Deriving Mode Logic for Autonomous Resilient Systems

Inna Vistbakka[1](✉), Amin Majd[1], and Elena Troubitsyna[1,2]

[1] Åbo Akademi University, Turku, Finland
{inna.vistbakka,amin.majd}@abo.fi
[2] KTH, Stockholm, Sweden
elenatro@kth.se

Abstract. Ensuring system resilience – dependability in presence of changes – is a complex engineering task. To achieve resilience, a system should not only autonomously cope with non-deterministically changing internal state and external operating conditions but also proactively reconfigure to maintain efficiency. To facilitate structuring and verifying such complex system behavior, in this paper, we demonstrate how to derive resilience-enhancing mode transition logic from the goals that the system should achieve. Our approach is formalised in Event-B that allows us to reason about resilience mechanisms at different architectural levels. We illustrate the proposed approach by an example – safe and efficient navigation of a swarm of drones.

1 Introduction

Resilience [6] is an ability of a system to deliver its services in a trustworthy way despite changes. Often resilience is reasoned about using the notion of *goals* – functional and non-functional objectives that a system should achieve [5]. Resilience can be seen as an ability of a system to reach its functional goals or maintain a required level of satisfaction of non-functional goals (e.g., efficiency).

A resilient system should autonomously, i.e., without a human intervention, recognise the changes, evaluate their impact on reachability and degree of satisfaction of goals and adapt. The adaptation process, either triggered by failures of system components or external changes, usually requires complex component coordination and system reconfiguration. Due to highly non-deterministic nature of the system and a large number of components (especially in such autonomous systems as swarms of drones), ensuring correctness of component interactions and the overall system resilience is a challenging and error-prone task.

In this paper, we propose an approach to a formal development of resilient autonomous systems. Our approach allows a developer to derive a resilience-enhancing mode logic in a structured disciplined way. We use *modes* [7] as a main mechanism to structure system behaviour. The goals, which the system should fulfil, serve as a basis for defining the mode transition logic. We formally define reachability conditions for functional goals and degree of satisfaction of non-functional ones. Changes in complying to these conditions trigger mode transitions.

https://doi.org/10.1007/978-3-030-02450-5_19

We consider distributed autonomous systems that are composed of asynchronously communicating heterogeneous components – agents. Each agent has certain capabilities. Our goal reachability and degree of satisfaction conditions are defined as corresponding functions over the agent capabilities.

Since mode transitions, in general, incur complex agent coordination and system reconfiguration, we need a formal structured approach to ensure correctness of mode transition logic. In this paper, we rely on Event-B [1] – a state-based approach to correct-by-construction system development to specify and verify mode logic. We propose a specification pattern for modelling mode transitions triggered by changes in reachability and degree of satisfaction conditions.

The main development technique of Event-B – refinement – supports stepwise construction and verification of complex specifications and allows us to iteratively use the proposed pattern at different architectural levels. In the refinement process, a high-level abstract specification is incrementally augmented to unfold the entire multi-layered architecture and coordination between the components at different levels of architectural hierarchy. The approach is illustrated by an example – development of a resilient swarm of drones. Abstraction, refinement and proofs as well as automated tool support allow us to scale the formal development to such complex autonomous systems.

2 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 to be preserved are defined as model invariants. A machine usually has the accompanying component, called context. A context may include user-defined carrier sets, constants and their properties (defined as model axioms).

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

$$\mathsf{e} \mathrel{\widehat{=}} \textbf{any} \;\; a \;\; \textbf{where} \;\; G_e \; \textbf{then} \; R_e \;\; \textbf{end},$$

where e is the event's name, a is the list of local variables, and (the event *guard*) G_e is 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.

Event-B employs a top-down refinement-based approach to system development. 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 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. The Rodin platform [17] provides tool support for modelling and verification. In particular, it automatically generates all required proof obligations and attempts to discharge them. When the proof obligations cannot be discharged automatically, the user can attempt to prove them interactively using a collection of available proof tactics.

3 Resilience-Enhancing Mode Transition Logic

To achieve resilience, an autonomous system should be able to adapt to nondeterministically changing internal state and external operating conditions. In our work, we study *reconfigurability* as an essential mechanism of achieving resilience of autonomous distributed systems. Since the collaborative aspect of the component behaviour is important for our study, we adopt the agent-based approach, i.e., we consider the system components as agents and the overall system as a multi-agent system [10], correspondingly.

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 [5]. 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 or maintain a required level of satisfaction of non-functional goals.

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 to ensure goal reachability or contribute to goal satisfaction. Next we discuss how notions of goals and agents can be used to reason about behaviour of an autonomous resilient system.

3.1 Reasoning About Resilience-Enhancing Mode Transitions

We assume that there is a number of main (global) goals defined for the system. Let $G = \{G_1, G_2, \dots, G_n\}$ 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 can be independent and might even be seen as conflicting.

