Adaptive Execution of Workflows in Emergency Response

Massimo Cossentino
National Research Council of Italy (CNR)
massimo.cossentino@icar.cnr.it

Davide Andrea Guastella
Machine Learning Group
Université Libre de Bruxelles
davide.andrea.guastella@ulb.be

Salvatore Lopes
National Research Council of Italy (CNR)
salvatore.lopes@icar.cnr.it

Luca Sabatucci
National Research Council of Italy (CNR)
luca.sabatucci@icar.cnr.it

ABSTRACT

In emergencies, preparation is of paramount importance but it is not sufficient. As we know, emergency agencies develop extensive (text) plans to deal with accidents that could occur in their territories; their personnel train to enact such procedures, but, despite that, the unpredictable conditions that occur during an emergency require the ability to adapt the plan promptly. This paper deals with the last mile of a process we defined for enabling the adaptive execution of such emergency plans. In previous works, we discussed how to convert a free-text plan into a structured-text form, represent this plan using standard modelling notations, and extract goals that plans prescribe to be fulfilled. In this paper, we propose an approach for executing these plans with a workflow execution engine enriched by the capability to support runtime adaptation.

Keywords

Emergency management; Disaster response; Adaptive workflow execution; Standard Operating Procedure (SOP)

INTRODUCTION

Disasters such as earthquakes, floods, hurricanes, tsunamis, terror attacks, and disease epidemics affect citizens' lives and several socio-economic aspects of societies. Emergency response must address a twofold objective. The first one is the efficient allocation of resources and prompt enactment of the procedures that can handle the emergency. The second objective is adapting the planned procedure to cope with unforeseen events and environmental changes, often occurring during the enactment of the planned emergency procedures.

Public organizations and administrations continuously aim at improving the effectiveness of disaster management through preparedness and estimation of disasters to limit their impact. Nevertheless, disaster management must consider unpredictable events in highly dynamic environments. The challenge is to improve performance and strategically align ICT use with emergency management's needs and requirements of emergency management (Hellmund et al. 2021). Integrating ICT in the emergency domain is challenging: usually, emergency plans are written in a free-form text by domain experts (who are not necessarily ICT experts) (Cossentino, Guastella, et al. 2022a), and may not express the actions to pursue under specific environmental conditions, requirements or evolution of the emergency. In this case, ICT must provide methods and tools that adapt to dynamic environments, thus allowing coping with unforeseen events and ensuring a prompt response to disasters.

In previous papers, we discussed how to convert the free-text emergency plan into a structured-text form, how to represent this plan using standard modelling notations, and how to extract the goals that the plan prescribes to be fulfilled. Now, we propose an approach for executing these plans using a BPMN-compliant workflow execution engine enriched by the capability to support runtime adaptation. The novelty of our proposal lies in the conception of a transformation process that, starting from the goals extracted from the free-text plan, can ensure the generation of a fulfilling workflow that may be executed by a reliable industrial tool. The peculiarity of this workflow is to be

able to adapt at runtime because of the introduction of some specifically conceived decoration layers in the BPMN description of the plan.

Our proposal is being developed in the context of the EU-funded N.E.T.TUN.IT project, whose goal is to improve the collaboration of different countries in the Mediterranean Sea Sicilian channel in case of an emergency that could affect several of them. One of the contributions of the ICT partners in the project is to define a generic pipeline for going from free-text emergency plans to executable plans using process execution engines supporting the standard BPMN 2.0 notation. Herein we refer to *generic* as the technique is not specific to an environment, a rescue context, or an emergency plan. Instead, we aim to define a method that can be coupled with existing process execution engines to continuously monitor emergency tasks and adaptation capabilities in case of unforeseen events. The steps related to converting the initial free-text plan into a structured-text version, its representation in the form of BPMN, CMN, DMN models, and identifying the plan goals have already been discussed in Cossentino, Guastella, et al. 2022a; Cossentino, Guastella, et al. 2022b.

This paper is organized as follows: first, we introduce the problem of the adaptive workflow execution in the context of emergency management, and we position this work according to our previous results. Then, we discuss state-of-the-art propositions to support emergency procedures with ICT solutions, particularly by using workflow management systems. Then, we present a method that introduces the adaptive execution of workflows in the emergency domain. Then, we present the architecture that composes the N.E.T.TUN.IT software system, and we describe how the proposed approach relates to the other components of the system that are needed to fulfill the emergency goals. Finally, we summarize the work in the conclusions and point out future works.

THE PROBLEM

The critical aspect of the transition from free-form text plans to executable emergency workflows is the management of tasks and resources when unpredictable events occur. Nowadays, to the best of our knowledge, there is no commercial workflow management system that allows supporting the adaptive execution of emergency workflows. This creates a gap between the reliability of commercial products and the need for adaptive workflow execution: on the one hand, it is preferable to employ reliable products for executing processes involving critical activities as in the emergency domain; on the other hand, there is a need for adaptive execution of the workflow, which is not provided yet by these tools. To address this issue, this paper proposes a methodology for supporting the adaptive execution of emergency workflow that integrates with any process execution engine supporting standard BPMN 2.0 notation.

Our goal is to conceive a methodology to move from a free-text emergency plan towards the execution of the corresponding supporting workflow. The proposed methodology is currently being developed in the context of the N.E.T.TUN.IT[1] project. This project aims at implementing a fully operational platform for cross-border data collaboration to cope with risks and disasters due to emergency scenarios that involve both partners from Italian and Tunisian countries. The project objective is that the Italian and Tunisian sides collaborate in response to a simulated accident causing health risks and atmospheric and marine pollution that potentially impact the population on both the coasts of the Sicilian channel. In such a context, one of the challenges is that two different nations with different legal frameworks should cooperate in the ongoing emergence.

Figure 1 shows the three macro-phases that allows addressing the previous goal: 1) structuring the free-text emergency plan, 2) modelling the emergency plan using standard business notations, 3) producing executable workflows.

