Automated Process Adaptation in Cyber-Physical Domains with the SmartPM System

Andrea Marrella*, Massimo Mecella*, Pätris Halapuu† and Sebastian Sardiña‡
*Sapienza Università di Roma, Italy. *Email: {marrella,mecella}@dis.uniroma1.it*
†University of Tartu, Estonia. *Email: patris.halapuu@ut.ee*
‡RMIT University, Melbourne, Australia. *Email: sebastian.sardina@rmit.edu.au*

***Abstract*—Cyber Physical Systems (CPSs) refer to a new generation of embedded ICT systems (PCs, smartphones, sensors, actuators, etc.) that are interconnected and collaborating to provide users with a wide range of innovative applications and services. Many application domains, e.g., emergency management, factories of the future, personalized healthcare, just to name a few, require the definition, design and development of systems able to carry out complex processes that coordinate the services offered by the CPS in the "physical" real world. The physical world, however, is not entirely predictable, and such processes must be robust to unexpected conditions and adaptable to unanticipated exceptions. This demands a more flexible approach in process design and enactment, recognizing that in real-world environments it is not adequate to assume that all possible recovery activities can be predefined for dealing with the exceptions that can ensue. In this paper, we tackle the above issue and we propose an approach and a process management system implementation, called SmartPM, for automatically adapting processes enacted in cyber-physical domains in case of unanticipated exceptions and exogenous events.**

I. Introduction

As Information and Communication Technologies (ICTs) are being increasingly integrated and embedded into our everyday environment, the design of embedded ICT from components (PCs, smartphones, sensors, actuators, etc.) to *cyber-physical systems* is becoming a reality. A cyber-physical system (CPS) is a system of *interconnected* and *collaborating* computational elements controlling physical components that provide real world entities (e.g., people, machines, robots, agents, etc.) with a wide range of innovative applications and services [1]. CPSs are designed to support and facilitate collaboration among people and software services on complex tasks. On the other side, the Business Process Management (BPM) discipline has gained an increasing importance in describing complex correlations between distributed systems and offer a powerful representation of collaborative activities [2]. In the field of online trading and manufacturing, for example, modelling and execution languages for business processes, such as BPMN [3] and BPEL [4], have proven to be well suited to formalize high-level sequences of tasks and activities involving web service invocations and human interaction.

The current maturity of process management systems (PMSs) can led to the application of *process-oriented* approaches in new challenging *cyber-physical domains* beyond business computing [5], [6], such as personalized healthcare [7], emergency management [8] and factories of the future [9]. Such domains are characterized by the presence of a CPS coordinating heterogeneous ICT components and involving real world entities that perform complex tasks in the "physical" real world to achieve a common goal. In this context, a PMS is used to manage the life cycle of the collaborative 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 long-term objective of CPSs is to create a strong link between the physical world and the cyber world to support their users while performing their tasks [10]. The physical world, however, is not entirely predictable. CPSs do not necessarily and always operate in a controlled environment, and their collaborative processes must be robust to unexpected conditions and adaptable to exceptions and external exogenous events. *Exception handling* is one of the most important tasks that process designers undertake during process modelling and execution [11]. An *anticipated exception* can be planned at design-time and incorporated into the process model, i.e., a (human) process designer can provide an exception handler that is invoked during run-time to cope with the exception. Conversely, *unanticipated exceptions* refer to situations, unplanned at design-time, that may emerge at run-time and can be detected by monitoring discrepancies between the real-world processes and their computerized representation. To cope with those exceptions, a PMS is required to allow ad-hoc process changes for adapting running process instances in a context-dependent way.