The system consists of a number of agents (components, in general). Let $A = \{a_1, a_2, , \dots, a_m\}$ be a set of system agents. To contribute to goal achievement, the agents have to utilise their capabilities. Let $C = \{c_1, c_2, , \dots, c_k\}$ be a set of all agent capabilities. Then, for each agent, we can define the set of its capabilities as a structure AC – *agent capabilities* – with the following property:

$$\forall\, a_i : a_i \in A \Rightarrow AC(a_i) \subseteq C.$$

Agent failures make their capabilities unavailable. In the similar way, the changes in the operating environment might prevent an agent from utilising its capabilities. Thus agent capabilities AC is a dynamic structure, i.e., during system execution a set of current agent capabilities can vary.

Based on their capabilities, the agents perform the tasks contributing to achieving the system goals. To associate such goals with the agent capabilities, we define a logical function GC – *goal reachability function* over agent capabilities:

$$GC \in T \times G \times C \rightarrow BOOL.$$

For every goal $G_i \in G$ this function determines whether or not a certain capability $c_i \in C$ is required to achieve this goal G_i.

In general, a number and types of capabilities can vary depending on system needs and overall goals. The examples of capabilities include "an ability to collect data" or "an ability to send data". As a result of agent failures or change in operational conditions, some agent capabilities might become unavailable. "Degradation" of any agent capability might also slow down or aggravate the goal achievement or goal maintenance process.

To detect any changes in overall goal achievement, we also introduce a fitness function GS – *goal satisfaction function* – that evaluates the level (degree) of the goal achievement during system functioning:

$$GS \in T \times G \times C \rightarrow REAL.$$

This function is also dynamic, i.e., its value depends on time and current available capabilities of agents that can vary during system execution.

A decrease in goal satisfaction function as well as changes of logical goal function indicate hindering achieving the desired system goals. To achieve resilience, a system should monitor its goals and reconfigure to maintain the required level of goal satisfaction. In our work we propose to use *modes* as the main mechanism for structuring the behaviour of the system [7]. Modes define coarse-grained representation of system behaviour. Changes in system states trigger a change of a mode – a mode transition. In our work, we propose to connect the states of the system agents with the goals and trigger a mode transition every time when the level of satisfaction of system goals changes. Thus the goals, which the system should fulfil, serve as a basis for defining the mode transition logic.

To achieve resilience, the system architecture should contain a monitor for detecting internal and external changes and evaluating their impact on the logical goal function or goal satisfaction function. As a result of impact evaluation, a mode transition might be triggered. We say that a mode transition is triggered whenever the following condition (*) holds:

$$\begin{aligned}(GC(t1, G_i, c_i) = TRUE \wedge GC(t2, G_i, c_i) = FALSE) \vee \\ (GS(t2, G_i, c_i) < GS(t1, G_i, c_i)), where\ t1 < t2. \qquad (*)\end{aligned}$$

Naturally, the condition (*) serves as a condition on a mode transition: when a logical goal condition on the required capability for a goal has been broken or a degree of goal satisfaction lowered from the previous monitored cycle, mode transition is triggered.

In case of a logical goal condition violation (first part of (*)), a transition to the nominal mode, will be triggered as soon as the logical goal condition on capability will be re-established. In its turn, when the goal satisfaction function again reaches the necessary (desired) level the transition back, to the nominal mode, will be triggered.

As discussed earlier, we consider reconfiguration to be the essential mechanism of achieving resilience of autonomous systems. It is triggered by the corresponding mode transition. The reconfiguration is based on reallocation of responsibilities between agents to ensure that the healthy (i.e., operational) agents can either substitute the failed ones or be utilised more efficiently to partially cover up for them. Obviously, reconfiguration requires a sophisticated agent coordination. To reason about correctness of agent coordination, we propose an Event-B specification pattern for modelling mode transitions triggered by changes in goal reachability and degree of satisfaction conditions.

3.2 Modelling Mode Transitions in Event-B

To derive a mode-structured coordination scheme for an autonomous resilient system, we rely on formal modelling in Event-B. We represent the introduced above notions and definitions in terms of the corresponding Event-B elements. Then we derive a generic specification pattern that can be used to model resilience mechanisms at different levels of abstraction.

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 reasoning include the set of all possible goals, agents and capabilities (G, A and C, respectively) as well as different static structures defining various interdependencies between elements. The latter include (initial) values for agent capabilities, logical goal function on capabilities and goal satisfaction function (AC_init, GC_init and GS_init, correspondingly). We introduce static notions as sets and constants of a model context and define their properties as a number of context axioms. The corresponding context is presented in Fig. 1.