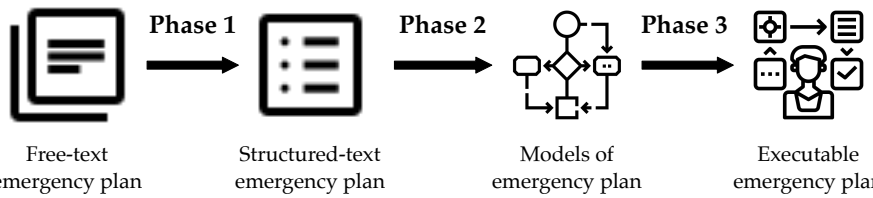

Figure 1. The proposed process for transitioning from free-text emergency plans to executable plans.

The objective of phase 1 is to manually process a free-text plan to reduce the ambiguity of the natural language and to obtain a structured, not-ambiguous version of the same document, preserving, organizing and correlating all the

[1]Net de l'Environnement Transfrontalièr TUNisie-ITalie (N.E.T.TUN.IT), Cross-border Cooperation Programme of the EU Community, Italy-Tunisia 2014-2020, project started on 03.09.2020 and concluding on 30.09.2023, https://www.italietunisie.eu/projets/les-projets/nettunit/

essential information. For this purpose, we created a semantic layer that highlights what is important and which kinds of relationships are essential and must be highlighted (Cossentino, Lopes, et al. 2021).

The objective of phase 2 is to represent the emergency plan in a notation that enables the automatic execution of the plan, eliminating any ambiguity of interpretation, clarifying tasks and responsibilities of each role, and highlighting all criteria for decisions to take during the emergency (Cossentino, Guastella, et al. 2022a). It is worth noting that both the previous phases require the intervention of requirement analysts and system designers to perform some of the prescribed activities.

This paper completes the pipeline shown in Figure 1. In particular, it deals with phase 3 by proposing a method to support the adaptive execution of emergency procedures under the constraint of employing industrial process execution engines supporting the standard BPMN 2.0 notation.

STATE OF THE ART

Most emergency plans are defined by competent agencies using a free text form. Often, the country's legislation profoundly impacts the general outline and several parts of that. A plan, for instance, describes the stakeholders' responsibilities in managing specific aspects of the emergencies like a fire, a mountain rescue, and so on.

In (Sell and Braun 2009), the authors identify an interesting list of problems related to the usage of paper-based plans in emergency management: *(i)* limited situation awareness, *(ii)* lack of support for resource management, *(iii)* lack of flexibility in adapting the plan to the mutable needs of the emergency response, *(iv)* no support for the delegation of tasks from one actor to another. In response to these problems, many authors agree that a WF-based approach to a computer-supported solution may be useful if it satisfies the following requirements: *(i)* listing and allocation of resources to the activities, *(ii)* representation of the current state of advancement in the execution of the workflow, *(iii)* adaptation of the plan in terms of deletion/insertion of activities and resource allocation, *(iv)* delegation of responsibility for the execution of some task and resource management, *(v)* execution of the workflow.

In this section, we also report some interesting contributions, from the literature, about the issues (and challenges) related to the execution in the form of a workflow of a text plan. We will not deal with the previously cited legal concerns because of the specific scope of this work.

Sell and Braun 2009 propose a workflow metamodel that resembles the BPMN one, with interesting novelties like the explicit introduction of resources and delegation; besides, as regards the need for adaptation, they define that as: "deletion, addition and configuration of activities, connections and resource allocations". They do not deal with the conception of alternative plans when one of the current activities cannot be completed (for instance because of lacking resources). They leave the definition and introduction of new activities to the user, providing no computerized support for that.

In (Peinel et al. 2012), the authors analyze an interesting series of experiences done in research projects also with stakeholders in the field. From their experiences, they deduced some considerations about applying conventional workflow management systems (WFMS) to the emergency management (EM) domain. They see a significant limitation of current approaches because they are motivated by cost/time optimization, which is not the primary factor in emergency response (although time often is). The authors suggest that in this domain, the plan and the execution of its activities are primarily goal-oriented and give great relevance to this aspect. An interesting comparison between conventional BPMN features and the requirements for an application to EM provides other insights like the need (for EM) to deal with skeletons of plans to be filled at the time of the emergency rather than with completely detailed plans to be developed before that, again, as in other works we found the requirement to deal with resources overloading and double allocation. A peculiarity identified in this paper regards the need to support the "escalation/de-escalation" of processes consequent to the change in the level of alert; this implies the conception of specific transition procedures that are not part of conventional WFMS.

Despite there is general agreement about the need for dynamic workflows, a debate is still in progress about which is the best way to capture the dynamism of the business process within the workflow model.

One of the first approaches was anticipating changes during the business analysis phase, thus including exceptions in the process-logic (Ceri et al. 1997). To support this solution, modern BPM languages directly provide the exception mechanism (OMG 2011). However, anticipating problems at design time has shortcomings: 1) modelling adaptation within the basic control flow sometimes causes exponential growth on the model to be developed; 2) designers are generally not capable of predicting all the possible exceptions and events beforehand and capturing them in the design of a workflow; moreover, due to dynamic contexts, understanding when and how flow deviations may appear may be difficult (Strong and Miller 1995). One alternative approach consists in providing run-time flexibility to the process. Supporting run-time modification of the workflow ensures the ability to overcome unexpected situations

while maintaining the size of the workflow model (Reichert and Dadam 1998; Kammer et al. 2000). It is necessary to consider that removing any control or constraints wipes out the very reason for which workflow technology was introduced. The challenge is to solve a trade-off between rigidity and flexibility.

In particular, Reichert and Dadam 1998 present a conceptual and operational framework for the support of dynamic structural changes of workflows, preserving correctness and consistency. They advocate the need to separate the application's control structures from the implementation of its tasks. This contributes to simplifying and fastening application development, and it enables run-time coordination and scheduling of activities.

Modifying a process definition at run-time is crucial for workflow process execution and evolution over time. An interesting approach comes from ProAdapt (see Aschoff and Zisman 2012) that proposes to support workflow adaptation via error prediction and autonomic service composition. This framework can identify problems that cause the composition to stop its execution, but also the emergence of new requirements or the introduction of better services.

