An Adaptive Process Management System Implementation based on Situation Calculus, Indigolog and Classical Planning

Andrea Marrella and **Massimo Mecella**
Sapienza - Universitá di Roma, Italy
{marrella—mecella}@dis.uniroma1.it

Sebastian Sardina
RMIT University, Melbourne, Australia
sebastian.sardina@rmit.edu.au

Abstract

In this paper, we introduce an adaptive Process Management System implementation that combines business process execution monitoring, unanticipated exception detection and automated resolution strategies leveraging on well-established formalisms developed for reasoning about actions in Artificial Intelligence, including the situation calculus, IndiGolog and classical planning. Such formalisms provide a natural framework for the formal specification of explicit mechanisms to model world changes and responding to anomalous situations, exceptions, exogenous events in an automated way during process execution.

1 Introduction

In recent years, *Business Process Management* (BPM) approaches and technologies received considerable attention, as they are highly relevant from a practical point of view while offering many technical challenges for researchers. BPM focuses on overseeing how work is performed in a company by managing and optimising its *business processes*. 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, domotics or emergency management. More and more such processes are *cyber-physical*, as the information flowing through the process is often produced by human activities or is acquired by sensors and software services. Consequently, the execution context of these processes is more complex and not entirely predictable, as increasing amounts of data may influence unexpectedly the running of process instances.

Today there is an abundance of *Process Management Systems* (PMSs) driven by semi-formal activity-centric process modeling languages to enact and manage traditional business processes. However, such PMSs shy away from dealing with the inherent dynamic nature of processes enacted in cyber-physical domains, which must be robust and adaptable to unexpected conditions [Di Ciccio *et al.*, 2015].

In this paper, we tackle the above challenge by presenting SmartPM, a PMS implementation for automatically adapting processes enacted in cyber-physical domains in case of unanticipated exceptions and exogenous events. SmartPM aims at demonstrating that a targeted use of some well-established formalisms for reasoning about actions in Artificial Intelligence (AI), such as situation calculus [Reiter, 2001], IndiGolog [De Giacomo *et al.*, 2009] and classical planning [Ghallab *et al.*, 2004], provides a natural framework for the formal specification of explicit mechanisms to model world changes and responding to anomalous situations in an automated way during process execution. Specifically (the formal model underlying SmartPM is described in [Marrella *et al.*, 2014]): *(i)* situation calculus theories are used to model the process data and to represent the set of tasks of the application domain of interest; *(ii)* the IndiGolog high-level agent language provides the formal executable semantics for processes in cyber-physical domains; *(iii)* classical planning techniques are used to synthesize resolution strategies in case of unanticipated exceptions and exogenous events.

2 The Approach and the System

SmartPM relies on an approach (cf. Figure 1) that builds on the dualism between an *expected reality* Ψ, the (idealized) model of reality used by the PMS to reason, and a *physical reality* Φ, the real world with the actual values of conditions and outcomes. While Φ records what is *concretely* happening in the real environment during a process execution, Ψ reflects what it is *expected* to happen. Process execution steps and exogenous events have an impact on Φ and any deviation from Ψ results in the invocation of a state-of-the-art planner, which synthesises a recovery procedure to adapt the faulty process instance by removing the gap between the two realities.

To realize this approach, the implementation of SmartPM covers the modeling, execution and monitoring stages of the process life-cycle, by capturing the connection of implemented processes with the real-world objects of the cyber-physical domain of interest. To that end, the architecture of SmartPM relies on five architectural layers.

The *Presentation Layer* provides a graphical editor developed in Java that 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., capabilities, ser-

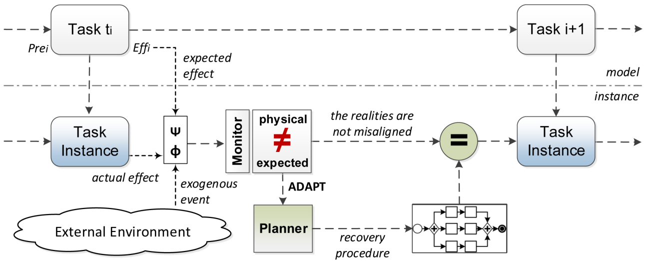

Figure 1: An overview of the SmartPM approach.

vices, etc.), while atomic terms can be used to express properties of domain objects (and relations over objects). *Tasks* are collected in a repository and are described in terms of *preconditions* - defined over atomic 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 atomic terms will be modified after their occurrence. Once a valid domain theory is ready, the process designer uses the graphical editor to define the process control flow through the standard BPMN notation among a set of tasks selected from the tasks repository.

The *Execution* and *Service Layers* are in charge of managing the process enactment. First of all, the domain theory specification and the BPMN process are automatically translated into situation calculus and IndiGolog readable formats. Situation calculus is used for providing a declarative specification of the domain of interest (i.e., available tasks, contextual properties, tasks preconditions and effects, what is known about the initial state). Then, 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; *(iv)* monitor contextual data to identify changes or events which may affect process execution. Once a task is ready for being executed, the IndiGolog engine assigns it to a proper process participant (that could be software applications, human actors, robots, etc.) that provides all the required capabilities for task execution. Process participants interact with the engine through a *Task Handler*, an interactive GUI-based application realized for Android devices that supports the visualization/execution of assigned tasks by selecting an appropriate outcome. The communication between the IndiGolog engine and the task handlers is mediated by the *Communicator Manager* component and through the Google Cloud Messaging service.

To enable the automated synthesis of a recovery procedure, the *Adaptation Layer* relies on the capabilities provided by a PDDL-based planner component (the LPG-td planner[2]), 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.

[1]http://sourceforge.net/projects/indigolog/

[2]http://lpg.unibs.it/lpg/

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 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 process designer can try to manually adapt the faulty process instance.

The *Cyber-Physical Layer* is tightly coupled with the physical components available in the domain of interest. Since the IndiGolog engine can only work with defined discrete values, while data gathered from physical sensors have continuous values, the system provides 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, we developed a web tool (as a Google Maps plugin) that allows to mark areas of interest from a real map (by selecting latidude/longitude values) and associate them to the discrete locations defined during the design stage of a process. The mapping rules generated are then 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.

SmartPM has been validated through several experiments that confirm the effectiveness of its AI-based approach for adapting processes in medium-sized cyber-physical domains (cf. [Marrella *et al.*, 2014]). More information about the system can be found at: http://www.dis.uniroma1.it/∼smartpm.

Acknowledgments. This work has been partly supported by the Italian projects SM&ST, RoMA and NEPTIS. The authors would like to thanks Pätris Halapuu for his valuable help in the SmartPM development.

References

[De Giacomo *et al.*, 2009] Giuseppe De Giacomo, Yves Lespérance, Hector Levesque, and Sebastian Sardina. IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents. In *Multi-Agent Prog.* Springer, 2009.

[Di Ciccio *et al.*, 2015] Claudio Di Ciccio, Andrea Marrella, and Alessandro Russo. Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches. *J. Data Semantics*, 4(1), 2015.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[Marrella *et al.*, 2014] Andrea Marrella, Massimo Mecella, and Sebastian Sardina. SmartPM: An Adaptive Process Management System through Situation Calculus, IndiGolog, and Classical Planning. In *KR*, 2014.

[Reiter, 2001] Raymond Reiter. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT press, 2001.