Context ARSystem_cnt
Sets $G, A, C, MODES, \ldots$
Constants $AC\_init, GC\_init, GS\_init, \ldots$
Axioms
$\ldots$
axm4: $G \neq \varnothing$
axm5: $A \neq \varnothing$
axm6: $C \neq \varnothing$
axm7: $AC\_init \in A \rightarrow \mathbb{P}(C)$
axm8: $GC\_init \in G \times C \rightarrow BOOL$
$\ldots$

Fig. 1. A generic structure of the specification pattern: context part

```
Machine ARSystem_Abs Sees ARSystem_cnt
Variables $mode, AC, GC, GS\_prev, GS, status, \ldots$
Invariants $mode \in MODES \wedge AC \in A \rightarrow \mathbb{P}(C) \wedge GC \in G \times C \rightarrow \text{BOOL} \wedge \ldots$
Events ...
AgentFailure $\hat{=}$                                // agent failure detection
  any $a_i, c_i$
  where $a_i \in A \wedge c_i \in AC(a_i) \wedge \ldots$
  then $AC(a_i) := AC(a_i) \setminus \{c_i\} \mid\mid GC(g_i, c_i) := FALSE$
  end
ModeTransition $\hat{=}$                              // transition to a generic reconfiguration mode
  any $a_i, c_i, g_i$
  where $mode{=}NOM \wedge$
      $(GC\_init(g_i, c_i) = TRUE \wedge GC(g_i, c_i) = FALSE) \vee GS(g_i, c_i) < (GS\_prev(g_i, c_i)) \vee \ldots$
  then $mode{:=}RECONF$
  end
RestoreCapability $\hat{=}$                           // scheme of reconfiguration
  any $a_i, g_i, c_i$
  where $mode{=}RECONF \wedge (GC(g_i, c_i) = FALSE \wedge GC\_init(g_i, c_i) = TRUE)\ldots$
  then $AC(a_i) := AC(a_i) \cup \{c_i\} \mid\mid GC(g_i, c_i) := TRUE$
  end
NominalModeTransition $\hat{=}$                       // transition back to the nominal mode
  any $g_i, c_i$
  where $mode{=}RECONF \wedge (GC\_init(g_i, c_i) = GC(g_i, c_i) \wedge GS\_prev(g_i, c_i) \leq GS(g_i, c_i))\ldots$
  then $mode{:=}NOM$
  end
```

Fig. 2. A generic structure of the specification pattern: machine part

The system dynamics is modelled by the events in the machine of the Event-B specification. The related notions – logical goal function and goal satisfaction function, the mode transition conditions, mode transitions, agent failures etc. – are represented as model variables, invariants, predicate expressions, or specific events. GC and GS can be represented as the system variables whose values might be changed during system functioning modelled as an execution of events. The general structure of the abstract Event-B specification is shown in Fig. 2.

To model possible agent failure and, as a consequence, the loss of some agent capability, we define an event AgentFailure. This event models non-deterministic failure of a_i agent. As a result of an event execution, a capability c_i will be lost. When the monitored component detects such a change as violation of logical goal function, it triggers a dedicated mode transition. This behaviour is specified by an event ModeTranstion. Here, in the event guard, we formulate a condition on the event to fire (this condition is based on the logical expression (*) with small modifications). We check that the capability c_i, required to accomplish a goal g_i, is not available any more (or, in general, the level of fitness function has been decreased). We store the current value for goal satisfaction function in GS variable, while its previous value in the variable GS_prev. Then RestoreCapability and NominalModeTransition events model a simple case of agent reconfiguration (as a restoring of the lost capability) and a transition back to the Nominal mode.

Let us note that in this specification pattern we consider a simple case of reconfigurability – when an agent is able to restore its capability by itself (e.g., restoring communication after a transient communication failure). In more complex cases (as we will discuss in Sects. 4 and 5), reconfiguration can be based on agent cooperation, and might involve changes in relationships between agents.

The presented design Event-B pattern only reflects the main concepts of the goal-based mode transition logic and represents generic modelling solutions that can be reused in the development of resilient autonomous systems. In the next section we demonstrate how to derive mode transition logic using the proposed approach for a swarm of drones. Further, in Sect. 5, we present its Event-B development relying on the generic specification pattern described above.

4 Autonomous Swarm-Based System

The swarms of drones are increasingly used for surveillance, shipping, rescue etc. A swarm is a group of drones that, in a coordinated manner, executes a mission. For instance, a mission can be "video surveillance of a certain area". A video surveillance mission can be represented by a (generic) goal:

G1**:** *Periodically send the images covering certain sectors of the monitored area.*

For a swarm of drones, we can identify the following generic subgoals contributing to achieving the overall goal ***G1***:

G2**:** *Produce the payload data (e.g., images) with the required quality level.*

G3**:** *Guarantee survivability of drones allowing them to complete the mission.*

To achieve ***G3***, we have to ensure that the following subgoals are satisfied:

G4**:** *The drones do not prematurely deplete their batteries, i.e., they are navigated in an efficient way.*

G5**:** *The drones do not collide with each other and static obstacles.*

G6**:** *The drones do not collide with the unforeseen dynamically appearing objects.*

The goals are interdependent and might even be seen as conflicting, e.g., the travel distance has to be increased to guarantee safety and produce the payload data of the required quality. Hence, the controlling software should rely on sophisticated coordination mechanisms to ensure that all the goals remain satisfiable thought the mission execution.

The system architecture is presented in Fig. 3. The decision center (DC) – is an intelligent component which is responsible for generating the efficient navigation strategies according to the mission goals and preventing unsafe behaviour, i.e., it navigates the drones to avoid collisions with each other and static obstacles. DC runs high-performance machine learning and evolutionary algorithms proposed in our previous work [8,9]. They allow us to safely navigate the drones and optimise travel distance, resource consumption and quality of payload data ratio. The algorithms ensure inter-drone and drone-obstacle collision avoidance.