THE PROPOSED APPROACH

In the context of the N.E.T.TUN.IT project, we defined a pipeline for going from a free-text emergency plan to an executable plan. In our previous works, we have already tackled the non-trivial challenges of structuring free-form emergency plans:

- in (Cossentino, Guastella, et al. 2022b), we proposed a human-based approach for converting a free-form plan document into a structured version. The approach focuses on inconsistencies, redundancies, and ambiguities that hinder understanding and formalizing emergency plans, and tackles linguistic issues by mapping (emergency) domain concepts into a set of keywords; this mapping process results in a structured version of an emergency plan that does not present ambiguities or flaws deriving from the use of natural language;
- in (Cossentino, Guastella, et al. 2022a), we proposed a novel notation that is based on *de facto* business process standards (BPMN, CMMN, DMN, see Herrera and Díaz 2019). The proposed notation allows describing dependencies and relationships among the responsibilities of the roles involved in a specific emergency, and also the events that trigger the need for adapting the emergency plans.

We complete the definition of the pipeline by proposing a technique to support the adaptive execution of workflows fulfilling emergency procedures modeled from a set of plans in structured and semi-structured text. The proposed method provides adaptive execution capability to most of the commercial workflow management system supporting BPMN 2.0. Figure 2 shows the main steps of the approach to support the adaptive execution of workflows in the emergency domain and the intermediate results from each step. Steps in the figure are labeled as numbers in green circles (❶ to ❻).

The input of the overall approach is a free-text emergency plan converted to a structured text (Phase 1) and then converted to a model including BPMN, DMN, CMMN diagrams, and goal diagrams (Phase 2). The contribution of this paper focuses on phase 3, which objective is to support the adaptive execution of the emergency plan using a BPMN process engine. The input of phase 3 is a goal model expressed using the GoalSPEC notation (Sabatucci, Ribino, et al. 2013). GoalSPEC is a formalism for expressing objectives to be pursued within a contingency plan through a formal notation. During the second phase of the process in Figure 1, requirement engineers define goals for the emergency response. In this phase, they are supported by structured text documents that are a semi-formal description of the emergency plan in which some collaboration patterns have been identified and highlighted. They also are supported by classical BPMN, CMMN and DMN models describing various aspects of an emergency response. The outcome is a description of the expected results during an emergency. In practice, goals are formalized as a combination of linear temporal logic and a goal-tree structure that represent what is desired to achieve during the emergency response. One of the principle, when writing goals, is to minimize the constraints of temporal dependencies among the activities: this is the key for defining the space for future adaptation. For more details about emergency goals and how to identify and refine them, please refer to (Cossentino, Guastella, et al. 2022b; Cossentino, Guastella, et al. 2022a).

GoalSPEC is the input of a pipeline of automatic model transformation in phase 3 (see the Plan Generator and Grounder box in Figure 2). It is responsible of creating an emergency response plan (as a workflow) by composing available actions thus to fulfill input the expected objectives, but leaving space for possible workflow reshaping when

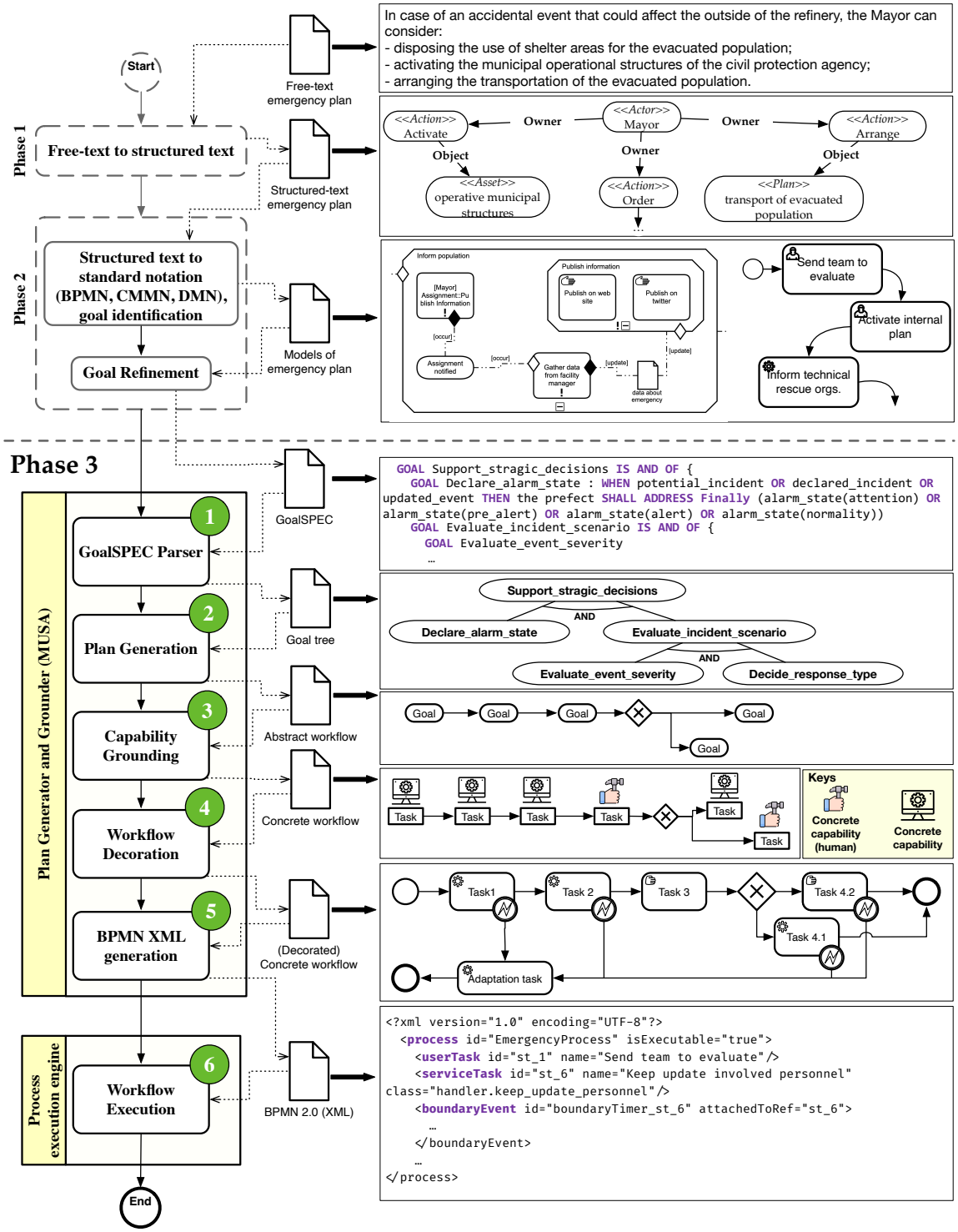

Figure 2. Main steps of the proposed method to support the adaptive execution of workflows for emergency plans. The dashed tasks refer to phases 1 and 2, that are the object of our previous works.

