Chapter 2
Adaptive Process Management in Cyber-Physical Domains

Andrea Marrella and Massimo Mecella

Abstract The increasing application of process-oriented approaches in new challenging cyber-physical domains beyond business computing (e.g., personalized healthcare, emergency management, factories of the future, home automation, etc.) has led to reconsider the level of flexibility and support required to manage complex processes in such domains. A cyber-physical domain is characterized by the presence of a cyber-physical system coordinating heterogeneous ICT components (PCs, smartphones, sensors, actuators) and involving real world entities (humans, machines, agents, robots, etc.) that perform complex tasks in the "physical" real world to achieve a common goal. The physical world, however, is not entirely predictable, and processes enacted in cyber-physical domains 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 chapter, we tackle the above issue and we propose a general approach, a concrete framework 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. The adaptation mechanism provided by SmartPM is based on declarative task specifications, execution monitoring for detecting failures and context changes at run-time, and automated planning techniques to self-repair the running process, without requiring to predefine any specific adaptation policy or exception handler at design-time.

Keywords Process adaptation · Process management system · Cyber-physical system · Emergency management · Knowledge representation · Automated planning

A. Marrella (✉) · M. Mecella
Sapienza Università di Roma, Rome, Italy
e-mail: marrella@diag.uniroma1.it

M. Mecella
e-mail: mecella@diag.uniroma1.it

A. Marrella · M. Mecella
Dipartimento di Ingegneria Informatica Automatica e Gestionale, Rome, Italy

G. Grambow et al. (eds.), *Advances in Intelligent Process-Aware Information Systems*, Intelligent Systems Reference Library 123,
DOI 10.1007/978-3-319-52181-7_2

2.1 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., humans, machines, agents, robots, 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 offers 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 activities involving web service invocations and human interaction.

Nowadays, the current maturity of process management systems (PMSs) and methodologies has led to the application of process-oriented approaches in new challenging *cyber-physical domains* beyond business computing [5, 6], such as personalized healthcare [7–9], emergency management [10, 11], factories of the future [12] and home automation [13]. Such domains are characterized by the presence of a CPS coordinating heterogeneous ICT components with a large variety of architectures, sensors, 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, 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. To guarantee a better control over the interaction that PMS has with the real world, it continuously collects contextual information from the specific cyber-physical domain it is employed in.

The long-term objective of CPSs is to create a strong link between the physical world and the cyber world to support their users with performing their tasks [14]. 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. To this end, we define an *exception* as any deviation from an "ideal" collaborative process that uses the available resources to achieve the task requirements in an optimal way [15].

Exception handling is one of the most important tasks that process designers undertake during business process modelling and execution [16]. Exceptions can be either anticipated or unanticipated. 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 which is invoked during run-time to cope with the exception. Conversely, unanticipated exceptions generally refer to situations,

unplanned at design-time, that may emerge at run-time and can be detected by monitoring discrepancies and inconsistencies 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 situation- and 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 [12], the management of processes enacted in such domains requires a PMS providing real-time monitoring and adaptation features during process execution. This requires the formalization of explicit mechanisms to model world changes and responding to anomalous situations, exceptions, exogenous events in an *automated* way, in order to achieve the overall objectives of the processes still preserving their structure without (or by minimising) any human intervention.

In this chapter, we tackle the above challenge by presenting a general approach, a concrete framework 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. Even more importantly, the adaptation mechanisms provided by SmartPM allow deviations at run-time from the execution path prescribed by the original process without altering its process model, a feature that makes SmartPM particularly suitable for managing processes in cyber-physical domains.

The rest of the chapter is organized as follows. In Sect. 2.2 we describe the state-of-the-art approaches to process adaptation, by investigating existing techniques to deal with anticipated and unanticipated exceptions in BPM. In Sect. 2.3, we first present a concrete running example of a process enacted in a cyber-physical environment; then, we derive a list of characterizing features that a PMS managing processes in cyber-physical domains should provide. To meet the identified features, in Sect. 2.4 we introduce the general approach to handle with unanticipated exceptions and exogenous events as defined in the SmartPM framework, and we present the architecture of the implemented SmartPM system. Then, in Sect. 2.5 we provide a critical discussion about the general applicability of the SmartPM approach and we trace the future challenges related to the management of processes in cyber-physical domains. Finally, Sect. 2.6 concludes the chapter.

2.2 Related Work

Over the last years, there was a trend in providing PMSs with a growing support for adapting business processes to deal with exceptions, changing environments and evolving needs [16, 17]. If not detected and handled effectively, exceptions can result in severe impacts on the cost and schedule performance of PMSs [18].

Process adaptation techniques rely on the assumption that exceptions and deviations are detectable [19]. When detection capabilities are provided by the PMSs, mainly in the case of anticipated exceptions, the modeling and execution environment enables process designers to define events, triggers and conditions (e.g., timers, error messages, pre- and post-execution constraints, etc.) whose run-time occurrence or violation is recognized as an exception. When the exceptions and deviations are unanticipated or caused by external factors not under the control of the PMS, users (or external systems) are often allowed to explicitly notify the PMS about the detected exception or deviation.

In this section, we describe the state-of-the-art approaches to process adaptation considering to what extent users are involved in the process of defining exception conditions and handling policies (as summarized in Fig. 2.1), which directly influences the degree of automation provided in the exception resolution and process adaptation stages. Specifically, we first outline traditional exception handling techniques used to deal with anticipated exceptions (Sect. 2.2.1). Then, we review the existing approaches allowing ad hoc process changes for adapting running process instances in case of unanticipated exceptions (Sect. 2.2.2). Finally, we analyze a number of techniques from the field of AI that have been applied to BPM with the aim of increasing the degree of automated process adaptation at run-time (Sect. 2.2.3).

2.2.1 Exception Handling

Initial research efforts addressing the need for exception handling in PMSs can be traced back to the late nineties and early two thousands [15, 20–25]. Although possible sources of anticipated exceptions are different (as outlined in [21, 22],

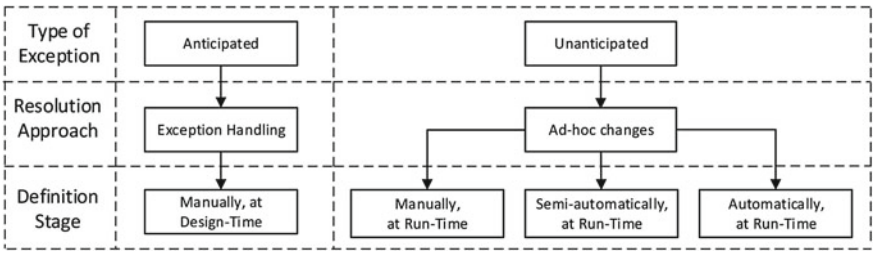


Fig. 2.1 Exception handling and process adaptation approaches

they can be related to activity failures, deadline expirations, resource unavailabilities, constraint violations and external events) and go beyond technical failures, not surprisingly exception handling approaches in PMSs trace and resemble exception handling mechanisms in programming languages. Abstracting from the specific techniques and implementations, a common behavioral pattern can be identified. 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 (including multiple activities), or for the overall process (as in the case of a `try` block in programming languages). The main process logic is thus clearly separated from the exception handling logic. 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. The handler defines specific activities to be performed to recover from the exception, so that process execution can be possibly resumed.