At each cycle DC receives the payload (e.g., imaging) and telemetry data from the swarm and processes this information and if required, generates a new routing for the swarm. The information obtained from the Dynamic Monitoring component allows DC to detect the changes in the drone swarm and in the flying zone. Such changes may invoke swarm reconfiguration and regeneration of the drone routes.

The Navigation Centre (NC) communicates with the drones by sending them the flying plan received from DC. In their turn, the drones periodically send their

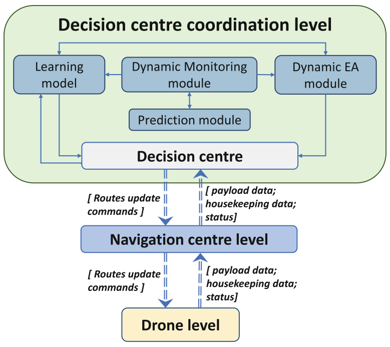
Decision centre coordination level
Learning model
Dynamic Monitoring module
Dynamic EA module
Prediction module
Decision centre
[Routes update commands]
[payload data; housekeeping data; status]
Navigation centre level
[Routes update commands]
[payload data; housekeeping data; status]
Drone level

Fig. 3. Overview of a system architecture

payload and telemetry data (current status, position, battery level, etc.) to NC, which packages them, (sometimes) preprocesses and forwards to DC.

Drones communicate with NC and each other in order to achieve their individual and common goals. Since communication with NC is typically long range, it consumes significant energy. To alleviate the problem of fast energy depletion, the swarm of the drones can be organised hierarchically and form a tree-structure depending on its different capabilities: more powerful drones – *the leaders* and less powerful drones – *the slaves* – that communicate with their leaders using less power consuming means. Moreover, we distinguish a *sink* drone – a dedicated leader drone – what besides area monitoring tasks transmits data between NC and drones at the leader level. The drones of the leader level send data to the sink. Each leader has a number of slave drones and periodically gathers information from its corresponding slaves. Finally, drones of the slave level exchange information with their leaders and receive new commands. Since some drones might change their predefined routes or even fail, to maintain an efficient drone configuration, at each cycle DC assesses the current state of the swarm and might reconfigure the tree.

Moreover, each drone (at any level) has its own local collision avoidance mechanism – *drone reflexes computation module* – a module that overrides the goals received from DC and commands a drone to move away when a camera or radar of a drone detects an obstacle. When a drone detects a possible collision with an unforeseen obstacle, the reflexes computation module quickly computes a reflex movement for a drone to prevent or mitigate the collision.

The top-most layer – DC – is responsible for achieving goals ***G3***–***G5***, i.e., it controls the swarm to ensure quality, efficiency and implement *preventive* safety. The on-board drone software is responsible for satisfying goal ***G2*** and ***G6***, in the latter case implementing *defensive* safety.

Next we discuss the coordination of drones and their collaborative behaviour as well as the resilience aspect of controlling the swarm of drones.

4.1 Mode Transition Logic for a Swarm of Drones

Before deriving a mode transition logic for the discussed swarm of drones using the approach presented in Sect. 3, let us now describe the capabilities of drones of the different levels:

- The drones of the *slave level* have the capabilities to:
    - collect data from the assigned sectors of the monitored area;
    - send the collected data and house keeping data to the next drone level.
- The drones of the *leader level* have capabilities to:
    - collect data from the assigned sectors of the monitored area;
    - aggregate data received from the slave drone level;
    - send all collected data to the sink level.
- Finally, the *sink* drone has capabilities to:
    - collect data from the assigned sectors of the monitored area;
    - aggregate data received from the drones of the leader level;
    - send all collected data to NC.

Such capabilities allow a drone of any layer to achieve its goals and contribute to the overall goal achievement and maintenance. However, failures of the drones, communication loss as well as changes in the operating environment affect the level of satisfaction of the system goals as discussed in Sect. 3.

In nominal situation (called *Nominal* mode), the drones fly according to the plan issued by DC. Upon receiving new commands from DC the drones change their current routes and perform reconfiguration if it is commanded by DC. In this case, reconfiguration means that logical relationships between the drones (i.e., *sink-leader* and *leader-slave* relationships) might be changed according to a new update of a drone tree structure recalculated by DC.

Next we will analyse the factors affecting the goal satisfaction and define the corresponding mode logic that allows the system to achieve the overall system goals despite failures and deviations.

Appearing an Unpredictable Obstacle. Unpredictable obstacles appearing in a drone flying zone might prevent a drone from achieving the goal ***G6***. Thus, when a drone detects a possible collision with an unforeseen obstacle, the monitoring component evaluates the goal satisfaction function and issues a transition to the *Reflection Activation* mode. The drone reflexes computation module quickly computes a reflex movement for a drone to prevent and mitigate the collision. After the collision is avoided, the goal satisfaction function is recalculated and a transition to the *Nominal* mode is triggered.