unexpected things will happen. This pipeline is based on MUSA (Middleware for User-driven Service Adaptation)[2], an open-source tool for generating the workflows that fulfill the emergency goals (Sabatucci and Cossentino 2015). MUSA is a middleware for self-adaptation for implementing dynamic workflows. It enables decoupling the "*what*" (i.e. the goals to achieve) from the "*how*" (the set of capabilities used to fulfill the goals). The peculiarity of MUSA is the adoption of run-time models for goals and capabilities and an engine for performing a Proactive Means-end Reasoning (PMR, see Sabatucci and Cossentino 2015). The PMR sub-system creates several alternative abstract

[2] https://github.com/icar-aose/adaptivity-framework

workflow solutions that find a concretization in the binding phase, where abstract capabilities are realized by real services composed to fulfill the goals, thus obtaining the executable workflow. If the selected solution cannot accomplish its objectives, for instance because of a fault in some service, one of the alternative solutions may be used or MUSA may trigger a new PMR session thus to compute new solutions that properly take into account the current state of the world and available capabilities.

The parsing operation in step ❶ transforms the goals from text format to an internal representation that can be used in subsequent modeling steps.

In step ❷, the goals are injected into MUSA, which derives an abstract workflow by finding a functional bridge between goals and capabilities. Capabilities are independent computational units that bind services whose execution allows for addressing the input goals. The capabilities are stored in a centralized computation unit and can invoke local or remote services (for example, on the cloud). The workflows provide a technological means to support emergency operations, specifically by managing cross-border communications in the context of the N.E.T.TUN.IT project, to promptly address the rescue operations and avoid loss of human lives.

In step ❸, MUSA grounds the capabilities in the abstract workflow to concrete capabilities: this grounding operation associates each capability in the abstract workflow with an executable service (a concrete capability). The step ❸ results in a concrete workflow. In step ❹, the concrete workflow is enriched with constructs that allow supporting the adaptive execution of the emergency plan. In step ❺ the solution is derived into a standard BPMN 2.0 notation for the execution with a process execution engine (step ❻).

The process in Figure 2 is executed after the activation of a new emergency plan. In N.E.T.TUN.IT, we assume that the decision to activate a new emergency plan, and consequently to organize a workflow to support the plan, stems from the need for cross-border cooperation to cope with emergencies. Typically, in the Italian legislature, the prefect is the actor who decides whether to activate emergency plans.

The capabilities included in the output workflow are chosen by the Plan Generation component: a capability is included in a workflow if it addresses one or more input goals, and if doesn't violate some goal constraints (such as temporal prescriptions or resource availability).

In the following section, we detail the proposed method.

Plan Generation (step ❷)

We use MUSA (Sabatucci and Cossentino 2019) to produce an executable workflow starting from a set of goals that specify the objectives of an emergency plan.

Declarative Specification of Capabilities

The main building block of a MUSA solution is the capability, the means by which goals can be fulfilled. The concept of capability comes from artificial intelligence (planning actions, see Gelfond and Lifschitz 1998) and service-oriented architecture (micro-services, see Namiot and Sneps-Sneppe 2014). Indeed, MUSA considers two different levels of abstraction for capabilities: **abstract capability**, that is a description of the effect of an action that can be performed (together with required pre-conditions), and **concrete capability** that is an independent, composable unit of computation able to produce some concrete result.

An example of description of capabilities is provided in (Sabatucci, Lodato, et al. 2015) in the context of a smart travel domain: each capability encapsulates a web service for reserving some kind of travel service (hotel, flight, local events).

This solution has the following benefits:

- each capability is relatively small, and therefore easier for a developer to implement,
- it can be deployed independently of other capabilities,
- it is easier to organize the overall development effort around multiple teams,
- it supports self-adaptation because of improved fault isolation.

The abstract capability language is a STRIP-like language (Fikes and Nilsson 1971), and it presents many similarities with PDDL (Edelkamp and Hoffmann 2004). The BNF grammar is summarized here:

```
capability := (capability) (identifier)
              ({) params? pre post scenario+ (})
params     := (params:) data_list
data_list  := param (,) data_list
            | param
param      := (identifier) (−) type
pre        := (pre:) formula
post       := (post:) formula
scenario   := (scenario) (identifier) ([) evolution (])
evolution  := effect
            | evolution (,) effect
effect     := (add) predicate
            | (remove) predicate
```

where *formula* is a FOL logic formula

In the paper (Cossentino, Sabatucci, et al. 2018), we conducted an experiment to support the claim that decoupling abstract and concrete capabilities makes it easier to deploy frequently new versions of the software. By exploiting three different applications, we got some findings about the easiness of continuously evolving a system. The conclusions are that providing capabilities as run-time entities constitutes the basis for continuous data exchange between human and agents, and therefore it leads to system evolution.

Self-Configuring a Solution

In our approach, goals and capabilities are two fundamental run-time entities for enabling self-adaptation. To support automatic configuration of *ad hoc* solutions, we provide a reasoner that selects and associates capabilities to goals (Sabatucci and Cossentino 2015). The basic idea consists in exploring a space of solutions where goals represent points of the space that must be reached, and capabilities provide evolution-functions that allow to move through the space.

Self-configuration is a space search problem. The algorithm for solving this problem is a symbolic nondeterministic planning algorithm. It incrementally builds a *computational graph* where nodes represent stable states the system may generate when executing its capabilities. The algorithm denotes a symbolic representation of the world (Newell 1982) because it deals with categories rather than with instances of objects (that would have disastrous consequences in terms of temporal and spatial complexity): ignoring a number of distinctions –that are irrelevant for the solution– allows working with an abstract view of the problem space.

