Reliability Engineering and System Safety 132 (2014) 220–232

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Automated synthesis of dependable mediators for heterogeneous interoperable systems

F. Di Giandomenico [a], M.L. Itria [a], P. Masci [c], N. Nostro [a,b,*]

[a] ISTI-CNR, Pisa, Italy
[b] University of Florence, Italy
[c] Queen Mary University of London, United Kingdom

ARTICLE INFO

Article history:
Received 10 January 2013
Received in revised form
30 July 2014
Accepted 9 August 2014
Available online 19 August 2014

Keywords:
Model-based analysis
Dependability
Performance
Dependability mechanisms
Dynamic networked systems
Interoperability

ABSTRACT

Approaches to dependability and performance are challenged when systems are made up of networks of heterogeneous applications/devices, especially when operating in unpredictable open-world settings. The research community is tackling this problem and exploring means for enabling interoperability at the application level. The EU project Connect has developed a generic interoperability mechanism which relies on the on-the-fly synthesis of "Connectors", that is software bridges that enable and adapt communication among heterogeneous devices. Dependability and Performance are relevant aspects of the system. In our previous work, we have identified generic dependability mechanisms for enhancing the dependability of Connectors. In this work, we introduce a set of *generic strategies* for automating the selection and application of an appropriate dependability mechanism. A case study based on a global monitoring system for environment and security (GMES) is used as a means for demonstrating the approach.

1. Introduction and motivation

The classic and well understood way of building dependable systems [1] is based on the application of rigorous development methods. Special programming techniques are used for software, such as model-driven development [2], and specific architectures are used for hardware, such as modular redundancy [3]. Dependability-critical domains, such as avionics and power plants, *require by law* the adoption of these techniques, and define standards that must be followed.

This classic approach to dependability is challenged when critical systems are made up of networks of heterogeneous devices from different manufacturers. The GMES (Global Monitoring for Environment and Security) European Programme for the establishment of a European capacity for Earth Observation provides an excellent example of heterogeneity in interoperable applications and devices for critical applications. It started in 1998, and includes six main thematic areas: land monitoring, marine environment monitoring, atmosphere monitoring, emergency management, security and climate change. The emergency management service directs efforts towards a wide range of emergency situations; in particular it covers different catastrophic circumstances: floods, forest fires, landslides, earthquakes and volcanic eruptions and humanitarian crises. As another example, in the healthcare domain, currently there is not a standard for medical device interoperability. Nevertheless, this has not prevented the adoption in hospitals of networks of heterogeneous technologies. In some cases the lack of interoperation just causes minor disturbances, e.g., patients not recognised by palm-sized wireless medical devices because the devices are not enabled to gather this information on-the-fly from a central database [4]. In other cases, problems are more serious, e.g., surgical fires caused by lack of dependable interoperation between electrosurgical devices and oxygen-delivery devices [5].

The problem is that standardised interoperability at the application level is essentially *non-existent*. In fact, standards like Universal Serial Bus (USB) and IEEE 802.11 (WiFi) enable interoperability at a level *lower* than the application logic. The consequence of this is that heterogeneous networked devices might be able to interoperate but at the same time they might not be able to fully benefit from each other's services. This situation might create serious problems, e.g., in safety-critical systems safety interlocks defined at the application level may be ignored or overridden. Even more challenging is the situation where the heterogeneous systems have a dynamic and evolving behaviour, thus requiring adaptation if interoperability is to be enabled.

* Corresponding author at: ISTI-CNR, Pisa, Italy.
E-mail addresses: f.digiandomenico@isti.cnr.it (F. Di Giandomenico), massimiliano.leone.itria@isti.cnr.it (M.L. Itria), paolo.masci@eecs.qmul.ac.uk (P. Masci), nicola.nostro@unifi.it (N. Nostro).

http://dx.doi.org/10.1016/j.ress.2014.08.001
0951-8320/© 2014 Elsevier Ltd. All rights reserved.

1.1. Problem statement

The problem at stake is *application-level interoperability* in networks of heterogeneous devices. Interoperability is the ability for a device to connect to or be used with another device, and perform individual functions without alteration of the individual device. In heterogeneous networks, devices may have compatible transmitters and receivers that allow us to exchange messages, but interoperability may still be *not* enabled because of mismatches in communication protocols used at the application level by the devices. For instance, assume two devices $D1$ (e.g., a mobile phone) and $D2$ (e.g., a printer) which need to interoperate to accomplish a task (e.g., print an electronic document). If $D1$ requires handshake H 1 to start communication, and device $D2$ only accepts handshake H 2, then the two devices are not able to interoperate at the application level, and so the task cannot be accomplished. Another example is when the two devices have identical protocols but different dependability or performance requirements. For instance, device $D1$ requires maximum latency X, but device $D2$ can only guarantee latency $Y > X$ during communication. In this case too, the application-level interoperability is not possible.

To enable interoperability, different communication protocols need to be *harmonised*. A generic approach to harmonise heterogeneous communication protocols relies on the synthesis of *mediators* that bridge functional (i.e., semantic) gaps between communication protocols, and non-functional (i.e., dependability- or performance-related) mismatches between protocols.

Generic approaches for addressing functional mediation have been investigated since Yellin and Strom's seminal work on component adaptors [6]. Communities that are particularly active on this topic are those of Self-Adaptive Systems (e.g., see [7,8]) and Service Oriented Architectures (e.g., see [9–11]).

Generic approaches for addressing non-functional mediation have been largely neglected. Only few examples can be found in the literature. In [12], the control science theory is used to define a framework with composable modules and an overlay of agents that enable security, privacy and dependability in heterogeneous networks of embedded systems. Another example is [13], where an approach based on stochastic modelling is explored to synthesise mediators that meet given performance requirements.

In our previous work [14,15] we have presented a generic model-based framework to support the synthesis of *dependable mediators*, that is mediators that meet dependability and performance requirements. Recently, in [16], we have identified generic templates which can be used in several practical cases to enhance the dependability and performance level of synthesised mediators. In this work, we unify the two contributions of our previous works, and illustrate in detail the model-based approach used to automate the synthesis of dependable mediators.

1.2. Contribution

The contributions of this work are (i) an automated approach to select and instantiate generic dependability and performance templates during the synthesis of dependable mediators; (ii) a detailed example based on a global monitoring system for environment and security (GMES) that demonstrates the proposed approach.

1.3. Structure of the paper

The presentation proceeds as follows. In Section 2, we provide an overview of the Connect framework, as this work is contextualised within it. In Section 3, the performed model-based dependability analysis is presented. In Section 4, we illustrate the proposed generic methodology for selecting dependability mechanisms in networks of heterogeneous interoperable devices. In Section 5, we demonstrate the benefits of the proposed approach within an example based on a global monitoring system. The selected scenario is one of the demonstrative examples developed in the Connect project. Section 6 describes related work and conclusions are drawn in Section 7.

2. Context

The context of this work is that of Connect,[1] a research project that explored generic approaches to the automated synthesis of "Connectors", software mediators that enable application-level interoperability. A model-based approach is used to identify gaps and mismatches between communication protocols, and then generate a Connector that bridges the identified gaps and mismatches. Modelling is composed of two phases: (i) building of a model that reflects the behaviour of the components of the system and their interactions; (ii) analysis of the model to obtain a Connector that enables application-level interoperability. The Connect framework supports this model-based approach using an overlay network of five types of active units: Discovery, Learning, Synthesis, Dependability, and Monitoring. The role of these five units is now illustrated.

Discovery and Learning: These units gather information about functionalities requested and provided by networked systems. Specifically, the Discovery unit discovers mutually interested devices, and retrieves information about their interface behaviours. The unit assumes that devices are discovery enabled, i.e., they provide a minimal description of their intent and functionalities. When a networked system just provides a partial specification of its behaviour, the Learning unit completes the specification through a learning procedure (e.g., usage on model-based testing and model inference [17]).

Synthesis: This unit performs the dynamic synthesis of mediating Connectors to enable *functional* interoperation among mutually interested devices. The unit performs a graph-based analysis to identify mismatches between the communication protocols identified by Discovery and Learning. A formal definition of the synthesis approach has been presented in [11]. The approach consists of the following steps:

1. The functional specification of the protocols identified by Discovery and Learning is translated into Labelled Transition Systems (LTSs). An LTS is a directed labelled graph used to represent state machines: nodes in the graph represent machine states; directed edges represent transitions between states; labels on the edges identify the event that triggers the transition. In this case, nodes represent protocol states.
2. Ontologies [18] are used to establish a mapping relation between events in the heterogeneous protocols.
3. Communication protocols are "sliced" according to the mapping relation, and the trace of events is systematically generated for each slice. Differences between traces generated for corresponding slices identify mismatches between the protocols.
4. A new LTS (the Connector) is generated to reconcile mismatching traces and thus enable interoperability.

Dependability: This unit supports Synthesis during the generation of Connectors to estimate whether given *non-functional* requirements are met by the synthesised Connector. To this end, the unit performs a stochastic model-based analysis that takes into account the structure of the synthesised Connector and the

[1] http://www.connect-forever.eu

non-functional aspects of the system used for Connector deployment. If the analysis reveals that given dependability and performance requirements may not be satisfied by the Connector, then the Dependability unit instructs Synthesis about enhancements that can be applied to the Connector. This is done using a set of generic templates of dependability and performance mechanisms, such as Retry, Probing or Error Correction. A detailed illustration of the Dependability unit and of a core set of essential dependability and performance templates is presented further below, in Sections 3 and 4.

