SmartPM: Automated Adaptation of Dynamic Processes

Andrea Marrella[1], Massimo Mecella[1], Sebastian Sardina[2], and Paola Tucceri[1]

[1] Sapienza University of Rome, Italy
{marrella,mecella}@dis.uniroma1.it, paola.tucceri@gmail.com
[2] RMIT University, Melbourne, Australia
sebastian.sardina@rmit.edu.au

Abstract. In this demonstration paper, we present the first working version of SmartPM, a Process Management System that is able to automatically adapt dynamic processes at run-time when unanticipated exceptions occur, thus requiring no specification of recovery policies at design-time.

Keywords: Process Management System, Adaptation, Dynamic Scenario

Introduction. Nowadays, the maturity of process management methodologies has led to the application of process-oriented approaches in new challenging domains beyond business computing [2], such as healthcare, emergency management, and domotics. In those *dynamic* settings, process enactment is influenced by user decision making and coupled with contextual data and knowledge production. During process enactment, variations from structured reference models are common due to exceptional circumstances arising (e.g., autonomous user decisions, exogenous events, or contextual changes), thus requiring the ability to properly *adapt* the process behavior. According to [6], *process adaptation* can be seen as the ability of a process to react to *exceptional circumstances* (that may or may not be foreseen) and to modify its structure accordingly. In dynamic scenarios, traditional manual implementation of exception handlers at design time is not feasible for the process designer, who has to anticipate all potential problems and ways to overcome them in advance [5]. Furthermore, many *unanticipated exceptional circumstances* may arise during process execution, and their handling requires a manual intervention of a domain expert at run-time. However, the complexity of the operational context may transform the manual definition of a recovery procedure at run-time in a time-consuming and error-prone task. To tackle the above issues, we present the first working version of SmartPM, a Process Management System (PMS) that is able to *automatically adapt dynamic processes at run-time* when *unanticipated exceptions* occur, thus requiring *no specification of recovery policies at design-time*.

Demonstration Scenario. We consider the emergency management situation described in Fig. 1(a), in which a train derailment is depicted in a grid-type map. A possible concrete realization of an incident response plan for our scenario is shown in Fig. 1(c), through a BPMN process composed of three parallel branches, with tasks instructing first responders to act for evacuating people from train coaches, taking pictures of the locomotive, and assessing the gravity of the accident. To execute the process, a response team is sent to the derailment scene. The team is composed of four first responders, called *actors*, and two *robots*, initially all located at location cell $loc00$. It is assumed

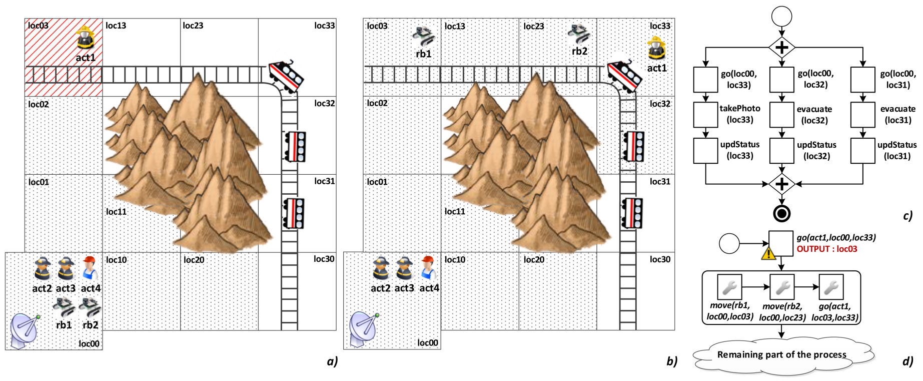
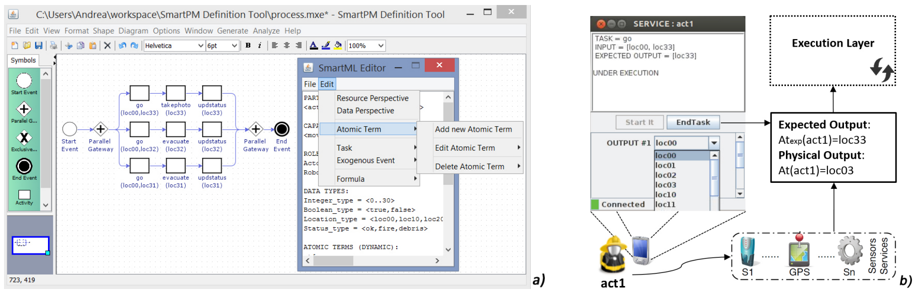