As extensively discussed in [26], exception handling capabilities provided by academic prototypes and commercial PMSs can be reconducted to the abstract framework introduced before. The different approaches vary in the exception types that can be handled and in the way they support the definition and selection of exception handlers, which can be completely predefined, contextually selected from a repository or instantiated from templates. Typical strategies applied when defining exception handlers for anticipated exceptions have been systematized in the form of *exception handling patterns* [16, 27, 28]. When for a given exception no explicit handling logic is defined or the handler is not able to resolve the issue, a process participant may be notified and involved in the definition of corrective actions.

Several exception detection and handler activation techniques [20, 24, 29] 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 that the specific *condition* holds. ECA rules represent a good way for separating the graphical representation of the process with the "exception handling flow". A similar principle has been applied in YAWL [30], where for each exception that can be anticipated, it is possible to define an exception handling process, named *exlet*, which includes a number of exception handling primitives (for removing, suspending, continuing, completing, failing and restarting a workitem/case) and one or more compensatory processes in the form of *worklets* (i.e., self-contained YAWL specifications executed as a replacement for a workitem or as compensatory processes). Exlets are linked to specifications by defining specific rules (through the Rules Editor graphical tool), in the shape of *Ripple Down Rules* specified as "if condition then conclusion", where the condition defines the exception triggering condition and the conclusion defines the exlet.

2.2.2 Ad Hoc Process Change

Even though the handling of anticipated exceptions is fundamental for every PMS, the latter also needs to be able to deal with unanticipated exceptions. Research efforts dealing with unanticipated exceptions have established the area of *adaptive process management* [16, 31]. While the introduction of exception handling techniques for anticipated exceptions increases process flexibility and adaptation capabilities, a different approach is required for handling unanticipated exceptions and deviations occurring at run-time. 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. The need to perform complex behavioral changes over a process instance requires *structural adaptation* of the corresponding process model, which leads to adaptations of the process instance state.

As in the case of exception handling, structural adaptation techniques have been systematized through the identification and definition of adaptation patterns [32, 33]. At a low-level of abstraction, structural model adaptations can be performed by applying change primitives such as adding/removing nodes, routing elements, edges and other process elements. At a higher level of abstraction, change operations provide a set of adaptation patterns to perform model adaptations, such as adding, deleting, moving or replacing activities or process fragments. A single change operation corresponds to the application of multiple change primitives, hiding the complexity of the model editing task. Adaptation patterns are not limited to the control flow perspective and also cover other process perspectives to perform changes, e.g., at the level of the data flow schema or on process resources. In addition, change operations performed for one perspective (e.g., control flow) may affect the other perspectives (e.g., the data flow) as well, resulting in so-called secondary changes. Notice that ad hoc changes must preserve the correctness of the process model and the executability of the process instance [34].

While a good level of support can be provided to ensure correctness and compliance when high-level change operations are performed, the degree of automation in performing these changes is generally limited. In fact, ad hoc changes are often manually performed by experienced users: process execution is suspended and the model and state of the affected instance are adapted by relying on the capabilities of the modeling environment. In an attempt to increase the level of user support, *semi-automated* approaches have been proposed [35]. They aim at storing and exploiting available knowledge about previously performed changed, so that users can retrieve and apply it when adapting a process. Knowledge retrieval and reuse requires establishing a link between performed changes and the application context, including the occurred exception and the process state. Contextual information allows, in turn, identifying similarities between the current exceptional situation and previous cases. The available knowledge on how similar cases were handled in the past is used to assist the users, provide recommendations and suggest possible changes to be

applied. Such an approach has been concretely put into practice using case-based reasoning techniques [36, 37].

Strong support for adaptive process management and exception handling is provided by the ADEPT system and its evolutions [38–41]. ADEPTflex offers modeling capabilities to explicitly define pre-specified exceptions, and supports changes of process instances to enable different kinds of ad hoc deviations from the pre-modeled process models in order to deal with run-time exceptions. These features have been extended and improved in ADEPT2, which provides full support for the structural process change patterns defined in [32], and in ProCycle, which combines ADEPT2 with conversational case-based reasoning (CCBR) methodologies. On the basis of the ADEPT technology, the AristaFlow BPM Suite was developed, with the aim of transferring process flexibility and adaptation concepts into an industrial-strength PMS. Similarly, AgentWork [42] relies on ADEPTflex and exploits a temporal ECA rule model to automatically detect logical failures and enable both reactive and predictive process adaptation of control- and data-flow elements. Here, exception handling is limited to single tasks failures, and the possibility exists for conflicting rules to generate incompatible actions, which requires manual intervention and resolution.

If compared with traditional exception handling approaches (cf. Sect. 2.2.1), adaptive PMSs deal with unanticipated exceptions by automatically deriving the `try` block 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 `catch` block should include those recovery procedures required for realigning the computerized processes with the real-world ones.

2.2.3 AI-based Process Adaptation

The AI community has been involved with research on process management for several decades, and AI technologies can play an important role in the construction of PMS engines that manage complex processes, while remaining robust, reactive, and adaptive in the face of both environmental and tasking changes [43]. One of the first works dealing with this research challenge is [44]. It discusses at high level how the use of an intelligent assistant based on planning techniques may suggest compensation procedures or the re-execution of activities if some anticipated failure arises during the process execution. In [45] the authors describe how planning can be interleaved with process execution and plan refinement, and investigates plan patching and plan repair as means to enhance flexibility and responsiveness. Similarly, the approach presented in [46] highlights the improvements that a legacy workflow application can gain by incorporating planning techniques into its day-to-day operation.

A goal-based approach for enabling automated process instance change in case of emerging exceptions is shown in [47]. If a task failure occurs at run-time and leads to a process goal violation, a multi-step procedure is activated. It includes the termination

of the failed task, the sound suspension of the process, the automatic generation (through the use of a partial-order planner) of a new complete process definition that complies with the process goal and the adequate process resumption. A similar approach is proposed in [48]. The approach is based on learning business activities as planning operators and feeding them to a planner that generates a candidate process model that is capable of achieving some business goals. If an activity fails during the process execution at run-time, an alternative candidate plan is provided with the same business goals. The major issue of [47, 48] lies in the replanning stage used for adapting a faulty process instance, which forces to completely redefine the process specification at run-time when the process goal changes (due to some activity failure), by revolutionizing the work-list of tasks assigned to the process participants (that are often humans).

