Cautious Adaptation of Defiant Components

Paulo Henrique Maia*, Lucas Vieira*, Matheus Chagas*, Yijun Yu†, Andrea Zisman†, and Bashar Nuseibeh†
*State University of Ceará, Fortaleza, CE, Brazil
pauloh.maia@uece.br, {lucas.vieira, matheus.chagas}@aluno.uece.br
†The Open University, Milton Keynes, United Kingdom
{yijun.yu, andrea.zisman, bashar.nuseibeh}@open.ac.uk

***Abstract*—Systems-of-systems are formed by the composition of independently created software components into a single system. These participating components are designed to satisfy their own requirements, and may not satisfy the overall requirements of the system-of-systems. We refer to components that cannot be adapted to meet both individual and global requirements as "defiant" components. We propose a "cautious" adaptation approach that supports changing the behaviour of such defiant components under exceptional conditions to satisfy global requirements, while continuing to guarantee the satisfaction of the components' individual requirements. The approach uses scenarios to represent normal and exceptional conditions; models the behaviour of exceptional conditions as wrappers implemented using an aspect-oriented technique; and adapts accordingly at runtime. We consider both single and multiple defiant components, with several instances of these components operating at the same time. We evaluated the implementation of our approach using an organ delivery drone application, conceived by our industrial partners, the results of which are presented and discussed in the paper.**

***Index Terms*—Defiant Component, Adaptation, Scenarios, Aspects**

I. Introduction

It is now 10:00am in busy London. The kidney transplant department in St Thomas Hospital (Hospital A) has been informed that a matching kidney of a patient in Guy's Hospital (Hospital B) is now available for its patient waiting for surgery. The organ transplant department of both hospitals contact the SOSDronePayload company and make all the necessary arrangements for the organ to be transferred by drone from Hospital B to Hospital A. Drone_DR1235 is selected by SOSDronePayload to deliver the organ through its flying corridor over the Thames River, to avoid land traffic. At a certain point during the journey the battery of Drone_DR1235 reduces dramatically and when it reaches 10%, Drone_DR1235 attempts to land before reaching its final destination (Hospital A). However, at this point, Drone_DR1235 was only 2 km away from Hospital A and, given the favourable wind conditions at the time, Drone_DR1235 could have reached Hospital A with its 10% battery capacity and delivered the critical organ.

In the scenario, *Drone_DR1235* was developed independently of the payload organ delivery application, and was not intended to change its behaviour during its execution, regardless of the application in which it is used.

Many applications are developed by the composition of independently created software components into a single system. These systems are known as systems-of-systems [1] and allow the execution of certain functionalities that cannot be achieved by individual participating components on their own. In such situations, it is necessary to support emergent behaviours that appear due to the combination of existing components into new larger systems, or even due to new requirements or context changes [2]. The participating software components have been designed to satisfy predefined requirements following predefined specifications and are not necessarily intended to change their behaviour during execution, nor to support some global requirements of the systems-of-systems applications. We refer to components that resist such changes, despite the need, as *defiant* components. Existing adaptation approaches have not considered such defiant components [3]–[7].

In adaptive approaches models are often specified based on finite-state machine formalisms such as a Labelled Transition Systems (LTS) [8] or Markov chains [9], to represent system behaviour, verify property violation, and check the correctness of the adaptation. However, dealing with these models requires significant mathematical skills from the software engineers, which may hinder their adoption by users [6]. Such behaviour models also require a complete description of the system behaviour for a fixed scope and span [10], which may not be possible for adaptive systems since not all adaptive behaviours can be modelled in advance due to unexpected requirements or environment uncertainty.

In this paper we suggest a novel approach that considers the existence of defiant components. More specifically, we present a *cautious adaptation* approach in which changes in the behaviour of the components are accommodated, in order to satisfy global requirements in exceptional circumstances. The adaptation is *cautious* because it guarantees that changes will not interfere with the original functionality of the participating components during their normal use, and will only be triggered in exceptional conditions. For example, in the above scenario, the drone is considered a *defiant* component since it has been developed with a specification in which it should make a safe landing when its battery reaches 10%. However, from the perspective of the payload organ delivery application, it should be possible to force the drone to continue flying, given the exceptional and favourable wind conditions, which will allow the drone to complete its journey.

Our approach relies on the use of *wrappers*, implemented using an aspect-oriented programming technique [11], to support exceptional conditions that are formalised with behavioural semantics using LTS [12]. In our approach, exceptional conditions are represented by join-points of scenario-based aspect weaving [13], given that they can handle exceptions. The use of scenarios to identify exceptional conditions has been demonstrated successfully in the literature [14] [15], and used to support behavioural model checking [10]. The high abstraction level provided by scenarios facilitates identification and understanding of requirements and involvement of different stakeholders in an application. The use of scenarios provides a framework for analysing future events of an application and helps with decision-making.

Aspect-oriented programming provides wrappers to allow for the introduction of changes into defiant components without requiring consent from the designer of the components. It avoids the need to redesign a component to satisfy emergent behaviours, as in the case when using dependency injection techniques [16] or plugin-based approaches [17]. On the other hand, the use of aspect-oriented programming techniques can be risky, since changes introduced by the use of aspects may violate the original (local) requirements of the components. Our work uses aspect-oriented techniques in a way that guarantees the original requirements of the components under normal execution conditions.

The remainder of the paper is structured as follows. Section II describes an overview and formalisation of the approach to support cautious adaptation of defiant components. Section III presents the detailed process of the cautious adaptation approach. Section IV describes the implementation and the evaluation of the work in a payload organ delivery scenario. Section V discusses related work. Section VI concludes the paper and discuses future work.

II. Approach Overview

Figure 1 presents an overview of the cautious adaptation approach. As shown in the figure, the process is divided into three phases, namely: (i) *defiant component identification*, (ii) *wrapper design and implementation*, and (iii) *runtime cautious adaptation*. The first two phases occur at design time, while the last phase occurs at run time.

The *identification of defiant components* (phase 1) is supported by the use of scenarios to model the system and to formally verify defiant behaviour of components. Scenarios can be used to represent both exceptional and normal conditions of a system, and have been widely used for modelling *what-if* situations [18]. In general, scenarios assume the use of off-the-shelf software components, with predefined requirements and specifications. Our approach uses Message Sequence Chart [19] (MSC) specifications, a standard International Telecommunication Union (ITU) notation for describing the interaction between communicating processes, and to model both ordinary and exceptional scenarios.

The *wrapper design and implementation* (phase 2) addresses the defiant behaviour of one or more components identified in phase 1, and uses aspect-oriented programming (AOP) [11] to implement wrappers. The use of wrappers is to support changes in the behaviour of the defiant components while keeping the satisfaction of the global system requirements. Exceptional conditions identified in the scenarios in phase 1, indicate where changes in the system should occur; we call these places *joint points*. The identified join points are represented as regular expressions as *point-cuts*. In this case, a replacement of the behaviour of a defiant component after join points can be done through aspect in terms of *advices*. Aspect advices handle exceptions by either invoking existing functionalities of the defiant components or introducing new functionalities and conditions to be executed. The approach uses precedence order among wrappers when more than one wrapper exists for the same exceptional condition.

In the *runtime cautious adaptation* (phase 3), the system is *weaved* with wrappers and the respective functionalities are executed. This phase is based on the MAPE-K loop approach [20]. In this phase, monitors are used to verify contextual variables that characterise the surroundings of the system based on data from the environment (e.g., wind condition) or from sensors (e.g., Global Positioning Systems, accelerometer, battery level). The approach analyses the exceptional conditions defined in phase 2. When exceptional conditions are satisfied, the wrapper is invoked and the exceptional functionalities, as implemented by the aspect, are executed.

The general problem being tackled can be formalised using the semantics of Problem Frames [21], as follows.

At the design time of a component (c), given its world context (W_c), its specification (S_c) must satisfy its requirements (R_c), i.e., $W_c, S_c \models R_c$. Suppose that component c is being used as part of a system-of-systems (s). Based on the analysis of exceptional scenarios concerning a specific context of the system (W_s), which satisfies the condition $W_s \implies W_c$, component c cannot satisfy global requirements of s. In other words, $W_s, S_s \not\models R_s$, where S_s contains S_c.