Local Communication Failure. Each drone has capabilities to identify its local communication failure. Communication failure might prevent a drone from achieving the goal ***G2***. When a drone detects such a failure, the goal satisfaction function is recalculated and a transition to the *Local Communication Failure* mode is triggered. Upon this transition, every drone should move to reconnect with NC and reunite with a swarm. This is a self-triggered mode transition, i.e., the drones perform it independently upon detection of a failure. When a drone re-establish connection with a swarm, satisfaction function will be recalculated and a transition to the *Nominal* mode is triggered.

Slave Failure. A slave failure prevents a drone from achieving the goal ***G2***. Upon detection a slave failure (by the corresponding leader drone), the satisfaction function is recalculated and the *Slave Failure* mode is triggered. This is a local leader-triggered mode transition meaning that it does not affect other drones. The leader drone tries to re-establish connection with the failed slave drone within the time bound period and, in case of unsuccessful outcome considers this slave as failed. Further, the health status of every slave will be transmitted to the sink drone and finally will reach DC. Let us note, that if the failed slave was a candidate for the next leader then the new candidate is recalculated.

Leader Failure. In case of a leader failure (that affects achieving ***G2***), detectable by the sink drone, the sink should trigger the *Leader Failure* mode transition. The corresponding reconfiguration procedure is performed to substitute the failed leader by the predefined slave of the failed leader.

Sink Failure. NC is able to identify the health status of the sink drone. In case of a sink failure, the satisfaction function will be recalculated and NC triggers a transition to the *Sink Failure* mode. A sink failure can have severe consequences and might prevent a system from achieving all ***G1–G6*** goals. The reconfiguration is triggered to substitute the failed sink by the predefined leader. In this case, NC retransmits the DC commands to the "new" sink. Moreover, if the leader drones detect a sink failure before NC does, all healthy leaders should issue the commands to its corresponding slaves to slow down the flying speed.

Despite the small number of modes, the mode logic is complex due to the highly non-deterministic nature of the conditions triggering mode transitions. Ensuring correctness of coordinated behaviour of a collaborative swarm of drones is a challenging engineering task. To approach it in a systematic rigorous way, we rely on Event-B and its main development technique – refinement. In the next Sect. 5, we will demonstrate how to derive and verify properties of the multi-layered drone coordination in a structured rigorous way.

5 Formal Development of a Resilient Swarm of Drones

In this section, we outline the formal development of the coordinated mode logic for the discussed swarm of drones in Event-B. The full development can be found in [21]. We start from specifying the high-level general requirements and unfold the entire coordination logic in the refinement process.

Abstract Model. The initial model represents the global control cycle spanning over all layers of the architecture shown in Fig. 4. At each cycle, DC analyses the telemetry data and either maintains the previously calculated routing or generate a new one. The routing commands are transmitted from DC to NC and then from NC to the sink. Next, the sink broadcasts the received information to all the drones at the leader level. In its turn, upon receiving commands from the sink each leader further distributes the commands to its corresponding slaves.

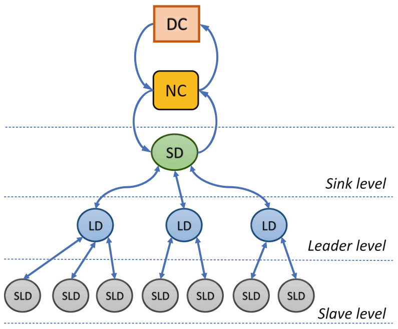

Fig. 4. System layered architecture

Once per cycle, the collected information about the monitored area and housekeeping data (e.g., battery level) are sent by slaves to their corresponding leaders. When all the required information is gathered by the leaders, they transmit data to the sink. Then the sink drone sends this information to NC and, NC forwards it to DC. DC analyses the received data and, if it is needed, issues the new commands to the drones as well as triggers the drone reconfiguration.

Next we refine the abstract model to represent the coordination required to model mode transitions in all possible nominal and off-nominal situations and the corresponding data flow.

Introducing Drones and Drone Failures. In our first refinement, we introduce a representation of the behaviour of the system components, in particular, we augment the specification by representation of drones and their failures. We model the impact of such failures on the system dynamics and resilience. In this case, reconfiguration would involve changing the relationships between drones (at every layer) in order to optimise routing, coverage, energy and safety ratio.

We distinguish the permanent drone failure (e.g., due to a physical drone damage) and transient drone failure (e.g., due to loss of communication). If a transient failure occurred then after some time a drone (of any layer) can restore the connection with the swarm and continue to function. This behaviour is modelled by the transition to the *Local Communication Failure* mode and then returning back to the *Nominal* mode.

In the case of a permanent drone failure (of any layer), the corresponding drone of the upper layer or NC will detect this failure and, eventually, DC will be notified about the loss in the swarm. In this case, the transition to the corresponding *Sink Failure*, *Leader Failure* or *Slave Failure* mode is triggered.

In case of a leader failure, as a part of reconfiguration, some predefined slave drone associated with the failed leader will become a new leader. When the other leaders detect a failure of a leader, they send the corresponding commands to their slaves to slow down their speed of the flying, until the new commands from the DC will be issued. The scheme of the leader failure reconfiguration is presented in Fig. 5.

