n = A$. We assume that all of them are disjoint.

During the system functioning, the agents have to utilise their capabilities in order to contribute to overall goal achievement. We define C to be a set of all agent capabilities. Then, we also define a relation AC (called *agent capabilities*) between the agents and their capabilities as follows:

$$AC : A \leftrightarrow C \tag{1}$$

It associates agents with their capabilities. In general, agents might have many capabilities, and different agents might have the same capabilities. Changes in operating environment or internal failures can prevent them from utilising their capabilities, i.e., AC is a dynamic structure meaning that at the run-time, a set of current agent capabilities might be changing. Therefore, AC is a state-dependant relation.

In practice, if the system has a small number of agent types and their capabilities, the capabilities can be represented by corresponding separate variables.

Based on their capabilities, the agents perform the tasks contributing to achieving the system goals. We define the following function GC_Rel to associate the goals with the agent capabilities:

$$GC\_Rel : G \leftrightarrow C. \tag{2}$$

Therefore, for any goal g and agent capability c_j, the expression $(g \mapsto c_j) \in GC\_Rel$ implies that capability c_j is required to achieve the goal g.

For example, a mobile robot might have a capability "bring a box", which might become unavailable if it experiences a grip failure, which, in turn, would result in hindering achieving the goal "collect the items in a shipment".

Additionally, we introduce the dynamic agent attribute *Active*, which defines a set of the active (healthy) system agents. We call *active* those agents that can carry out the tasks in order to achieve the system missions. In its turn, *inactive* agents are those agents that are not currently in the system or those that are failed and thus incapable of carrying out any tasks.

Typically, in a multi-agent system, agents interact with each other in order to achieve their individual or common goals. Interactions might be simple, e.g., information

exchange, or complex, e.g., involving requests for service provisioning from one agent to another [15].

In our work, we assume that agent interactions in a multi-agent system are based on the specific logical connections between agents called *relationships*. There can be as many relationships as necessary to describe all such connections between agents. A relationship r between two agents can be defined as follows:

$$AA\_Rel_r : A \leftrightarrow A, \tag{3}$$

where r is an identifier of a relationship. AA_Rel_r establishes the logical connections between two system agents of the same or different classes. In a general case, if we want to specify a relationship involving more than two agents, we can define AA_Rel_r as follows:

$$AA\_Rel_r : A \leftrightarrow A \leftrightarrow \ldots \leftrightarrow A \tag{3'}$$

In Event-B modelling, it is more convenient to operate with a pair-wise definition of relationship (3) rather than a general one (3'). Nevertheless, such a modelling convenience does not prevent us from considering a general case as well.

Similarly to agent capabilities, the agent relationships are dynamic and might change during the system execution. If this relation holds for several agents then these agents might be or are currently engaged in a certain collaboration required to provide a predefined system function.

We consider agent interactions to be the essential supporting mechanism of achieving system goals. Namely, to perform the required system functions, the system agents should interact and collaborate with each other. Thus, in our work, we represent system functions as collaborative activities of autonomous system agents. Next we present a detailed formal analysis of component activities and component interactions while providing a certain function and/or participating in a specific collaboration.

2.3 Agent interactions and system reconfigurability

Let us now focus on defining the essential properties of agent interactions in cooperative activities. As a result, we will derive the constraints that should be imposed on them to achieve resilience.

In multi-agent systems, we distinguish between two types of agent relationships: static and dynamic. The static relationships are known at the system initialisation. They do not change during the system execution. The dynamic relationships might change during the system functioning. The dynamic relationships can be *pending* (i.e., incomplete) and *resolved* (i.e., completed). The pending relationships are often also caused by a failure or disconnection of the agents previously involved in a relationship. Moreover, an existing agent may initiate a new pending relationship with other agents.

Next we formulate a number of required properties that determine the rules for regulating correct interactions and collaborative agent activities in a multi-agent system.

Property 1 *Let EAA be all interaction activities defined between agents and let EAI be all individual agent activities. Moreover, for each agent* $a \in A$*, let* E_a *be a set of*

activities in which the agent a can be involved. Then

$$\forall\, a \cdot a \in Active \wedge E_a \neq \varnothing \Rightarrow E_a \subseteq EAA \cup EAI$$

and

$$\forall\, a \cdot a \in A \wedge a \notin Active \wedge E_a \neq \varnothing \Rightarrow E_a \subseteq EAI \setminus EAA.$$

This property defines agent interactions with respect to the agent health status. If the agent is recovering from the failure and it is involved in some activities, these activities are individual and not cooperative. Therefore, while modelling agent interactions, we have to consider the agent status. However, there might be a situation, when while participating in collaborative activity an agent might fail.

The next property concerns collaborative activities between the agents and how these activities are linked with the inter-agent relationships.

Property 2 *Let EAA be all the interactions in which active agents are involved. For each collaborative activity $ca \in EAA$, let $AA\_Rel\_Set_{ca}$ be a set of all the relationships associated with this collaborative activity. Finally, for each collaborative activity $ca \in EAA$, let A_ca be a set of all agents involved in it. Then, for each $ca \in EAA$, $AA\_Rel_{ca} \in AA\_Rel\_Set_{ca}$, and any $a_1, a_2 \in A\_ca$,*

$$(a_1 \mapsto a_2) \in AA\_Rel_{ca} \text{ or } (a_2 \mapsto a_1) \in AA\_Rel_{ca}$$

This property regulates the interaction activities between the agents—only the agents that are linked by relationships can be involved into cooperative activities. In general, some of the relationships might be pending.

Let us note that for the case, when more than two agents are to be involved in the activity, this property can still be formulated in the same way by taking into account the arity of the relationships AA_Rel_j.

Property 3. Let $CA\_g \in EAA$ be an agent collaborative activity associated with the achievement of goal $g \in G$ and GC_g be a required subset of agents capabilities defined by GC_Rel. Moreover, let A_g be a set of all agents involved in a collaboration for achieving goal g. Then for every capability $cp \in GC\_g$

$$\exists\, a.\, a \in A\_g \wedge\ a \in Active \wedge a \mapsto cp \in AC$$

This property describes the agent interaction activity required for goal fulfilment—the agents, involved into the activity for the goal accomplishment should have the required capabilities to achieve this goal.