However, in cyber-physical domains, 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, anticipated exceptions cover only partially relevant situations, as in such scenarios many unanticipated exceptional circumstances may arise during the process execution. While most PMSs of today shy away from dealing with the inherent dynamic nature of cyber-physical domains [9], the management of processes enacted in such domains requires a PMS providing the formalization of explicit mechanisms to model world changes and responding to anomalous situations and exceptions in an *automated* way, in order to achieve the overall objectives of the processes still preserving their

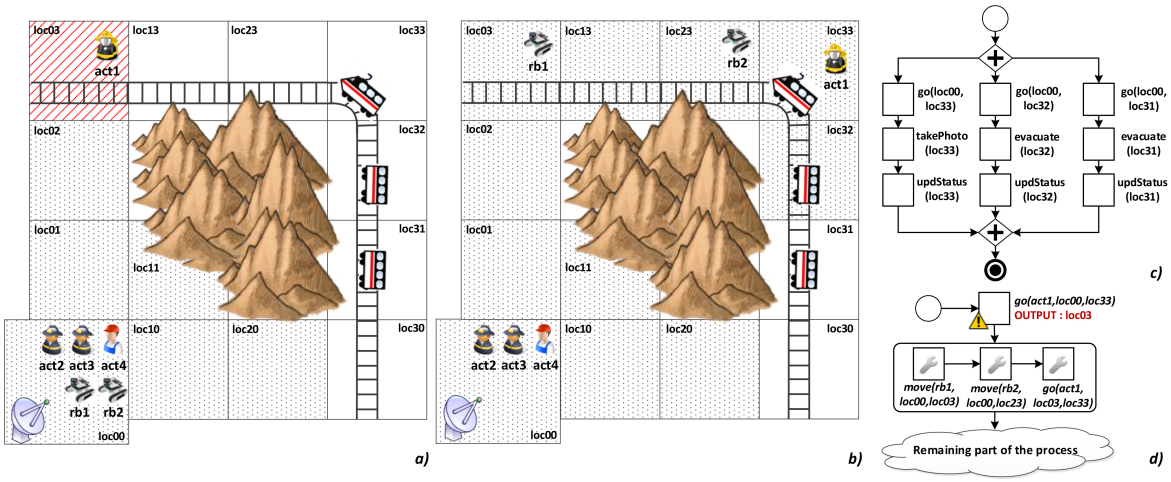

Fig. 1. A train derailment situation; area and context of the intervention.

structure without (or by minimising) any human intervention.

In this paper, we tackle the above challenge by presenting a general approach and a PMS implementation, called SmartPM (Smart Process Management) for automatically adapting processes enacted in cyber-physical domains in case of unanticipated exceptions and exogenous events. SmartPM is based on declarative task specifications, process execution monitoring for detecting failures and context changes at run-time, and automated exception handling and resolution strategies on the basis of well-established Artificial Intelligence (AI) techniques, including the Situation Calculus [12], IndiGolog [13] and classical planning [14]. Specifically, the paper is organized as follows. In Section II, we present a concrete running example. In Section III we introduce the general approach of SmartPM, and we present the architecture of the implemented SmartPM system. Finally, in Section IV we discuss the state-of-the-art approaches to process adaptation, while in Section V we conclude the paper by providing a critical discussion about the applicability of SmartPM in cyber-physical domains.

II. Running Example

The trend of managing processes in cyber-physical domains has been fueled by the increased availability of sensors disseminated in the world, which has lead to the possibility to monitor the evolution of several real-world objects of interest [15]. The knowledge extracted from such objects allows to depict the context in which processes are carried out, by consenting a fine-grained monitoring and decision support for them.

To make our discussion more concrete, let us consider an application scenario that comes from the *emergency management* domain and is inspired to a real disaster response plan investigated by the authors during the European project WORKPAD [16]. Specifically, in Fig. 1(a), 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 her/him 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 $act1$ (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

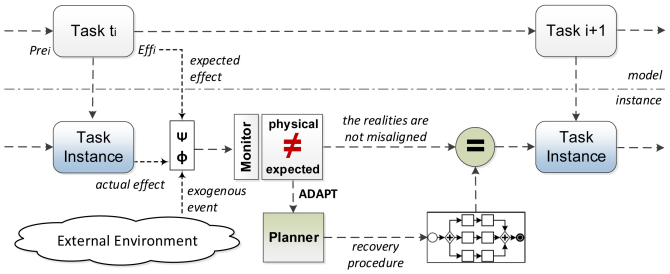

Fig. 2. An overview of the SmartPM approach.