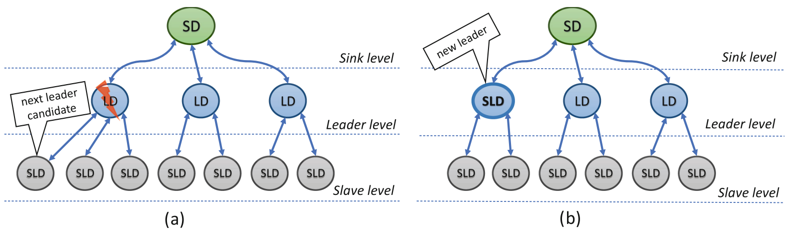

Fig. 5. Leader failure

As a result of a transition to the *Sink Failure* mode, reconfiguration of the system is also activated. Namely, the predefined leader drone becomes a new sink and the predefined slave drone replaces it by becoming a leader. The impact of the sink failure on the system architecture is represented in Fig. 6.

To model the behaviour described above, we refine our initial model by introducing a number of new variables, events and refining some abstract events. In particular, we define variables to specify the set of all drones, leaders and slaves and the sink drone (by corresponding variables *drones*, *leaders*, *slaves*, *sink*):

$$\{sink\} \cup leaders \cup slaves = drones, \;\; drones \subseteq SWARM.$$

Here the swarm is represented by a finite non-empty set of drones $SWARM$. It can be seen as a set that contains the ids of all drones in the swarm.

The new variable *slaves_of_leaders* established the relationship between a leader and slaves it supervises:

$$slaves\_of\_leaders \in leaders \rightarrow \mathbb{P}(slaves).$$

To model the health state of the drones, we introduce a variable *status*. It is defined as a function:

$$status \in drones \rightarrow STATUSES,$$

where $STATUSES$ is a set consisting of the constants OK, $FAILED$ and $DISCON$ representing correspondingly the nominal, failed and disconnected

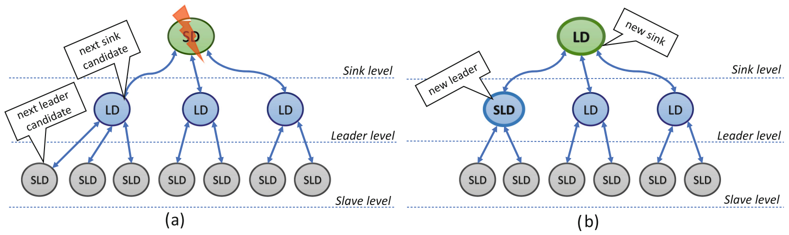

Fig. 6. Sink failure

drone status. In the system implementation, the decision about the drone status is made on the basis of the analysis of the currently received telemetry data and the routing plan. Let us note that we intentionally introduce statuses instead of modelling drones capabilities. Such an abstraction of drone statuses allows us to avoid introducing all drones capabilities of the different layers and to have only three states covering all the cases that might effect goal achievement.

Performing this refinement step we apply our modelling pattern proposed in Sect. 3. A number of new events are introduce to model possible drones failures as well as system reaction on them. We introduce SINK_Failure, SINK_discon, LEADER_Failure, etc. Upon execution of these events, the value of *status* variable is changed. As soon as a leader failure is detected, as modelled by the new event LEADER_FailureMode, then the "new" leader should be chosen from one of its slave drones (modelled by LEADER_Failure_Reconfiguration event). In this case, *slaves_of_leaders* as well as *leaders*, *slaves* and *leader_alt* variables are updated. Similarly, new events are introduced to model a slave and sink failure as well as events modelling reconfigurations and transitions back to the Nominal mode. An excerpt from the first refinement step is presented in Fig. 7.

At this refinement step we formulate and prove the correctness of coordinated reconfiguration involving all the layers of the architecture. For instance, we prove that no slaves become dispatched from some leader:

$$\forall\, sl.\ sl \in slaves \ \Rightarrow\ (\exists\, ld.\ ld \in leaders \ \wedge\ sl \in slaves\_of\_leaders(ld)).$$

Multi-level Drone Communication. The goal of our second refinement is to introduce a communication model between the sink and NC as well as between the drones. Next we discuss a simple communication scheme that can be instantiated to implement communication between the drones at any level.

Lets consider *Sink-Leader* communication. At every cycle, the sink initiates the communication with a leader. The sink checks status of a leader and if it is *OK*, then the sink sends the data via the inter-drone communication link. Upon delivery of the message, a leader updates its route commands and sends the acknowledgement to the sink. In its turn, the sink waits for the acknowledgement from a leader. Upon receiving the acknowledgement, the sink considers the data transition to be successfully completed. If no acknowledgement is received, the sink triggers the transition to *Leader Failure* mode.

The communication between the leaders and their slaves as well as between NC and the sink can be implemented in the similar way.

Data Flow Modelling and Introducing Reflexes Mechanisms. In the further refinement steps, we model data flow between all system components at the different layers and also specify the local drone safety reflex mechanisms.