In the work [49] the authors propose a goal-driven approach for service-based applications to automatically adapt business processes to run-time context changes. Process models include service annotations describing how services contribute to the intended goal, and business policies over domain elements. Contextual properties are modeled as state transition systems capturing possible values and possible evolutions in the case of precondition violations or external events. Process and context evolution are continuously monitored and context changes that prevent goal achievement are managed through an adaptation mechanism based on service composition via automated planning techniques. However, this work requires that the process designer explicitly defines the policies for detecting the exceptions at design-time.

A work dealing with process interference is that of [50]. Process interference is a situation that happens when several concurrent business processes depending on some common data are executed in a highly distributed environment. During the processes execution, it may happen that some of these data are modified causing unanticipated or wrong business outcomes. To overcome this limitation, the work [50] proposes a run-time mechanism that uses (i) *Dependency Scopes* for identifying critical parts of the processes whose correct execution depends on some shared variables; and (ii) *Intervention Processes* for solving the potential inconsistencies generated from the interference, which are automatically synthesised through a domain independent planner based on CSP (*Constraint Satisfaction Problems*) techniques.

2.3 Managing Processes in Cyber-Physical Domains

CPSs are having widespread applicability and proven impact in multiple areas, like aerospace, automotive, traffic management, healthcare, manufacturing, emergency management, entertainment, and consumer appliances [14, 51]. According to [1], any physical environment that contains computing-enabled devices can be considered as a cyber-physical domain. The trend of managing processes in cyber-physical domains has been fueled by two main factors. On the one hand, the recent development of powerful mobile computing devices providing wireless communication capabilities have become useful to support mobile workers to execute tasks in such dynamic

settings [52]. On the other hand, the increased availability of sensors disseminated in the world has lead to the possibility to monitor in detail the evolution of several real-world objects of interest. The knowledge extracted from such objects allows to depict the contingencies and the context in which processes are carried out, by consenting a fine-grained monitoring, mining, and decision support for them.

However, if compared with traditional business domains, additional challenges need to be considered when managing processes in cyber-physical domains. On the one hand, there is the need of representing explicitly real-world objects and "technical" aspects like device capability constraints, wireless networking, device mobility, etc. On the other hand, since cyber-physical domains are intrinsically "dynamic", a PMS that runs a process in such domains must be able to adapt itself to the current real world entities and environment.

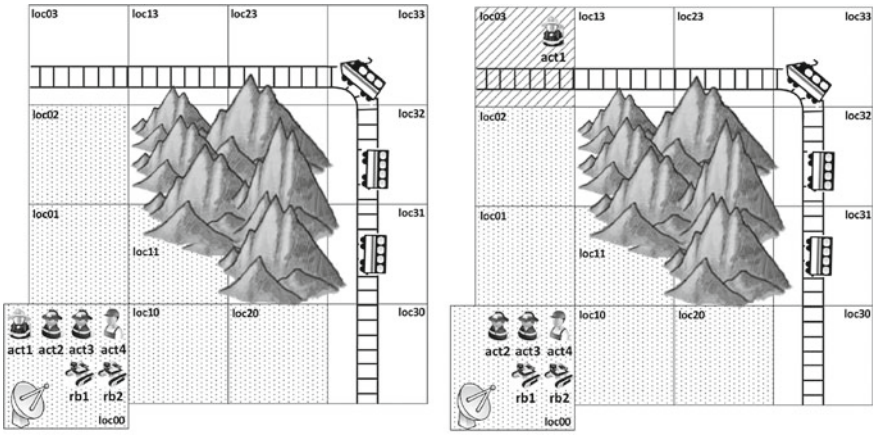
To make our discussion more concrete, in Sect. 2.3.1 we present an application scenario (as running example) that comes from the emergency management domain and is inspired to a real disaster response plan investigated by the authors during the European project WORKPAD[1] [53–56]. Then, starting from the analysis of the application scenario and from the experience gained from participating to several European Projects involving CPSs, in Sect. 2.3.2 we identify a list of high-level features that a PMS aiming at managing and adapting processes in cyber-physical domains should provide.

2.3.1 A Running Example from the Emergency Management Domain

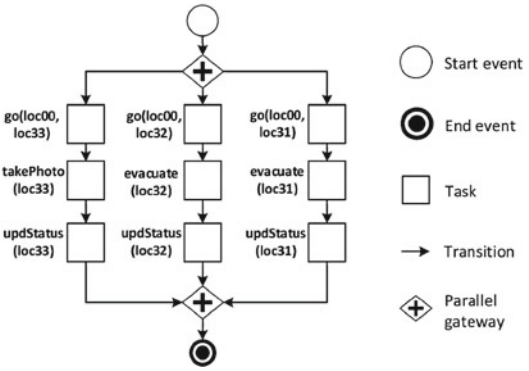
As an application scenario, let us consider the *emergency management* domain, in which teams of first responders act in disaster locations with the main purpose of assisting potential victims and stabilizing the situation. A CPS composed by first responders' mobile devices, robots and wireless communication technologies is coupled with a process-oriented approach for team coordination. A response plan encoded as a process and executed by a PMS deployed on mobile devices can help to coordinate the activities of first responders acting on the field.

To be more concrete, let us consider the emergency management situation described in Fig. 2.2a, in which a train derailment is depicted in a grid-type map. For the sake of simplicity, the train is composed of a locomotive (located at *loc*33) and two coaches (located at *loc*32 and *loc*31, respectively). In our train derailment situation, the goal of an incident response plan is to evacuate people from the coaches and take pictures for evaluating possible damages to the locomotive. To that end, 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 *loc*00. It

[1]The WORKPAD Project (http://www.dis.uniroma1.it/~workpad) investigated how the use of a process-oriented approach can enhance the level of collaboration and support provided to first responders that act in emergency/disaster scenarios.



(a) Train derailment scenario. (b) Failed GO($loc00, loc33$).



(c) Main process.

Fig. 2.2 A train derailment situation; area and context of the intervention

is assumed that actors are equipped with mobile devices for picking up and executing tasks, and that each provides specific capabilities. For example, actor *act*1 is able to extinguish fire and take pictures, while *act*2 and *act*3 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, actor *act*4 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 *loc*00, whose range is limited to the dotted squares in Fig. 2.2a. Such a coverage can be extended by robots *rb*1 and *rb*2, 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. An incident response plan is defined by a set of activities that are meant to be executed on the field by first responders, and are predicated on specific contexts. Therefore, the information

collected on-the-fly is used for defining and configuring at run-time the incident response plan at hand. A possible concrete realization of an incident response plan for our scenario is shown in Fig. 2.2c, using the BPMN modeling language. The process under investigation is composed of three parallel branches, with tasks instructing first responders to act for evacuating people from train coaches in *loc*31 and *loc*32, taking pictures of the locomotive, and assessing the gravity of the accident.