On the other side, the use of nondeterministic capabilities, (actions with multiple alternative outcomes) allows creating decision points and loops into the structure. In particular, for rendering a plan as a workflow, we exploit three control flow patterns: sequence, exclusive-choice, and structured-loop. In a *sequence*, the completion of a task enables the execution of the next one in the process. The Sequence pattern serves as the fundamental building block for processes. In an *exclusive-choice* pattern (also known as XOR-split), a branch splits into two or more branches such that when the incoming branch is enabled, the thread of control is immediately passed to one of the outgoing branches based on a design-time condition. A *structured-loop* pattern describes the ability to execute a task repeatedly. The loop is characterized by a condition that is either evaluated at the beginning or the end of the loop (with a single entry and exit point).

Capability Grounding (step ❸)

The output from the previous step is an **abstract workflow**. We distinguish abstract workflows from concrete workflows: the former consist of abstract capabilities, and provide a high-level description as well as the flow of operations to be performed, without any reference on how activities should be performed. The latter is a realization of abstract workflows, that is, having the same semantic content (same flow, same capabilities) but abstract capabilities are grounded to concrete capabilities.

The separation between abstract and concrete workflow is mainly driven by reusability and abstraction objectives. A workflow that is tightly coupled to the real services cannot be re-used in different contexts. Herein, the context refers to parameters such as the environmental configuration or resources availability. For example, in the emergency domain, the environmental conditions could suggest a different resource allocation, or a different sequence of execution of the emergency operations. A solution could consist in defining many emergency plans, one for each possible environmental configuration. However, this solution is unsuitable because of the unpredictable dynamics

of the physical environment; it would be extremely difficult to enumerate the plans for all the possible environmental configurations. Instead, starting from an abstract workflow definition, its concrete counterpart can be opportunely calibrated to cope with unforeseen environmental conditions without changing the abstract definition of the workflow for a specific emergency plan, thus making the same workflow definition reusable under different conditions and requirements.

We define a **grounding process** (step ❸) to specialize an abstract workflow into a concrete one. Grounding an abstract workflow basically binds each abstract capability to a concrete capability. Differently from abstract ones, concrete capabilities are Java classes that contain the code to realize a given service.

The grounding process consists of the following steps, executed in this order:

1. retrieving, from the repository, the concrete capabilities that match the services included in the abstract workflow;
2. filtering out the concrete capabilities that are not applicable due to time constraints, service or resources unavailability, at the time the workflow is grounded;
3. choosing the most pertinent concrete capability for a given abstract capability.

Filtering the capabilities requires monitoring the environment to discover the parameters that could have an impact on the execution of the committed solution.

Let us suppose we have an abstract workflow containing, among the others, the following abstract capability, devoted to report the status of the response to an emergency to one of the actors involved in the plan:

```
capability := Notify competent body
pre        := internal_plan_active(active) AND alarm_state(attention)
post       := informed_authority(prefect, attention) AND
                informed_authority(mayor, attention)
effect     := (add) informed_authority(prefect, attention)
            | (add) informed_authority(mayor, attention)
```

The abstract capability is being grounded to its concrete counterpart, which refers to a service that realizes the communication task. In the presence of multiple concrete capabilities for the same service, a user-defined strategy can be employed to prioritize capabilities, for example on the basis of some Quality-of-Service (QoS) metrics, resource availability, or time constraints.

Figure 3 shows two concrete capabilities for the abstract one described before (Notify competent body). Despite the two capabilities address the same service, they differ in the way they realize the notification functionality: in the former case (`notify_competent_body_GSM`), communication is done by a GSM device, in the latter case (`notify_competent_body_satellite`) through a satellite communication channel. We assume that when the GSM communication is not available (for instance because of a network issue), the workflow undergoes a reorganization process, and the satellite communication service is chosen to fulfill the notification goal.

The parameters of the capability have the following meaning: parameter (1) is a numerical value that uniquely identifies the concrete capability, parameter (2) is a self-explanatory name of the concrete capability, parameter (3) is a Boolean variable which value is true if the concrete capability refers to an action that must be carried out by one (or several) human operator(s), false if the capability involves operations that are executed by the software system, parameter (4) is the string identifier of the real service the capability wraps, parameter (5) is the name of the Java class containing the code executed by the capability, parameters (6) and (7) are the optional name of the Java classes respectively executed before and after the execution of the capability itself.

The output of the grounding process is a workflow where all capabilities have been grounded, that is, associated with concrete services.

Workflow Decoration (step ❹)

In the emergency domain, workflow management is typically done through commercial tools also because they deliver a high degree of reliability. However, existing commercial workflow engines do not support run-time adaptation in process execution. Herein, the unexpected events yields adaptation needs, and that requires a reconfiguration of the workflow's activities during the execution. Such unexpected events can be triggered by several

Figure 3. Example of concrete capabilities, binding an abstract service (Notify competent body) to multiple Java code implementations.

conditions: network issues, human errors, and unavailable services or resources. Commercial and open-source workflow management systems do not natively support the ability to adapt at run-time. Conversely, the customization of such tools is not a trivial task, because APIs and documentation often are not available. To tackle this issue, we propose to maintain the use of BPMN as standard way to model the process, but adding details about how to behave when adaptation is required. These additional elements enrich the original model by using the same BPMN notation; for this reason, we call them 'decoration layers', that through a combination of standard BPMN 2.0 constructs, provide support for run-time adaptation in existing workflow engines.

The application of decoration layers must ensure that the output workflow specification is compliant with the BPMN 2.0 standard. Also, the semantics of the input workflow must remain the same after the application of the decoration layers: in other words, it must always be possible to trace back to the initial, non-decorated solution. Figure 4 shows an example of concrete workflow, converted into BPMN 2.0 notation (Figure 4a), and its decorated counterpart (Figure 4b).

The workflow in Figure 4a is decorated with an **adaptation layer**, providing support for the adaptive execution. The following steps must be done, in order, to decorate a workflow with an adaptation layer:

1. decorate each domain-specific task with a boundary error event;
2. link each boundary error event to the adaptation task;
3. link the adaptation task to a boundary event.