The goal of the mission is to produce the payload data. As a part of the mission, the drones periodically send the collected data to DC. Upon receiving these data, DC makes a decision to recalculate the current route commands or restructure drone tree-structure. To reflect the required data flow, we introduce a number of events and variables and refine our model.

```
Machine SwarmOfDrones_m1 refines SwarmOfDrones_m0 Sees SwarmOfDrones_c1
Variables $phase, mode, drones, sink, leaders, slaves, status, slaves\_of\_leaders, sink\_alt, \ldots$
Invariants $phase \subseteq PHASES \wedge mode \subseteq MODES \wedge drones \subseteq SWARM \wedge leaders \subseteq drones \wedge$
           $slaves \subseteq drones \wedge sink \in drones \wedge sink\_alt \in drones \wedge$
           $slaves\_of\_leaders \in leaders \rightarrow \mathbb{P}(slaves) \wedge$
           $\forall sl.\ sl \in slaves \Rightarrow (\exists ld.\ ld \in leaders \wedge sl \in slaves\_of\_leaders(ld)) \wedge$
           $status \in drones \rightarrow STATUSES \wedge \ldots$
Events ...
SINK_Failure_Reconfiguration $\widehat{=}$
 any $ld\_alt, new\_ld\_alt, sls$
 where $\ldots \wedge mode = SINK\_FAILURE\_RECONF \wedge status(sink) = FAILED \wedge$
       $status(sink\_alt) = OK \wedge sls = slaves\_of\_leader(sink\_alt) \setminus \{ld\_alt\} \wedge$
       $ld\_alt = leader\_alt(sink\_alt) \wedge new\_ld\_alt \in sls$
 then
   $sink := sink\_alt$
   $sink\_alt := ld\_alt$
   $leaders := (leaders \setminus \{sink\_alt\}) \cup \{ld\_alt\}$
   $slaves := slaves \setminus \{ld\_alt\}$
   $leader\_alt(ld\_alt) := new\_ld\_alt$
   $slaves\_of\_leader := (\{sink\_alt\} \lhd slaves\_of\_leader) \cup \{ld\_alt \mapsto sls\}$
 end
LEADER_Failure_Reconfiguration $\widehat{=}$
  any $ld, ld\_alt, new\_ld\_alt, sls$
  where $\ldots \wedge mode = LEADER\_FAILURE\_RECONF \wedge ld \in leaders \wedge$
        $sls = slaves\_of\_leader(ld) \setminus \{ld\_alt\} \wedge ld\_alt = leader\_alt(ld) \wedge$
        $new\_ld\_alt \in sls \wedge status(ld\_alt) = OK$
  then
   $slaves\_of\_leader := (\{ld\} \lhd slaves\_of\_leader) \cup \{ld\_alt \mapsto sls\}$
   $leader\_alt(ld\_alt) := new\_ld\_alt$
  end
...
```

Fig. 7. The machine SwarmOfDrones_m1

Moreover, for each drone, we model possibility to react on particular hazardous situations – an unexpected appearance of an obstacle in the drone flying zone. In our proposed approach, when a drone detects a possible collision with an unforeseen obstacle, the drone safety reflex computation module quickly computes a reflex movement for the drone to prevent the collision.

To model drone safety reflex mechanisms, first we model possibility of appearing an obstacle in a drone flying zone (at any level of hierarchy). Then, upon detection an obstacle, a drone triggers mode transition to the *Reflection Activation* mode. Let us note that the drones perform this transition autonomously and independently upon detection of an obstacle. Upon triggering a transition to this "local" mode, a drone computes the best safe position and moves there. The nominal mode is restored after DC receives the update about the current drone positions and calculates the routing for the swarm. We introduce the new events Unpredictable_Obstacle and Reflection_Activation and refine the number of old events, e.g., Update_Local_Routes (omitted due to the lack of space).

6 Related Work and Conclusions

During last decades the problem of resilience and motion safety of autonomous robotic systems attracts significant research attention. A comprehensive overview of the problems associated with autonomous mobile robots is given in [18]. The analysis carried out in [20], shows that the most prominent routing schemes do not guarantee motion safety. Our approach resolves this issue and ensures not only safety but also efficiency of routing.

A layered architectural solution for robot navigation has been proposed in [3]. The authors focus on a problem of safe navigation of a vehicle in an urban environment. Similarly to our approach, they distinguish between a global route planning and a collision avoidance control. However, in their work, they focus on the safety issues associated with the navigation of a single vehicle and do not consider the problem of route optimization that is especially acute in the context of swarms of robots.

Modelling and verification of a system architecture using Event-B in the context of multi-agent and multi-robotic systems has also been investigated in works [12–14]. Moreover, in [15] we verified by proofs correctness and safety of agent interactions. In [4] the interactions between agents have been studied using goal-oriented perspective. In this work, the roles were defined as agent capabilities to perform certain tasks in order to accomplish the entire mission.