Monitoring: This unit becomes operational when the Connector is deployed. The unit continuously monitors the Connector in order to update the other units of the Connect framework with run-time data. This allows adaptation and evolution of the Connector.

2.1. *Connectors life-cycle*

The life-cycle of a Connector starts with a networked device broadcasting a “Connect request”. This happens whenever a device requires a service. The Connect request contains a functional description of the required service, together with a specification of dependability requirements associated with it. The request is processed as follows within the Connect framework:

1. Discovery captures the Connect request and looks for networked devices that can provide the requested service. Learning completes the specification of the available networked systems through a learning procedure when needed. If a device is found that can satisfy the request, Synthesis is activated and a Connector that enables functional interoperation is generated (a *null* Connector will be generated if the communication protocols of the two networked devices are already compatible).
2. Synthesis generates the specification of a mediating Connector. This is done on the basis of the specification of the communication protocols. Before deploying the synthesised Connector, Synthesis activates the Dependability unit to assess whether the Connected system meets given dependability requirements.
3. Dependability performs a model-based evaluation of the Connected system. When given dependability requirements are not met, the unit informs Synthesis about how the synthesised Connector can be improved. In [19] the adaptive approach under development has been illustrated, which shows the integration between Synthesis and Dependability modules at design time and run-time. The suggestion on how to strengthen the Connector, from the point of view of dependability and performance, is derived from the analysis results obtained by exercising the model, automatically improved with a dependability mechanism selected from the predefined library, described in detail in [16], until a successful mechanism is found. From the point of view of the synthesis of Connectors, the implementation of the mechanism identified to improve the Connector is performed in the same way by selecting the equivalent mechanism from a set of generic templates.
4. Once the Connector is deployed, its runtime behaviour is monitored through the Monitoring unit, which will notify the other units of relevant changes of functional and non-functional aspects of the deployed Connector.

More details about the Connector life-cycle in a dynamic setting can be found in [19].

3. Dependability unit: architecture and model-based analysis

The Dependability unit is the focus of this work. It performs a state-based stochastic analysis that uses state-space mathematical models to express probabilistic assumptions about time durations and transition behaviours. State-space models allow explicit specification of complex relationships concerning failure and repair processes, as well as sequencing information.

The input–output relation between the Dependability unit and the other units of the Connect framework is shown in Fig. 1: the non-functional requirements for the Connected system are provided by the Discovery/Learning unit; the specification (nominal behaviour and exceptional conditions) of the connected system is provided by the Synthesis unit; run-time statistical data on the execution of the deployed Connector is provided by the Monitoring unit; dependability and performance enhancements for a synthesised Connector are identified by the Dependability unit and used by Synthesis.

To support the above input–output relations, the Dependability unit is logically split into four main functional modules (see Fig. 2): Builder, Analyser, Evaluator and Enhancer.

Builder: This module generates a model of the Connected system for dependability and performance analysis. The model is obtained by decorating the functional specification provided by Synthesis with non-functional information about the Connector deployment. Examples of non-functional information include time to complete given events, typical failure modes, and typical failure probability.

Analyser: This module uses the decorated model to perform a quantitative assessment of given non-functional requirements. This is done by specifying dependability and performance metrics through *reward functions* that collect information on the Connected system state. Depending on the non-functional requirements that need to be evaluated, information is collected either at precise instant of time or accumulated over a period of time.

Evaluator: This module checks the results of the quantitative assessment of given non-functional requirements. If the results indicate that the requirements are met, the Evaluator acknowledges

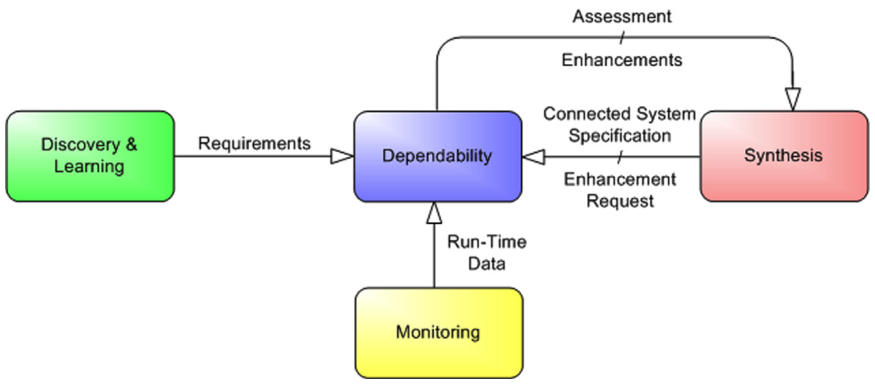

Fig. 1. Input–output relations of the Dependability unit.

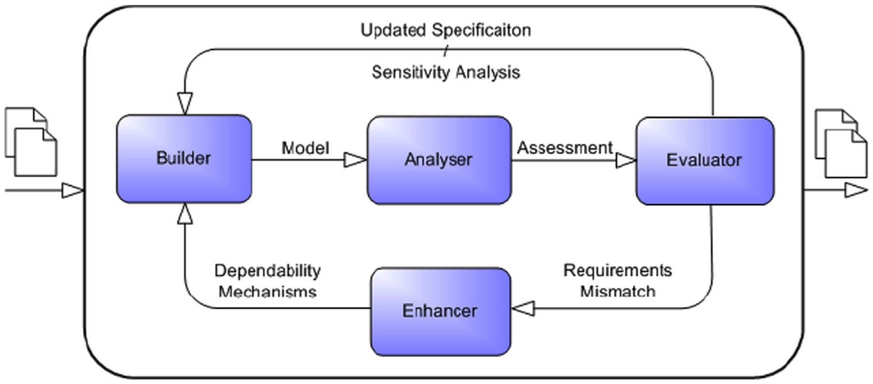

Fig. 2. Architecture of the Dependability unit.

Synthesis of the analysis success. Otherwise, the Evaluator sends a warning message to Synthesis, which in turn may send back a request to identify possible enhancements that can be applied to improve the dependability level of the Connected system.

Enhancer: This module becomes operational when Synthesis sends a enhancement request. The module explores whether alternative Connector deployment is available (e.g., deployment on a system with lower failure rates), or dependability/performance mechanisms (e.g., message retransmission techniques) can be used to improve the non-functional characteristics of the Connector. If a suitable enhancement is found, the Enhancer module reports the specification of the new Connector to Synthesis, and the enhanced Connector will be deployed.

4. Automated selection and application of dependability and performance templates

A novel approach for the automatic selection of a dependability mechanism for enhancing synthesised Connectors is now presented. First, in Section 4.1, the rationale behind the approach for automatic selection and application of dependability mechanisms is illustrated; second, the approach is implemented in the context of Connect in Sections 4.2 and 4.3.

4.1. Rationale behind the approach

The selection of dependability mechanisms for enhancing Connectors is typically driven by (i) *application constraints* imposed by the application domain, e.g., remote-control applications may have different constraints in terms of timing and tolerance to degraded service from video-based applications; (ii) *fault and failure assumptions* for devices during their operational life, e.g., transient faults or permanent faults; (iii) *dependability metrics* relevant to given dependability requirements that must be met, e.g., message delivery time, coverage of receivers.

Application constraints may have an impact on the system tolerance to degraded service (meaning degradation both in the value and time domains), and the ability to deploy specific solutions. In some cases this may hinder the benefits of certain dependability mechanisms. For instance, consider a dependability mechanism based on error-correction that can be used to detect transmission errors and thus reconstruct the original error-free data. Given that the Connect framework aims to modify only the Connector, the mechanism can be applied only if the original error-free data are available at the transmitter-side of the Connector, e.g., either when the Connector can be deployed on the transmitter, or when a reliable channel exists between the transmitter and the infrastructural element with the deployed Connector.

Fault and failure assumptions are at the basis of dependability and performance models. They are typically based on assumptions about the malfunctions that may undermine the (dependability or performance) indicators of the system under analysis during its operational life. The nature of the assumed faults and/or failures guides the selection of countermeasures. Avizienis et al. [1] proposed a taxonomy of the faults and failures in the dependability community. For the generic dependability and performance mechanisms considered in this work, the interest is mainly in (i) the persistence of the fault, that is whether the fault is transient (maybe, in bursts, but lasting a limited amount of time) or permanent; (ii) the objective of the fault, that is if it is accidental or intentionally introduced in the system; (iii) the failure domain, that is content failures when the content of the information delivered at the service interface deviates from implementing the system function, or timing failures when the time of arrival or the duration of the information delivered at the service interface deviates from implementing the system function. We will discuss further below how the dependability mechanisms we adopt relate with these faults and failures assumptions.

Dependability metrics guide the definition of the assessment model, which has to faithfully include all the system aspects with a relevant impact on the measure under evaluation while abstracting or even neglecting all the other behaviours/phenomena with negligible impact, in order to keep the model as much as possible manageable and controllable. The measure also influences the choice of the dependability mechanism, characterised by differing structure and operational behaviours. In the context of Connect, we mainly addressed two categories of measures for stochastic quantitative analysis: performance-related one (e.g., variants of the latency indicator), and dependability-related one (e.g., different forms of coverage as percentage of networked services receiving/offering a service with respect to the full population).