In our work, we study *reconfigurability* as an essential mechanism of achieving resilience of multi-agent systems. If under the current configuration the system is not able to achieve a certain goal, it should perform a *reconfiguration*. As a result of reconfiguration, an agent might receive additional responsibilities, i.e., it could become involved into an execution of tasks that were not assigned to it initially.

We assume that agents are co-operative, i.e., they always accept the new responsibilities. In this case, a new relationships between agents can be established to allow them

to collaboratively contribute to goal achievement. At the same time, the agents are unreliable, i.e., they might fail and cease performing their functions. This might also trigger system reconfiguration. As a result, the responsibilities of the failed agents can be re-allocated to the healthy ones. If an agent is healthy and idle, it can be deployed to perform the functions of failed agents or it might also become engaged in an execution of some other task, e.g., to improve the system performance and/or increase the likelihood of successful task completion.

The reconfiguration mechanisms ensure that the system progresses towards achieving its goals despite agent failures or becomes more performant by using its agents more efficiently. Since reconfiguration is a powerful technique for achieving resilience, we have proposed a general formalisation of the reconfigurability concept, by connecting it with the system goals, agents, agent capabilities and their inter-relationships. In this paper, we demonstrate how our generic formalisation can be supported by an automated formal framework—Event-B, which we overview next.

3 Modelling and refinement in Event-B

Event-B is a state-based formal approach that promotes the correct-by-construction development paradigm and formal verification by theorem proving. In Event-B, a system model is specified using the notion of an *abstract state machine* [1]. An abstract state machine encapsulates the model state, represented as a collection of variables, and defines operations on the state, i.e., it describes the dynamic behaviour of a modelled system. The important system properties that should be preserved are defined as model invariants. Usually a machine has the accompanying component, called context. A context is the static part of a model and may include user-defined carrier sets, constants and their properties (defined as model axioms).

The system dynamic behaviour is described by a collection of atomic *events* defined in the machine part. Generally, an event has the following form:

$$\mathsf{event_e} \;\widehat{=}\; \textbf{any}\ x_e\ \textbf{where}\ G_e\ \textbf{then}\ R_e\ \textbf{end}$$

Here $\mathsf{event_e}$ is the unique name of the event, x_e is the list of local variables, and G_e is the event guard—a predicate over the model state. The body of an event is defined by a *multiple* (possibly nondeterministic) assignment to the system variables. In Event-B, this assignment is semantically defined as the next-state relation R_e. The event guard defines the conditions under which the event is *enabled*, i.e., its body can be executed. If several events are enabled at the same time, any of them can be chosen for execution nondeterministically.

System development in Event-B is based on a top-down refinement-based approach. A development starts from an abstract specification that nondeterministically models the most essential functional system behaviour. In a sequence of refinement steps, we gradually reduce nondeterminism and introduce detailed design decisions. In particular, we can add new events, refine old events as well as replace abstract variables by their concrete counterparts. The *gluing invariants* are used to link the abstract and

concrete state variables. A correct refinement ensures that the properties of defined in an abstract specification are also preserved in the concrete one.

The consistency of Event-B models—verification of model well-formedness, invariant preservation as well as correctness of refinement steps—is demonstrated by discharging the relevant proof obligations. For instance, to verify *invariant preservation*, we should prove the following logical formula:

$$A(d, c),\ I(d, c, v),\ G_e(d, c, x, v),\ R_e(d, c, x, v, v') \vdash I(d, c, v'),$$

where A are the model axioms, I are the model invariants, d and c are the model constants and sets respectively, x are the event's local variables and v, v' are the variable values before and after event execution. The full definitions of all the proof obligations are given in [1].

The Rodin platform [2] provides an automated integrated support for formal modelling and verification in Event-B. The platform provides us with the facilities for creating and editing models as well as model animation. Moreover, it also generates and tries to automatically prove the required proof obligations. When the proof obligations cannot be discharged automatically, the user can attempt to prove them interactively using a collection of available proof tactics.

4 Modelling agent interactions in Event-B

In this section, we demonstrate how the generic formalisation presented in Sect. 2 can be instantiated within Event-B framework.

Event-B separates the static and dynamic parts of a model, putting them into distinct yet dependent components called a *context* and a *machine*. All the static notions of our formalisation including the set of all possible goals, agents and capabilities (G, A and C, respectively) as well as different static structures defining various interdependencies between the elements are defined in *context*. The latter also includes the (initial) values of agent capabilities, the logical goal function over the required capabilities and the initial agent relationships (AC_init, GC_Rel, $AA\_Rel\_init_i$, correspondingly). We introduce static notions as sets and constants of a model context and define their properties as a number of context axioms.

The machine part of the Event-B specification defines system dynamic. Therefore, in the machine part, we should represent all dynamic notions introduced in our formalisation, e.g., $Active$, AA_Rel_i, etc. They are modelled as the corresponding model variables. The types of the variables as well as logical relations between the defined notions are represented as model invariants and predicate expressions. Agent activities are modelled by the corresponding model events. Below we will discuss several generic cases of agent activities and agent interactions typical for a multi-agent system and show how they can be modelled in Event-B.

We start by defining a variable $Goals\_state$ that models the current state of the system goals:

$$Goals\_state \in G \rightarrow STATES,$$

where $STATES = \{incompl, compl\}$. The variable $Goals\_state$ obtains the value *compl* when the main goal is achieved. Otherwise, it has the value *incompl*. Initially, none of the goal is completed, i.e., the status of a goal is *incompl*. After a successful accomplishment, the goals status changes to *compl*, i.e., $Goals\_state(g) = compl$.

To abstractly model the process of achieving the goal, we define the event Reaching_Goal. It might change the value of the variable $Goals\_state$ from *incompl* to *compl*. The system continues its execution until all goals are reached. Note that this event is parametrised - the parameter g designates the id of the goal in process of getting achieved.

The anticipated status of the event indicates that we promise to prove the convergence of this event, thus showing reachability of any system goals. The actual proof of such convergence is postponed until some later refined model, which has enough implementation details to prove the overall convergence based on a formulated variant expression. Alternatively, we can rely on ProB, a model checker for Event-B, and verify goal reachability by formulating and checking the corresponding temporal logic property for the considered system models.

