SmartPM: An Adaptive Process Management System for Executing Processes in Cyber-Physical Domains*

Andrea Marrella[1], Pätris Halapuu[2], Massimo Mecella[1], and Sebastian Sardiña[3]

(1) Sapienza Università di Roma, (2) University of Tartu, (3) RMIT University, Australia

Demo Abstract. Nowadays, the automation of business processes not only spans classical business domains (e.g., banks and governmental agencies), but also new settings such as healthcare, smart manufacturing, domotics and emergency management [2]. Such domains are characterized by the presence of a Cyber-Physical System (CPS) coordinating heterogeneous ICT components with a large variety of architectures, sensors, actuators, computing and communication capabilities, and involving real world entities that perform complex tasks in the "physical" real world to achieve a common goal. In this context, Process Management Systems (PMSs) are used to manage the life cycle of the processes that coordinate the services offered by the CPS to the real world entities, on the basis of the contextual information collected from the specific cyber-physical domain of interest. The physical world, however, is not entirely predictable. CPSs do not necessarily and always operate in a controlled environment, and their processes must be robust to unexpected conditions and adaptable to exceptions and external exogenous events. In this paper, we tackle the above issue by introducing the SmartPM System (`http://www.dis.uniroma1.it/~smartpm`) an adaptive PMS which combines process execution monitoring, unanticipated exception detection (without requiring an explicit definition of exception handlers), and automated resolution strategies on the basis of well-established Artificial Intelligence techniques, including the Situation Calculus and IndiGolog [1], and classical planning [3].

Significance to the BPM field. Exception handling is one of the most important tasks that process designers undertake during process modelling and execution [5]. In cyber-physical domains, the fact is that the number of possible anticipated exceptions is often too large, and 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. Furthermore, in such domains many unanticipated exceptional circumstances may arise during the process execution, requiring to adapt running process instances in a situation- and context-dependent way. While most PMSs of today shy away from dealing with the inherent dynamic nature of cyber-physical domains by providing *manual* or *semi-automated* techniques to deal with unanticipated exceptions, the management of processes enacted in such domains requires a PMS providing *real-time monitoring* and *automated adaptation* features during process execution. In this direction, SmartPM allows *(i)* the continuous screening of real-world objects performed by the physical sensors disseminated in the cyber-physical domain of interest, by transforming the "continuous" knowledge extracted from the domain in its digital counterpart; and *(ii)* the formalization of explicit mechanisms to model world changes and responding to anomalous situations, exceptions, exogenous events in an

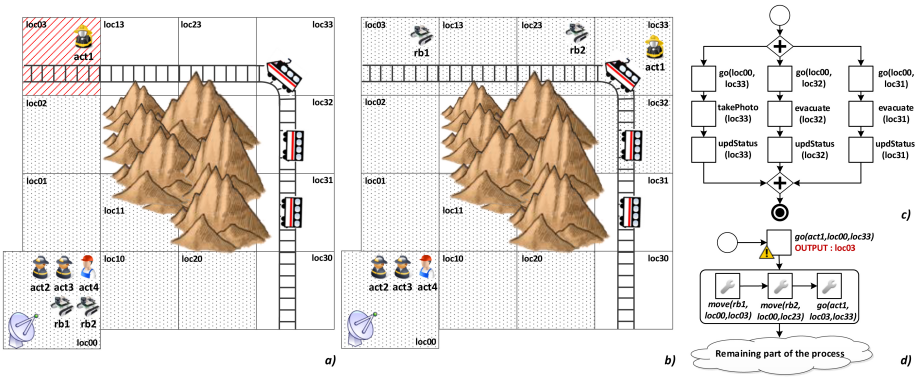

Fig. 1. A train derailment situation; area and context of the intervention

automated way, in order to achieve the overall objectives of the processes still preserving their structure without (or by minimising) any human intervention.

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 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 $\text{GO}(loc00, loc33)$ 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 $\text{GO}(loc03, loc33)$ is reassigned to actor

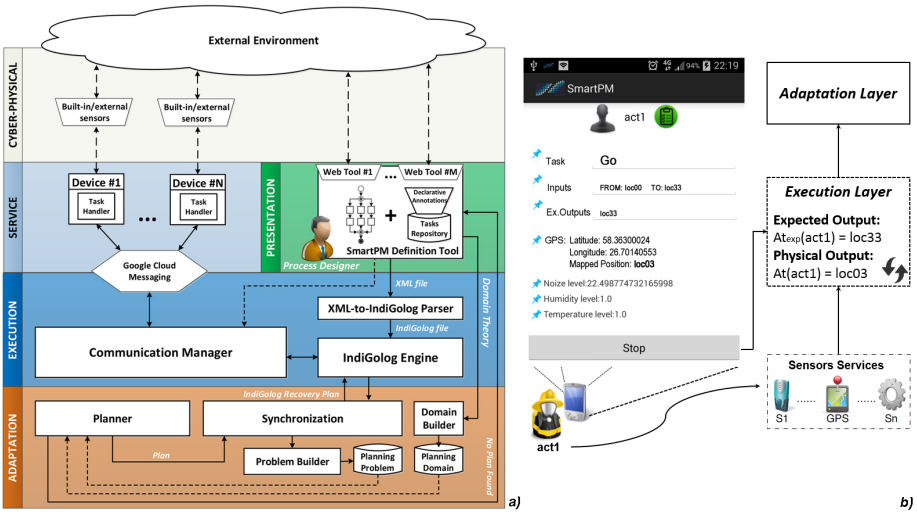

Fig. 2. The SmartPM architecture (a) and a screenshot of the Task Handler (b)