In the literature, there is no evidence that relying just on one of the factors illustrated above is the best choice. Therefore, our proposed selection method (illustrated further below in Section 4.2) will consider a combination of these identified factors.

4.2. A method for automatic selection of dependability mechanisms

The Connect project uses ontologies [18] as the basis for specifying the behaviour of networked systems. That is, a semantic description of the behaviour of the networked systems is used to identify peers that need to be connected to enable a service. In Connect, this semantic description is referred to as *affordance*. Given semantically matching affordances, their associated communication protocols must be checked for the potential to interoperate, possibly under the mediation of a Connector. According to the classification of Connectors adopted by the Connect project, and the related reference ontology, the following coordination models have been identified: client-service, message-orientation, publish-subscribe and shared memory.

Following an ontology-based approach similar to the one used for functional mediation, a general method for selecting a dependability mechanism for Connectors is now outlined. It consists of the following main steps:

Step 1: An ontology-based characterisation of the dependability mechanisms is created that points out how the mechanism is linked to application constraints, fault and failure assumptions, and dependability metrics.

Step 2: A semantic matching based on the developed ontology is performed when a dependability mechanism is required for the Connector. The aim is to identify and give a rating to those dependability mechanisms that are relevant to the type of mismatch revealed by the performed model-based dependability and performance analysis of the Connector.

Step 3a: If only one identified mechanism is rated the highest, then this mechanism is selected.

Step 3b: If more than one mechanism have same (highest) rating, then a prioritisation among application constraints, fault and failure assumptions, and dependability metrics are used for ranking the identified mechanisms.

The *Step* 1 of the method has been implemented by creating a table of the ontology of dependability mechanisms for Connectors. Table 1 shows such an ontology and defines a mapping relation among dependability mechanisms, mitigated threats, affected dependability metrics (both enhanced and deteriorated), and application constraints to be satisfied. Specifically, Table 1 specifies the following for threats: type (e.g., omission or value), duration (transient or permanent) and nature (accidental, i.e., natural or human made but without malicious intent, or intentional [1]) of the threat (column *Mitigated Threat*). Concerning the dependability metrics, the table indicates which metrics can be enhanced by the dependability mechanism (column *enhanced metric*), and which are potentially deteriorated (column *deteriorated metric*). We focused on the classical dependability and performance metrics (e.g., reliability, safety, integrity, latency); however, it is possible to extend this set to include other dependability and performance related metrics, such as the *coverage* metric analysed in the case study at Section 5. Finally, Table 1 shows, in the last column (*satisfied constraint application*), the application constraints satisfied by the mechanism. As briefly discussed, such application constraints are application specific and are mainly related with the ability of the application to tolerate degraded services and to provide enabling facilities to deploy dependability and performance mechanisms. Since we do not target any specific application, only the classical timing constraint is considered at the moment, grading it into *none*, *soft*, *medium*, and *hard*. Again, extensions to consider other kinds of application constraints can be easily accommodated. The ontology table includes the five basic dependability mechanisms already defined in [16] as a suitable core to enhance, especially, Connector dependability. Let us focus on the *Retry* mechanism. This mechanism consists in re-sending messages that get corrupted or lost during communications. A typical implementation of the Retry mechanism uses time-outs and acknowledgements: after transmitting a message, the sender waits for a message of the receiver that acknowledges successful communication. If the acknowledgement is not received within a certain time interval, the sender assumes that the communication was not successful, and re-transmits the message. The ontology therefore indicates that the Retry mechanism can be used to enhance the reliability of a Connector when non-intentional transient faults result in omitted data or data with wrong values. The ontology indicates, however, that the mechanism is likely to deteriorate two other quality parameters of the Connector: latency and throughput. Given the longer execution time due to the Retry, it complies with applications having none or soft timing constraints only.

In the following, we define the characteristics of the other dependability mechanisms included in Table 1. For these mechanisms, the entries in the table are built following the same approach illustrated for the Retry mechanism.

- *Probing:* This mechanism exploits redundant paths and periodic *keep-alive* messages for enabling reliable communication in face of path failures. The basic idea is to continuously collect statistics about the characteristics of the communication channels, and to select the best channel on the basis of such statistics. Table 1 shows that the Probing mechanism is useful to enhance the reliability and the latency of the Connector, and it is compliant with hard timing constraints. However,

Table 1
Example of ontology of dependability enhancement mechanisms for Connectors.

Ontology of dependability mechanisms for Connectors

Mechanism	Mitigated threat	Enhanced metric	Deteriorated metric	Satisfied application constraint
Retry	Transient, accidental, omission or value failure	Reliability	Latency, throughput	Soft timing
Probing	Transient or permanent, accidental, omission failure	Reliability, latency	Throughput	Hard timing
Majority voting	Transient or permanent, accidental or intentional, value failure	Integrity, safety, reliability, maintainability	Latency, throughput	Medium/Hard timing
Error correction	Transient, accidental or intentional, omission or value failure	Reliability, integrity	Latency,throughput	Medium timing
Parallel retry	Transient or permanent, accidental, omission or value failure	Latency, reliability, maintainability	Throughput	Hard timing

implementing such mechanism leads to a throughput deterioration of the system, due to additional procedures to select the best channel.

- *Majority Voting:* This is a fault-tolerant mechanism that relies on a decentralised voting system for checking the consistency of data. Voters are software systems that constantly check each other's results, and have been widely used for developing resilient systems in the presence of faulty components. In a network, voting systems can be used to compare message replicas transmitted over different channels. It is rather expensive in redundancy, with benefits from the point of view of enhancements of several properties (integrity, safety, reliability, and maintainability). But the redundancy degree, the synchronisation among the multiple executions and the additional voting execution result in a degradation of latency and throughput aspects with respect to the non-redundant CONNECTor solution.
- *Error Correction:* This mechanism deals with the detection of errors and re-construction of the original, error-free data. A widely used approach for enabling hosts to automatically detect and correct errors in received messages is *forward error correction* (FEC). The mechanism requires the sender host to transmit a small amount of additional data along with the message, thus causing a worsening of latency and throughput, but providing an improvement on reliability and integrity. With respect to potential timing constraints imposed by the applications, this mechanism is classified as suitable to comply with up to a *medium* level.
- *Parallel Retry:* This is similar to the Retry mechanism already introduced, but exploits channels redundancy (instead of successive retries on the same channel) to speed up the communication time. In this way it allows us to improve latency, reliability, and maintainability, but at the same time it may reduce the throughput of the system due to the redundancy. Concerning its ability towards timing constraints, this mechanism appears to be suitable to satisfy up to *hard* constraints.

Two final notes about Table 1 are the following: First, it has been built considering general characteristics of the selected dependability mechanisms. Of course, a more accurate matching with respect to their ability to enhance (or decrease) dependability and performance metrics, as well as to satisfy application constraints depends on the specific implementation of such mechanisms, and on the quantification of the grades of timing constraints (here simplified in none, soft, medium, hard), which maybe application dependent. The second observation is that the mechanisms listed in the table primarily address dependability improvements, with the consequent negative impact on performance related metrics. In the considered set, only *Probing* and *Parallel Retry* mechanisms have ability to also enhance latency. More solutions targeting performance enhancements can be however added, once defined.

By using the defined ontology, the proposed approach for automated selection of a dependability mechanism for CONNECTors illustrated at the beginning of this section leads to Table 2. The table presents the mechanisms following general risk assessment principles: it establishes a relationship among threats, application constraints, and risk reduction strategies (dependability mechanisms). In this table, combinations of *threats* and *application constraints* are presented, and for each of them a set of dependability mechanisms suitable to cope with them are selected from Table 1. Of course, this initial table is not exhaustive; it is expected to be extended with additional combinations of *threats* and *application constraints*, as well as additional dependability and performance mechanisms to enable CONNECTors enhancement in different application scenarios.

4.3. *Where to apply the dependability mechanism*

The most accurate method to identify element(s) of the CONNECTor that should be enhanced through the selected dependability mechanism is based on a systematic exploration of the benefits of the mechanism to each individual element in the CONNECTor. The element for which the benefit is the highest is the best candidate. While very accurate, this approach is time consuming. We have developed an alternative method that provides a good trade-off between time and accuracy in several practical cases. The idea of the approach is to perform an exploratory investigation based on simple probabilistic formulations for finding out where to apply the dependability mechanism. The approach is tailored to two specific cases: dependability-related metrics and performance-related metrics.

The exploratory investigation is now illustrated in detail. A typical situation encountered in model-based analysis of interoperable devices is considered. Communication protocols are specified in terms of timed activities. Each timed activity a_i is characterised by a time to complete t_i and a failure probability q_i, independently from the others. For the illustrative purpose of this work, we assume that dependability-related metrics are mainly influenced by failure probability, while performance-related metrics are mainly influenced by the time to complete. Under these assumptions, two automated strategies (one for dependability-related metrics, one for performance-related metrics) can be defined for identifying elements in the CONNECTor that should be enhanced with dependability mechanisms.