Reaching_Goal status **anticipated**
  any g, $result$
  where $g \in G$
    $Goals\_state(g) = incompl$
    $result \in STATES$
  then $Goals\_state(g) := result$
  end

Next we model a simple case of agent local activities—joining and leaving the system location (system environment). This abstraction is suitable for modelling agent failures and introducing new agents into the system (e.g., to model an agent recovery or replacement). In the machine part of Event-B specification, we define the corresponding events Activation and Deactivation as presented below.

Activation $\widehat{=}$
  any a
  where $a \in A \wedge a \notin Active$
  then $Active := Active \cup \{a\}$
  ...
  end

Deactivation $\widehat{=}$
  any a
  where $a \in Active \wedge$
    $\forall i.\ a \notin dom(AA\_Rel_i) \wedge$
    $\forall i.\ a \notin ran(AA\_Rel_i)$ ...
  then $Active := Active \setminus \{a\}$
  ...
  end

They model simple cases when an agent a joins or leaves the system. Here we do not put any specific restrictions on when such behaviour might occur, since it depends on the specific system properties. In the event Deactivation, we only check that that an agent should not be involved in any relationships with the other agents before leaving the system.

The event InteractionActivity abstractly models a possible interaction between two agents $a1$ and $a1$ in order to achieve the goal g. Here, in the event guard, we specify conditions when this interaction can happen. While formalising these conditions, we take into consideration the properties defined in our generic formalisation in Sect. 2. In particular, we require that only active agents can interact with each other. Moreover, each agent should also have specific capabilities to participate in the interaction asso-

ciated with achieving the goal g. Here we describe a generic case of agent interaction and do not specify which particular actions are performed upon this event execution.

InteractionActivity $\widehat{=}$
any $a1, a2, c1, c2, g$
where $a1 \in Ag \wedge a2 \in Ag \wedge a1 \neq a2 \wedge$
$a1 \in Active \wedge a2 \in Active \ \wedge$
$a1 \mapsto c1 \in AC \ \wedge a2 \mapsto c2 \in AC \wedge$
$g \in G \wedge \text{ran}(\{g\} \lhd GC\_Rel) = \{c1, c2\} \wedge$
$((a1 \mapsto a2) \in AA\_Rel\_ca \vee (a2 \mapsto a1) \in AA\_Rel\_ca) \wedge \ldots$
then $\ldots$
end

An initiation of a new relationship between agents can be specified by the event $\text{InitiateRelationship}_{\text{ca}}$ given below. In the event guards, we check that all the required agents are active, eligible and ready to enter the relationship. Here the condition $Elig\_ca(c1, c2) = TRUE$ abstractly models specific eligibility conditions of the agents that should be checked before their interaction ca can be initiated.

InitiateRelationship $\widehat{=}$
any $a1, a2, c1, c2$
where $a1 \in A \wedge a2 \in A \ \wedge a2 \neq a2 \wedge$
$a1 \in Active \wedge a1 \in Active \wedge$
$a1 \mapsto c1 \in AC \ \wedge a2 \mapsto c2 \in AC \wedge$
$a1 \mapsto a2 \notin AA\_Rel\_ca \wedge$
$Elig\_ca(c1, c2) = TRUE \ \wedge \ldots$
then $AA\_Rel\_ca := AA\_Rel\_ca \cup \{a1 \mapsto a2\}$
$AA\_Rel\_ca := AA\_Rel\_ca \cup \{a2 \mapsto a1\}$
$\ldots$
end

Similarly, we can model collaborating activities involving any number of agents.

Next we will discuss how such agent interactions allow us to build different mechanisms to ensure system resilience.

Modelling Resilience Mechanisms in Multi-Agent Systems. To model different resilience mechanisms in the context of multi-agent systems, we rely on the concepts and properties discussed above. The resilience mechanisms can be introduced at both—system and local (i.e., individual agent) levels. The system-level mechanisms involve a number of agents, where the number depends on a scale of occurred failure or a change. We can distinguish between the structural resilience mechanisms (i.e., forming the new collaborations) and the compensating resilience mechanisms (i.e., introducing new agents or capabilities into the system). Small scale failures or changes do not require system-level coordination and can be handled locally, i.e., by an agent itself. For instance, a robot by itself can handle its internal transient failures or mitigate an impact of an unexpected change, e.g., perform an obstacle avoidance maneuver to avoid a collision with an unexpectedly appeared object.

To model possible loss of some agent capability (e.g., due to agent failure), we define an event LoseCapability. As a result of an event execution, a capability c will be lost. The RestoreCapability event models a simple case of agent reconfiguration (as a restoring of the lost capability)

LoseCapability $\widehat{=}$
any a, c
where $a \in A \ \wedge a \in Active \wedge$
$a \mapsto c \in AC \wedge \ldots$
then $AC := AC \setminus \{a \mapsto c\}$
end

RestoreCapability $\widehat{=}$
any a, c
where $a \in A \wedge a \mapsto c \in AC\_init \wedge$
$a \mapsto c \notin AC \wedge \ldots$
then $AC := AC \cup \{a \mapsto c\}$
end

A local resilient mechanism can be modelled in Event-B as the following generic event LocalResilientMechanism given below. Upon detection a change in the system or its environment, an agent performs the required remedy actions to tolerate this disturbance. Here we should check that an agent is healthy, has required capabilities and eligible to perform these actions.

LocalResilientMechanism $\widehat{=}$
any $a1, c1, d\_lm$
where $disturb\_condition(d\_lm) = TRUE$
$a1 \in A \wedge a1 \in Active \wedge a1 \mapsto c1 \in AC \wedge$
$Elig\_lm(c1) = TRUE \wedge \ldots$
then . . . // core agent functionality
end

The reconfiguration mechanism can also be supported by collaborative agent behaviour, where agent collaborations are regulated by relationships between agents. As we discussed before, we can specify an initiation of a new relationship between agents by the event InitiateRelationship. However, when some agents of the initiated relationship are still unknown (e.g., should still be selected), this situation can be defined by the following event InitiatePendingRelationship.