Due to the high dynamism of the environment, there are a wide range of exceptions that can ensue. Because of that, there is not a clear anticipated correlation between a change in the context and a change in the process. So, suppose for instance that actor *act*1 is sent to the locomotive's location, by assigning to it the task GO(*loc*00, *loc*33) in the first parallel branch. Unfortunately, however, the actor happens to reach location *loc*03 instead. The actor is now located at a different position than the desired one, and most seriously, is out of the network connectivity range (cf. Fig. 2.2b). Since all participants need to be continually inter-connected to execute the process, the PMS has to first find a recovery procedure to bring back full connectivity, and then find a way to re-align the process. We notice that the execution of an emergency management process can also be jeopardized by the occurrence of *exogenous events* (e.g., a fire burnt up into a coach, a rock slide collapses in a location, etc.). Indeed, exogenous events could change, in asynchronous manner, some contextual properties of the scenario in which the process is under execution, hence possibly requiring the process to be adapted accordingly.

The above example (though it is very simple) shows that in a cyber-physical domain it is inadequate to assume that a process designer can pre-define all possible recovery activities for dealing with the exceptions that can ensue. The recovery procedures will depend on the actual context (e.g., the positions of process participants, the range of the main network, robot's battery levels, whether a location has become dangerous to get it, etc.) and there are too many of them to be considered at design-time. This emphasizes the fact that for processes enacted in cyber-physical domains there is a critical need of explicit mechanisms to model world changes and responding to them in a fully automated way.

2.3.2 High-Level Features for Managing Processes in Cyber-Physical Domains

The management of processes enacted in cyber-physical domains requires a PMS providing real-time monitoring and automated adaptation features during process execution [16]. To this end, the role of the data perspective becomes fundamental. Data, including information processes by process tasks as well as contextual information, is the main driver for triggering process adaptation, as focusing on the control flow perspective only would be insufficient. In fact, in a cyber-physical domain, a process is genuinely knowledge and data centric: the process control flow must be coupled with contextual data and knowledge production and process progres-

sion may be influenced by user decision making. This means that procedural and imperative models have to be extended and complemented with the introduction of declarative elements (e.g., tasks preconditions and effects) which enable a precise description of data elements and their relations, so as to go beyond simple process variables, and allow establishing a link between the control flow perspective and the data perspective.

Starting from the above considerations, coupled with the experience gained in the area and lessons learned from several European Projects involving CPSs (a partial list can be found in the Acknowledgements section at the end of the chapter), we derive a set of 5 high-level characterizing features that must be provided by a PMS that wants to successfully manage processes enacted in cyber-physical domains:

- [F1] *Representing digitally real-world objects*. The screening of real-world objects performed by the physical sensors disseminated in the world must be taken into consideration when planning and executing a collaborative process in cyber-physical domains. To make the PMS aware of physical reality, a *physical-to-digital bridge* that transforms the knowledge extracted from real-world objects in their digital counterpart is required.
- [F2] *Modeling contextual data*. Contextual data representing the cyber-physical domain in which the process will be enacted and all relevant data affecting the process and manipulated by it need to be formalized and encoded in an *information model*, so as to define data objects and information to be considered as part of the process context and execution state. A process designer should be also allowed to *express conditions over process data*, if needed.
- [F3] *Representing data-driven activities*. A process executed in a cyber-physical domain is characterized by activities whose enactment is related to the evolution of the information model. Such activities are *enriched with declarative elements and constraints* (e.g., preconditions and effects) defined on contextual data, which specify when a particular activity can be executed in a specific state of the contextual scenario, the execution dependencies between activities and the effects that activity executions have on the current state.
- [F4] *Monitoring and exception detection*. The PMS should automatically detect exceptional situations, i.e., any mismatch between the computerized version of the process and its corresponding real-world version. This requires to *monitor running process instances* against the evolution of the process execution context, to identify when a process instance is deviating from the intended behavior.
- [F5] *Exceptions resolution*. The PMS should react to any event that represents a risk for process continuity. If a detected anomalous situation may prevent process progressing, the PMS needs to *automatically deriving and enacting a recovery procedure* that allows the process to progress as expected.

If compared with the features provided by traditional control-flow oriented PMSs (a comprehensive list can be found in [57]), it is clear that processes enacted in cyber-physical domains reveal some challenging features (e.g., data orientation, low predictability, etc.) that pose serious problems for their support through the use of existing approaches [5]. While there is the lack of a holistic approach that allows to

tackle the set of identified features as a whole and to provide a right support for them, we argue that the realization of such an approach can be regarded as a key success factor for the fruitful application of BPM in new domains different from the business one, and represents one main challenge that is currently under investigation by the research community [5, 16]. As a step towards this goal, in the following section we introduce the SmartPM approach and the corresponding implemented system. SmartPM provides a flexible approach to manage the life-cycle of processes enacted in cyber-physical domains, with a targeted support for the set of features discussed above.

2.4 The SmartPM Approach and System

SmartPM[2] (Smart Process Management) [58] is a model and a PMS implementing a set of techniques that enable automatically adapting process instances at run-time in the presence of unanticipated exceptions, without requiring an explicit definition of handlers/policies to recover from tasks failures and exogenous events, and without the intervention of domain experts at run-time.

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 (or exception) to be removed to allow process progression. If an exception 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.

To meet the high-level features described in Sect. 2.3.2, SmartPM relies on and combines well-established AI techniques and frameworks, including the Situation Calculus [59], the IndiGolog framework [60] and automated planning [61]. 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. In the field of BPM, many other formalisms and technologies are being used, such as Petri Nets [62], Coloured Petri Nets [63], Workflow Nets [64], YAWL nets [30], BPMN [3] and process algebras [65], with varying degrees of automated reasoning support over them. While Petri Nets and Worklow Nets do not support data-based decisions as well as data-driven execution of any kind due to the lack of data-awareness [66], other formalisms such as Coloured Petri Nets, YAWL Nets, BPMN and Process Algebras are potentially all fine solutions for realizing our framework. However, the level of abstraction provided for manipulating data values and reasoning over dynamic changes is not formally specified (in the case of YAWL), performed at

[2] http://www.dis.uniroma1.it/~smartpm.

shallow level (in the case of BPMN) or at very low level (in the case of Coloured Petri Nets and Process Algebras), since such formalisms mainly focus on the control-flow perspective of a business process. Conversely, the AI field is rich of algorithms and systems that support the user in the creation, acquisition, adaptation, evolution, and sharing of data knowledge for specifying and implementing dynamic systems [59, 67, 68].