In this paper, we have presented a novel approach to formal modelling of resilient autonomous systems. Our approach allows a designer to derive the resilience-enhancing mode logic from the goals that the system should fulfil. We have considered both functional and non-functional goals and demonstrated how to define the conditions for monitoring goal reachability or degree of goal satisfaction. Using multi-agent modelling paradigm, we have demonstrated how to define such monitoring conditions as the functions over the capabilities of the system component – agents. Furthermore, we have proposed a generic Event-B specification pattern for modelling mode transitions triggered by changes in the monitored conditions at different architectural layers and demonstrated how to derive the complex mode-transition logic by refinement. The approach was illustrated by a case study – deriving mode logic of a resilient swarm of drones.

Our formal development was greatly facilitated by the Rodin platform. Reliance of refinement, proofs and powerful tool support has allowed us to derive a specification of a complex distributed system in a systematic rigorous way. The proposed technique is not constrained by the number of the architectural layers or of system components. Hence, it can potentially scale to the development of realistic autonomous systems.

In the future work, we are planing to extend our approach and focus on its communication model. Indeed, communication is a critical aspect in ensuring correct coordination and safety of the autonomous swarms of drones. To extend the communication model we can rely on our approach discussed in [19].

During the presented in this work refinement process we arrived at a centralised specification of the multi-layered swarm-based system. Our next goal can also focus on deriving its distributed implementation by refinement. We can employ modularisation facilities of Event-B [2,16] to achieve this. We can further decompose a system-level model and derive the interfaces of the drones and guarantee that their communication supports correct coordination despite unreliability of the communication channel and drones failures. To achieve it our current work can be complemented with our approaches proposed in [11,19].

References

1. Abrial, J.R.: Modeling in Event-B. Cambridge University Press, Cambridge (2010)
2. Iliasov, A., et al.: Supporting reuse in Event B development: modularisation approach. In: Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (eds.) ABZ 2010. LNCS, vol. 5977, pp. 174–188. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11811-1_14
3. Macek, K., Govea, D.A.V., Fraichard, T., Siegwart, R.: Safe vehicle navigation in dynamic urban scenarios. In: Proceedings of 11th International IEEE Conference on Intelligent Transportation Systems, pp. 482–489. IEEE (2008)
4. Laibinis, L., Pereverzeva, I., Troubitsyna, E.: Formal reasoning about resilient goal-oriented multi-agent systems. Sci. Comput. Program. **148**, 66–87 (2017)
5. van Lamsweerde, A.: Goal-oriented requirements engineering: a guided tour. In: RE 2001, pp. 249–263. IEEE Computer Society (2001)
6. Laprie, J.: From dependability to resilience. In: 38th IEEE/IFIP International Conference on Dependable Systems and Networks, pp. G8–G9 (2008)
7. Leveson, N., Pinnel, L.D., Sandys, S.D., Koga, S., Reese, J.D.: Analyzing software specifications for mode confusion potential. In: Human Error and System Development, pp. 132–146 (1997)
8. Majd, A., Ashraf, A., Troubitsyna, E., Daneshtalab, M.: Integrating learning, optimization, and prediction for efficient navigation of swarms of drones. In: PDP 2018. IEEE (2018)
9. Majd, A., Troubitsyna, E.: Integrating safety-aware route optimisation and runtime safety monitoring in controlling swarms of drones. In: ISSRE Workshops, pp. 94–95. IEEE Computer Society (2017)
10. OMG Mobile Agents Facility (MASIF). www.omg.org
11. Pereverzeva, I., Troubitsyna, E.: Formalizing goal-oriented development of resilient cyber-physical systems. In: Alexander Romanovsky, F.I. (ed.) Trustworthy Cyber-Physical Systems Engineering, chap. 6 (2017)
12. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: A case study in formal development of a fault tolerant multi-robotic system. In: Avgeriou, P. (ed.) SERENE 2012. LNCS, vol. 7527, pp. 16–31. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33176-3_2
13. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: Formal development of critical multi-agent systems: a refinement approach. In: EDCC 2012, pp. 156–161. IEEE Computer Society (2012)
14. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: Formal goal-oriented development of resilient MAS in Event-B. In: Brorsson, M., Pinho, L.M. (eds.) Ada-Europe 2012. LNCS, vol. 7308, pp. 147–161. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30598-6_11
15. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: A refinement-based approach to developing critical multi-agent systems. IJCCBS **4**(1), 69–91 (2013)
16. Rodin: Modularisation Plug-in. http://wiki.event-b.org/index.php/Modularisation_Plug-in
17. Rodin: Event-B platform. http://www.event-b.org/

18. Siegwart, R., Nourbakhsh, I.R.: Introduction to Autonomous Mobile Robots. MIT Press, Cambridge (2004)
19. Tarasyuk, A., Pereverzeva, I., Troubitsyna, E., Latvala, T.: The formal derivation of mode logic for autonomous satellite flight formation. In: Koornneef, F., van Gulijk, C. (eds.) SAFECOMP 2015. LNCS, vol. 9337, pp. 29–43. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24255-2_4
20. Fraichard, Th.: A short paper about motion safety. In: Proceedings of the IEEE International Conference on Robotics and Automation. IEEE (2007)
21. Vistbakka, I., Majd, A., Troubitsyna, E.: Autonomous resilient systems: derivation of mode logic using Event-B. Technical report 1199, Turku Centre for Computer Science (2018)