To verify that the wrapper executes its purpose, we formalise three conditions to be checked as follows:

$$\begin{array}{ll} W_s, S_s \not\models R_s & \text{(defiant component identification)} \\ W_s, S_s|_{c \to w(c)} \models R_s & \text{(defiant behaviour removal)} \\ W_c \setminus W_s, S_{w(c)} \models R_c & \text{(safety assurance)} \end{array}$$

- Defiant component identification checks that component's configurations do not satisfy global requirements.
- Defiant behaviour removal checks that after wrapping up the component with new behaviour, the global requirement can be restored in exceptional conditions.
- Safety assurance checks that after wrapping up the defiant component with new behaviour, the local requirements are still satisfied in normal conditions.

The introduction of a *wrapper* $w(c)$ changes the defiant behaviour of the component c, i.e. $W_s, S_{w(c)} \not\models R_c$, retaining the essential satisfaction of global requirements when c is substituted with $w(c)$: $W_s, S'_s \models R_s$ where $S'_s = S_s|_{c \to w(c)}$. Furthermore, the adaptation is *cautious* because other than the exceptional condition W_s, the wrapped component should behave like the original designed component, i.e., $(W_c \setminus W_s), S_{w(c)} \models R_c$.

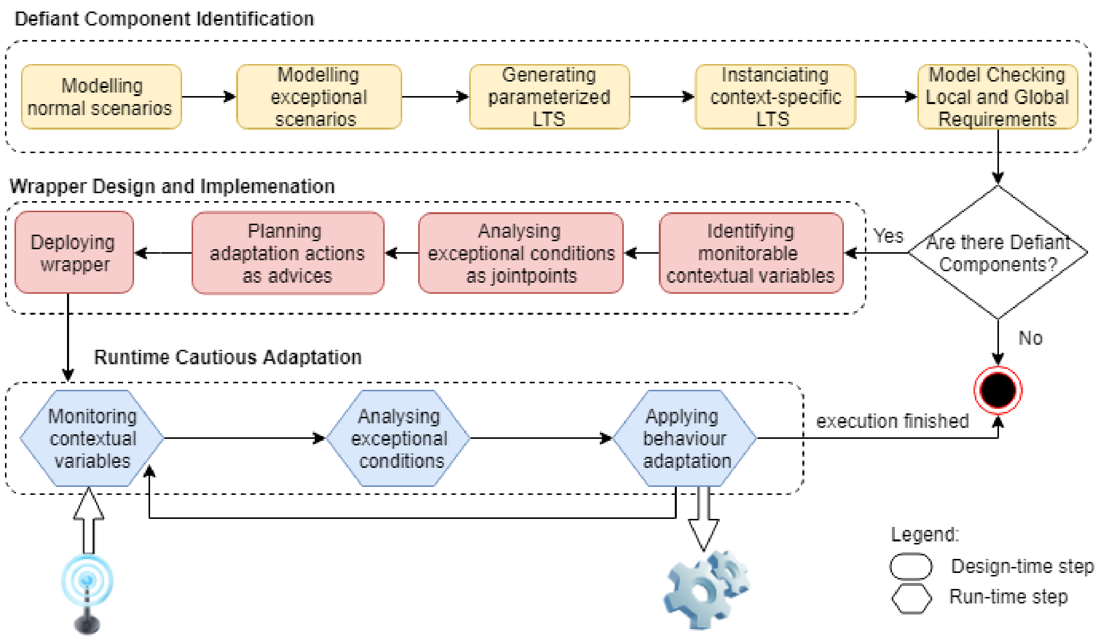

Defiant Component Identification
Modelling normal scenarios
Modelling exceptional scenarios
Generating parameterized LTS
Instanciating context-specific LTS
Model Checking Local and Global Requirements
Wrapper Design and Implemenation
Are there Defiant Components?
Yes
No
Identifying monitorable contextual variables
Analysing exceptional conditions as jointpoints
Planning adaptation actions as advices
Deploying wrapper
Runtime Cautious Adaptation
Monitoring contextual variables
Analysing exceptional conditions
Applying behaviour adaptation
execution finished
Legend:
Design-time step
Run-time step

Fig. 1. Overview of the cautious adaptation approach in a workflow diagram

III. Cautious Adaptation Approach: Detailed Process

In this section we describe each phase of the approach depicted in Figure 1 in details. We use the payload organ delivery example presented in Section I to illustrate the approach. Suppose two global requirements for the system as:

- GR1: the drone must take the payload organ from the sender to the receiver hospital.
- GR2: in the case in which it is not possible to deliver the payload, the drone should not lose it (e.g., lost of the payload by landing on the water).

A. Defiant component identification

As shown in Figure 1, the defiant component identification phase is composed of five steps. The first two steps consist of *modelling normal and exceptional scenarios* of a system-of-systems. The approach advocates the use of scenarios to model core functional requirements of the system-of-systems and local requirements for each component participating in the system. Scenarios are described using Message Sequence Chart specification through two main syntactic components: *basic MSC* (bMSC) and *high MSC* (hMSC). A bMSC represents a set of asynchronous processes (or instances) with their exchanged messages. A hMSC consists of a graph of a set of connected bMSCs representing parallel, sequential, iterating, and non-deterministic executions. In a hMSC graph the nodes refer to bMSCs and the edges represent execution sequences.

Figure 2 shows the hMSC specification for the payload organ delivery scenario. Each box in the hMSC corresponds to one bMSC, while the arrows represent the execution flow. The guards in the bMSC transitions indicate the conditions that have to be satisfied in order to execute the next functionalities

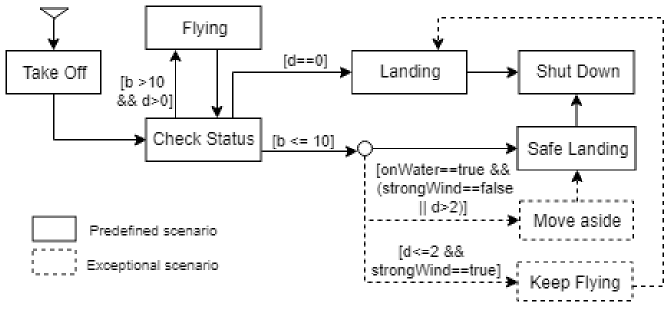


Fig. 2. hMSCs for the drone scenario

represented in a bMSC. Figure 3 shows part of the scenarios of the bMSCs in Figure 2 for the organ payload delivery example.

In Figure 2, the predefined scenarios are represented by a full rectangle and are part of the normal specification of a drone. We assume that, with a controller, the pilot can control the drone to take off (bMSC *Take Off*). During a flight, a drone periodically checks the status (bMSC *Check Status*) of its internal devices and sensors such as battery level and distance from the destination, represented by b and d, respectively, in the guard conditions over the transitions. In the example, if the battery level is above the expected threshold $\theta = 10\%$, and the drone is not yet in its destination, the pilot can keep manoeuvring the drone (bMSC *Flying*). In the scenario, when a drone arrives at its destination, the pilot sends a landing command to the drone (bMSC *Landing*), that is acknowledged when the drone lands on the ground, and after it shuts down (bMSC *Shut Down*). If the battery is below the expected threshold, the drone performs a safe landing (bMSC *Safe Landing*) in accordance to its specification, and shuts down. The above scenario corresponds to the original behaviour of a drone, and represents its local requirements (LRs).