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.

III. The SmartPM Approach and System

SmartPM is an approach and a PMS implementing a set of techniques that enable to automatically adapt process instances at run-time in the presence of unanticipated exceptions and exogenous events, without requiring an explicit definition of recovery policies. The approach, which is schematized in Fig. 2, builds on the dualism between an *expected reality* ψ_s, the (idealized) model of reality that is used by the PMS to reason, and a *physical reality* ϕ_s, the real world with the actual values of conditions and task outcomes. While the physical reality records what is *concretely* happening in the real environment during a process execution, the expected reality reflects what it is *supposed* to happen in the environment. A misalignment of the two realities often stems from errors in the tasks outcomes or is the result of exogenous events coming from the environment. Specifically, when a task is executed and completed, the physical/expected reality will reflect the actual/intended outcome of the task execution, according to the specification of task's effects. Conversely, exogenous events modify asynchronously only the physical reality, by leaving untouched the expected one. A recovery procedure is needed if the two realities are different from each other. If an exception/exogenous event invalidates the enactment of the process being executed, 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, in order to allow process progression. In this paper, we do not focus on the formal model underlying the SmartPM approach, which is described in [17], but we rather explore the architecture of the implemented system, which covers the modeling, execution and monitoring stages of the process life-cycle and captures the connection of implemented processes with the real-world objects of the cyber-physical domain of interest. To that end, as shown in Fig. 3, the architecture of the SmartPM system relies on five architectural layers.

The Presentation layer. The *Presentation layer* provides a GUI-based tool called SmartPM Definition Tool, which assists the process designer in the definition of a *process model* at design-time, i.e., it allows to *(i)* build a *tasks repository*, *(ii)* define the process *control flow* and *(iii)* formalize the *contextual knowledge* of the cyber-physical domain in which the process will be enacted. Contextual knowledge is represented as a *domain theory* that includes all the information of the application domain, such as the people/services that may be involved in performing the process, the exogenous events, the contextual data and so forth. Data are represented through some *atomic terms* that range over a set of *data objects*. In short, a data object depicts an entity of interest (e.g., a location, a capability, a service, etc.), while atomic terms can be used to express properties of domain objects (and relations over objects). In our running 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. A process designer can also specify which *exogenous events* may be catched at run-time and which atomic terms will be modified after their occurrence. Concerning the definition of process *tasks*, the process designer is required to specify which tasks are applicable to the scenario under study. Tasks will be stored in a specific repository, and can be used for composing the control flow of the process and for adaptation purposes. Each task is described with *(i)* typed *input parameters*, *(ii) pre-conditions* - defined over atomic and complex terms - that constrain the task assignment, and *(iii) deterministic effects*, which establish the outcome of a task after its execution in terms of a change of the value of one or more atomic terms. For example, the task GO involves two input parameters $from$ and to of type $Location\_type$, representing a starting and an arrival location. An instance of this task can be executed only if the process participant $SRVC$ that will execute it at run-time is at the starting location $from$ and is connected to the network. As a consequence of task execution, the actor moves from the starting to the arrival location, and this is reflected by assigning to the term $At[SRVC]$ the value to in the effect. Once a valid domain theory and a tasks repository are 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 outcome of the process design activity is a XML-encoded process specification that is passed to the *Execution layer*.