InitiatePendingRelationship $\widehat{=}$
any a, c, c_p
where $a \in A \wedge a \in Active \wedge a \neq a0 \wedge$
$a \mapsto c \subseteq AC \ \wedge c\_p \in C \wedge$
$Elig\_ca(c, c\_p) = TRUE$
$a \mapsto a0 \notin AA\_Rel\_Pending\_ca \ \wedge \ldots$
then $AA\_Rel\_Pending\_ca := AA\_Rel\_Pending\_ca \ \cup \{a \mapsto a0\}$
. . .
end

Here we use the pre-defined element $a0$ to designate a missing agent in the pending relationship. In this event, an agent a initiates a new pending relationship, where the place for a second agent of the particular type is currently vacant (i.e., is marked by $a0$). The resulting pending relationships is added to the set of pending relationships $AA\_Rel\_Pendingd\_ca$.

The pending relationship is resolved, when the corresponding agent "joins" this collaborative activity. The event AcceptRelationship abstractly models this situation.

AcceptRelationship $\widehat{=}$
any $a1, a2, c1, c2$
where $a1 \in Active \wedge a2 \in Active \wedge$
$a1 \mapsto c1 \in AC \wedge a2 \mapsto c2 \in AC \wedge$
$Elig\_ca(c1, c2) = TRUE \wedge$
$a1 \mapsto a0 \in AA\_Rel\_Pending\_ca \wedge \ldots$
then $AA\_Rel\_ca := AA\_Rel\_ca \cup \{a1 \mapsto a2\}$
$AA\_Rel\_Pending\_ca := AA\_Rel\_Pending\_ca \setminus \{a1 \mapsto a0\}$
$\ldots$
end

Let us note, that in a similar way, we can model all collaborating activities involving any number of agents.

In our work, we rely on the assumption that agents behave in a cooperative way. Therefore, the reconfiguration mechanisms are enabled by a collaborative agent behaviour, where agent collaborations are regulated by relationships between agents as defined by properties 2 and 3 defined in Sect. 2. The system reconfiguration mechanisms can be based on a reallocation of the execution of certain functional tasks from some components (e.g., failed) to the another (e.g., healthy) ones. Such a mechanism guarantees system resilience in the presence of agent failures or other changes.

Next we will demonstrate an application of the proposed formal framework and present a case study—a smart warehouse system. We will show how our generic formalisation presented in Sect. 2 and the Event-B modelling patterns can be instantiated and used to model a resilient multi-robotic systems.

5 Autonomous resilient smart warehouse system

5.1 Case study description

A smart warehouse is a fully automated storage and shipment facility. It is equipped with the autonomous robots that can transport labelled boxes (goods) between the multi-level shelves and collection points. Since arrival and dispatch of the boxes is outside of the scope of our study, we assume that the boxes just appear on the conveyor belt when they arrive to the warehouse and should be transported to the shelves. In the similar way, the boxes disappear from the conveyor belt when they are brought to it for shipment.

Each box has a unique RFID tag attached to it. When a box arrives at the warehouse, the warehouse management system (WMS) assigns it the place at which it should be stored. Correspondingly, in its database WMS keeps track of box-place assignments.

Each robot is equipped with an extendable arm, which can pick up a box from a shelf and put it on the robot's storage space (located on its base). It can also take the box from the robot's base and put it on the shelf. The arm is equipped with the RFID tag reader, i.e., it can check the RFID of the box that it handles.

Each robot has a unique ID known to WMS. The robot can communicate with WMS. It receives the orders to bring the box from the corresponding place or fetch

and bring it to the collection points. WMS sends the robot both the ID of a place and RFID of the box. WMS also sends the routes to the robots.

The robot has a battery and a corresponding sensor that detects the level of the battery charge. If robot's battery reaches its critical value (but still enough to perform some actions), the robot should stop the execution of the current task, move and leave the box at the specific place, and then travel to a charging station.

If a robot requires to charge its battery, it first sends a request for charging to its predefined (attached) station. If this station is able to charge this robot, it confirms the request. However, due to a possible failure of a charging station or its overload, the attached charging station can be changed. In this case, a robot will contact other station in its proximity until some station agrees to provide it with charging.

A robot communicates with WMS when it fails to complete its operation due to some reasons. It also should send a notification to WMS when it decides to abort its current assignment and move to a charging station. In this case, WMS will reassign robot's task to the next robot.

Each robot is also equipped with a radar. It allows a robot to detect obstacles on its way. It also recognizes whether the obstacles are moving or static and estimates a distance to them. In general, WMS plans routing for all robots in such a way that the obstacles are avoided. However, it applies to the obstacles known at the time of route planning. Hence, if a box is accidentally dropped or some robot's motor fails and the robot stops then a moving robot can encounter an obstacle at an unexpected location. In this situation, the robot should on its own, i.e., without notifying WMS, execute a collision avoidance maneuver. Since such a maneuver might result in a deviation from the planned route. Hence, after avoiding a collision with an obstacle, the robot should also notify WMS, which should decide whether the route should be recalculated

Due to some unforeseen deviations while en-route, some robots might run into a risk of collision. For instance, they might be moving towards each other and their planned paths intersect. Such a situation should be handled by the collaborative robots actions. The robots will follow a predefined procedure to determine the manoeuvres to be performed in order to avoid the imminent collision. Such unforeseen situations are handled by the robots locally, i.e., without coordination of WMS. After robots perform the collision avoidance procedure, their routes will be recomputed by WMS.

The described warehouse system has a heterogeneous architecture and consists of different types of agents (robots and charging stations). The possible changes in the system and its operating environment include components failures (both robots and charging stations), static and dynamic obstacles appearance as well as sudden robot's battery depletion. Thus to achieve resilience a system should stay operational despite all such unpredictable changes.

To achieve overall goals, the components of the smart warehouse system should behave cooperatively. However, heterogeneity of the robots and variety of possible conditions pose a significant challenge in ensuring correctness of system behaviour and resilience. Hence, we will rely on formal Event-B modelling to derive a specification of a resilient WMS.

5.2 Event-B development of a smart warehouse system

Let us now overview the key modelling aspects of our Event-B development of a smart warehouse system (SWS). The main focus of our development is a specification of complex collaborative behaviour of agents in SWS. In particular, we focus on modelling the collaborative behaviour of agents within battery charging procedure and collision avoidance.

While modelling, we rely on our generic formalisation presented in Sect. 2 that covers the notions of system agent, agent capabilities and statuses as well as agent relationships. Moreover, we employ the Event-B refinement technique to gradually unfold the system architecture and functionality. This allows us to represent the system agents, model their local behaviour (both normal and abnormal) as well as introduce agent collaborative interactions for ensuring system resilience. We will use the generic development solutions discussed in Sect. 4.