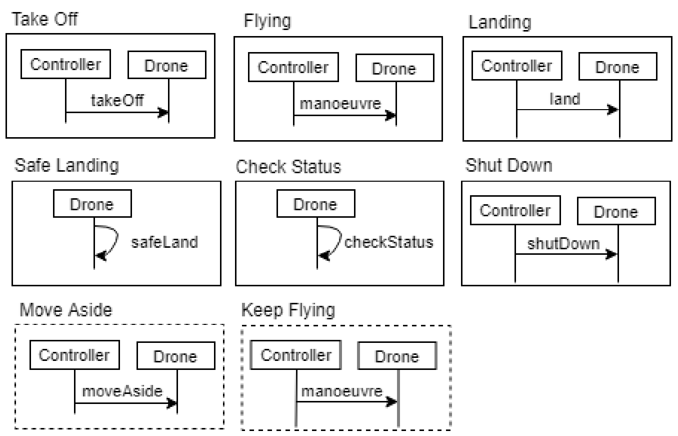


Fig. 3. bMSCs for the drone payload delivery scenario

In order to *model exceptional scenarios*, the MSC specifications are amended with *interception points*. These interception points represent points in which conditions are analysed in order to decide whether the original expected scenario is executed or an introduced exceptional scenario has to take place. Several exceptional conditions can be intercepted in the same point, indicating other possible behaviours according to alternative contexts. If there is no exceptional scenario in an interception point, the expected default scenario is performed. It may also be necessary to identify new contextual variables and conditions to be used in the transitions.

Exceptional scenarios are represented as dashed bMSCs in Figure 2. In our example, the exceptional conditions added in the interception point state that if the distance to the expected destination is less than 2km and the wind is strong (condition *strongWind==true*), then the pilot can keep the drone flying (bMSC *Keep Flying*), even in a low-level battery situation, and land afterwards. In this case, the drone is able to complete its overall goal of delivering the organ payload successfully (global requirements GR1, above). In the case in which the battery level is low, the drone is flying over the river (condition *onWater==true*), but it is more than 2km away or the wind is not strong, then the action is to move the drone aside (bMSC *Move Aside*) in order to land it safely on the ground and, therefore, satisfy the global requirement GR2.

After the normal and exceptional scenarios are modelled, the next step consists of automatically transforming the MSC specifications into a *parameterised Labelled Transition System* (pLTS). This is to allow for correctness checks of the model before and after adaptation, and to support the lack of knowledge of the ranges of values of the context variables at this stage. The conditions in the transitions in the MSC specification are represented as guards in the state transitions. The approach uses an adaptation of the algorithm to generate LTS from MSC specifications proposed in [18], with the addition of the scenario transition guards in the resulting LTS. Figure 4 depicts the pLTS for the example of Figure 2.

After the parameterized LTS is generated, the values of the context variables can be instantiated by the developer, and

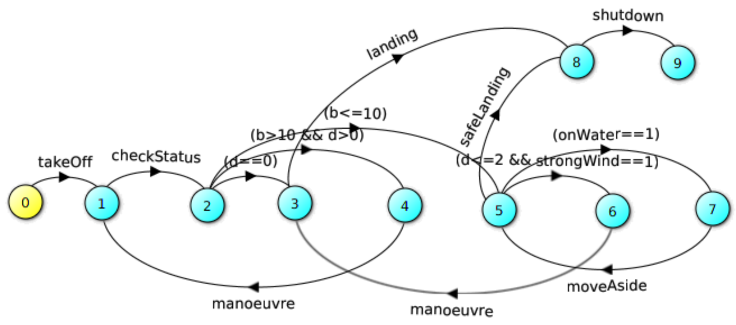


Fig. 4. Parameterised LTS illustrates the behaviours when the contextual variables are parameters bound to user specified constants.

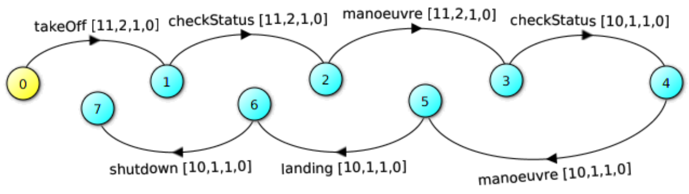


Fig. 5. LTS of Landing: the partial behaviour of landing bMSC, where the unnamed transitions are the transitions between different bMSCs. Though an ϵ-reduction, these transitions will be removed.

a *context-specific LTS* representing the system behavior for those values is generated. Figure 5 illustrates the specific LTS generated when the developer sets the variables *b = 12%*, *d==2*, *strongWind==true*, and *onWater==true*. The variable values, in that order, are shown as parameters in the transitions in Figure 5. We assume that, after each drone movement, both battery and distance are decreased by one unit. Steps one to four above were implemented as extensions of Lotus [6].

The context-specific LTS can be exported to LTSA [8] in order to check both local and global requirements specified in terms of properties. A *model checker* is applied to verify whether the participating components can satisfy global requirements of the system-of-systems, for which not necessarily it was designed. In the case when the component satisfies the requirements, it can be used as-is in the system-of-systems. However, when the component cannot satisfy global requirements (i.e., the case of a defiant component), a wrapper is specified in order to weave exceptional conditions into the component specification in LTS, and it is verified by the LTSA model checker against the properties. The adaptation is considered *cautious* only when **all** properties (local and global) are satisfied by the component within the system-of-systems context. If the verification step through model checking is considered successful, the defiant behaviours can be addressed by design-time simulation and verification.

B. Wrapper Design and Implementation

The wrapper design and implementation phase is composed of four steps, as shown in Figure 1. The first step consists of *identifying monitorable context variables* for analysing global requirements. In the payload organ delivery example, the monitorable context variables are: battery level, distance to destination, position of drone in relation to water, and wind condition. These variables may not be available or used when analysing local requirements of participating components.

In the next two steps the approach maps exceptional conditions and scenarios to corresponding concepts in the aspect-oriented paradigm [11]. The main goal in the second step is to *analyse exceptional conditions* and identify when they are triggered in the MSC specification in order to associate them with joinpoints when mapping into aspect-oriented technique.

The interception points in the MSC specifications represent the point in which exceptional conditions should be analysed and they are mapped to joinpoints. For instance, considering our drone example, the only interception point is the one between the bMSCs *Check Status* and *Safe Landing*, when it is checked whether the battery level reached 10%. Subsequently, we have to define pointcuts in which the code will be intercepted. For our example, we defined that method *safeLanding()* needs to be intercepted when it is called, in order to analyse exceptional conditions.

In the third step, exceptions are addressed by *planning adaptation actions as advices* in the components. Using existing functionalities (e.g., maneuvering), while disabling certain functionalities (e.g., safe landing), it is possible to switch to a different solution through adaptation. These adaptions are implemented as *advices* in aspect-oriented programming language, with a type (before, after or around) dependant on the control flow of the exceptional scenarios.

The following code shows an excerpt of the *DroneAspect* implemented to wrap one defiant behaviour of the drone. We defined pointcut *checkExceptionalConditions()* that intercepts the call of the drone's original *safeLanding()* method (when the battery level reaches 10%).

```
1  public aspect DroneAspect {
2
3      pointcut checkExceptionalConditions():
4          call (void safeLanding());
5
6      void before(): checkExceptionalConditions() {
7          if (isOverWater()
8           && (getDistanceTargetHospital() >=2
9            || !isStrongWind()))
10             getDrone().moveAside();
11     }
12
13     void around(): checkExceptionalConditions() {
14         if (isOverWater()
15          && getDistanceTargetHospital() <=2
16          && isStrongWind())
17            getDrone().manoeuvre();
18     }
19 }
```

The above code shows two kinds of syntactic advices: *before* and *around*. The *before* clause (Line 6) is executed when the drone is over the water, and either the distance to the target hospital is no less than 2km or the wind is not strong (Lines 7-9). In this case, the drone executes a *modeAside()* method (Line 10), which moves the drone to fly over the ground and to execute the original *safeLanding()* method.

On the other hand, the *around* clause (Line 13) is executed when the drone is no more than 2km away from the destination, is flying over the water, and the wind is strong (Lines 14-16). Unlike the previous *before()* clause, new behaviour *manoeuvre()* (Line 17) replaces the *safeLanding()* method, which is no longer executed. When the drone reached the destination it performs the original *Landing()* method.

The implemented *wrappers* are deployed in the system so they can be used at runtime to adapt the behaviour of defiant components. This is done by weaving the wrapper aspects with the bytecode of the simulated system using the aspectJ compiler. If the defiant component is implemented in other programming languages, the wrapper aspects will need to be written in corresponding AOP languages [11].