Fig. 1. A train derailment situation; area and context of the intervention.

that actors are equipped with mobile devices for picking up and executing tasks, and that each provide specific capabilities. For example, $act1$ is able to extinguish fire and take pictures, while $act2$ and $act3$ can evacuate people from train coaches. The two robots, in turn, are designed to remove debris from specific locations. When the battery of a robot is discharged, $act4$ can charge it. In order to carry on the response plan, all actors and robots ought to be continually inter-connected. The connection between mobile devices is supported by a fixed antenna located at $loc00$, whose range is limited to the dotted squares in Fig. 1(a). Such a coverage can be extended by robots $rb1$ and $rb2$, which have their own independent (from antenna) connectivity to the network and can act as wireless routers to provide network connection in all adjacent locations. Due to the high dynamism of the environment, there is a wide range of exceptions that can ensue. So, suppose for instance that actor $act1$ is sent to the locomotive's location, by assigning to it the task $\textsc{go}(loc00, loc33)$ in the first parallel branch. Unfortunately, however, the actor happens to reach location $loc03$ instead. The actor is now located at a different position than the desired one and is out of the network connectivity range (cf. Fig. 1(a)). Therefore, the PMS initially has to find a recovery procedure to bring back full connectivity, and then find a way to re-align the process. To that end, provided robots have enough battery charge, the PMS may first instruct the first robot to move to cell $loc03$ in order to re-establish network connection to actor $act1$, and then instruct the second robot to reach location $loc23$ in order to extend the network range to cover the locomotive's location $loc33$. Finally, task $\textsc{go}(loc03, loc33)$ is reassigned to actor $act1$ (cf. Fig. 1(b)). The corresponding updated process is shown in Fig. 1(d), with the encircled section being the recovery procedure. We note that the execution of a dynamic process can be also jeopardized by the occurrence of *exogenous events* (e.g., a fire burnt up into a coach) that could change, in asynchronously manner, some contextual properties of the scenario, by possibly requiring the process to be adapted accordingly.

The SmartPM Approach and System. The SmartPM approach builds on the dualism between an *expected reality*, the (idealized) model of reality that is used by the PMS to reason, and a *physical reality*, the real world with the actual values of conditions and outcomes. Process execution steps and exogenous events have an impact on the



```
TASK = go
INPUT = [loc00, loc33]
EXPECTED OUTPUT = [loc33]

UNDER EXECUTION
```

```
DATA TYPES:
Integer_type = <0..30>
Boolean_type = <true,false>
Location_type = <loc00,loc10,loc2
Status_type = <ok,fire,debris>

ATOMIC TERMS (DYNAMIC):
```

Expected Output:
At$_{exp}$(act1)=loc33
Physical Output:
At(act1)=loc03

Fig. 2. A screenshot of the SmartPM Definition Tool (a) and the Task Handler (b).

physical reality and any deviation from the expected reality results in a mismatch to be removed to allow process progression. At this point, an external state-of-the-art planner is invoked to synthesise a recovery procedure that adapts the faulty process instance. The implementation of the SmartPM approach relies on three architectural layers that cover the modeling, execution and monitoring stages of the process life-cycle.