The Execution and Service layers. The *Execution Layer* is in charge of managing and coordinating 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, agents, etc.). The BPMN process and the associated domain theory are taken as input from the *XML-to-IndiGolog Parser* component,

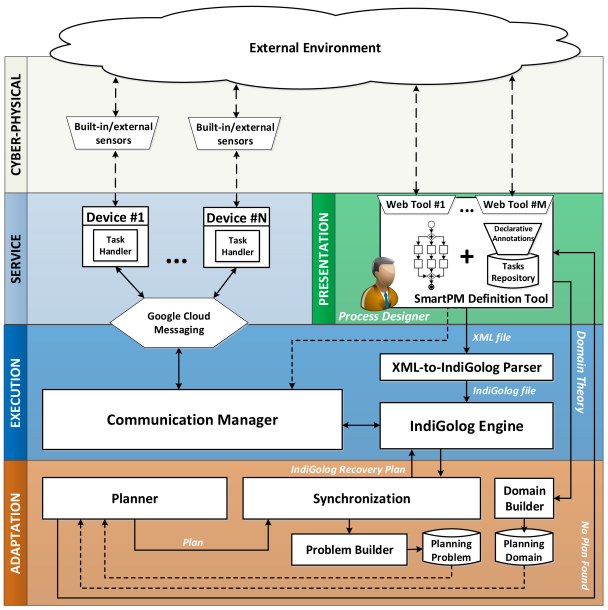

Fig. 3. The SmartPM architecture.

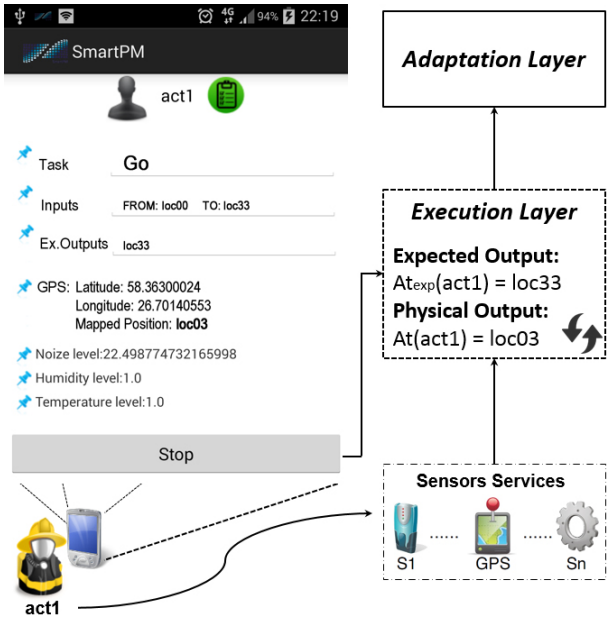

Fig. 4. The SmartPM Task Handler.

a Java module that translates them into Situation Calculus and IndiGolog readable formats. The Situation Calculus is a logical language designed for representing and reasoning about dynamic domains [12]. On top of that, we use the IndiGolog high-level agent programming language [13] for the specification of the process control flow. Hence, while from a user perspective the process control flow is defined using a subset of the modeling constructs provided by the BPMN notation, 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 process routing; *(iii)* collect exogenous events from the external environment. Once a task is ready for being executed, the IndiGolog engine is in charge of assigning it to a proper service that provides all the required capabilities for task execution. Process participants interact with the engine through a *Task Handler* (cf. Fig. 4), an interactive GUI-based software application that supports the visualization of assigned tasks and enables starting task execution and notifying of task completion by selecting an appropriate outcome. The SmartPM Task Handler is realized for Android devices from version 4.0 and up. The communication between the IndiGolog engine and the task handlers is mediated by the *Communicator Manager* component (which is essentially a web server) and established using the Google Cloud Messaging (GCM) service.[2]

The Adaptation layer. The IndiGolog engine is also in charge of monitoring contextual data to identify changes or events which may affect process execution, and notify them to the *Adaptation layer*. Specifically, given a process instance δ, 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 we consider our running example, when the task GO$(loc00, loc33)$ completes, we show that 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 '$loc33$' (cf. Fig. 4). This means that the two realities are misaligned, and the faulty process instance δ needs to be adapted. 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 [18]), 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, the *Domain Builder* component translates the domain theory defined at design-time into a planning domain, while the *Problem Builder* component converts *(i)* 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 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.