C. Runtime Cautious Adaptation

The last phase encompasses the execution of the recently weaved system that, due to our wrapper solution, becomes a self-adaptive system. The steps of this phase are derived from the activities of the MAPE-K [20] control loop.

During its execution, the system keeps *monitoring contextual variables* by either checking internal resource levels or receiving environmental data provided by sensors. It may be necessary to execute some calculations in order to obtain the real value of the monitorable context variables. For instance, the value of variable distance from the target (d) is calculated by the difference between the GPS position of the target hospital and current drone location.

After reading context information, the wrapper *analyses the exception conditions* by intercepting the system execution in the defined pointcuts. In the example, this happens after the interception of *checkExceptionalConditions()* pointcut.

The last step corresponds to both Planning and Execution activities of the MAPE-K loop. When an exceptional condition is satisfied, the wrapper *applies the behaviour adaptation* by executing the exceptional scenario according to the rules implemented in the advices. The process continues to be executed while the execution of the system is not finished.

IV. Implementation and Evaluation

In order to evaluate our approach, we developed a prototype tool to simulate an extension of the payload organ delivery drone application described in Section I. We evaluated our approach for both single and multiple defiant components in a system-of-systems, with several instances of these components operating concurrently. In this section, we present the evaluation setup with its research questions, describe the different configurations used in the evaluation, present the prototype tool implemented to simulate all possible scenarios, and discuss the results of the evaluation and some threats to validity.

A. Evaluation setup

We used an extension of the organ payload delivery application described in Section I and with the global requirements GR1 and GR2 described in Section III. Figure 6 shows the hMSC of the extended example used in the evaluation, including all normal and exceptional scenarios. Apart from the *safe landing* functionality described in the example in

Section III, we introduced two new functionalities in a drone, namely: *return to home* and *economy mode*.

The *return to home* functionality is executed whenever a drone bypasses a bad connection area (for instance, near communication towers) and loses connection with the pilot. This is a common safety procedure currently present in most existing drones. The exceptional scenario for this case consists of allowing the drone to *glide* while waiting for the connection to return and, therefore, maximizing the chances of delivery. As shown in Figure 6, bMSC *Return to Home* happens when the contextual variable *bc* (bad connection) is true. When the drone arrives at the depart point (contextual variable *ds* equals to 0), it lands as usual. The exceptional scenario is represented by the bMSC *Glide*, which is performed when a bad connection is detected, the distance from the target (contextual variable *dt*) is less than the distance from the source hospital, and the battery level is greater than 10%.

For the *economy mode* functionality some internal resources of a drone (e.g., sensors and cameras) can be switched off in order to save battery when the battery level reaches a certain level (e.g., 15%) and the drone is within the visual line-of-sight (VLOS) of the pilot (e.g, distance from the source hospital, where the pilot is supposed to be, is up to 500m according to CAP 722 [22]). However, in this case a pilot can lose control of the drone, which may put the organ payload at risk. The exceptional scenario for this case consists of always flying the drone in *normal mode* to avoid the drone to be sent to an incorrect direction and, therefore, not landing at the target hospital. As shown in Figure 6, bMSC *Economy Mode* is performed when the battery reaches 15% and the distance from the source hospital is less than 0.5km. Its correspondent exceptional scenario is represented by bMSC *Normal Mode*, which indicates that the drone keeps its normal behaviour with all internal resources on.

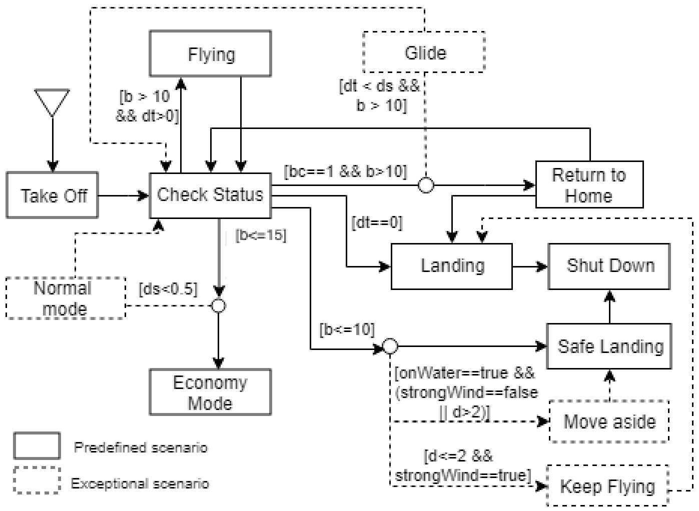


Fig. 6. Simulation scenario

We evaluated the above scenarios in three distinct configurations, representing different numbers of functionalities in the drones, that can cause defiant situations in the organ payload delivery scenarios due to global requirements GR1 and GR2. The first configuration consists of drones with the safe landing functionality and its exceptional condition. This configuration is the same one used throughout Section III to explain the cautious adaption approach. The second configuration consists of drones with both safe landing and return to home functionalities, and their exceptional conditions. The third configuration consists of drones with the three functionalities, namely: safe landing, return to home, and economy mode, and their respective exceptional conditions.

In the evaluation we were interested in the following questions:

- RQ1: can the cautious adaptation approach be used to achieve the global requirements of the system-of-systems?
- RQ2: how does the approach behave when the number of drones increases?
- RQ3: can the approach support different numbers of concurrent defiant components?

B. Implementation

We built a prototype with a graphical user interface that simulates the drone scenario used in this evaluation. In the simulator, the user can configure one or several drones, determine the source and target hospital locations, draw a river flow between the hospitals, and set the context variables (e.g, initial battery level, battery consumption rate, wind strength, drone above water). The user can manoeuvre the drones either manually or using an automatic pilot control, in which the drones try to reach the destination by executing a minimal distance routing algorithm. A snapshot of the simulator is depicted in Figure 7.

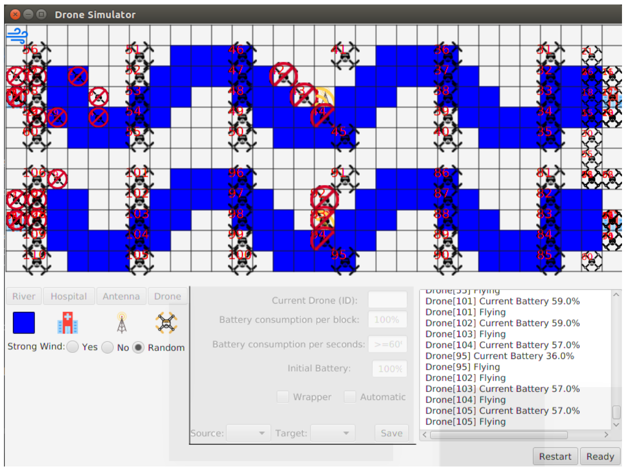
In the simulator, it is possible to track each drone by checking a panel that shows the traces outputted by the drones in terms of performed scenarios, which are in accordance with the ones modelled in the hMSC (in this case, the one depicted in Figure 6). The panel also shows the updates of the battery level periodically.

The prototype is an open source tool available at GitHub [1] that can be used by the community to add new scenarios. It is also possible to change the wrapper implementation by a customized one and reuse the same environment. The simulator also offers an API so users can create programs to configure the environment, and plan and execute the journey of one or a fleet of drones.

C. Experiments and Results

For each configuration, we used the API of the simulator to run three experiments for different numbers of drones. We ran the experiments with groups of 100 (group G1), 250 (group G2), and 400 (group G3) drones, flying at the same time. For each group of drones, half of the drones had their original functionalities, while the other half consisted of drones with wrappers. In all configurations and groups, the drones

[1]https://github.com/cautiousadaptation/Drone-Route-Simulator



Strong Wind: Yes No [✓ Random]

Wrapper Automatic

```
Drone[101] Current Battery 59.0%
Drone[101] Flying
Drone[102] Current Battery 59.0%
Drone[103] Flying
Drone[104] Current Battery 57.0%
Drone[95] Current Battery 36.0%
Drone[95] Flying
Drone[102] Flying
Drone[103] Current Battery 57.0%
Drone[104] Flying
Drone[105] Current Battery 57.0%
Drone[105] Flying
```

Fig. 7. Screenshot of the simulation prototype

where simulated as automated pilot setting, with no human interference.