While the formal model underlying SmartPM is described in detail in [58], in this section we aim at providing an overview of the SmartPM approach (cf. Sect. 2.4.1), its concrete implementation (cf. Sect. 2.4.2) and application (cf. Sect. 2.4.3) to the running example introduced in Sect. 2.3.1.

2.4.1 Overview of the Approach

Process Representation

In SmartPM a process model includes a set T of n task definitions. Each task $t_i \in T$ is described in terms of its preconditions Pre_i and effects Eff_i, and can be considered as a single step that consumes input data and produces output data. Data are represented through a set F of *fluents* whose definition strictly depends on the specific process domain of interest. In AI, a fluent is a condition that can change over time. Such fluents can be used to constrain the task assignment (in terms of *task preconditions*), to assess the outcome of a task (in terms of *task effects*) and as guards for decision points and routing elements (e.g., for cycles or conditional statements).

SmartPM adopts a *service-based approach* to process management, that is, tasks are executed by services (that could be software applications, human actors, robots, agents, etc.). Choosing the fluents that are used to describe each activity falls into the general problem of *knowledge representation*. To this end, the environment, services and tasks are grounded in domain theories described in Situation Calculus [59]. Situation Calculus is specifically designed for representing dynamically changing worlds in which all changes are the result of task executions. Situation Calculus is thus used for providing a declarative specification of the domain (i.e., available tasks, contextual properties, tasks preconditions and effects, what is known about the initial state) where a process has to be executed. This declarative specification also covers the resource perspective, with a definition of the available services and the capabilities they provide, to be matched with capability requirements defined for the tasks.

On top of Situation Calculus, SmartPM relies on the IndiGolog high-level agent programming language for the specification of the process control flow. IndiGolog [60] enables the definition of programs with cycles, concurrency, conditional branching and interrupts that rely on program steps that are actions of some domain theory expressed in Situation Calculus. The dynamic world of SmartPM is modeled as progressing through a series of *situations*, where each situation s is the result of the

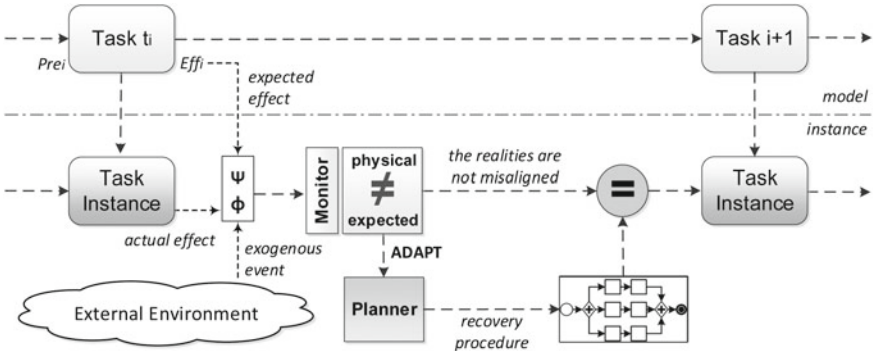


Fig. 2.3 Execution monitoring and adaptation in SmartPM

various tasks performed up to that point. In this context, fluents can be considered as "properties" of the world whose values may vary across situations.

Process Monitoring and Adaptation

SmartPM provides mechanisms for adapting process models that require no predefined handlers. To this end, a specialized version of the concept of adaptation from the field of agent-oriented programming [69] is used. The approach is schematized in Fig. 2.3. Specifically, adaptation in SmartPM can be seen as reducing the gap between the *expected reality*, i.e., the (idealized) model of reality that is used by the PMS to reason, and the *physical reality*, i.e., the real world with the actual values of conditions and outcomes.

The physical reality ϕ_s reflects the concept of "now", i.e., what is happening in the real environment while the process is under execution. The physical reality ϕ_s captures exactly the value assumed by each fluent in the situation s. In general, a task t_i can only be performed in a given physical reality ϕ_s if and only if that reality satisfies the preconditions Pre_i of that task. Moreover, each task has also a set of effects Eff_i that change the current physical reality ϕ_s into a new physical reality ϕ_{s+1}.

A PMS that takes as input a process specification should guarantee that each task is executed correctly, i.e., with an output that satisfies the process specification itself. In fact, at execution time, the process can be easily invalidated because of task failures or since the environment may change due to some external exogenous event. For this purpose, the concept of expected reality ψ_s is introduced. The expected reality in a situation s is given by the set of fluents that are supposed to hold. Basically, when a task is executed and completed, both the physical and expected realities are updated so that:

- the physical reality reflects the actual outcome produced by the task execution;
- the expected reality reflects the intended outcome of the task execution, according to the specification of task's effects.

A recovery procedure is needed in a specific situation if the two realities are different from each other. A misalignment of the two realities often stems from errors in the tasks outcomes (e.g., incorrect data values) or is the result of exogenous events coming from the environment. An execution monitor is responsible for detecting whether the gap between the expected and physical realities is such that the process instance cannot progress. In that case, the PMS has to find a recovery procedure whose execution removes the gap between the physical reality and the expected one.

SmartPM allows the synthesis of a recovery procedure at run-time by invoking an external state-of-the-art planner [61]. Given as the goal condition the process state reflecting the expected reality, the planner searches for a plan that may turn the physical reality into the expected reality. The recovery procedure will be built by composing tasks stored in a specific repository. The repository contains both tasks used for defining the specific process instance under execution and other tasks built on the same contextual scenario and possibly used in past executions of the process. If a recovery plan exists, it will be executed by SmartPM for adapting the faulty process instance.

2.4.2 The SmartPM Environment and Architecture

The concrete implementation of the SmartPM approach has required to cover the modeling, execution and monitoring stages of the process life-cycle and to make explicit the connection of implemented processes with the real-world objects of the cyber-physical domain of interest. To that end, as shown in Fig. 2.4, the architecture of the SmartPM system relies on five architectural layers.

Presentation Layer

The *Presentation Layer* provides a GUI-based tool called SmartPM Definition Tool (cf. Fig. 2.5), which assists the process designer in the definition of a process model at design-time. The SmartPM Definition Tool has been developed using the Java SE 7 Platform, and the JGraphX open source graphical library.[3] To define a process model with the SmartPM Definition Tool means (i) to build a *tasks repository*, (ii) to define the process *control flow* and (iii) to 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*, defined over some *data types*. In short, a data object depicts an entity of interest (e.g., a location, a capability, a service, etc.), while each data type explicitly specifies the data objects that represent the domain of that type. Under this representation, possible values of a data type univocally identify data

[3] http://www.jgraph.com/.