[1] https://bitbucket.org/ssardina/indigolog
[2] https://developer.android.com/google/gcm/index.html

The Cyber-Physical layer. This layer is tightly coupled with the concrete physical components available in the cyber-physical domain under consideration. For automating the data collection from the environment by using external/internal sensors built-in in the mobile devices, several plugins have been created for the Task Handler. For example, location data can be obtained using built-in GPS sensors. In addition, external sensors can be taken into use to gather automatic measurements. For example, the Task Handler can take advantage of the Arduino platform[3], which provides several sensors to measure different environmental values, such as the gas level in the air, water quality, radiation level, etc. Arduino can be connected with the Task Handler via Bluetooth for transferring the data. We notice that the IndiGolog engine of SmartPM can only work with defined discrete values, while data gathered from physical sensors have naturally continuous values. Therefore, 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 through the SmartPM Definition Tool. Similarly, we developed further web tools for the other developed sensors (temperature, humidity, 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 (after any task completion) 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. 4).

IV. Related Work

Initial research efforts addressing the need for exception handling in PMSs can be traced back to the late nineties and early two thousands [19]–[21]. Although possible sources of anticipated exceptions are different and go beyond technical failures, not surprisingly exception handling approaches in PMSs trace and resemble exception handling mechanisms in programming languages. At design-time, the process designer identifies possible exceptions that may occur, defines exception triggering events and conditions, and specifies exception handlers associated with the predefined process model. Exception handlers can be defined for single activities, for selected process regions, or for the overall process (as in the case of a `try` block in programming languages). During process execution, timers, messages, errors, constraint violations and other events might interrupt the process flow: the exception is detected and thrown. The run-time environment checks for the availability of a suitable exception handler, which is then invoked to catch the exception (as in the case of a `catch` block). Typically, the process (or sub-parts of it) is interrupted and the flow of control passes to the exception handler, which defines specific activities to be performed to recover from the exception, so that process execution can be possibly resumed. As extensively discussed in [22], exception handling capabilities provided by academic prototypes and commercial PMSs can be reconducted to the abstract framework introduced before. Several exception detection and handler activation techniques [21], [23], [24] adopt a rule-based approach, typically relying on some form of Event-Condition-Action (ECA) rules. ECA rules have the form "on *event* if *condition* do *action*" and specify to execute the *action* (i.e., the exception handler) automatically when the *event* happens (i.e., when the exception is caught), provided the a specific *condition* holds.

Research efforts dealing with unanticipated exceptions have established the area of *adaptive process management* [11]. The handling of unanticipated exceptions does not assume the availability of predefined exception handlers and relies on the possibility of performing ad hoc changes over process instances at run-time. However, the degree of automation in performing these changes is generally limited, as they are often manually performed by experienced users: process execution is suspended and the state of the affected instance is adapted by relying on the capabilities of the modeling environment. In an attempt to increase the level of user support, semi-automated approaches based on case-based reasoning techniques have been proposed [25], [26]. They aim at storing and exploiting available knowledge about previously performed changed, so that users can retrieve and apply it when adapting a process. Strong support for adaptive process management and exception handling is also provided by the ADEPT system and its evolutions [27]–[29] and by several AI-based techniques [30]–[34]. If compared with traditional exception handling approaches, we note that adaptive PMSs deal with unanticipated exceptions by automatically deriving the `try` block (with the manual definition of some business policies at design-time, in the case of [33], [34]) as the situation in which the PMS does not adequately reflect the real-world process anymore. As a consequence, one or several process instances have to be adapted with ad hoc process changes, and the common strategy used by the adaptive PMSs is to *manually* or *semi-automatically* define at run-time the `catch` block. However, in dynamic working environments and cyber-physical domains, analyzing and defining these adaptations "manually" becomes time-demanding and error-prone. Indeed, the designer should have a global vision of the application and its context to define appropriate recovery actions, which becomes complicated when the number of relevant context features and their interleaving increases. Conversely, the adaptation mechanism provided by SmartPM is based on execution monitoring for detecting failures and context changes, and allows to *automatically synthesize at run-time* the recovery procedures, without requiring to predefine