In each experiment, we used two identical environments to run the drones with and without wrappers (original drones), so that each group of drones can go through the same conditions during the simulation. To make the experiment more realistic, we chose different values for the initial battery levels of the drones. In the same group of drones, some drones will have enough battery to arrive at the destination, while others will only be able to arrive with the help of the wrapper. We randomly selected initial battery values of the drones from an interval between 66% and 70%, since for any value above (below) this range, we observed that all (none of the) drones would have arrived at the destination.

Configuration 1: safe landing. Figure 8 shows the results of the experiments. In both approaches (with original and wrapped drones), the number of drones that had enough battery to reach the destination successfully (shown in column "Landed at Destination Normally", i.e. they arrived in the target hospital with a battery level greater than 10%) was the same: 12 (24%), 33 (26,4%) and 40 (20%) for groups G1, G2 and G3, respectively. This data represent the number of drones that satisfied global requirement GR1.

We observed that 38 drones of group G1 did not have enough battery to complete their mission and had to follow a safe landing procedure. From them, 19 (38% of the total) landed on the ground and 19 (38% of the total) on the water. The former group comprises the drones that satisfy global requirement GR2, while the latter group represents the situation that both drones and payloads are lost. Considering groups G2 and G3, 32% and 37% of the drones satisfied GR2, while 41,6% and 43% of the drones were lost, respectively.

On the other hand, with our cautious adaptation approach (using wrappers), out of the 38 drones of group G1 that were supposed to perform a safe landing, 18 drones were able to perform the "Keep Flying" scenario (they were less than 2km away and used the favourable wind condition to reach the target hospital). We configured the wind to be strong with a chance of 50% at each second. Therefore, the number of drones that satisfied GR1 increased to 30 (i.e. 60%).

In addition, nine out of those 38 drones moved aside and landed on the ground due to our approach. Those drones, along with eleven other drones that landed on the ground, because they did not manage to reach an exceptional scenario, add up to 20 drones (i.e. 40%), and satisfied GR2. None of the drones landed on water and did not cause loss of the payload.

In group G2, 51% of drones satisfied GR1, while 41.6% of the drones were lost. Similarly, in group G3, 55% of the drones satisfied GR1, while 43% of the drones were lost. Figure11 presents the results of the satisfaction of global requirements GR1 and GR2, and for landing the drones on water (column LW), for all three configurations.

Configuration 2: safe landing and return to home. In this configuration, the bad connection area is represented by a communication tower with a probability of 30% of bad communication[2]. Figure 9 shows the experiment results.

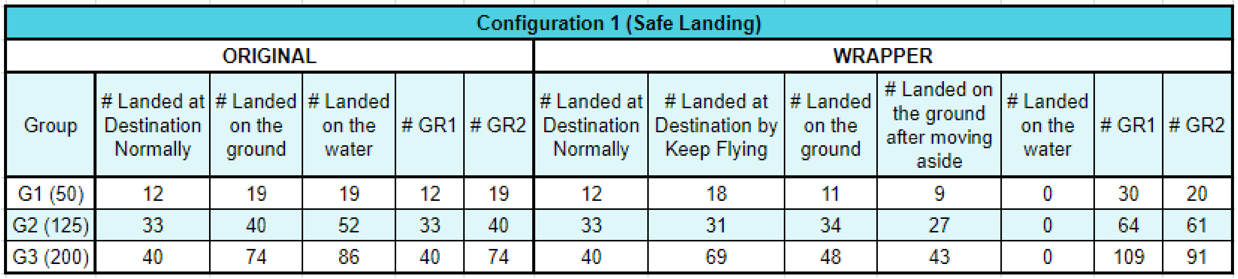
In the case of drones without wrappers (original drones) in group G1, only one drone managed to arrive at the destination normally, while 23 drones landed on the ground (either during the ongoing flight or during the return to home journey), and five drones returned to home. Moreover, as shown in Figure 11, only one drone (2%) satisfied GR1, 28 drones (56%) satisfied GR2, and 21 drones (42%) landed on water. In group G2, 4.8%, 61.6% and 33.6% of the drones satisfied GR1, GR2, and got lost, respectively. For the drones in group G3, the values for satisfaction of GR1 and GR2, and for getting lost were 3.5%, 54%, 42.5%, respectively.

Considering drones with wrappers, although the number of drones that arrived normally at the destination was the same as their non-wrapped counterparts, the cautious adaptation approach helped ten drones accomplish their mission: seven drones performed a "keep flying" after a "safe landing"; one drone glided for a while, but continued the journey and reached the destination; and two drones glided and performed a safe landing after a keep flying scenario. In this way, the number of drones that satisfied GR1 increased to 11 (22%). In addition, thirteen drones were able to land on the ground because they performed the functionality "move aside". Without the approach, those drone would have landed on water. In total, 39 drones (78%) satisfied GR2 (landed on ground or returned to home) and no drone landed on water.

As shown in as shown in Figure 11, in group G2, 29.6% and 70.4% of the drones satisfied GR1 and GR2, respectively; while in group G3 27.5% and 72.5% of the drones satisfied GR1 and GR2, respectively. For this configuration, in both groups G1 and G2, none of the drones landed on water.

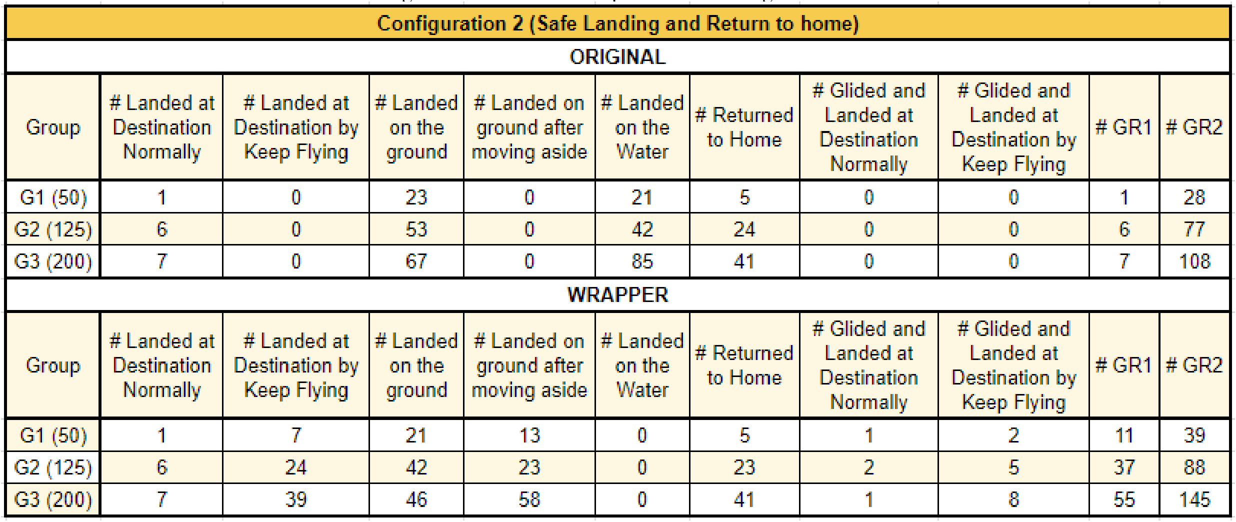
Configuration 3: safe landing, return to home, and economy mode. In this configuration, we simulated the effect of the economy mode functionality (e.g., camera off) by increasing the path to reach the hospital destination. In this case, the pilot

[2]This value can be changed for other simulations.