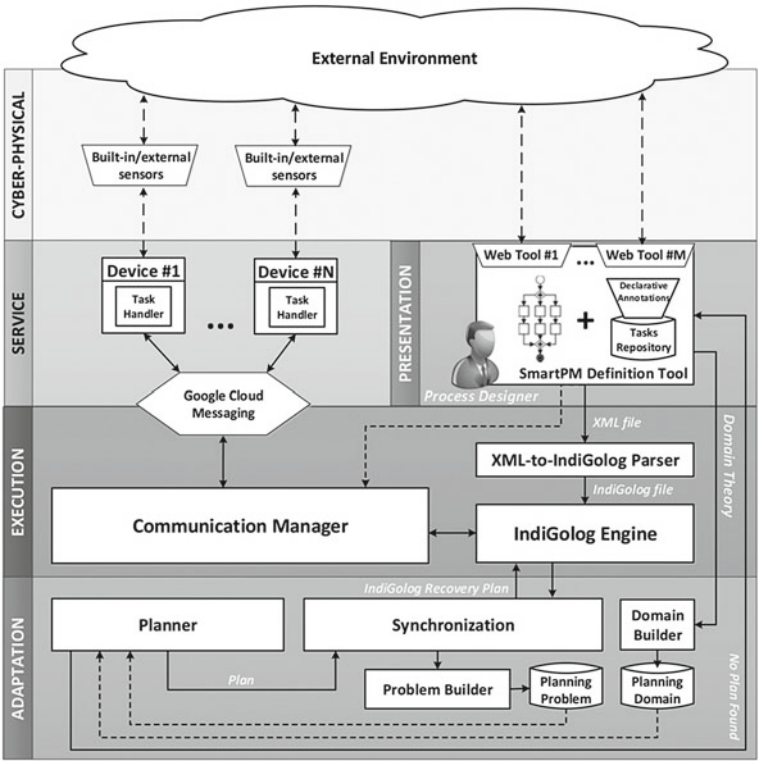


Fig. 2.4 The SmartPM architecture

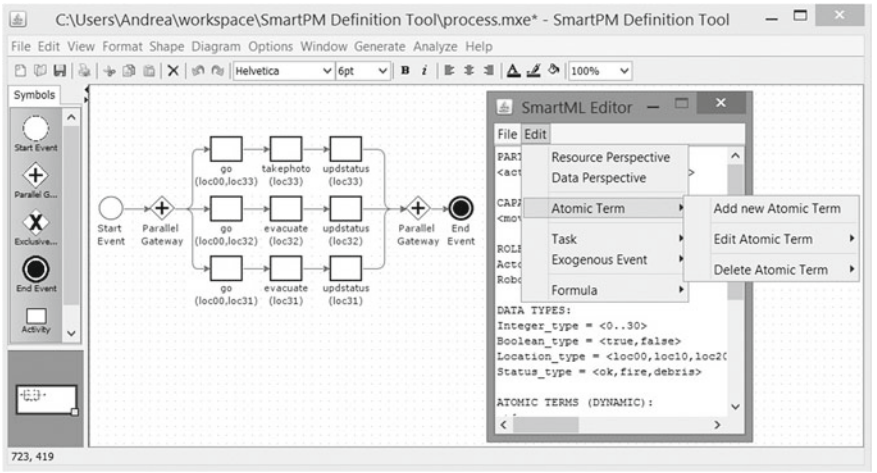


Fig. 2.5 A screenshot of the SmartPM Definition Tool

objects in the scenario of interest. *Atomic terms* can be used to express properties of domain objects (and relations over objects) and argument types of a term—taken from the set of the available data types—represent the finite domains over which the term is interpreted. For example, if we consider the emergency management domain discussed in Sect. 2.3.1, the term $At[act : Actor] = (loc : Location\_type)$ is used for recording the position of each actor in the area. Similarly, the numeric term $BatteryLevel[rbt : Robot] = (int : Integer\_type)$ records the battery level of each robot. 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. For each atomic/complex term, the process designer has to decide which ones are *relevant* for adaptation and which ones have not to be considered for that. An atomic term that is considered as relevant for adaptation will be continuously monitored by the PMS, and if its value becomes different from the one expected, then the adaptation mechanisms provided by SmartPM will be triggered. 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 dynamic scenario under study. Tasks will be stored in a specific *tasks repository* and can be used for composing the control flow of the process and for adaptation purposes. Each task can be considered as a single step that consumes input data and produces output data, and is described with (i) typed input parameters, (ii) preconditions—defined over atomic and complex terms—that constrain the task assignment and must be satisfied before the task is applied, 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 actor *SRV* that will execute it at run-time is at the starting location *from* and provides the required capabilities for executing the task. As a consequence of task execution, the actor moves from the starting to the arrival location, and this is reflected by assigning to the atomic term $At[SRV]$ the value *to* in the effect.

Notice that the definition of a valid domain theory and of tasks specifications allows to meet the features F2 and F3 introduced in Sect. 2.3.2. At this point, 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 editor provides visual, graphical editing and creation of BPMN 2.0 business processes.[4] It is important to notice that atomic/complex terms can be used as guards for decision points and routing elements (e.g., for cycles or conditional

[4]The SmartPM Definition Tool provides a relevant subset of the BPMN modeling constructs to define the control flow of a process, including basic activities, start/end events and parallel/exclusive gateways.

statements). The outcome of the process design activity is a complete XML-encoded process specification that is passed to the *Execution Layer*.

Execution and Service Layers

The *Execution Layer* is in charge of managing and coordinating the process enactment. The BPMN process and the associated domain theory are taken as input from the *XML-to-IndiGolog Parser* component, a Java module that translates them into situation calculus [59] and IndiGolog [60] readable formats (cf. Sect. 2.4.1). It is interesting to notice that 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,[5] written in the well-known open source SWI-Prolog environment,[6] to (i) build a physical reality by taking the initial context from the external environment; (ii) build an expected reality (initially equal to the physical one) that records the expected process state after each task execution or exogenous event occurrence; (iii) manage the process routing and decide which tasks are enabled for execution; (iv) 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 (which may be a human actor, a robot, a software application, etc.) that is available (i.e., free from any other task assignment) and that provides all the required capabilities for task execution.

Process participants interact with the engine through a *Task Handler*, 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 (cf. Fig. 2.6a). The SmartPM Task Handler is realized for Android devices from version 4.0 and up. Each device has an unique ID that matches the service name defined in the domain theory by the designer. Every step of the task life cycle—ranging from the assignment to the release of a task—requires an interaction between the IndiGolog engine and the task handlers. Such an interaction is mainly intended for notifying the device corresponding to the human actor of actions performed by the IndiGolog engine as well as for notifying the engine of actions executed by actor through the task handler of the corresponding device. The communication between the IndiGolog engine and the task handler is mediated by the *Communicator Manager* component (which is essentially a web server) and established using the Google Cloud Messaging (GCM) service.[7]

As previously discussed, the IndiGolog engine is in charge of monitoring contextual data to identify changes or events which may affect process execution, and notify them to the *adaptation layer*. This allows to meet the feature F4 introduced in Sect. 2.3.2. Specifically, given a process instance δ, after each task completion (or exogenous event occurrence), the physical and expected realities are updated to reflect

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

[6] http://www.swi-prolog.org/.