Initial model: System Goal Modelling We start our development with an abstract model, an excerpt from which is shown below. Essentially, it representing the behaviour of a smart warehouse system as a process of achieving the main goal—handling requests for services, which arrive from the system responsible for the logistics. WMS receives such requests and processes them. The actual execution of such requests is handled by the robots.

Machine SWS_m1 refines SWS_m0
Sees SWS_c1
Variables $requests, request\_status$
Invariants $requests \subseteq REQUESTS \wedge request\_status \in requests \rightarrow STATUS \wedge \ldots$
Events

Initialisation $\widehat{=} \ldots$

RequestArrival $\widehat{=}$
any rq
where $rq \in REQUESTS \wedge rq \notin requests$
then $requests := requests \cup \{rq\}$
$request\_status(rq) := incompl$
end

RequestService $\widehat{=}$
any rq, res
where $rq \in requests \wedge request\_status(rq) = incompl \wedge res \in STATUS$
then $request\_status(rq) := res$
end
$\ldots$

Modelling Agents and their Interdependencies In our first refinement, we introduce system agents, define some relationships between them as well as model the main agent activities.

In the context part of our Event-B specification, we represent system components by a finite non-empty set of agents *AGENTS* and its partition to sets of *ROBOTS* and *CSTATIONS*, modelling robots and charging stations correspondingly. This set might contain the ids of all agents in the system.

In the machine part, we define the variable $robots \subseteq ROBOTS$ to model the active robot agents and the variable $cstations \subseteq CSTATIONS$ to model the active charging stations. By "active" we mean such robots and stations that are currently present in the warehouse location and are functional. The events ActivateRobot, DeactivateRobot, ActivateStation, DeactivateStation model joining and leaving warehouse location by the system agents.

Each robot joining the system should be associated with a charging station. To model this relationship, we introduce the variable *Attached*, which is defined as a total

function associating the robots with the charging stations:

$$Attached \in robots \rightarrow cstations.$$

Attached is a representation of one of possible relationships between system agents discussed in Sect. 2. In the event ActivateRobot, we specify to which charging station a new robot will be attached:

ActivateRobot $\widehat{=}$
any rb, cs
where $rb \in ROBOTS \wedge rb \notin robots \wedge cs \in cstations$
then $robots := robots \cup \{rb\}$
$Attached := Attached \cup \{rb \mapsto cs\}$
end

Here the guard $cs \in cstations$ ensures preservation of a specific instance of the **Property 1**: only the active charging stations are assigned to the robots. The remaining events model agents leaving a system in a similar way.

WMS issues orders—assigns the tasks to bring a box from/to a shelf position—to robots. To assign such a task, WMS chooses an idle robot and requests its battery status. Then WMS either commands the robot to bring a box or move to a certain charging station. In the former case, WMS chooses another robot for the assignment. We model this behaviour by abstract events RequestBatteryLevel and AssignRobotTask (given below). While assigning a task, we ensure that a robot is idle and a task is not currently being performed by any other robot. Here we also check whether the current battery level is sufficient to perform a task. The current battery level defines one of the conditions of availability of the robot capability required to perform a task. We use an abstract function b_min that returns battery level required for a task. Let us note that in our case study, it is more convenient to model different agent capabilities as the corresponding model variables instead of aggregating them into one theoretical concept *AC* introduced in Sect. 2. While leaving the generic notion of agent capability *AC* intact, such a modelling style improves readability of Event-B specification and simplifies the proofs.

AssignRobotTask $\widehat{=}$
any rb, tk
where $rb \in robots \wedge rb \notin \text{ran}(Assigned\_Task) \wedge task\_status(tk) = incompl \wedge$
$tk \in TASKS \wedge tk \notin dom(Assigned\_Tasks) \wedge battery(rb) > b\_min(tk)$
then
$Assigned\_Tasks := Assigned\_Tasks \cup \{tk \mapsto rb\}$
end

The dynamic system behaviour is represented by the process of achieving system goals by decomposing them into tasks and assigning to the agents—*goal assignment*. A task can be "assigned" to a robot, which will try to perform it:

$$Assigned\_Tasks \in TASKS \rightarrowtail ROBOTS.$$

Here $\rightarrowtail$ denotes a partial *injection*. The function is injective because we assume that an agent can not perform more than one task simultaneously. Obviously, only uncompleted task can be assigned to a robot for execution. This property is formulated

as a model invariant:

$$\forall\, tk.\ tk \in dom(Assigned\_Tasks) \Rightarrow Task\_status(tk) = incompl.$$

The robot failures have impact on the whole behaviour of the warehouse system. Even if a task has been assigned to a healthy robot, a task cannot be completed if that robot fails during task execution or becomes incapable of completing its assigned task due to, e.g., battery depletion. To model this behaviour, we define two events RobotTaskSuccess and RobotTaskFailure, which respectively model successful and unsuccessful execution of a task by the robot. If the robot fails to achieve the assigned task, its task can be reassigned to another robot capable of achieving it.

A robot failure results in loosing some capability. Therefore, we rely on our definitions of agent capabilities (1) and association of goals with capabilities (2) to decide whether a failure prevents a robot from achieving its assigned task. Moreover, we rely on the same definitions to select a robot capable of carrying the task, which should be (re)-assigned.

Modelling Agent Interactions In next refinement steps, we model the agents interactions required to contribute to overall goals achievement. While modelling such interactions, we should introduce restrictions on the conditions under which these activities can happen, e.g., only the agents that are linked by specific dynamic relationships can be involved in the corresponding interaction.

First, we discuss a collaborative behaviour between a robot and a charging station. Let us consider a case when a robot needs to charge its battery. During a task execution, a robot constantly monitors its battery level. When a robot detects that it is required to charge the battery, it halts the current task and, if it carries a box, leaves a box in a designated temporal storage area. Then a robot sends a request for charging to its attached charging station. If this charging station can serve a robot (it is either free, or can put a robot into a queue), it accepts the request from a robot. Otherwise, the station rejects the request and the robot re-sends its request to another station. When the charging station is confirmed, the robot moves to this station. After completing charging, a robot notifies WMS and becomes ready to continue its service.