4.3.1. *Dependability-related metrics case*

We start with the simple case where (i) the metric under analysis is a function of the failure probabilities of all the activities

Table 2
Example of table for automated selection of dependability mechanisms for CONNECTors.

Table for automated selection of dependability mechanisms for CONNECTors

Threat	Application constraint	Mechanism
Transient, accidental, omission failure	Hard timing	Parallel Retry, Probing
Transient, accidental, omission failure	None	Parallel Retry, Probing, Error Correction Retry
Transient, accidental, omission failure	Medium timing	Parallel Retry, Probing, Error Correction Retry
Permanent, accidental, omission failure	Medium timing	Parallel Retry, Probing
Transient, accidental or intentional, value failure	None	Majority Voting, Error Correction Retry
Transient, accidental or intentional, value failure	Medium timing	Majority Voting, Error Correction Retry
Permanent, accidental, value failure	Medium timing	Majority Voting, Parallel Retry
Permanent, accidental, value failure	None	Majority Voting, Parallel Retry
Permanent, intentional, value failure	Medium timing	Majority Voting
Permanent, accidental, value failure	Hard timing	Parallel Retry, Majority Voting

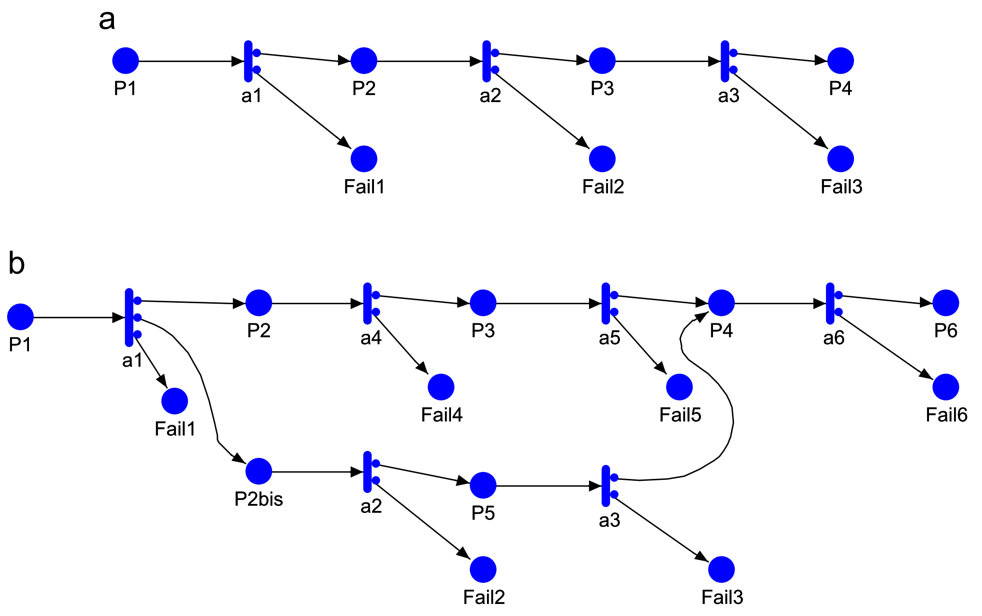

Fig. 3. Example of a basic Connector model (a), and of a more complex Connector with two branches (b).

Table 3
Illustrative example of the selection approach for dependability-related metrics.

Activity	q_i	$\overline{q}_i$(Maj. Voting)	$\overline{q}_i$(Retry)	Q_i(Maj. Voting)	Q_i(Retry)
a_1	0.20	0.104000	0.0400	0.66104320	0.6368320
a_2	0.03	0.002646	0.0009	0.68882555	0.6882808
a_3	0.35	0.281750	0.1225	0.66558280	0.5914360

in the model representing the Connector; (ii) each activity can either be successfully completed or fails. This means that there is only one path in the model that results in a successful execution of the Connector. Fig. 3(a) illustrates this case.

Let us consider that N activities are included in the model of the Connector under analysis. Then, the selection of the activity to be enhanced is simply determined through the following steps:

1. for each activity a_i, determine the new value $\overline{q}_i$ resulting from applying the dependability mechanism to activity a_i;
2. for each activity a_i, compute the value of the failure probability Q_i of the whole Connector when using the new value $\overline{q}_i$. The general formulation of the Connector failure probability Q is given by the expression

$$Q = q_1 + \sum_{i=2}^{N} q_i \cdot \prod_{j=1}^{i-1} (1-q_j)$$

   For each index i, Q_i is obtained by substituting $\overline{q}_i$ to q_i in the formula of Q;
3. select the activity a_j whose new probability value $\overline{q}_j$ determines a Q_j that is the minimum among the N values of Q_i.

To exemplify the approach, let us consider the model in Fig. 3 (a) and the failure probabilities q_i for two dependability mechanisms (Majority Voting and Retry), as reported in Table 3. According to the values in the table, activity a_1 should be enhanced when applying majority voting, while activity a_3 should be enhanced when using Retry.

Generalisations of this simple case can be performed in order to take into account that (i) not all the activities a_i are involved in the dependability metric under analysis, and (ii) more than one path is possible within the dependability model to reach a_i from the start activity (e.g., as illustrated in Fig. 3(b)).

With respect to point (i), the generalisation is easily performed by determining the new values $\overline{q}_i$ only for those activities a_i involved in the formula expressing the metric and consequently restricting the calculation of Q_i only to the $\overline{q}_j$ computed. Then, the last step remains the same.

The second generalisation does not require any actual modification to the described basic method. What changes is just the formula expressing the Connector failure probability Q, which has to take into account the different paths which may lead to failure. The formulation is straightforward and omitted here to save space.

An interesting observation concerns the formula expressing the improvement in reliability gained through the application of the schemes for which a dependability model has been generated (e.g., those in Table 2), necessary to derive the new values $\overline{q}_i$ associated with activities a_i. It could be stored together with the mechanism model as a parametric reliability expression, and be efficiently adapted with the current probability failure parameter q_i of the activity at hand when a specific Connector needs to be enhanced. This allows us to speed up the application of the selection procedure.

Finally, by executing a campaign of experiments starting with basic failure probabilities q_i lower than 0.01, we found that the approach described above could be reasonably approximated by directly choosing the activity with the highest failure probability value. Again, this promotes efficiency of the selection process.

4.3.2. *Performance-related metrics case*

In the case of performance-related metrics, the choice of the model elements to be enhanced is expected to be made among those representing communications with longer time to complete. We recall that performance metrics we are dealing with are classical metrics, typically the degree to which a system or a component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage, as defined in [20]. Therefore, with reference to our Connect context, the dependability mechanisms should act to improve transmission time essentially in the presence of possible congestion of the communication channels, responsible for longer times. Among the dependability mechanisms shown in Table 2, Parallel Retry and

Probing are the only ones adequate to address performance improvements: by using redundant transmission channels, these mechanisms allow us to exploit the most performable ones, e.g., the Probing mechanisms, by continuously monitoring the efficiency of both channels available for transmission, at each sending uses the most efficient one.

The approach to select the activity to improve is similar to the dependability-related metric case, but specular under the time domain. The steps are as follows:

1. identification of those activities that are involved in the performance metric (unless we are dealing with an end-to-end metric, a subset of activities is typically involved);
2. for each activity a_i in the involved set, re-determine the value of the transmission time parameter $\overline{t}_i$ resulting from applying the dependability mechanism;
3. since it is expected that the performance-related metric is based on the summation of the execution time of the activities under consideration, select the activity a_j for which the difference between the original time parameter value t_i and the new one $\overline{t}_j$, as calculated at Step 2, is the largest.

Similar to Table 3, Table 4 elaborates a simple example for the performance-related case, using the Probing and Parallel Retry (using three channels) as dependability mechanisms.

From the table, it can be observed that the Parallel Retry results in much better values than those of the Probing, at the cost of a higher redundancy. The choice of the tuning of the mechanisms parameters (such as the number of additional channels for the Parallel Retry) depends on the level of performance that needs to be guaranteed for the specific application at hand.

4.3.3. General observations

Following the above developments, some indications about further extensions are addressed in the following.

First, in this study we concentrated separately on dependability and performance properties, since it is rather common for applications to have requirements which pertain either one or the other. However, it is also true that dependability and performance properties are more and more required to be considered in strict relationship in a variety of critical applications. Therefore resorting to separate evaluation models would result inadequately, since they fail to capture the consequences of failures on the degradation of the performance. Performability metrics have been proposed [21] as a unification of performance and dependability, that is system's ability to work in the presence of errors and failures. In contrast with pure performance, which refers to how efficiently a system delivers a specified service assuming that it is delivered correctly, with performability effects of faults being considered and, at the same time, failures due to faults are not the only relevant phenomena to assess.