In the adaptation layer, boundary error events catch exceptions or malfunctions that could occur during the execution of concrete capabilities related to the tasks of the workflow. Once an exception is caught, the execution flow of the workflow is diverted to the adaptation task. The adaptation task performs an adaptation request (Figure 6): the task communicates to MUSA that the execution of a task failed, and a reorganization of the emergency plan is necessary. In the subsequent plan (re)generation of the workflow, MUSA will choose another solution for addressing the same goal. After the execution of the adaptation task, the execution of the aborted workflow is replaced by the new one, computed as discussed in (Sabatucci, Lodato, et al. 2015; Sabatucci and Cossentino 2019), and it will start from the same state in which the previous workflow has been interrupted.

Multiple decoration layers can be applied on top of the same workflow. Let us suppose we want to add the support for handling time constraint failures in the workflow reported in Figure 4a. For instance, the execution of some

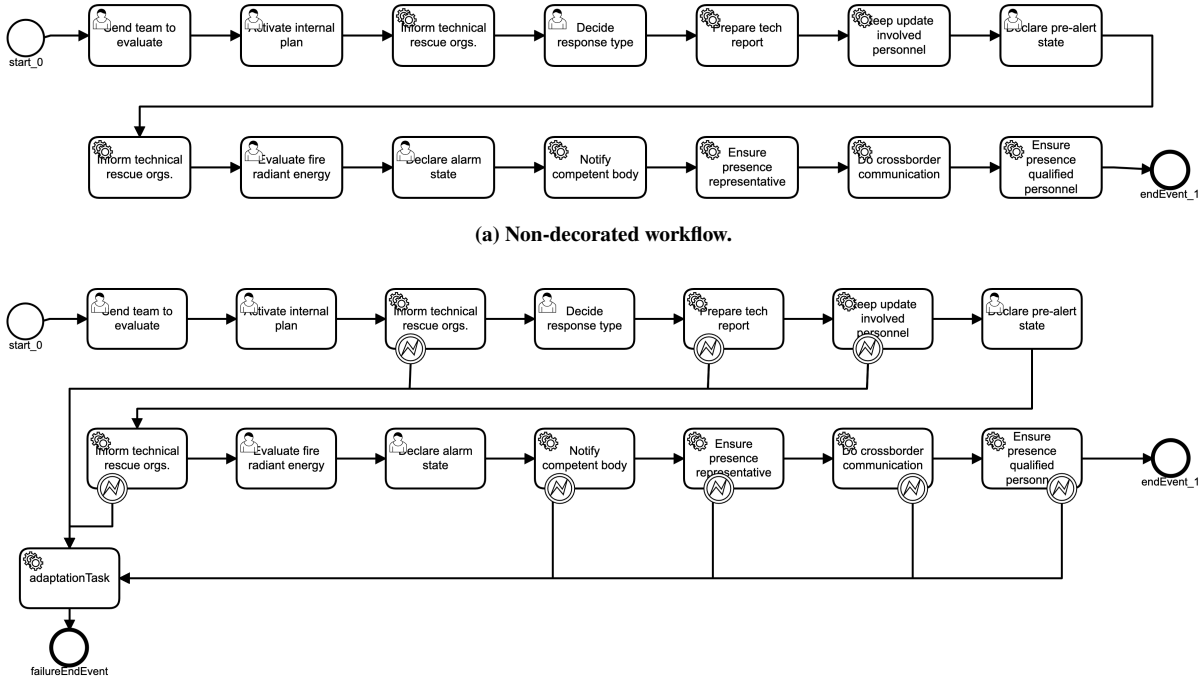

Figure 4. The decoration of the workflow (Figure 4a) through an adaptation layer undergoes a process of enrichment of the workflow with further constructs (Figure 4b), while maintaining the same semantic validity.

tasks must not last more than 10 seconds, otherwise an error is generated, and a re-organization of the workflow is needed. To handle time constraints, we add a decoration layer that adds boundary timer events to tasks. Each boundary timer event is associated with a rule that triggers the event in case of time constraint violation and deviates the execution flow towards the adaptation task (just like we reported before for the adaptation layer). This results in the workflow proposed in Figure 5, that combines the two discussed decoration layers: one for handling tasks failure, one for handling time constraints.

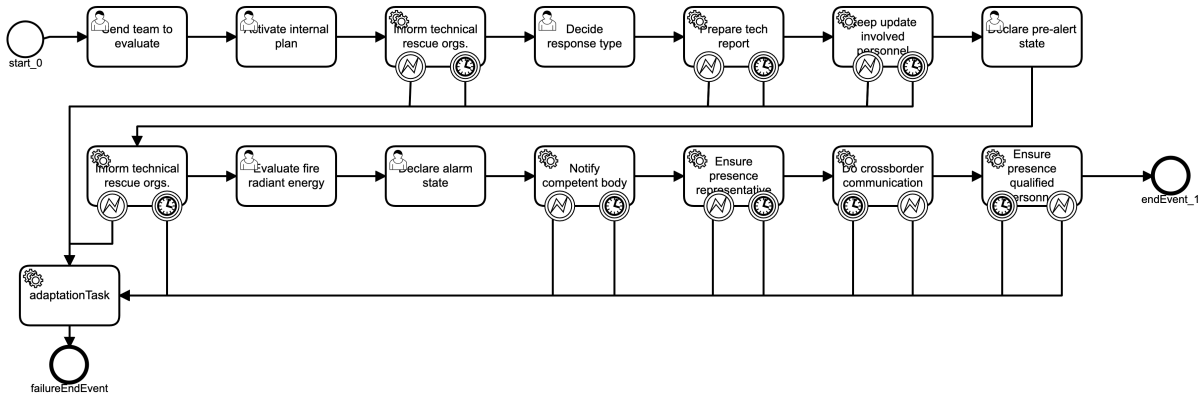

Figure 5. Result of the decoration of the workflow in Figure 4a with a layer for handling tasks failures and a layer for handling time constraints.

Adaptive Workflow Execution (step ❺-❻)

The concrete workflow, resulting from the previous step, is now converted to a xml file that follows the BPMN 2.0 schema (step ❺). The objective is to obtain something that a standard workflow execution engine may directly process (step ❻). The choice of the standard BPMN 2 notation opens the choice to many commercial process execution engines. The advantage of coupling the decoration strategy with a commercial engine is to support self-adaptive workflow execution without the constraint of adopting research products. In addition, the advantage of being independent of the process execution engine lies in the portability: the same workflow, decorated with

constructs that allow adaptive execution, can be used in different process execution engines without particular configurations or modification to the workflow itself, thus reducing the effort required to develop generic workflows for the emergency domain.