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

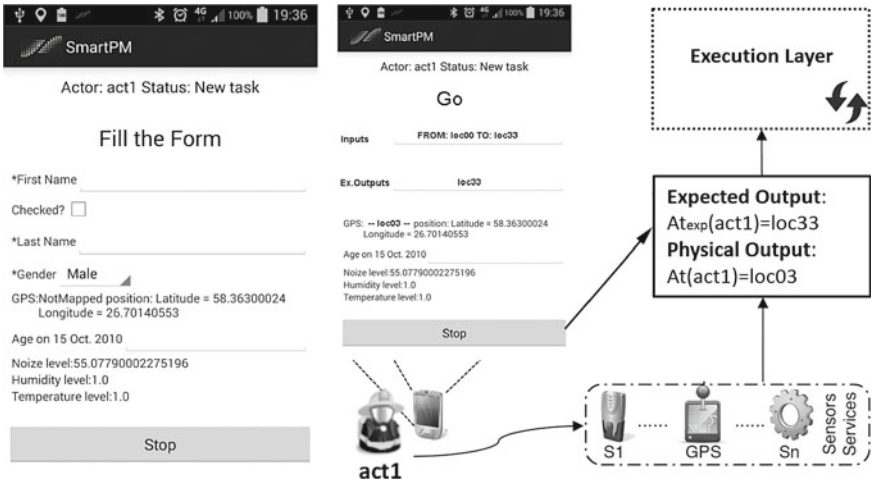


Fig. 2.6 The SmartPM Task Handler

the actual and intended (according to the specification of task's effects) outcome of task performance (or the contextual changes produced by an exogenous event). If we consider the first example shown in Sect. 2.3.1, when the task GO(*loc*00, *loc*33) completes, it means that the output value for *At*[*act*1] (generated as an effect of the task GO) is '*loc*03', that is different from the task's expected outcome, that is '*loc*33'. Hence, the two realities are misaligned, and the faulty process instance δ needs to be adapted (cf. Fig. 2.6b).

Adaptation Layer

To enable the automated synthesis of a recovery procedure, and to provide the right support to feature F5 discussed in Sect. 2.3.2, the *Adaptation Layer* of SmartPM resorts to classical planning techniques. Process adaptation relies on the capabilities provided by a PDDL-based planner component (the LPG-td planner [70]), which assumes the availability of a so-called *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. To this end, if process adaptation is required, the *Domain Builder* component translates (i) the domain theory defined at design-time into a planning domain, while the *Problem Builder* component converts (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 represent the input for the planner component and, in particular, the planning problem reflects the gap between the two realities. 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 and builds an adapted process $\delta'' = (\delta_a; \delta')$. Notice that, whenever a process δ needs to be adapted, every running task is interrupted,

since the recovery sequence of tasks δ_a has to be executed before that the remaining part of the process instance δ' can progress. Thus, active branches can only resume their execution after the repair sequence has been executed. This is fundamental to avoid the risk of introducing data inconsistencies during the repair phase. Finally, the *Synchronization* component converts δ'' 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 and no handling strategy can be automatically derived for the specific deviation, the control passes back to the process designer, who can try to manually manage the exception and adapt the process instance.

Cyber-Physical Layer

The cyber-physical 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, different built-in and external sensors can be used with the SmartPM Task Handler. To exploit sensors that are built 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. Similarly, using the microphone, it is possible to automatically get the current noise level near the device. In addition, external sensors can be taken into use to gather automatic measurements—for prototyping purposes, the Arduino platform can be used.[8] The task handler can take advantage of this technology for gathering environmental data: Arduino has a large variety of sensors available to measure different environmental values, for example different gas levels in the air, water quality, radiation level, etc.; Arduino can be connected with Android 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, to meet feature F1 introduced in Sect. 2.3.2, 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. Notice that in SmartPM finiteness is crucial, as it is one of the main assumptions that makes classical planning possible to the computation of a recovery plan. For example, in the case of the GPS sensor, we developed a location 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., *loc*00, *loc*01, etc.) defined during the design stage of a process through the SmartPM Definition Tool. Figure 2.7 shows a screenshot of the location web 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.

[8] 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.

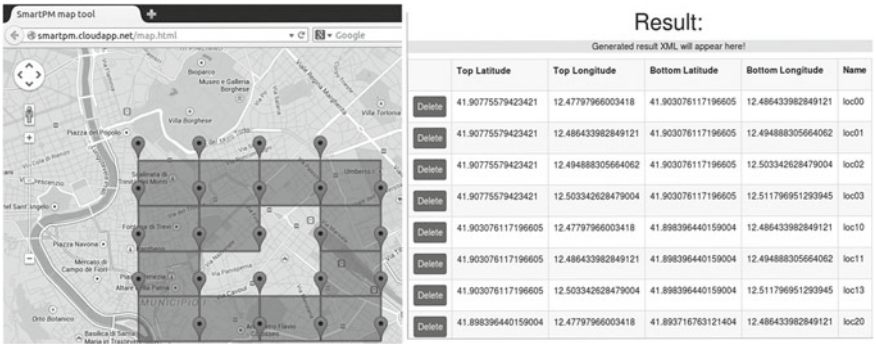


Fig. 2.7 A screenshot of the location web tool provided by the SmartPM Definition Tool

2.4.3 Applying SmartPM to the Running Example

While in the previous sections we discussed the approach underlying SmartPM and the architecture of the implemented SmartPM system, in this section we present a practical application of SmartPM with respect to the running example introduced in Sect. 2.3.1. As anticipated in Sect. 2.4.2, the design of a process to be enacted in a cyber-physical domain starts from the SmartPM Definition Tool, which supports the process design activity. In SmartPM, the process design activity consists of defining the domain theory, the tasks repository and the control flow of the process.