Although not fully addressed in this paper, we outline a basic approach to address performability as a criterion to select where a fault tolerance mechanism would bring a significant benefit to the system at hand. Once a dependability mechanism suitable to cope with both dependability and performance aspects has been selected (e.g., the Probing mechanism, according to Table 2), its impact on each of the activities the Connector under enhancement is determined in terms of both reliability (following the steps at Section 4.3.1) and execution time (following the steps in Section 4.3.2). Therefore, for each activity a_i, both the failure probability Q_i and the execution time T_i of the Connector are determined and obtained when the dependability mechanism is applied to activity a_i. Then, a performability measure M can be defined in terms of a reward structure that assigns a cost C in case of connection failure and a gain G when the connection completes within the calculated time T:

$$M = \frac{(1-Q)\cdot G}{T} - (Q \cdot C)$$

According to the above formulation, in case the connection fails, the cost C (whose value is chosen in agreement with the criticality of the applications willing to interoperate) has to be afforded, while in case of success the gain G is obtained, whose amount (again, chosen depending on the applications under consideration) decreases at increasing the connection duration time, to favour efficient Connectors. By properly evaluating the values of Q_i and T_i resulting from applying the dependability mechanism to a_i (for each a_i) and assigning values to C and G (cost and gain, respectively), the activity a_i which leads to the highest *Performability_Measure* (by substituting Q_i and T_i to Q and T in the performability formula) can be determined. Full exploration of the approach, as well as investigating other formulation for a performability measure and related trade-offs of costs and gains, is postponed to future work.

Another aspect that we would like to point out is that, after the application of our method to select a model element for enhancement through the dependability mechanism (either for dependability- and performance-related measures), and the consequent extension of the Connected system model for a new analysis phase, the results that are not yet satisfactory with respect to given requirements could happen. This means that, iteratively, the selection of an additional element needs to be carried out on repeating the same steps as before, until the requirement is met or the whole model has been enhanced without success. It would be interesting to investigate methods to predict, with a satisfactory level of accuracy, the minimum set of elements that would be necessary for requirement satisfaction within the next analysis phase. This would greatly reduce the exploration time. Also this research direction is part of our future research agenda.

Table 4
Illustrative example of the selection approach for performance-related metrics.

Activity	t_i	$\overline{t}_i$(Probing)	$\overline{t}_i$(Parallel Retry)	$t_i-\overline{t}_i$(Probing)	$t_i-\overline{t}_i$(Parallel Retry)
a_1	4.664	4.530	1.589	0.134	3.075
a_2	2.490	2.302	0.794	0.188	1.696
a_3	1.747	1.497	0.529	0.250	1.218

5. GMES case study

In this section, we present our demonstrative scenario, based on the GMES (Global Monitoring for Environment and Security)[2] European Programme for the establishment of a European capacity for Earth Observation, in order to show how the proposed method can be applied to select proper dependability mechanisms and where to apply them.

GMES services address six main thematic areas: Land Monitoring, Marine Environment Monitoring, Atmosphere Monitoring, Emergency Management, Security, and Climate Change. The GMES emergency management service provides all actors involved in the management of natural disasters. In particular, it covers different catastrophic circumstances: floods, forest fires, landslides, earthquakes and volcanic eruptions and humanitarian crises.

[2] http://www.gmes.info/

5.1. *Forest-fire emergency*

In this work, we concentrate on the Forest fire emergency situation [22]. The scenario describes the management of forest-fire, close to a border village and a factory between two different countries, Country A and Country B. Forest monitoring and forest fire management in the country A are under the responsibility of Country A Command and Control fire operations centre (C2-A).

The Forest-fire scenario addresses different phases. We focus on the *Reinforcement integration phase*, where Country B provides an unmanned aerial vehicle (UAV) to Country A. The UAV is equipped with various video cameras that allow us to get a better view of the fire front close to the village in order to be able to proceed to its evacuation in time. Country B grants access to its weather forecast service, in order to continuously provide information about temperature, humidity and wind of the area interested by the fire.

During the crisis phase, Country A's Command and Control fire operations centre (C2-A) is in charge of the management of the forest fire crisis. This phase involves a number of sensors and human actors. In case the fire goes wider and there is a direct threat to the village and the factory, Country A asks support to Country B (*reinforcement phase*). The reinforcement phase starts when the support resources, provided by Country B, are deployed and controlled by C2-A. Once deployed on the emergency area, resources from Country B are seen by C2-A as resources are available and usable during the fire-fighting phase. Resources provided by Country B have the functionalities similar to those of Country A's resources (e.g., UAVs provide high quality images or weather information of the area interested by the fire), but use different protocols.

Networked systems involved in this scenario are C2-A, the Weather Service, and the UAV with integrated video camera. For the illustrative purpose of this work we consider only C2-A and the UAV, as this is sufficient to demonstrate the approach. We included the UAV instead of the Weather Service because it provides a more interesting scenario.

The behaviour of the considered networked systems is described in the following subsections.

5.1.1. *Command and Control centre of Country A*

The Command and Control centre represents the first networked system which needs to use specific resources. When C2-A wants to access the resources equipped with one (or more) video camera(s) and operating in the field, like an unmanned ground vehicles (UGV), it first needs to authenticate with the resource sending a *getToken* message. After that, it receives back a message containing the *token* useful to access the resources. Once C2-A has a token, it can instruct the resource to move forward, backward, left and right, by sending the corresponding message, which is followed by an acknowledgement message confirming the movement. At the same time C2-A can select the camera installed on the resource through the message *selectCamera* and then it can receive the video stream with a specified zoom level by sending the message *getVideo*.

5.1.2. *UAV and integrated video camera of the Country B*

The user of the UAV first needs to authenticate with the service by sending a `getIdentifier` request. After that it receives back an identifier (`idResp`) and can trigger the flight phase through the `takeOff` message. After the takeoff, the user can order the UAV to move left, right, front, back, up or down, or to land. For each order of movement, the UAV replies with an acknowledgement message. Moreover, at anytime during the flight phase, the user can get, from the integrated video camera, the video stream showing in real time the specified area of interest; the user can also perform zoom-in and zoom-out commands.

5.1.3. *CONNECTing C2-A and the UAV*

The application behaviour of the CONNECTed system is depicted by the sequence diagram in Fig. 4. Given the heterogeneity between the Command and Control centre and the UAV involved in this GMES scenario, it is necessary to synthesise on-the-fly a CONNECTOR to allow the communications and the exchange of information between the two Networked Systems. A CONNECTOR is therefore synthesised and analysed to assess whether it meets the dependability and performance requirements.

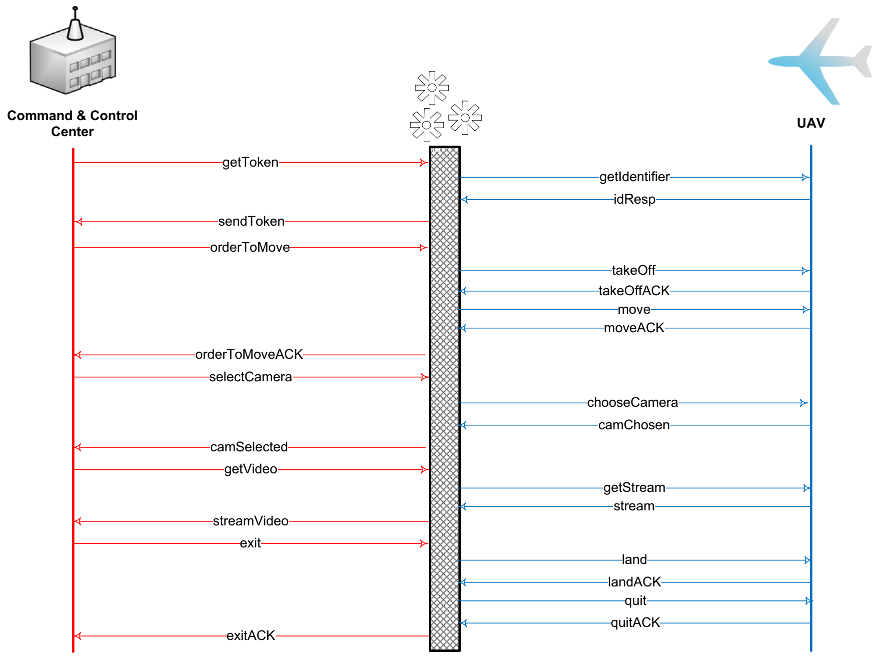

Fig. 4. Sequence diagram of the CONNECTed system.

5.2. Connector's model

From the specification of the Connected system, we generated a model suitable to assess dependability and performance related metrics. This task is accomplished in the Connect project by an automated process implemented in the Dependability and Performance unit mentioned in Section 2. The adopted model formalism is SAN (Stochastic Activity Network) [23] and the analysis is carried out through the Möbius tool [24]. The choice of the SAN formalism is not exclusive, other modelling formalisms presenting equivalent characteristics could also be used for the purpose.

SANs [23] are a stochastic extension of Petri nets; they are based on four primitive objects: *places*, *activities*, *input gates*, and *output gates*. Places and activities represent the state and the actions of the modelled system, respectively. There are two types of activities: *instantaneous* and *timed*. Timed activities represent actions that have a duration expressed via a time distribution function. Both instantaneous and timed activities may have case probabilities. Each case probability stands for a possible outcome of the activity, and can be used to model probabilistic aspects of the system, e.g., probability for a component to fail. Input gates, that are red triangle oriented to the left, control the enabling of activities and define the marking changes that will occur when an activity completes (fires). Output gates, that are black triangles oriented to the right, define the marking changes of the state of the system when an activity completes. The attributes of the SAN primitives are defined by using sequences of C++ statements.