The *Presentation Layer* provides a GUI-based tool called SmartPM Definition Tool[3] (cf. Fig. 2(a)), which assists the process designer in the definition of the process model at design-time. Process knowledge is represented as a *domain theory* that includes all the contextual information of the domain of concern, such as the people/services that may be involved in performing the process, the tasks, the data and so forth. Data are represented through some *atomic terms* that range over a set of *data objects*, which depict entities of interest (e.g., locations, capabilities, services, etc.), while atomic terms can be used to express properties of domain objects (and relations over objects). For example, the term $At[act : Actor] = (loc : Location\_type)$ is used for recording the position of each actor in the area. In addition, the designer can define *complex terms*. They are declared as basic atomic terms, with the additional specification of a well-formed first-order formula that determines the truth value for the complex term. For example, the complex term $Connected[act : Actor]$ can be defined to express that an actor is connected to the network if s/he is in a covered location or if s/he is in a location adjacent to a location where a robot is located. *Tasks* are collected in a specific repository and are described in terms of preconditions - defined over atomic and complex terms - and effects, which establish their outcomes. Finally, a process designer can specify which *exogenous events* may be catched at run-time and which atomic terms will be modified after their occurrence. Once a valid domain theory is ready, the process designer uses the BPMN graphical editor provided by the SmartPM Definition Tool to define the process control flow among a set of tasks selected from the tasks repository.

The *Execution Layer* is in charge of managing and coordinating the execution of dynamic processes. First of all, the domain theory specification is translated into situation calculus and IndiGolog [1] readable formats.[4] The situation calculus is a logical

[3] It was developed with the JGraphX graphical library (`http://www.jgraph.com/`).
[4] The formal model underlying the SmartPM system is described in [4].

language designed for representing and reasoning about dynamic domains. On top of that, we use the IndiGolog high-level agent programming language for the specification of the process control flow. Hence, an executable model is obtained in the form of an IndiGolog program to be executed through an IndiGolog engine. To this end, we customized an existing IndiGolog engine[5] to *(i)* manage the process routing and decide which tasks are enabled for execution; *(ii)* collect exogenous events from the external environment; *(iii)* monitor contextual data to identify changes or events which may affect process execution. Specifically, after each task completion (or exogenous event occurrence), the physical and expected realities are updated to reflect the actual and intended outcome of task performance (or the contextual changes produced by an exogenous event). If the two realities are misaligned, the running process instance needs to be adapted. Process participants interact with the engine through a *Task Handler*, an interactive GUI-based application that supports the visualization of assigned tasks and allows to notify task completion by selecting an appropriate outcome (cf. Fig. 2(b)).

To enable the automated synthesis of a recovery procedure, the *Adaptation Layer* of SmartPM relies on the capabilities provided by a PDDL-based planner component (the LPG-td planner [3]), which assumes the availability of a *planning problem*, i.e., an initial state and a goal to be achieved, and of a *planning domain* definition that includes the actions to be composed to achieve the goal, the domain predicates and data types. Specifically, if process adaptation is required, *(i)* we translate the domain theory defined at design-time into a planning domain, *(ii)* the physical reality into the initial state of the planning problem and *(ii)* the expected reality into the goal state of the planning problem. The planning domain and problem are the input for the planner component. If the planner is able to synthesize a recovery procedure, the plan is converted into an executable IndiGolog process so that it can be enacted by the IndiGolog engine. Otherwise, if no plan exists for the current planning problem, the control passes back to the process designer, who can try to manually adapt the process instance. More information about SmartPM can be found at: `http://www.dis.uniroma1.it/~smartpm`.

Acknowledgements. This work has been partly supported by the Sapienza award SPIRITLETS, the project PIA Regione Calabria COSM-FACTORY, and the EU project VOICE.

References

1. De Giacomo, G., Lespérance, Y., Levesque, H., Sardina, S.: Indigolog: A high-level programming language for embedded reasoning agents. In: Multi-Agent Prog. Springer US (2009)
2. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches. Journal on Data Semantics (2014)
3. Gerevini, A., Saetti, A., Serina, I., Toninelli, P.: LPG-TD: a Fully Automated Planner for PDDL2.2 Domains. In: ICAPS-04 (2004)
4. Marrella, A., Mecella, M., Sardina, S.: SmartPM: An Adaptive Process Management System through Situation Calculus, IndiGolog, and Classical Planning. In: KR'14 (2014)
5. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer (2012)
6. Sadiq, S.W., Sadiq, W., Orlowska, M.E.: Pockets of flexibility in workflow specification. In: ER'01 (2001)

[5] `http://sourceforge.net/projects/indigolog/`