$act1$ (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 process can be also jeopardized by the occurrence of *exogenous events* (e.g., a fire burnt up into a coach) that could asynchronously change 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 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 by removing the gap between the two realities. The formal model underlying the SmartPM approach is described in [4], and its implementation covers the modeling, execution and monitoring stages of the process life-cycle. To that end (cf. Fig. 2(a)), the architecture of the SmartPM System relies on five architectural layers.

The *Presentation Layer* provides a GUI-based tool called SmartPM Definition Tool (developed with the JGraphX graphical library), 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, etc.), while atomic terms are 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 expected outcomes. Finally, a process designer can specify which *exogenous events* may be catched at run-time and which terms will be modified after their occurrence. Once a valid domain theory is ready, the process designer uses the BPMN 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 and Service Layers* are in charge of managing the process enactment. SmartPM adopts a "service-based" approach to process execution, that is, tasks are executed by services (that could be software applications, human actors, robots, etc.). First of all, the domain theory specification and the BPMN process are translated into situation calculus and IndiGolog readable formats. The situation calculus is a logical 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 that end, we customized an existing IndiGolog engine[1] to *(i)* build a physical/expected reality by taking the initial context from the external environment; *(ii)* manage the appropriate process routing; *(iii)* collect exogenous events from the external environment; *(iv)* monitor contextual data to identify changes or events which may affect process execution. Process participants interact with the engine through a *Task Handler* (realized for Android devices from version 4.0 and up) (cf. Fig. 2(b)), an interactive GUI-based software application that supports the visualization/execution of assigned tasks by selecting an appropriate outcome. For example, if we consider our demonstration scenario, when the task GO$(loc00, loc33)$ completes, the output value for *At*$(act1)$ (generated as an effect of the task GO) is '$loc03$', that is different from the task's expected outcome, that is '$loc33$'. Therefore, the two realities are misaligned and the running process instance δ needs to be adapted. The communication between the IndiGolog engine and the task handlers is mediated by the *Communicator Manager* component (which is a web server) and established using the Google Cloud Messaging service.[2]

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, we translate *(i)* the domain theory defined at design-time into a planning domain, *(ii)* the physical reality into the initial state of the planning problem and *(iii)* the expected reality into the goal state of the planning problem. The planning domain and problem are the input for the planner component. If

[1] `https://bitbucket.org/ssardina/indigolog/`

[2] `https://developer.android.com/google/gcm/index.html`

the planner is able to synthesize a recovery procedure δ_a, the *Synchronization* component combines δ' (which is the remaining part of the faulty process instance δ still to be executed), with the recovery plan δ_a, builds an adapted process $\delta'' = (\delta_a; \delta')$ and converts it into an executable IndiGolog program 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.

The *Cyber-Physical Layer* is tightly coupled with the physical components available in the domain of interest. For automating the data collection from the environment by using the sensors that are built in the mobile devices, several plugins have been created for the Task Handler. However, since the IndiGolog engine can only work with defined discrete values, while data gathered from physical sensors have naturally continuous values, a mapping of such continuous values into their discrete counterparts is required. To tackle this issue, we enhanced the SmartPM Definition Tool by providing several web tools that allow process designers to associate some of the data objects defined in the domain theory with the continuous data values collected from the environment. For example, in the case of the GPS sensor, we developed a web tool (as a Google Maps plugin) that allows a process designer to mark areas of interest from a real map (by selecting latidude/longitude values) and associate them to the discrete locations (e.g., $loc00$, $loc01$, etc.) defined during the design stage of a process. Similarly, we developed further web tools for the other developed sensors (temperature, noise level, etc.). The mapping rules generated are then encoded in a XML file that is saved into the Communication Manager and retrieved at run-time to allow the matching of the continuous data values collected by the specific sensor into discrete data objects (cf. the matching between 'loc03' and concrete latitude/longitude values in Fig. 2(b)).

Maturity. The SmartPM System was validated through empirical experiments based on 3600 different process models having control flows with different structures and domain theories associated to them. On the one hand, the experiments confirm the feasibility of the planning-based approach of SmartPM for adapting processes in medium-sized cyber-physical domains from the timing performance perspective. On the other hand, SmartPM was able to complete 2537 process instances without any domain expert intervention, corresponding to an effectiveness of about 70,5% (cf. also [4]).

Acknowledgements. This work has been supported by the Sapienza award SPIRITLETS, the grants TESTMED and SUPER and Italian projects NEPTIS and RoMA.

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)