Several SAN models, called *atomic models*, can be composed of *Join* and *Rep* operators, in order to represent the whole system with a good modelling efficiency. Join is used to compose two or more SANs. Rep is a special case of Join, and is used to construct a model consisting of a number of replicas of a SAN. Models in a composed system interact via *Place Sharing*. Place Sharing is a composition formalism based on the notion of sharing places via an equivalence relation.

SAN models for the two networked systems (that are, the Command and Control centre and the UAV) and of the synthesised Connector are generated. Here, we just show in Fig. 5 the model of the Connector, since our approach to dependability and performance enhancement acts only on it (in fact, the networked systems are assumed to be observable only through their interface and there is no possibility to make changes into their internal structure and operation). Examples containing detailed descriptions of the formalism and SAN models, in the context of the Connect project, can be found in [14,25]. However, to simplify the reading of the paper, the SAN model of the Connector is briefly described in the following.

The atomic model Connector, shown in Fig. 5, represents the synthesised Connector generated to allow communication among C2-A, the UAV, and the Weather Service. One token in the place `p1` (in the upper left side) means that the Connector is waiting for the first networked system, that is C2-A, which can be interested to communicate with the Weather Service or the UAV. The upper part of the atomic model shows the Connection between C2-A and the Weather Service, while the lower and larger part shows the Connection between C2-A and the UAV. The activation of one of the two possible Connection is regulated by the input gates `ig1` and `ig5`, connected to the timed activities `getWeather` and `getToken`, respectively. We focus on the description of the (C2-A–Connector–UAV) branch, that is the one we analyse in this work. The activity `getToken` becomes enabled to complete when the place `p1` contains at least one token and the condition expressed by the input gate `ig5` is true, that is the place where `sCC3` contains at least one token $(\mathtt{sCC3}->\mathtt{Mark}()>0)$. The place `sCC3` represents a shared place between the two atomic SANs: C2-A and Connector. Tokens are put in `sCC3` by the C2-A in order to activate the communication with the UAV, representing the *getToken* message. Once the activity `getToken` completes, the Connector sends a *getIdentifier* request to the UAV, represented by the activity `getIdentifier`; when this activity completes, it adds one token in the place `p8` and, through the output gate `og5`, one token in the place `sUAV1`, that is a shared place between the two atomic SANs: Connector and UAV. At this point the UAV processes the received request and responds with an *identifier*. The identifier is received back by the Connector; it is represented by the activity `idResp`, which is enabled by predicate expressed in the input gate `ig6`: at least one token in the share place `sUAV2`. The description of the model proceeds following the description of the behaviour of the Connected system, shown in Fig. 4. As a general rule, we can observe that each activity has two cases the first from the top represents the correct operation, while the second one represents the failure of the activity, with a specified failure probability. The name of each shared place is preceded by the suffix *s*, followed by the name of the networked system that shares this place (e.g., UAV), finally an identification number is appended to the name to distinguish the various places.

5.3. Analysis

In order to cover both performance and dependability aspects, the measures assessed in the evaluation are, respectively, latency and coverage. It is important to note that the measures assessed refer to two different and independent analyses, detailed in the following.

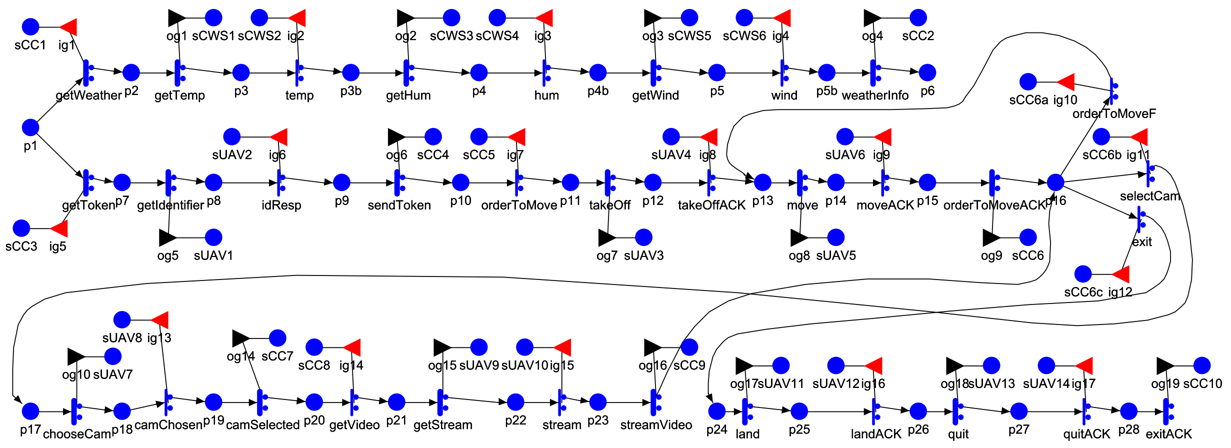
sCC1
ig1
og1 sCWS1 sCWS2 ig2
og2 sCWS3 sCWS4 ig3
og3 sCWS5 sCWS6 ig4
og4 sCC2
getWeather p2 getTemp p3 temp p3b getHum p4 hum p4b getWind p5 wind p5b weatherInfo p6
sCC6a ig10
orderToMoveF
p1
sUAV2 ig6
og6 sCC4 sCC5 ig7
sUAV4 ig8
sUAV6 ig9
sCC6b ig11
getToken p7 getIdentifier p8 idResp p9 sendToken p10 orderToMove p11 takeOff p12 takeOffACK p13 move p14 moveACK p15 orderToMoveACK p16
selectCam
sCC3 ig5
og5 sUAV1
og7 sUAV3
og8 sUAV5
og9 sCC6
exit
sCC6c ig12
sUAV8 ig13
og10 sUAV7
og14 sCC7 sCC8 ig14
og15 sUAV9 sUAV10 ig15
og16 sCC9
og17 sUAV11 sUAV12 ig16
og18 sUAV13 sUAV14 ig17
og19 sCC10
p17 chooseCam p18 camChosen p19 camSelected p20 getVideo p21 getStream p22 stream p23 streamVideo
p24 land p25 landACK p26 quit p27 quitACK p28 exitACK

Fig. 5. SAN of the Connector.

5.3.1. *Analysis of performance*

Latency is evaluated as a performance indicator and is measured from when the Command and Control centre sends one of the possible orders to move (`orderToMove`) to when it receives an acknowledgement back (`orderToMoveACK`). Given the initial parameter values characterising the timing aspects of the elements involved in the CONNECTed system model (namely, the time to communicate between the networked systems and the CONNECTOR), and the latency requirement stated beforehand, the analysis reveals that the CONNECTOR does not satisfy this last. It implies that some enhancement is necessary, and this is the point where our approach plays its part. In this case study, we limit our identification of the appropriate mechanism to select to those listed in Table 2.

In the case of latency requirement not satisfied, Probing and the Parallel Retry mechanisms are the candidate mechanisms to improve the CONNECTed system with respect to timing constraints. In the presence of limited availability of channels in exclusive usage by the CONNECTed system under analysis, preference goes to the Probing mechanism, which we applied to our example. Then, the second phase is to select the possible model element(s) to enhance with Probing. Following the steps delineated in Section 4.3, the element for which the mechanism brings the highest benefit in decreasing the latency metric is the `move` activity.

Fig. 6 shows the results obtained considering the measure of interest (on the *y*-axis expressed in time units) at varying rates of the distribution of the transmission time of the communication channel between 0.1 and 1 (on the *x*-axis). The plots in Fig. 6 show the latency results distinguishing the first movement, which includes the time needed for the takeoff operations, from all the other movements that are performed during the flight.

As expected, the latency during the first phase, which includes the takeoff, is greater than the flight phase. Fig. 6 also shows the results obtained including the use of the *Probing mechanism* [16] in the model of the CONNECTed system, thus allowing the enhancement of the requirements of latency in both the phases.

5.3.2. *Analysis of dependability*

The second analysis aims to assess aspects of dependability of the CONNECTed system through the *Coverage*, defined as the percentage of stream video the Command and Control centre correctly receives from the UAV, with respect to the number of video requests made.

Fig. 7 shows the trend of coverage (on the *y*-axis) at increasing values of streams requests from C2-A (on the *x*-axis). Moreover, the coverage threshold is shown in the figure at value of 0.75, as specified in the requirement. The figure shows that the analysis results, obtained considering the basic model of the CONNECTed system, satisfy the requirement only when the number of requests is at most 13.

As for the performance analysis, Table 2 needs to be examined to find the proper mapping between the mechanism and the characterising aspects when the coverage requirement is not met. Here, timing constraints are rather soft, while correctness is mainly relevant. Therefore, the viable candidate mechanisms are Retry and Majority Voting. For the sake of saving in additional resources to employ, the Retry mechanism is selected. By applying the procedure that selects the model element(s) that needs to be enhanced, we find that `getStream` is the activity that leads to the greatest improvement with the Retry mechanism. This activity models two communication channels between the CONNECTOR and the UAV. The results in Fig. 7 confirm that Retry applied to that activity provides an improvement on the measure of coverage with respect to the original model. The enhanced CONNECTOR meets the requirement up to a maximum of 18 requests.

In order to verify a further possible improvement, analyses were performed by applying the Retry mechanism on a further activity, `stream`, that is the second critical activity obtained through the selection procedure.

It is possible to note that the use of this dependability mechanism on two different activities allows us to fully meet the requirement of coverage.