Configuration 1 (Safe Landing)												
ORIGINAL						WRAPPER						
Group	# Landed at Destination Normally	# Landed on the ground	# Landed on the water	# GR1	# GR2	# Landed at Destination Normally	# Landed at Destination by Keep Flying	# Landed on the ground	# Landed on the ground after moving aside	# Landed on the water	# GR1	# GR2
G1 (50)	12	19	19	12	19	12	18	11	9	0	30	20
G2 (125)	33	40	52	33	40	33	31	34	27	0	64	61
G3 (200)	40	74	86	40	74	40	69	48	43	0	109	91

Fig. 8. Results for the experiment of configuration 1



Configuration 2 (Safe Landing and Return to home)										
ORIGINAL										
Group	# Landed at Destination Normally	# Landed at Destination by Keep Flying	# Landed on the ground	# Landed on ground after moving aside	# Landed on the Water	# Returned to Home	# Glided and Landed at Destination Normally	# Glided and Landed at Destination by Keep Flying	# GR1	# GR2
G1 (50)	1	0	23	0	21	5	0	0	1	28
G2 (125)	6	0	53	0	42	24	0	0	6	77
G3 (200)	7	0	67	0	85	41	0	0	7	108
WRAPPER										
Group	# Landed at Destination Normally	# Landed at Destination by Keep Flying	# Landed on the ground	# Landed on ground after moving aside	# Landed on the Water	# Returned to Home	# Glided and Landed at Destination Normally	# Glided and Landed at Destination by Keep Flying	# GR1	# GR2
G1 (50)	1	7	21	13	0	5	1	2	11	39
G2 (125)	6	24	42	23	0	23	2	5	37	88
G3 (200)	7	39	46	58	0	41	1	8	55	145

Fig. 9. Results for the experiment of configuration 2

cannot see the route of the drone and, therefore, the drone can fly for a bit longer taking other paths.

Figure 10 shows the results of the experiments. In group G1 for drones without wrappers (original), none of the 50 drones arrived normally at the destination, which means that 0% satisfied GR1; while 32 drones (64%) satisfied GR2, and 18 drones (36%) landed on water. For drones with wrappers in group G1, three drones arrived at the destination. While analysing the traces of the corresponding drones without wrappers, we observed that, as these drones enter the economy mode, they went through different paths. Due to that their battery reached 10% and they had to do a safe landing. On the other hand, the wrapped drones that kept the economy mode off all the time, managed to reach the destination with a battery level greater than 10%. In addition, seven drones arrived at the destination by performing the exceptional scenario "keep flying". The cautious adaptation approach helped to increase the number of drones that accomplished their mission, with 10 drones (20%) satisfying GR1. With respect to GR2, 40 drones (80%) either landed on ground or returned to home. Out of these 40 drones, thirteen drones avoided landing on the water by moving aside due to the use of wrappers. Similar to the other configurations, no drone was lost.

For drones in groups G2 and G3, the number of wrapped drones that arrived at the destination was higher than the number of their original counterparts. A complete comparison is shown in Figure 11.

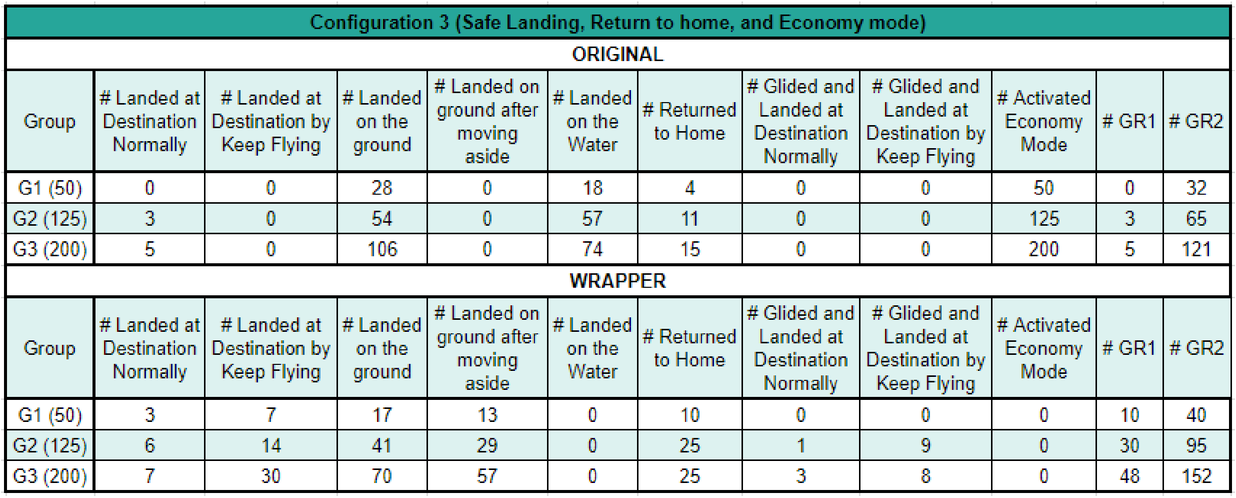
D. Discussion

We discuss below the results of the experiments with respect to each of the three research questions described above.

For *RQ1: can the cautious adaptation approach be used to achieve the global requirements of the system-of-systems?*, as shown in Figure 11, the cautious adaptation approach avoided drones getting lost, in all configurations and groups of drones. For drones without the use of wrappers, the experiments demonstrated a rate of drones that were lost from 38% to 43% for configuration 1; 33,6% to 42,5%, for configuration 2; and 36% to 45,6% for configuration 3.

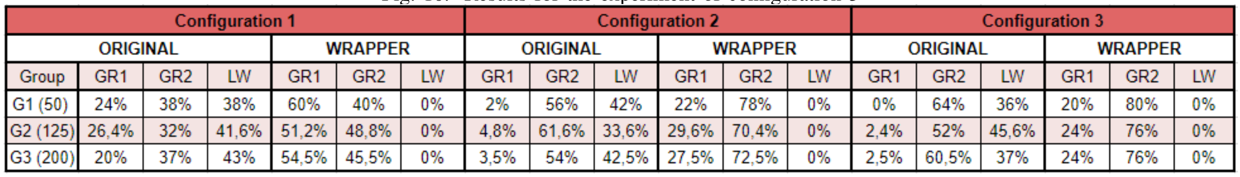
More specifically, when comparing the success rate of delivering the payload (GR1), we observed that the wrapper solution was much more effective than when not using wrappers. Although that rate decreases for both types of drones as the number of defiant functionalities increases, our caution adaptation approach was able to deliver the payload in at least 20% of the times.

Furthermore, we can also see that our approach improved the rate of drones that did not land on water (GR2). While the rates of drones without the wrapper that either landed on ground or returned to home varied from 32% to 38% for configuration 1; 54% to 61.6%, for configuration 2; and 52% to 64% for configuration 3, the rate of their wrapper counterparts soared to 40% to 48.5% for configuration 1; 70.4% to 78%, for configuration 2; and 76% to 80% for configuration 3.



Configuration 3 (Safe Landing, Return to home, and Economy mode)											
ORIGINAL											
Group	# Landed at Destination Normally	# Landed at Destination by Keep Flying	# Landed on the ground	# Landed on ground after moving aside	# Landed on the Water	# Returned to Home	# Glided and Landed at Destination Normally	# Glided and Landed at Destination by Keep Flying	# Activated Economy Mode	# GR1	# GR2
G1 (50)	0	0	28	0	18	4	0	0	50	0	32
G2 (125)	3	0	54	0	57	11	0	0	125	3	65
G3 (200)	5	0	106	0	74	15	0	0	200	5	121
WRAPPER											
Group	# Landed at Destination Normally	# Landed at Destination by Keep Flying	# Landed on the ground	# Landed on ground after moving aside	# Landed on the Water	# Returned to Home	# Glided and Landed at Destination Normally	# Glided and Landed at Destination by Keep Flying	# Activated Economy Mode	# GR1	# GR2
G1 (50)	3	7	17	13	0	10	0	0	0	10	40
G2 (125)	6	14	41	29	0	25	1	9	0	30	95
G3 (200)	7	30	70	57	0	25	3	8	0	48	152

Fig. 10. Results for the experiment of configuration 3



	Configuration 1						Configuration 2						Configuration 3					
	ORIGINAL			WRAPPER			ORIGINAL			WRAPPER			ORIGINAL			WRAPPER		
Group	GR1	GR2	LW	GR1	GR2	LW	GR1	GR2	LW	GR1	GR2	LW	GR1	GR2	LW	GR1	GR2	LW
G1 (50)	24%	38%	38%	60%	40%	0%	2%	56%	42%	22%	78%	0%	0%	64%	36%	20%	80%	0%
G2 (125)	26,4%	32%	41,6%	51,2%	48,8%	0%	4,8%	61,6%	33,6%	29,6%	70,4%	0%	2,4%	52%	45,6%	24%	76%	0%
G3 (200)	20%	37%	43%	54,5%	45,5%	0%	3,5%	54%	42,5%	27,5%	72,5%	0%	2,5%	60,5%	37%	24%	76%	0%

Fig. 11. Success and failure rates of satisfying the global requirements

In the case of *RQ2: how the approach behaves when the number of drones increase?*, the results in Figure 11 show that the success rate for each configuration with the cautious adaptation is similar in all groups of drones. Therefore, a large number of drones does not cause any impact in the effectiveness of the approach.