This scenario is an example of a collaborative activity between two agents—a robot and a charging station. It can be modelled according to the generic events InitiateRelationship and AcceptRelationship, InteractionActivity presented in Sect. 4 and relies on the definitions and properties of agent capabilities. Below the event ChargingRequest models sending a request from a robot to its corresponding available charging station, while the event AcceptChargingRequest models acceptance of a request by a charging station. Here we check capability of a station to serve a robot by formulating conditions on a station availability ($cs \notin occupied$) and station capacity ($capacity(cs) < max\_num$), for these events correspondingly.

ChargingRequest $\widehat{=}$
any rb, cs
where $rb \in robots \wedge cs \in cstations \wedge Attached(rb) = cs \wedge$
$rb \mapsto cs \notin RequestToStation \wedge cs \notin occupied$
then $RequestToStation := RequestToStation \cup \{rb \mapsto cs\}$
end

AcceptChargingRequest $\widehat{=}$
any rb, cs
where $rb \in robots \wedge cs \in cstations \wedge rb \mapsto cs \in RequestToStation \wedge$
$capacity(cs) < max\_num \wedge \ldots$
then $RequestToStation := RequestToStation \setminus \{rb \mapsto cs\}$
$ChargingRel := ChargingRel \cup \{rb \mapsto cs\}$
$capacity(cs) = capacity(cs) + 1$
end

If the attached charging station is not able to serve a robot, it rejects a request. In this case, the robot re-sends a request until another station accepts a request for charging. These behaviour is represented by RejectChargingRequest and ResendChargingRequest events. If all the stations are not able to serve a robot, a robot notifies WMS about the current situation. In this case, WMS will resolve this situation (e.g., WMS will "force" some charging station to put a robot in a queue).

Collaborative Collision Avoidance Interactions Next we focus on modelling of the robot cooperation, which is required to avoid a possible collision.

While moving around a warehouse location, a robot monitors an appearance of obstacles on its way. As soon as a robot's radar detects an object on its way and recognises whether it is moving or static, a robot stops its movement. Then the robot performs the corresponding collision avoidance procedure, which depends on whether the obstacle is static or moving. Next we discus the case when the detected obstacle is moving, i.e., it is another robot.

When a robot $rb1$ detects a possible collision with a dynamic obstacle—another robot, $rb2$, it initiates a collision avoidance routine. It tries to establish a communication with the robot $rb2$ that is also a subject to a collision. A robot $rb1$ initiates a new relationship by sending a request for collision avoidance (as modelled by the event RequestToAvoidCollision). Here the condition $CloseProximity(rb1) = rb2$ checks if both robots are linked by a relationship "close proximity" as regulated by Property 2.

As soon as a robot $rb2$ accepts this request, the robots will agree on the next steps to be performed (depending on where the robots are). Collision avoidance follows a protocol to determine the maneuvers to perform in order to avoid the imminent collision (for brevity, we omit its detailed modelling). After a danger of collision is removed, the robots notify WMS and continue executing their tasks.

RequestToAvoidCollision $\widehat{=}$
any $rb1, rb2$
where $rb1 \in robots \wedge rb2 \in robots \wedge CloseProximity(rb1) = rb2 \wedge$
$rb1 \mapsto rb2 \notin CollisionAvoidanceRequest \wedge$
$ca\_module(rb1) = TRUE \wedge ca\_module(rb2) = TRUE \wedge \ldots$
then $CollisionAvoidanceRequest := CollisionAvoidanceRequest \cup \{rb1 \mapsto rb2\}$
end

AcceptRequestToAvoidCollision $\widehat{=}$
any $rb1, rb2$
where $rb1 \in robots \wedge rb2 \in robots \wedge rb1 \mapsto rb2 \in CollisionAvoidanceRequest \wedge$
$ca\_module(rb1) = TRUE \wedge ca\_module(rb2) = TRUE \wedge \ldots$
then $CollisionAvoidanceRequest := CollisionAvoidanceRequest \setminus \{rb1 \mapsto rb2\}$
$CollisionAvoidance := CollisionAvoidance \cup \{rb1 \mapsto rb2\}$
end

In case a robot rb_1 that initiates a collision avoidance relationships does not get a reply from another robot $rb2$, it will notify WMS about this situation. WMS will try to communicate with the robot rb_2, and rb_1 will wait for next WMS control commands. Let us note, that in this case, we have also adopted the modelling patterns defined by the generic events InitiateRelationship and AcceptRelationship, InteractionActivity presented in Sect. 4.

Let us note that agent interactions and cooperative activities in the smart warehouse system are strongly dependant on communication. Communication is a critical aspect of ensuring system resilience. The robots communicate with each other to avoid possible collisions. Moreover, the robots communicate with the charging stations in order to charge their batteries and continue tasks. Finally, the reliable communication is required for the robots to receive the task assignments from WMS, report about task completion and their status or deviations in executing the assigned tasks. In this paper, since we focused on modelling resilience mechanisms relying on agent collaboration, we assumed that the communication is reliable. However, in our previous work [27], we have also studied the problem of unreliable communication and formally specified a communication protocol that ensures correct functioning of a multi-robotic system in presence of message losses and agent disconnections.

In our formal development, we have specified a number of collaborative activities, which the agents perform to achieve the system goals. The collaborations are established dynamically and their status changes when the state of the system or the agents changes. Event-B allowed us to formally define and verify inter-tangled agent interactions at different levels of abstraction. We have demonstrated the collaborative activities that are carried at the system level as well locally. Overall, the formal development in Event-B has resulted in building a clean and well-structured architecture of a multi-agent system.

6 Conclusions and future work

6.1 Conclusions

In this work, we have presented a formal approach to the development of resilient multi-agent systems. We have introduced a generic formalisation of the concept of a dynamic

reconfiguration based on the notions of agent capabilities and collaborations. Such a formalisation has facilitated defining the specification patterns for modelling resilient multi-agent systems in the formal modelling and verification framework—Event-B. In this paper, we focused on the formal analysis of dynamic system reconfiguration as the main mechanism for achieving system resilience. We have shown that the dynamic reconfiguration can be performed at different architectural levels. The system level reconfiguration requires a coordination between several agents, while a local reconfiguration can be performed by an individual agent.