Currently, we use Flowable[3], an open-source BPMN-compliant workflow execution engine. Flowable is written in Java, supports BPMN 2.0 notation, process instance creation, query execution, and access to active or historical process instances and related data. The Flowable engine can be integrated into a Java application or a service, and provides a REST API to interact via HTTP requests; this is useful for informing the engine about the completion of a task, or for querying the status of active processes or tasks. Flowable engine supports both service tasks and human tasks. We chose Flowable for enacting our emergency plans also because of its low-configuration, the easiness of use, the large user community and the continuous updates.

In our proposal, concrete capabilities are implemented as service tasks. In Flowable, a service task refers to a Java class containing the code that realizes a specific service. If the functioning of a service task undergoes some fault, then an exception is generated and caught by Flowable engine. In this case, if the process has been previously decorated with an adaptation layer, then the boundary error events catch the exceptions generated at run-time by the concrete capabilities, thus supporting the run-time adaptation of emergency operations.

However, an emergency plan does not contain only service tasks, which execution is delegated to the software, but also human tasks that must be carried out by human operators (the machine may only monitor the progresses). For example, a human task involves communications between different actors during rescue operations; these communications can be carried out through the communication component (Figure 6), or manually by the human operator (oral communications).

THE N.E.T.TUN.IT EXECUTION LAYER

In the context of the N.E.T.TUN.IT project, our goal is to define a platform to support operational data exchange between cross-border countries facing a common crisis. Consequently, the communication component must address the following objectives:

- supporting computer-mediated communications among actors involved in the emergency response. These communications cannot take place using traditional communication channels accessible to private citizens, and also require some specific knowledge about rescue teams that is not in the public domain. Thus, we assume that the communication component represents a means of communication exclusive to rescue teams and government organizations that can use it to manage internal communications during emergencies;
- providing support for logging computer-mediated communications between actors involved in emergency operations. Communications logging is useful to obtain data for process mining purposes, which can be beneficial for a post-disaster analysis of the rescue operations (emergency debriefing); the outcome of process mining techniques could support the improvement of emergency plans, based on both historical environmental context and responses, and improve preparedness activities.

Figure 6 shows the N.E.T.TUN.IT architecture with the components of the execution layer that allow the adaptive execution of workflows in emergency domain.

In the following, we describe the components that compose this execution layer:

- **Plan Generator component (MUSA)**: this component is responsible for producing workflows in standard BPMN 2.0, which execution allows addressing a set of input goals in the GoalSPEC notation;
- **Capability repository (MUSA)**: a container of services that can be invoked by the activities in an executable workflow;
- **Communication component**: the component responsible for dispatching communications to actors involved in the rescue operations of an emergency plan;
- **Process Execution Engine**: software responsible for a) the execution of workflows produced by MUSA, and b) monitoring the execution of each task.

[3] https://www.flowable.com/open-source

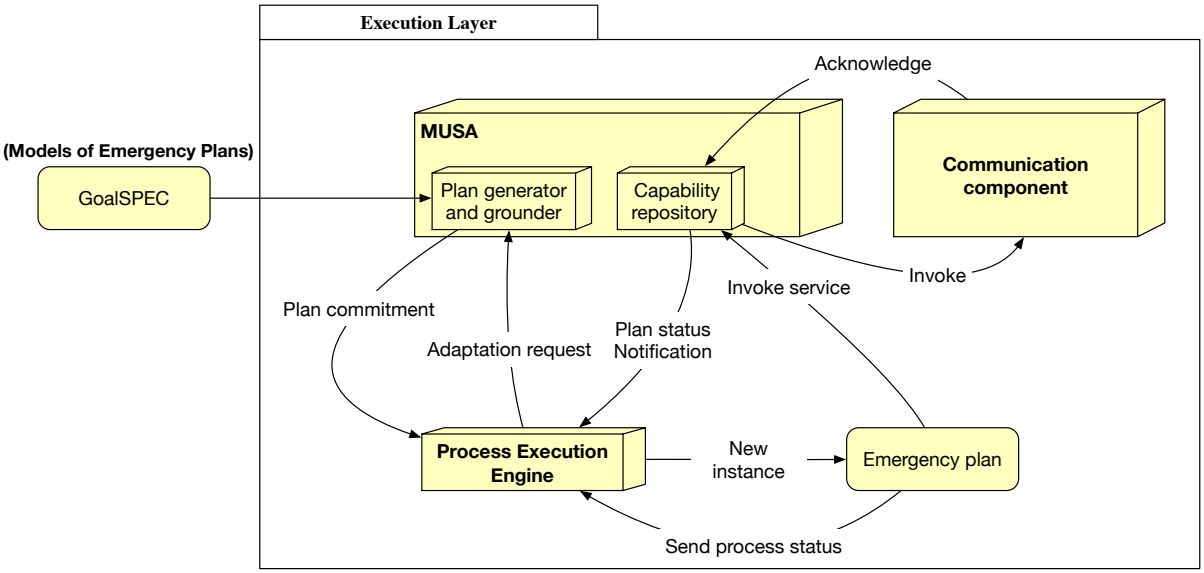

Figure 6. Components composing the execution layer.

CONCLUSIONS AND FUTURE WORKS

This paper completes the definition of a pipeline for going from emergency plans in free-form text to the adaptive execution of a workflow supporting emergency procedures. This pipeline has been developed in the context of the N.E.T.TUN.IT project, whose goal is to improve the collaboration between cross-border emergency operators in Italy and Tunisia to facilitate the response to common emergencies that may arise in the Mediterranean Channel.

In this work, we proposed a tool for generating executable workflows that fulfill goals extracted from emergency plans. Our proposal produces workflows in the standard BPMN 2.0 format that can be executed by compatible workflow execution engines.

Herein, adaptivity is intended as the capacity to respond to unpredictable events that could compromise the nominal execution of a process through a reorganization of the process itself. The reorganization results in a new workflow that satisfies the set of goals provided as an input using alternative tasks/services. To achieve adaptive execution, our proposal decorates workflows with standard BPMN 2.0 constructs that enable continuous monitoring of activities; by doing so, our tool provides on-demand reorganization of workflow for application in the emergency management domain.