For *RQ3: can the approach support different numbers of concurrent defiant components?*, the experiments have shown that it is possible to have conflicts between the various defiant components. For instance, in configuration 2, there were situations in which a drone, during the execution of the exceptional scenario "glide" (due to a bad connection), had its battery level decreased to below 10%, which triggered scenario "safe landing". In this situation, we have prioritized the safe landing component over the gliding component since a low battery situation is more critical and, not dealing with it, can cause possible hazards to the drone and to the payload. In Figure9, column "Gliding and Landed at Destination by Keep Flying", indicates the number of occurrences that the above mentioned conflict occurred. As can be seen, with this strategy, our approach was able to avoid losing two, five and eight drones for groups G1, G2 and G3, respectively.

Another similar situation occurred in configuration 3, in which a drone was gliding and then the battery reached 15%, which would have made it enter the economy mode functionality. Using a prioritization approach, the battery condition was checked before any other exceptional condition. In this case, the drone did not switch to the economy mode (as defined in the exceptional scenario) and kept on gliding.

Threats to validity. There are three types of threats to validity in our evaluation, discussed below.

Construct validity. We simulated up to 400 drones with three kinds of defiant components concurrently to see the scalability of the approach. However, in real-life cases, the contextual variables and their combinations can be much more complex. In our future work we plan to test the cautious adaptation approach in larger case studies involving a higher number of defiant scenarios.

Internal validity. Although we have implemented the entire process from formal specifications to technical verification and simulation, the wrappers have not been deployed to drones used in real world, which would require additional approvals by legal and ethics regulatory bodies. To address this threat, we are liaising with industrial organisations (e.g., Transport for London, Metropolitan Police, St Andrews Hospital) to deploy the solution into their system-of-systems.

External validity. Defiant components by our definition are a fairly generalisable concept which may manifest as physical hardware components (like the drones in this paper), as well as cyber-physical components including human in the loop. Although aspect-oriented programming is shown directly applicable to the defiant components in this work, we are aware that other wrapper techniques may be more suitable for other types of defiant components in domains such as self-driving cars and driverless trains. We plan to investigate this in future work.

V. Related Work

The cautious adaptation approach is related to works in a number of areas, including (i) self-adaptive systems, (ii) systems-of-systems and COTS adaptation, (iii) aspect-oriented programming and dependency injection, and (iv) safety assurance of drones. We discuss some of these works as follows.

Self-Adaptive Systems. Self-adaptive systems use feedback loops typically in monitoring-analysis-planning-execution-knowledge (MAPE-K) [20] to control uncertainty and provide self-managing capabilities for high-assurance at runtime [23], [24]. The introduction of MAPE-K are based on functional and non-functional requirements previously known [25]. When not all the requirements of the components are consistent to those of the global applications, high-variability approaches have been proposed [26] so that multiple alternative plans can be hot-switched at runtime [27]. Our approach uses scenarios to identify contextual variables to monitor behaviour models, requirement properties to be analysed as verifiable properties, and results in planning/execution actions through wrappers that modify the behaviour of defiant components. In this way, the need to prepare high-variability alternatives is replaced by identifying and handling exceptional scenarios and conditions.

SoSs and COTS Adaptation: One of the problems to manage the inconsistency of systems-of-systems (SoSs) is to identify conflicts among different components [28] and resolve conflicts through the use of a utility function by the MAPE-K self-adaptive feedback loops [29]. In this sense, defiant components compared to the rest of the systems-of-systems are a source of conflicts and our cautious adaptation approach can be seen as a means to resolve conflicts by design.

When global requirements are not achievable by the components off-the-shelf (COTS), adaptation would be required [30]. Although COTS adaptation has similar motivations to our cautious adaptation approach, the main difference lies in the support of multiple defiant components within COTS. Moreover, the proposed wrappers aim to partition defiant components by the internal behaviours rather than traditional adaptation at component interfaces).

AOP and DI: Modification of legacy systems can be supported by the use of aspect-oriented programming (AOP) [31] and dependency injection (DI) [16], in particular when it is not allowed to change the design of original system intrusively. Both AOP and DI could modify the behaviour of the original programs. AOP can be applied to both source and binary code, while DI is applicable typically when the programming languages support reflection [32]. In our work, AOP was used for implementing the adaptation concept. Compared to a simple application of AOP, our approach introduces additional "caution" by checking, proactively, that both local and global requirements of the system are satisfied for normal and exceptional conditions of the environment. When the components are high-level (rather than simulated programmatically in the running example), requirement-level aspect-oriented approach may also be considered [33] so that the advices can be implemented using traditional functions.

Safety Assurance of Drones: Formal methods have been applied to provide assurance for safety requirements satisfaction for UAVs [34]. However, runtime assurance of required properties can be weakened by exceptional conditions. Scenarios have been applied to address these exceptional conditions formally. However, as we pointed out, such application of scenarios was not enforced by the runtime system due to the black box nature of legacy systems. Our approach makes the early connection between scenario-based assurance of formal properties and their practical enforcement through wrappers.

Safety requirements, among many functional and non-functional requirements, are critical for transportation systems such as drones [35]. Although several attempts exist for safety assurance, incidents of drones still happen, often leading to interference in other transport systems settings like passenger aircraft and airports. In such cases, the protection of drones can be seen as a rather narrowed view of safety with respect to a broader, and much more challenging, safety and other critical requirements for the systems-of-systems. The running example of payload organ delivery illustrates that it is not sufficient to consider just the safety cases of the drones. Self-adaptive systems-of-systems have to address global requirements.

Simulation of defiant drones for safety requirements: While a swarm of drones are difficult to be evaluated, simulation is a recognized way to assure functionalities before physical tests in sky. Drone simulations such as Dronology [36] are an established way to simulate up to 100 drones for critical safety properties such as separation of distances. Although we simulate up to 400 drones for several exceptional conditions, these conditions may be much worse-case scenarios to expose. Even more when dealing with risks that are not easy to experience in physical simulation environments, since not all adversarial conditions may be present. Our approach provides the use of "wrappers" for more conservative assurance cases.

VI. Conclusion and Future Work

In this paper we proposed a novel approach to support changes in participating components in systems-of-systems that have been designed to satisfy predefined requirements, and not necessarily intended to change their behaviour in order to support global requirements of systems-of-systems. We call these *defiant* components. We presented a *cautious adaptation* approach that supports changes in the behaviour of defiant components in order to satisfy global requirements in exceptional conditions. The cautious adaptation guarantees that changes will not interfere with the original functionalities of the participating components under normal conditions.The approach uses scenarios to represent normal and exceptional conditions, uses wrappers implemented in aspect-oriented techniques to represent behaviour of exceptional conditions, and executes runtime adaptation. We use an example of a payload organ delivery drone application to illustrate and evaluate the work.

Currently, we are extending the approach to support on-the-fly identification of new exceptional conditions due to emergent behaviours, and creation of their respective wrappers. We are also evaluating the work in other large-scale scenarios.

References

[1] M. W. Maier, "Architecting principles for system of systems," vol. 1, 01 1998.