We have demonstrated the use of our approach by a case study—a formal specification of a smart warehouse system. We have shown how to rigorously define different reconfiguration mechanisms required to achieve resilience. Formal modelling and refinement have facilitated the process of specifying complex reconfiguration procedures at different levels of abstraction and formally verifying correctness of agent interactions not only in nominal conditions, but also in presence of failures or dynamically emerging unpredicted conditions.

In this work, we have relied on formal modeling and verification in Event-B. A system specification in Event-B is derived via a number of correctness-preserving refinement steps. In this paper, we used refinement to unfold system architecture and model reconfiguration mechanisms at different architectural levels. A gradual introduction of specification details helped us to derive the specifications of complex reconfiguration mechanisms in a systematic way. By incrementally increasing complexity of the introduced resilience mechanisms, we were able to systematically model intertangled agent interactions as well as represent both system-level and local reconfiguration mechanisms within a single system specification.

Event-B has a mature automated tool support—the Rodin platform. The platform has provided us with an integrated modelling and verification environment. Since Event-B adopts the proof-based approach to verification, in our modelling, we were not constrained by the state-space of the system. Hence, we could model non-deterministically occurring failures or changes and specify agents behaviour and collaboration in different situations. As a result, we were able to verify whether the introduced reconfiguration mechanisms allow the system to achieve its goals, i.e., ensure resilience. We believe that this is a promising direction in formal modelling and verification of multi-agent systems due to its scalability both in terms of the number of the agents and reconfiguration scenarios. The automated tool support—Rodin platform—automatically generated required proof obligations and discharged majority of them automatically.

The majority of the approaches for verifying properties of multi-agent systems rely on model checking. Model checking supports an explicit verification of goal reachability using a temporal logic representation of the reachability property. However, since it relies on checking all possible state transitions, to avoid a state explosion, it would also require to reduce the number of agents as well as modelled failures or deviations. Moreover, it would be hard to represent the architectural hierarchy of the resilience mechanisms, which would make reasoning about resilience less straightforward. In our work, goal reachability is modelled implicitly, i.e., by representing the fact, that all the tasks required to achieve the goal are either executed or executable, i.e., there are agents, which have the required capabilities to carry them. However,

in out approach we were free from the restrictions imposed by the model checking approach. Hence, we believe that our approach is beneficial for modelling complex resilient multi-agent systems with a large number of heterogenous agents, which is typical, e.g., for multi-robotic applications.

In this paper, we have taken a logic (qualitative) view on analysing system resilience and focused on development and verification of different reconfiguration mechanisms and agent collaboration. As a future work, it would be interesting to combine the proposed approach with quantitative stochastic reasoning [25]. This would enable not only design but also the assessment of different reconfiguration strategies as well as different system resilience attributes. Another interesting research direction, is to define a richer set of patterns modelling different forms of collaboration and reconfiguration.

7 Related work

A multi-agent system represents a popular paradigm for modelling complex and distributed systems. An overview of the literature on multi-agent systems (MAS) [28] reveals a significant amount of research devoted to different agent organisation concepts, agent specification languages and platforms, modelling and verification of the agent behaviour, etc. Various methodologies and tools have been proposed for design, development and verification of MAS: AUML [4], Gaia [26], MaSE [8], ADELFE [5], Tropos [7], etc. However these approaches are limited to provide rigorous reasoning about agent behaviour as well as agent interactions. In our work we attempt to formally model each individual agent as well as the dynamic behaviour of the overall system. Moreover, employed Event-B modelling method was capable of rigorously describing all the essential aspects of collaborative behaviour in MAS.

Similar to our work, the authors in [10] propose a set of general principles from which MAS may be designed (in particular, for capturing the organisational structure of MAS). However, our formalisation covers a more wide range of aspects of MAS and agent behaviour (agents capabilities, statuses, relationships, interactions and collaborative activities).

The work [12] presents the cooperative motion and task planning scheme for MAS. The presented approach is applicable to MAS where the agents have independently assigned local tasks. In contrast, in our work we consider cooperative agent behaviour, where an agent might take responsibility for a specific task or participate in a collaboration depending on its available capabilities.

Reconfiguration in MAS is studied also in work [22], where a framework for development, verification and execution of MAS is presented. In this work, the reconfiguration is triggered as soon as real-time requirements are not satisfied (e.g., a certain deadline for task accomplishment is expired). In contrast, in our approach, reconfiguration is triggered as soon as changes in system and its environment violate safety issues associated with a system behaviour or prevent a system from achieving its goals.

System adaptation based on the assume-guarantee concept has been studied in work [14]. Inverardi et al. propose a framework that allows the developers to efficiently define under which conditions adaptation can be performed by still preserving the desired system invariant properties. The framework also allows the designers to split

the system into parts that can be substituted. The special conditions are formulated and has to be proven at run-time to guaranteeing the correctness of adaptation. In our work, the reconfiguration mechanisms are already defined at development phase and are incorporated into the system architecture. And, in the case of failures or changes, the system is able to reconfigure by changing interdependencies among agents, as well as between agents and goals.

The work [29] introduce a meta-model of MAS that aims at defining the key concepts and interdependencies between them that should be addressed by a formal model. The authors also demonstrate how such a meta-model can facilitate construction of a formal model in Z. However, in our work the used refinement technique and associated automated verification tool support of Event-B provide us with a more scalable basis for constructing complex and detailed system specifications.

Fault tolerant aspects of MAS in Event-B have been undertaken by Ball and Butler in [3]. They present a number of informally described patterns that allow the developers to design fault tolerance mechanisms into formal models. In our approach fault tolerance mechanism becomes a part of actions for ensuring resilience of MAS. Moreover, we have formalised a more advanced fault tolerance scheme that relies on agent dynamic reconfiguration to guarantee system resilience and goals achievement.

In this work we focused on providing the logical reasoning of the relationships between agents and their interactions. However, we still have abstracted away from some features that could be interesting to study in the future. As a possible future direction, it would be interesting to combine the presented approach with the resilient-explicit goal-oriented refinement process that we proposed in [17]. In this work, the goal-oriented framework provided us with a suitable basis for reasoning about reconfigurability. Combined view would allow us to define reconfigurability as an ability of agents to redistribute their responsibilities via correct interactions and collaborations to ensure goal reachability. The resulting formal systematisation can be used then as generic guidelines for formal development of reconfigurable systems.