[3]Arduino is an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board, cf. `http://arduino.cc/en/guide/introduction`

any specific adaptation policy at design-time.

V. Conclusion

In this paper, we have introduced SmartPM, a framework and a concrete PMS for automated process adaptation in case of unanticipated exceptions and exogenous event, based on declarative task specifications and planning techniques. The choice of adopting AI technologies is motivated by their ability to provide the right abstraction level needed when dealing with dynamic situations in which data (values) play a relevant role in system enactment and automated reasoning over the system progress. However, the use of classical AI planning techniques for the synthesis of the recovery procedure imposes some restrictions for addressing more expressive problems, including incomplete information, preferences and multiple task effects. Furthermore, the need to explicitly model process execution context and annotate tasks with preconditions and effects may require some extra modeling effort at design-time, but the overhead is compensated at run-time by the possibility of automating exception handling procedures. 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. 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%. For a detailed discussion of the above experiments, the reader can refer to [17]. Future work will include an extension of our approach to "stress" the above assumptions by making the approach applicable to less-controllable cyber-physical domains (such as *smart museums*, and in general *smart spaces*), with the purpose to maintain the planning process very responsive.

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

References

[1] E. A. Lee, "Cyber physical Systems: Design Challenges," in *11th IEEE Int. Symp. on Object Oriented Real-Time Distributed Computing (ISORC)*. IEEE, 2008.

[2] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Springer, 2012.

[3] T. Allweyer, *BPMN 2.0: Introduction to the Standard for Business Process Modeling*. BoD–Books on Demand, 2010.

[4] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte *et al.*, "Business Process Execution Language for Web Services," 2003.

[5] C. Di Ciccio, A. Marrella, and A. Russo, "Knowledge-intensive Processes: An Overview of Contemporary Approaches," in *1st Int. Workshop on Knowledge-intensive Business Processes (KiBP 2012)*, 2012.

[6] C. Di Ciccio, A. Marrella, and A. Russo, "Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches," *Journal on Data Semantics*, pp. 1–29, 2014.

[7] R. Lenz and M. Reichert, "IT support for healthcare processes - premises, challenges, perspectives," *Data Kn. Eng.*, vol. 61, no. 1, 2007.

[8] A. Marrella, A. Russo, and M. Mecella, "Planlets: Automatically Recovering Dynamic Processes in YAWL," in *20th Int. Conf. on Cooperative Information Systems (CoopIS)*, 2012.

[9] R. Seiger, C. Keller, F. Niebling, and T. Schlegel, "Modelling Complex and Flexible Processes for Smart Cyber-Physical Environments," *Journal of Computational Science*, 2014.

[10] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-Physical Systems: the Next Computing Revolution," in *47th Design Automation Conference*. ACM, 2010.

[11] M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems. Challenges,Methods,Technologies*. Springer, 2012.

[12] R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, September 2001.

[13] G. De Giacomo, Y. Lespérance, H. Levesque, and S. Sardina, "Indigolog: A high-level programming language for embedded reasoning agents," in *Multi-Agent Programming*. Springer US, 2009, pp. 31–72.

[14] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Pub. Inc., 2004.

[15] A. Marrella, M. Mecella, and A. Russo, "Collaboration on-the-field: suggestions and Beyond," in *8th Int. Conf. on Information Systems for Crisis Response and Management (ISCRAM)*, 2011.