For illustrative purposes, in Fig. 8, the portion of the CONNECTOR model where the `getStream` activity has been replaced by the Retry Mechanism between the places `p21` and `p22` of Fig. 5 is

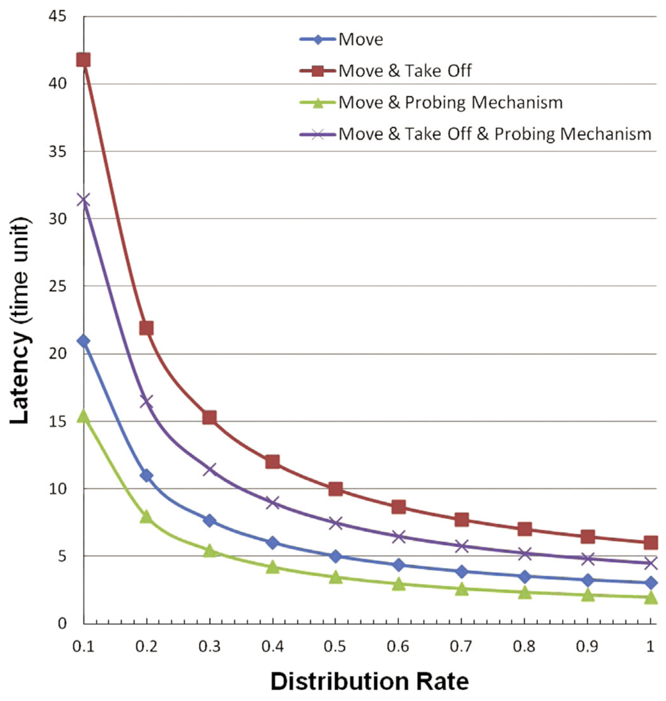

Fig. 6. Latency assessment at varying of the rate of the distribution.

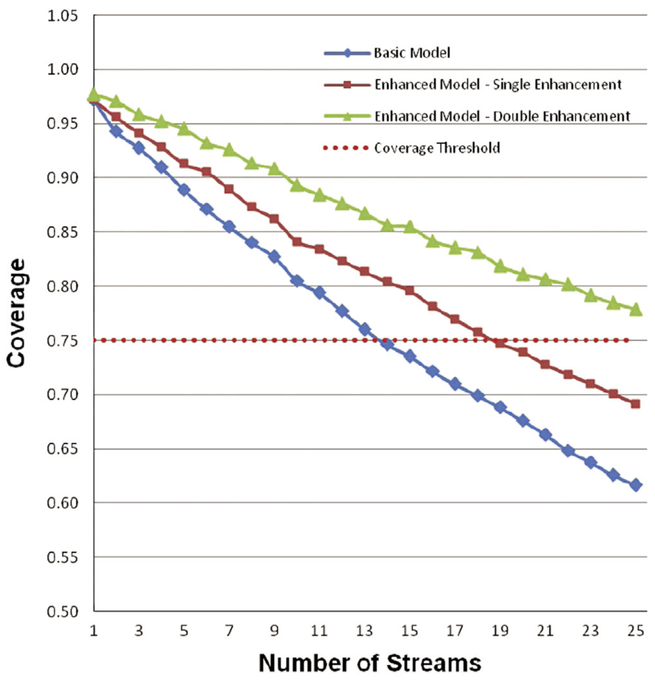

Fig. 7. Coverage assessment at varying of number of streams request.

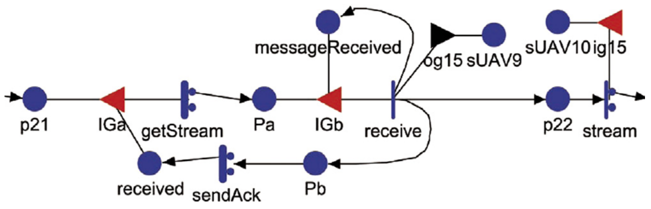

Fig. 8. Portion of the Connector model, replaced by the Retry mechanism.

shown. As explained in [16], the mechanism models are developed according to rules that allow us to automate the procedure for embedding them in the model of the synthesised Connector.

6. Related work

A number of solutions have been presented in the literature to deal with critical applications requiring degrees of fault tolerance or efficiency, including classical architectural mechanisms such as N-Version Programming, Recovery Block, Cooperative Backup and dynamic mechanisms such as dynamic function allocation [26–29]. The set of dependability mechanisms considered in this paper has fully inspired them. There are also studies where the most popular fault tolerance solutions are discussed and classified according to a variety of key characteristics, e.g., the main direct benefits of the mechanism in terms of improved system resilience or assurance of key system properties and the types of system within which this mechanism can be applied [26], with the intention to assist in the selection and configuration of mechanisms at design-time and run-time. Also, modelling and analyses of fault tolerant systems have been widely pursued, to help understanding the adequacy of employed fault tolerance mechanisms in terms of dependability, resilience and security indicators, under a variety of failure modes and critical system scenarios. Applied approaches include reliability block diagrams, fault trees, Markov models, Petri nets, Bayesian models, semantic technologies; a very rich literature has been produced which documents performed studies (such as [27,30–33] just to mention a few). A review of several technologies supporting dynamic selection of components and services, such as Multi-Agent Systems, GRID computing, Web Services, is also included in [26]. In [34] the authors address the interoperability problems raised by semantic heterogeneities in Web service communities by using a context-based approach that separates shared knowledge from the local contexts of Web service providers and developing mediation mechanisms that handle semantic data discrepancies between Web services and communities, thus enabling seamless interoperation at the semantic level. In [35] the authors examine challenges to interoperability; classify the types of heterogeneities that can occur between interacting services and present a possible solution for data interoperability using the mapping support provided by WSDL-S. Here, we are not tight to any specific technology, but rather we are focusing on generally applicable solutions in the specific context of dependable networks interoperability in heterogeneous environments.

Solutions for automatic addition of fault tolerance to fault intolerant programs have also been proposed, both resorting to a set of symbolic heuristics for synthesizing fault-tolerant distributed programs such as in [36] and by developing formal algorithms working in presence of a specific set of faults, a safety condition, and a reachability constraint as in [37], where it is also shown that the synthesis problem in the context of distributed programs is NP-complete in the state space of the given intolerant program. Further investigations on theoretical aspects and the development of a software framework for the synthesis of fault-tolerant programs are in [38].

Also, in the context of *reflective* systems, i.e., systems having the property of enabling observation and control of its own structure and behaviour from outside itself, independent development of fault tolerance libraries and applications has been carried out as a means to enhance the flexibility of dependable systems. The aim was to provide properties like ease of use and transparency of mechanisms for the application programmer, seamless reuse and extension of both functional and non-functional software and composition of mechanisms [39]. Moving to autonomic systems [40], their control loop requires assistance by a methodology and supporting technique to select and apply countermeasures to faults experienced during the system lifetime, possibly resorting to a library of fault tolerance patterns. However, to the best of our knowledge, there is no general approach in the literature, developed to select from a library of fault tolerance mechanisms, whose efficacy to meet specified dependability and performance requirements is checked through model-based analysis. Our work has been tailored to explore this direction.

7. Conclusions

We have illustrated an approach to automate the selection of a suitable dependability mechanism. This has been discussed in the context of networks of interoperable systems, as targeted by the EU project Connect. In particular, the approach targets generic mechanisms based on the synthesis of mediating software bridges (Connectors) that allow interoperability among heterogeneous devices. In addition, we have investigated a generic method for identifying elements in the Connector that must be reinforced with the selected dependability mechanism in order to improve performance- and dependability-related metrics. The approach has been demonstrated through a case study based on the GMES European Programme. Specifically, a scenario describing the management of forest-fire at the border between two different countries has been considered. In this scenario, resources provided by both countries provide the same functionalities but use different protocols and therefore a Connector is necessary to allow interoperation.

The proposed method for automating the selection of dependability mechanisms is based on ontologies. Alternative methods could be based on a sensitivity analysis. In fact, a sensitivity analysis, carried out on ranges of values for the model parameters, would point out which model parameters mostly impact the dependability measure that must be improved (e.g., time to complete a transition and its failure probability). Then, knowing the target parameter(s) values able to satisfy the dependability or performance metric under analysis, the most suitable dependability mechanism qualified to improve that (those) parameter (s) can be identified.

The work presented in this paper sets the basis for an automated approach to select a dependability mechanism and how to apply them on a model of the system in order to meet given requirements. Further investigations would be valuable to carry on, especially in the directions of (i) detailing the ontological matching approach by defining exemplary application-dependent ontologies for widely used networked applications; (ii) making more efficient the strategy devoted to select the elements where the dependability mechanism has to be applied by investigating methods to predict, with a satisfactory level of accuracy, the minimum set of elements that would be necessary for requirement satisfaction within the just one additional analysis phase; (iii) implement these methods in the dependability assessment tool (the DePer unit under development in the Connect project).

Acknowledgements

This work is partially supported by the EU FP7 Project Connect (FP7-231167), and by CHI+MED (Computer Human Interaction for Medical Devices, EPSRC research Grant EP/G059063/1).