[2] V. E. Silva Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos, "Awareness requirements for adaptive systems," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '11. New York, NY, USA: ACM, 2011, pp. 60–69. [Online]. Available: http://doi.acm.org/10.1145/1988008.1988018

[3] A. Filieri, C. Ghezzi, and G. Tamburrelli, "Run-time efficient probabilistic model checking," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 341–350. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985840

[4] R. de Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. R. Schmerl, D. Weyns, L. Baresi, N. Bencomo, Y. Brun, J. Cámara, R. Calinescu, M. B. Cohen, A. Gorla, V. Grassi, L. Grunske, P. Inverardi, J. Jézéquel, S. Malek, R. Mirandola, M. Mori, H. A. Müller, R. Rouvoy, C. M. F. Rubira, É. Rutten, M. Shaw, G. Tamburrelli, G. Tamura, N. M. Villegas, T. Vogel, and F. Zambonelli, "Software engineering for self-adaptive systems: Research challenges in the provision of assurances," in *Software Engineering for Self-Adaptive Systems III. Assurances - International Seminar, Dagstuhl Castle, Germany, December 15-19, 2013, Revised Selected and Invited Papers*, 2013, pp. 3–30.

[5] M. Szvetits and U. Zdun, "Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime," *Softw. Syst. Model.*, vol. 15, no. 1, pp. 31–69, Feb. 2016.

[6] D. M. Barbosa, R. G. de Moura Lima, P. H. M. Maia, and E. C. Junior, "[email protected]: A tool for runtime monitoring and verification of self-adaptive systems," in *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 24–30.

[7] P. Arcaini, E. Riccobene, and P. Scandurra, "Formal design and verification of self-adaptive systems with decentralized control," *ACM Trans. Auton. Adapt. Syst.*, vol. 11, no. 4, pp. 25:1–25:35, Jan. 2017.

[8] J. Magee and J. Kramer, *Concurrency: State Models and Java Programs*, 2nd ed. Wiley Publishing, 2006.

[9] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[10] S. Uchitel, D. Alrajeh, S. Ben-David, V. Braberman, M. Chechik, G. De Caso, N. D'Ippolito, D. Fischbein, D. Garbervetsky, J. Kramer, A. Russo, and G. Sibay, "Supporting incremental behaviour model elaboration," *Computer Science - Research and Development*, vol. 28, no. 4, pp. 279–293, Nov 2013. [Online]. Available: https://doi.org/10.1007/s00450-012-0233-1

[11] G. Kiczales and E. Hilsdale, "Aspect-oriented programming," *SIGSOFT Softw. Eng. Notes*, vol. 26, no. 5, pp. 313–, Sep. 2001. [Online]. Available: http://doi.acm.org/10.1145/503271.503260

[12] J. Magee and J. Kramer, *Concurrency: State Models and Java Programs*, 2nd ed. Wiley Publishing, 2006.

[13] D. Xu, I. Alsmadi, and W. Xu, "Model checking aspect-oriented design specification," in *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, vol. 1, July 2007, pp. 491–500.

[14] K. YoungGab and C. Sungdeok, "Threat scenario based security risk analysis using use case modeling in information systems," *Security and Communication Networks*, vol. 5, no. 3, pp. 293–300, 2011.

[15] X. Ban and X. Tong, "A scenario-based information security risk evaluation method," *International Journal of Security and Its Applications*, vol. 8, no. 5, pp. 21–30, 2014.

[16] M. Fowler, "Inversion of control containers and the dependency injection pattern," http://www.martinfowler.com/articles/injection.html, 2004, accessed: 2015-07-23.

[17] M. Wermelinger and Y. Yu, "Analyzing the evolution of eclipse plugins," in *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, ser. MSR '08. New York, NY, USA: ACM, 2008, pp. 133–136. [Online]. Available: http://doi.acm.org/10.1145/1370750.1370783

[18] S. Uchitel, J. Kramer, and J. Magee, "Synthesis of behavioral models from scenarios," *IEEE Trans. Softw. Eng.*, vol. 29, no. 2, pp. 99–115, Feb. 2003. [Online]. Available: https://doi.org/10.1109/TSE.2003.1178048

[19] D. Harel and P. S. Thiagarajan, "Message sequence charts," in *UML for Real: Design of Embedded Real-Time Systems*, L. Lavagno, G. Martin, and B. Selic, Eds. Boston, MA: Springer US, 2003, pp. 77–105.

[20] IBM, "An architectural blueprint for autonomic computing," IBM, Tech. Rep., Jun. 2005.

[21] M. A. Jackson, *Problem Frames - Analysing and Structuring Software Development Problems*. Pearson Education, 2000.

[22] CAA Safety and Airspace Regulation Group, "Cap 722: Unmanned aircraft system operations in uk airspace guidance," March 2015, pp. 1–165.

[23] B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, "Software engineering for self-adaptive systems," B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Software Engineering for Self-Adaptive Systems: A Research Roadmap, pp. 1–26.

[24] R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, and T. Kelly, "Engineering trustworthy self-adaptive software with dynamic assurance cases," *IEEE Trans. Software Eng.*, vol. 44, no. 11, pp. 1039–1069, 2018. [Online]. Available: https://doi.org/10.1109/TSE.2017.2738640

[25] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," in *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*, ser. RE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 125–134. [Online]. Available: http://dx.doi.org/10.1109/RE.2010.25

[26] A. Lapouchnian, Y. Yu, S. Liaskos, and J. Mylopoulos, "Requirements-driven design of autonomic application software," in *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '16. Riverton, NJ, USA: IBM Corp., 2016, pp. 23–37. [Online]. Available: http://dl.acm.org/citation.cfm?id=3049877.3049879

[27] M. Salifu, Y. Yu, and B. Nuseibeh, "Specifying monitoring and switching problems in context," in *15th IEEE International Requirements Engineering Conference, RE 2007, October 15-19th, 2007, New Delhi, India*, 2007, pp. 211–220. [Online]. Available: https://doi.org/10.1109/RE.2007.21

[28] T. Viana, A. Zisman, and A. K. Bandara, "Identifying conflicting requirements in systems of systems," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Sep. 2017, pp. 436–441.

[29] ——, "Towards a framework for managing inconsistencies in systems of systems," in *Proceedings of the International Colloquium on Software-intensive Systems-of-Systems at 10th European Conference on Software Architecture*, ser. SiSoS@ECSA '16. New York, NY, USA: ACM, 2016, pp. 8:1–8:7. [Online]. Available: http://doi.acm.org/10.1145/3175731.3176177

[30] D. Wile, R. Balzer, N. Goldman, M. Tallis, A. Egyed, and T. Hollebeek, "Adapting cots products," 10 2010, pp. 1 – 9.

[31] G. Kiczales and E. Hilsdale, "Aspect-oriented programming," in *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering 2001, Vienna, Austria, September 10-14, 2001*, 2001, p. 313. [Online]. Available: https://doi.org/10.1145/503209.503260

[32] S. Chiba and R. Ishikawa, "Aspect-oriented programming beyond dependency injection," in *Proceedings of the 19th European Conference on Object-Oriented Programming*, ser. ECOOP'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 121–143.

[33] Y. Yu, J. C. S. do Prado Leite, and J. Mylopoulos, "From goals to aspects: Discovering aspects from requirements goal models," in *12th IEEE International Conference on Requirements Engineering (RE 2004), 6-10 September 2004, Kyoto, Japan*, 2004, pp. 38–47. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/RE.2004.23

[34] M. Webster, M. Fisher, N. Cameron, and M. Jump, "Formal methods for the certification of autonomous unmanned aircraft systems," in *Proceedings of the 30th International Conference on Computer Safety, Reliability, and Security*, ser. SAFECOMP'11. Berlin,

Heidelberg: Springer-Verlag, 2011, pp. 228–242. [Online]. Available: http://dl.acm.org/citation.cfm?id=2041619.2041644

[35] C. Lin, D. He, N. Kumar, K. R. Choo, A. Vinel, and X. Huang, "Security and privacy for the internet of drones: Challenges and solutions," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 64–69, Jan 2018.

[36] J. Cleland-Huang, M. Vierhauser, and S. Bayley, "Dronology: an incubator for cyber-physical systems research," in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, 2018, pp. 109–112. [Online]. Available: https://doi.org/10.1145/3183399.3183408