[16] S. R. Humayoun, T. Catarci, M. de Leoni, A. Marrella, M. Mecella, M. Bortenschlager, and R. Steinmann, "The Workpad User Interface and Methodology: Developing Smart and Effective Mobile Applications for Emergency Operators," in *Universal Access in Human-Computer Interaction. Applications and Services*. Springer, 2009, pp. 343–352.

[17] A. Marrella, M. Mecella, and S. Sardina, "SmartPM: An Adaptive Process Management System through Situation Calculus, IndiGolog, and Classical Planning," in *Principles of Knowledge Representation and Reasoning: Proceedings of the 14th International Conference, KR*, 2014.

[18] A. Gerevini, A. Saetti, I. Serina, and P. Toninelli, "LPG-TD: a Fully Automated Planner for PDDL2.2 Domains," in *14th Int. Conference on Automated Planning and Scheduling (ICAPS-04)*, 2004.

[19] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi, "Specification and Implementation of Exceptions in Workflow Management Systems," *ACM Transactions on Database Systems (TODS)*, vol. 24, no. 3, 1999.

[20] J. Eder and W. Liebhart, "Workflow Recovery," in *1st IFCIS Int. Conf. on Cooperative Information Systems (CoopIS)*. IEEE, 1996.

[21] C. Hagen and G. Alonso, "Exception Handling in Workflow Management Systems," *IEEE Trans. on Soft. Eng.*, vol. 26, no. 10, 2000.

[22] M. J. Adams, "Facilitating Dynamic Flexibility and Exception Handling for Workflows," Ph.D. dissertation, Queensland University of Technology Brisbane, Australia, 2007.

[23] D. K. W. Chiu, Q. Li, and K. Karlapalem, "A Logical Framework for Exception Handling in ADOME Workflow Management System," in *12th Int. Conf. on Adv. Inf. Syst. Eng. (CAiSE)*. Springer-Verlag, 2000.

[24] A. H. M. ter Hofstede, W. M. P. van der Aalst, M. Adams, and N. Russell, *Modern Business Process Automation: YAWL and its Support Environment*. Springer, 2009.

[25] B. Weber, W. Wild, and R. Breu, "CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning," *Advances in Case-Based Reasoning*, 2004.

[26] M. Minor, R. Bergmann, and S. Görg, "Case-based Adaptation of Workflows," *Information Systems*, vol. 40, pp. 142–152, 2014.

[27] M. Reichert, S. Rinderle, and P. Dadam, "Adept Workflow Management System," in *Business Process Management (BPM)*. Springer, 2003.

[28] M. Reichert, S. Rinderle, U. Kreher, and P. Dadam, "Adaptive Process Management with ADEPT2," in *ICDE*, 2005.

[29] A. Lanz, M. Reichert, and P. Dadam, "Robust and Flexible Error Handling in the AristaFlow BPM Suite," in *Information Systems Evolution*. Springer, 2011.

[30] M. D. R-Moreno and P. Kearney, "Integrating AI planning techniques with Workflow Management System," *Knowl.-Based Syst.*, vol. 15, no. 5-6, 2002.

[31] H. Ferreira and D. Ferreira, "An Integrated Life Cycle for Workflow Management Based on Learning and Planning," *Int. J. Cooperative Information Systems*, vol. 15, 2006.

[32] A. Marrella, M. Mecella, and A. Russo, "Featuring Automatic Adaptivity through Workflow Enactment and Planning," in *7th Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*. IEEE, 2011.

[33] A. Bucchiarone, M. Pistore, H. Raik, and R. Kazhamiakin, "Adaptation of Service-Based Business Processes by Context-Aware Replanning," in *SOCA*, 2011, pp. 1–8.

[34] N. van Beest, E. Kaldeli, P. Bulanov, J. Wortmann, and A. Lazovik, "Automated Runtime Repair of Business Processes," *Information Systems*, vol. 39, pp. 45–79, 2014.