In future works, we will focus on validating the proposed method and assert the pertinence of the underlying adaptive execution mechanism; our objective is to prove that the decorating process enables the adaptive execution in several emergency contexts, with different resources, tasks, and time constraints. Also, we plan to investigate some challenges in process mining.

ACKNOWLEDGEMENTS

The research was partially funded by the DSB.AD008.645 “Net de l’Environnement Transfrontalière TUNisie-ITalie (N.E.T.TUN.IT)” research project, within the Cross-border Cooperation Programme of the EU Community, Italy-Tunisia 2014-2020.

REFERENCES

Aschoff, R. R. and Zisman, A. (2012). “Proactive adaptation of service composition”. In: *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*. IEEE, pp. 1–10.

Ceri, S., Grefen, P., and Sanchez, G. (1997). “WIDE-a distributed architecture for workflow management”. In: *Research Issues in Data Engineering, 1997. Proceedings. Seventh International Workshop on*. IEEE, pp. 76–79.

Cossentino, M., Guastella, D. A., Lopes, S., Sabatucci, L., and Tripiciano, M. (2022a). “From Textual Emergency Procedures to Executable Plans”. In: *19th International Conference on Information Systems for Crisis Response and Management, ISCRAM 2022, Tarbes, France, May 22-25, 2022*. Ed. by R. Grace and H. Baharmand. ISCRAM Digital Library, pp. 200–212.

Cossentino, M., Guastella, D. A., Lopes, S., Sabatucci, L., and Tripiciano, M. (2022b). "Linguistic and semantic layers for emergency plans". In: *Intelligenza Artificiale* 16.1, pp. 7–25.

Cossentino, M., Lopes, S., Sabatucci, L., and Tripiciano, M. (2021). "Towards a Semantic Layer for Italian Emergency Plans". In: *Proceedings of the 22nd Workshop "From Objects to Agents", Bologna, Italy, September 1-3, 2021*. Vol. 2963. CEUR Workshop Proceedings. CEUR-WS.org, pp. 144–161.

Cossentino, M., Sabatucci, L., and Seidita, V. (2018). "Engineering Self-adaptive Systems: From Experiences with MUSA to a General Design Process". In: *International Workshop on Engineering Multi-Agent Systems*. Springer, pp. 96–116.

Edelkamp, S. and Hoffmann, J. (2004). *PDDL2. 2: The language for the classical part of the 4th international planning competition*. Tech. rep. University of Freiburg: Technical Report 195.

Fikes, R. E. and Nilsson, N. J. (1971). "STRIPS: A new approach to the application of theorem proving to problem solving". In: *Artificial intelligence* 2.3-4, pp. 189–208.

Gelfond, M. and Lifschitz, V. (1998). "Action languages". In: *Computer and Information Science* 3.16.

Hellmund, T., Moßgraber, J., Schenk, M., Hertweck, P., Schaaf, H. van der, and Springer, H. (2021). "The Design and Implementation of ZEUS: Novel Support in Managing Large-Scale Evacuations". In: *18th International Conference on Information Systems for Crisis Response and Management, ISCRAM 2021, Blacksburg, VA, USA, May 2021*. Ed. by A. Adrot, R. Grace, K. A. Moore, and C. W. Zobel. ISCRAM Digital Library, pp. 1003–1014.

Herrera, M. P. R. and Díaz, J. S. (2019). "Improving Emergency Response through Business Process, Case Management, and Decision Models". In: *Proceedings of the 16th International Conference on Information Systems for Crisis Response and Management, València, Spain, May 19-22, 2019*. Ed. by Z. Franco, J. J. González, and J. H. Canós. ISCRAM Association.

Kammer, P. J., Bolcer, G. A., Taylor, R. N., Hitomi, A. S., and Bergman, M. (2000). "Techniques for supporting dynamic and adaptive workflow". In: *Computer Supported Cooperative Work (CSCW)* 9.3-4, pp. 269–292.

Namiot, D. and Sneps-Sneppe, M. (2014). "On micro-services architecture". In: *International Journal of Open Information Technologies* 2.9.

Newell, A. (1982). "The knowledge level". In: *Artificial intelligence* 18.1, pp. 87–127.

OMG, B. P. M. (2011). "Notation (BPMN) Version 2.0 (2011)". In: *Available on: http://www. omg. org/spec/BPMN/2.0*.

Peinel, G., Rose, T., and Wollert, A. (2012). "The myth of business process modelling for emergency management planning." In: *ISCRAM*.

Reichert, M. and Dadam, P. (1998). "ADEPT flex—supporting dynamic changes of workflows without losing control". In: *Journal of Intelligent Information Systems* 10.2, pp. 93–129.

Sabatucci, L. and Cossentino, M. (2015). "From means-end analysis to proactive means-end reasoning". In: *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, pp. 2–12.

Sabatucci, L. and Cossentino, M. (2019). "Supporting Dynamic Workflows with Automatic Extraction of Goals from BPMN". In: *ACM Transactions on Autonomous and Adaptive Systems* 14.2, pp. 1–38.

Sabatucci, L., Lodato, C., Lopes, S., and Cossentino, M. (2015). "Highly Customizable Service Composition and Orchestration". In: *Service Oriented and Cloud Computing*. Ed. by S. Dustdar, F. Leymann, and M. Villari. Vol. 9306. Lecture Notes in Computer Science. Springer International Publishing, pp. 156–170.

Sabatucci, L., Ribino, P., Lodato, C., Lopes, S., and Cossentino, M. (2013). "Goalspec: A goal specification language supporting adaptivity and evolution". In: *Engineering Multi-Agent Systems: First International Workshop, EMAS 2013, St. Paul, MN, USA, May 6-7, 2013, Revised Selected Papers 1*. Springer, pp. 235–254.

Sell, C. and Braun, I. (2009). "Using a workflow management system to manage emergency plans". In: *Proceedings of the 6th International ISCRAM Conference*. Vol. 41, p. 43.

Strong, D. M. and Miller, S. M. (1995). "Exceptions and exception handling in computerized information processes". In: *ACM Transactions on Information Systems (TOIS)* 13.2, pp. 206–233.