References

[1] Avizienis A, Laprie J-C, Randell B, Landwehr C. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Dependable Secur Comput 2004;1(1):11–33. http://dx.doi.org/10.1109/TDSC.2004.2.
[2] France R, Rumpe B. Model-driven development of complex software: a research roadmap. In: 2007 Future of software engineering, FOSE '07. Washington, DC, USA: IEEE Computer Society; 2007. p. 37–54. doi: http://dx.doi.org/10.1109/FOSE.2007.14.
[3] Avizienis A, Kelly JPJ. Fault tolerance by design diversity: concepts and experiments. Computer 1984;17(8):67–80. http://dx.doi.org/10.1109/MC.1984.1659219.
[4] Masci P, Furniss D, Curzon P, Harrison M, Blandford A. Supporting field investigators with pvs: a case study in the healthcare domain. In: Avgeriou P, editor. Software engineering for resilient systems. Lecture notes in computer science, vol. 7527. Berlin, Heidelberg: Springer; 2012. p. 150–64.
[5] US Food and Drug Administration, FDA Safety Communication: Preventing Surgical Fires (October 2011). URL ⟨http://www.fda.gov/MedicalDevices/Safety/AlertsandNotices/ucm275189.htm⟩.
[6] Yellin DM, Strom RE. Protocol specifications and component adaptors. ACM Trans Program Lang Syst 1997;19(2):292–333.
[7] Cheng B, Lemos R, Giese H, Inverardi P, Magee J, Andersson J, et al. Software engineering for self-adaptive systems: a research roadmap. In: Software engineering for self-adaptive systems. Lecture notes in computer science, vol. 5525. Berlin, Heidelberg: Springer; 2009. p. 1–26.
[8] Pastrana J, Pimentel E, Katrib M. Qos-enabled and self-adaptive connectors for web services composition and coordination. Comput Lang Syst Struct 2011;37(1):2–23.
[9] Alam S, Chowdhury M, Noll J. Interoperability of security-enabled internet of things. Wirel Pers Commun 2011;61(3):567–86.
[10] Kyusakov R, Eliasson J, Delsing J, van Deventer J, Gustafsson J. Integration of wireless sensor and actuator nodes with it infrastructure using service-oriented architecture. IEEE Trans Ind Informatics 2013;9(1):43–51.
[11] Spalazzese R, Inverardi P. Mediating connector patterns for components interoperability. In: Proceedings of the fourth European conference on software architecture, ECSA'10. Berlin, Heidelberg: Springer-Verlag; 2010. p. 335–43. URL ⟨http://dl.acm.org/citation.cfm?id=1887899.1887928⟩.
[12] Fiaschetti A, Suraci V, Priscoli FD. The shield framework: how to control security, privacy and dependability in complex systems. In: Complexity in engineering (COMPENG 2012). Aachen, Germany: IEEE; 2012. p. 1–4.
[13] Di Marco A, Inverardi P, Spalazzese R. Synthesizing self-adaptive connectors meeting functional and performance concerns. In: Proceedings of the eighth international symposium on software engineering for sdaptive and self-managing systems, SEAMS'13. San Francisco, CA, USA: IEEE Press; 2013. p. 133–42.
[14] Di Giandomenico F, Kwiatkowska M, Martinucci M, Masci P, Qu H. Dependability analysis and verification for connected systems. In: Margaria T, Steffen B, editors. Proceedings of ISOLA 2010—leveraging applications of formal methods, verification, and validation. Lecture notes in computer science. Heraklion, Crete, Greece: Springer; 2010. p. 263–77.
[15] Masci P, Martinucci M, Giandomenico FD. Towards automated dependability analysis of dynamically connected systems. In: Proceedings of the 2011 10th international symposium on autonomous decentralized systems, ISADS '11. Washington, DC, USA: IEEE Computer Society; 2011. p. 139–46. doi: http://dx.doi.org/10.1109/ISADS.2011.23.
[16] Masci P, Nostro N, Di Giandomenico F. On enabling dependability assurance in heterogeneous networks through automated model-based analysis. In: Proceedings of the third international conference on software engineering for resilient systems, SERENE'11. Berlin, Heidelberg: Springer-Verlag; 2011. p. 78–92. URL ⟨http://dl.acm.org/citation.cfm?id=2045537.2045548⟩.
[17] Meinke K, Walkinshaw N. Model-based testing and model inference. In: Margaria T, Steffen B, editors. Leveraging applications of formal methods, verification and validation. Technologies for mastering change. Lecture notes in computer science, vol. 7609. Berlin, Heidelberg: Springer; 2012. p. 440–43.
[18] Gruber TR, Olsen GR. An ontology for engineering mathematics. In: Doyle J, Sandewall E, Torasso P, editors. KR. Bonn, Germany: Morgan Kaufmann; 1994. p. 258–69.
[19] Bertolino A, Calabro A, Di Giandomenico F, Nostro N, Inverardi P, Spalazzese R. On-the-fly dependable mediation between heterogeneous networked systems. In: ICSOFT 2011, CCIS 303. Berlin, Heidelberg: Springer Verlag; 2013. p. 20–37.
[20] IEEE std 610.12-1990 (n.d.). IEEE standard glossary of software engineering terminology. ⟨http://ieeexplore.ieee.org/⟩; 1990.
[21] Ann JFM, Tai T, Avizienis A. Software performability: from concepts to applications. Norwell, MA, USA: Kluwer Academic Publisher; 1996.
[22] CONNECT Consortium, Deliverable 6.3—experiment scenarios, prototypes and report iteration 2; 2012.
[23] Sanders WH, Meyer JF. Stochastic activity networks: formal definitions and concepts. Lectures on formal methods and performance analysis, Ed Brinksma, Holger Hermanns, and Joost-Pieter Katoen (Eds.). Springer Lectures On Formal Methods And Performance Analysis, Vol. 2090, Springer-Verlag New York, Inc., New York, NY, USA, 2002. p. 315–343.
[24] Daly D, Deavours DD, Doyle JM, Webster PG, Sanders WH. Möbius: ana extensible tool for performance and dependability modeling. In: Haverkort BR, Bohnenkamp HC, Smith CU, editors. The 11th international conference, TOOLS 2000. Lecture notes in computer science, vol. 1786. Schaumburg, IL, USA: Springer Verlag; 2000. p. 332–36.
[25] CONNECT Consortium, Deliverable 5.2—design of approaches for dependability and initial prototypes; 2011.
[26] ReSIST Consortium, EU project ReSIST: resilience for survivability in IST. deliverable d11: support for resilience-explicit computing—first edition, Technical report. ⟨http://www.resist-noe.org/⟩; 2007.
[27] Lyu MR. Software fault tolerance. New York, NY, USA: John Wiley & Sons, Inc.; 1995.
[28] Killijian M-O, Courtes L, Powell D. A survey of cooperative backup mechanisms. IAAS Technical report 06472; October 2006.
[29] Thildebrandt M, Harrison M. The temporal dimension of dynamic function allocation. In: Proceeding of 11th European conference on cognitive ergonomics (ECCE); 2002. p. 283–91.
[30] Ghosh R, Kim D, Trivedi KS. System resiliency quantification using non-state-space and state-space analytic models. Reliab Eng Syst Saf 2013;116:109–25.
[31] Dominguez-Garcia AD, Kassakian JG, Schindall JE, Zinchuk JJ. An integrated methodology for the dynamic performance and reliability evaluation of fault-tolerant systems. Reliab Eng Syst Saf 2008;93(11):1628–49.
[32] Flammini F, Marrone S, Mazzocca N, Vittorini V. A new modeling approach to the safety evaluation of n-modular redundant computer systems in presence of imperfect maintenance. Reliab Eng Syst Saf 2009;94(9):1422–32.
[33] Fiaschetti A, Lavorato F, Suraci V, Palo A, Taglialatela A, Morgagni A, et al. On the use of semantic technologies to model and control security, privacy and dependability in complex systems. In: SAFECOMP; 2011. p. 467–79.
[34] Mrissa M, Dietze S, Thiran P, Ghedira C, Benslimane D, Maamar Z. Context-based semantic mediation in web service communities. In: King I, Baeza-Yates R, editors. Weaving services and people on the world wide web. Berlin, Heidelberg: Springer; 2009. p. 49–66.
[35] Nagarajan M, Verma K, Sheth A, Miller J, Lathem J. Semantic interoperability of web services—challenges and experiences. In: International conference on web services, 2006. ICWS '06; 2006. p. 373–82.
[36] Bonakdarpour B, Kulkarni SS. Sycraft: a tool for synthesizing distributed fault-tolerant programs. In: CONCUR 2008: international conference on concurrency theory. Toronto, Canada: Springer Verlag; 2008. p. 167–71.
[37] Kulkarni SS, Arora A. Automating the addition of fault-tolerance. In: The 6th formal techniques in real-time and fault-tolerant systems; 2000. p. 82–93.
[38] Ebnenasir A. Automatic synthesis of fault-tolerance [Ph.D. dissertation]. Michigan State University; 2005.
[39] Ruiz J, Killijian M-O, Fabre J-C, Thvenod-Fosse P. Reflective fault-tolerant systems: from experience to challenges. IEEE Trans Comput 2003;52(2):237–54.
[40] Huebscher MC, McCann JA. A survey of autonomic computing: degrees, models, and applications. ACM Comput Surv 2008;40(3):237–54.