Funding Open access funding provided by Abo Akademi University (ABO).

References

1. Abrial J-R (2010) Modeling in Event-B. Cambridge University Press, Cambridge
2. Abrial J-R, Butler MJ, Hallerstede S, Hoang TS, Mehta F, Voisin L (2010) Rodin: an open toolset for modelling and reasoning in Event-B. STTT 12(6):447–466. https://doi.org/10.1007/s10009-010-0145-y

3. Ball E, Butler M (2009) Patterns Event-B, for specifying fault-tolerance in multi-agent interaction, methods, models and tools for fault tolerance. Springer, pp 104–129. https://doi.org/10.1007/978-3-642-00867-2_6
4. Bauer B, Müller JP, Odell J (2001) Agent UML: a formalism for specifying multiagent software systems. Int J Softw Eng Knowl Eng 11(3):207–230. https://doi.org/10.1142/S0218194001000517
5. Bernon C, Gleizes MP, Peyruqueou S, Picard G (2002) ADELFE: a methodology for adaptive multi-agent systems engineering, ESAW. Lecture Notes in Computer Science, 2577. Springer, pp 156–169. https://doi.org/10.1007/3-540-39173-8_12
6. Brambilla M, Ferrante E, Birattari M, Dorigo M (2013) Swarm robotics: a review from the swarm engineering perspective. Swarm Intell 7(1):1–41. https://doi.org/10.1007/s11721-012-0075-2
7. Bresciani P, Perini A, Giorgini P, Giunchiglia F, Mylopoulos J (2004) Tropos: an agent-oriented software development methodology. Autonom Agents Multi-Agent Syst J 8(3):203–236. https://doi.org/10.1023/B:AGNT.0000018806.20944.ef
8. DeLoach SA (2005) Multiagent systems engineering of organization-based multiagent systems. ACM SIGSOFT Softw Eng Notes 30(4):1–7. https://doi.org/10.1145/1082983.1082967
9. Ferber J (1999) Multi-agent systems: an introduction to distributed artificial intelligence, 1st edn. Addison-Wesley Longman Publishing Co. Inc, Boston
10. Ferber J, Gutknecht O, Michel F (2003) From agents to organizations: an organizational view of multi-agent systems, AOSE 2003. Lecture Notes in Computer Science, vol. 2935, Springer, pp 214–230. https://doi.org/10.1007/978-3-540-24620-6_15
11. Fisher M, Dennis LA, Webster MP (2013) Verifying autonomous systems. Commun ACM 56(9):84–93. https://doi.org/10.1145/2494558
12. Guo M, Dimarogonas DV (2015) Multi-agent plan reconfiguration under local LTL specifications, I. J Robot Res 34(2):218–235. https://doi.org/10.1177/0278364914546174
13. Huebscher MC, McCann JA (2008) A survey of autonomic computing–degrees, models, and applications. ACM Comput Surv 40(3):1
14. Inverardi P, Pelliccione P, Tivoli M (2009) Towards an assume-guarantee theory for adaptable systems. In: ICSE workshop on soft-ware engineering for adaptive and self-managing systems, SEAMS 2009. IEEE, pp 106–115
15. Jennings NR (2000) On agent-based software engineering. Artif Intell 117(2):277–296
16. Kephart JO, Chess DM (2003) The vision of autonomic computing. IEEE Comput 36(1):41–50
17. Laibinis L, Pereverzeva I, Troubitsyna E (2017) Formal reasoning about resilient goal-oriented multi-agent systems. Sci Comput Program 148:66–87. https://doi.org/10.1016/j.scico.2017.05.008
18. van Lamsweerde A (2001) Goal-oriented requirements engineering: a guided tour, RE'01. IEEE Computer Society, pp 249–263
19. Laprie JC (2005) Resilience for the scalability of dependability. In: Fourth IEEE International Symposium on Network Computing and Applications, pp 5–6. https://doi.org/10.1109/NCA.2005.44
20. Iocchi L, Nardi D, Salerno M (2000) Reactivity and deliberation: a survey on multi-robot systems, ECAI 2000. Lecture Notes in Computer Science, vol. 2103, Springer, 9–34. https://doi.org/10.1007/3-540-44568-4_2
21. Luckcuck M, Farrell M, Dennis LA, Dixon C, Fisher M (2018) Formal specification and verification of autonomous robotic systems: a survey. CoRR vol. abs/1807.00048. arXiv: 1807.00048
22. Moscato F, Venticinque S, Aversa R, Di Martino B (2008) Modeling formal, verification of real-time multi-agent systems: the REMM framework, IDC. Studies in computational intelligence, vol. 162, Springer, pp 187–196. https://doi.org/10.1007/978-3-540-85257-5_19
23. OMG Mobile Agents Facility (MASIF). http://www.omg.org
24. Pereverzeva I (2015) Formal development of resilient distributed systems, Ph.D. thesis No.203, Turku Centre for Computer Science. http://urn.fi/URN:ISBN:978-952-12-3253-4
25. Tarasyuk A, Laibinis L, Troubitsyna E (2015) Integrating stochastic reasoning into Event-B development. Formal ASP Comput 27(1):53–77. https://doi.org/10.1007/s00165-014-0305-z
26. Zambonelli F, Jennings NR, Wooldridge MJ (2003) Developing multiagent systems: the Gaia methodology. ACM Trans Softw Eng Methodol 12(3):317–370. https://doi.org/10.1145/958961.958963
27. Vistbakka I, Troubitsyna E, Majd A (2019) Multi-layered safety architecture of autonomous systems: formalising coordination perspective. In: 19th IEEE international symposium on high assurance systems engineering. HASE 58–65. https://doi.org/10.1109/HASE.2019.00019

28. Weyns D, Iftikhar MU, de la Iglesia DG Ahmad T (2015) A survey of formal methods in self-adaptive systems. In: Proceedings of the 5th international C* conference on computer science and software engineering, C3S2E 12. ACM, 2012, p 6779
29. Weyns D, Malek S, Andersson J (2012) FORMS: unifying reference model for formal specification of distributed self-adaptive systems. ACM Trans Auton Adapt Syst 7(1):8:1–8:61