When defining a new domain theory, the very first step to perform involves specifying the *resource perspective* of the process, i.e., the *services* that will be involved in tasks execution and the required *capabilities* to execute those tasks. If we consider the emergency management scenario depicted in Sect. 2.3.1, the following services and capabilities should be defined:

```
Service = {act1,act2,act3,act4,rb1,rb2}
Capability ={movement,hatchet,camera,gps,extinguisher,battery,
digger,powerpack}
```

We notice that the SmartPM Definition Tool allows to explicitly specify the *service providers*, i.e., the real-world entities offering services to perform specific process tasks. Examples of service providers are software components, smartphones, agents, humans, robots, etc. In our running example, two kinds of providers are required, *actors* and *robots*:

```
Actor = {act1,act2,act3,act4}
Robot = {rb1,rb2}
```

To make explicit which capabilities are provided by available services, a special atomic term $Provides[srv : Service, cap : Capability]$ (that is `true` if the capability *cap* is provided by *srv* and `false` otherwise) is used. For example, to state that actor $act1$ owns a mobile device with GPS capabilities, the term $Provides[act1, gps]$

will be set to `true`. Concerning the definition of data, two new data types (Boolean and Integer types are considered as predefined by the SmartPM Definition Tool) are required to capture the objects of interest in the emergency management domain under study.

```
Location_type = {loc00,loc10,loc20,loc30,loc01,loc11,loc02,loc03,
loc13,loc23,loc31,loc32,loc33}
Status_type = {ok,fire,debris}
```

The data type *Status_type* denotes the possible "states" of a location, while *Location_type* represents locations in the area. As discussed at the end of Sect. 2.4.2, data objects representing locations can be associated to real locations through a location web tool. The definition of data types and of the corresponding data objects allows the process designer to explicitly express the contextual properties of the cyber-physical domain under study. Such properties are captured through a finite number of *atomic* and *complex terms*. For our emergency management scenario, the following atomic and complex terms are required:

```
Evacuated[loc:Location_type] = (bool:Boolean_type)
BatteryLevel[rbt:Robot] = (int:Integer_type)
PhotoTaken[loc:Location_type] = (int:Integer_type)
At[srv:Service]= (loc:Location_type)
Status[loc:Location_type] = (st:Status_type)
MoveStep[] = (int:Integer_type)
DebrisStep[] = (int:Integer_type)
Neigh[loc1:Location_type,loc2:Location_type] = (bool:Boolean_type)
Covered[loc:Location_type] = (bool:Boolean_type)
Connected[act:Actor] = {
EXISTS(l1:Location_type, l2:Location_type, rbt:Robot).
   ((at[act]=l1) AND (Covered[l1] OR (at[rbt]=l2 AND
Neigh[l1,l2])))}
```

Therefore, we need boolean terms for indicating if people have been evacuated from a location (*Evacuated*), integer terms for representing the battery charge level of each robot (*BatteryLevel*) or for indicating the number of pictures taken in a specific location (*PhotoTaken*), and functional terms for recording the position of each actor/robot in the area (*At*) or for indicating if a specific location is safe, on fire or under debris (*Status*). Some atomic terms may be used as *constant values*. For example, the terms *MoveStep* and *DebrisStep* reflect the amount of battery consumed respectively when a robot moves from a location to another and when a robot removes debris from a specific location. Finally, atomic terms can also be used for expressing static relations over objects. For example, the atomic term *Neigh* indicates all adjacent locations in the area, while the atomic term *Covered* reflects the locations covered by the network provided by the fixed antenna. For each atomic term, the process designer may decide which ones are *relevant* for triggering the adaptation mechanisms provided by SmartPM. In our example, we can consider as relevant the atomic terms *At*, *Evacuated* and *PhotoTaken*. Finally, as anticipated in Sect. 2.4.2, our emergency management scenario requires also the definition of a complex term *Connected* to denote if an actor is connected to the network.

The definition of the domain theory is the basis to specify the tasks repository and the exogenous events required for the scenario under study. Our running example requires the following tasks and exogenous events:

```
Tasks Repository = {go, move, takephoto, evacuate, updstatus,
extinguishfire, chargebattery}
Ex_events = {photoLost, fireRisk, rockSlide}
```

For each task, the SmartPM Definition Tool provides a wizard-based editor to build a task specification and to define the single conditions composing the task preconditions and effects. We notice that the process designer is required to make explicit if a task effect can be considered as *supposed* or *automatic*. When a task returns some real-world outcome after its completion, we define that outcome as *supposed*, since its physical value may be different from the expected one as thought at design-time. This is the case, for example, of the effect of the task GO (the definition of the task GO has been provided in Sect. 2.4.2), whose consequence is to move an actor from a starting to an arrival location, which can be different from the one expected at design-time. Sometimes it may also happen that a task effect is *automatic*, i.e., it is applied every time a task completes its execution, independently from the outcomes returned by the task itself. For example, when a robot removes debris from a location, its battery decreases of a fixed quantity that does not depend on any physical outcome.

The procedure is similar for the definition of exogenous events. However, in this latter case there is no need to specify any precondition, while effects can only be considered as *automatic* (i.e., they are automatically applied to the involved terms when the exogenous event is catched). For example, the exogenous event ROCKSLIDE(loc) alerts about a rock slide collapsed in location loc, and its effect changes the value of the atomic term $Status[loc]$ to the value '*debris*'.

Starting from the domain theory and the tasks repository just defined, the control flow that captures the response plan of our running example can be built through the BPMN editor provided by the SmartPM Definition Tool (as shown in Fig. 2.5).

The very last step before executing the process consists of instantiating the domain theory with a *starting state*, which reflects an initial assignment of values to the atomic terms. This procedure is performed automatically by the SmartPM Definition Tool, which collects the values of the properties of the cyber-physical domain of interest by querying the sensors installed on services' devices. From a formal point of view, the definition of the starting state corresponds to the creation of the physical and expected realities. In the case of our running example, the initial physical and expected realities reflect the values of the contextual properties of the world before to execute any step of the emergency management process (cf. Fig. 2.2a). A fragment of two realities in the starting state S_0 is shown below:

- $\phi(S_0) = \{At[act1]=loc00, \ldots, Connected[act1]=\texttt{true}, \ldots, Status[loc31]=ok\}$
- $\psi(S_0) = \{At[act1]=loc00, \ldots, Connected[act1]=\texttt{true}, \ldots, Status[loc31]=ok\}$

During process enactment, SmartPM is in charge of assigning tasks to proper services and of continuously monitoring the evolution of the two realities. Let us

consider again our running example, and suppose that actor *act*1 is sent to the locomotive's location, by assigning to it the task GO(*loc*00, *loc*33) in the first parallel branch of the emergency management process defined in Fig. 2.2c. However, as depicted in Fig. 2.2b, the actor happens to reach location *loc*03 instead, meaning that it is now located at a different position than the desired one and is out of the network connectivity range. Consequently, the two realities change as follows:

- $\phi(S_1) = \{At[act1]=loc03, \ldots, Connected[act1]=\texttt{false}, \ldots, Status[loc31]=ok\}$
- $\psi(S_1) = \{At[act1]=loc33, \ldots, Connected[act1]=\texttt{true}, \ldots, Status[loc31]=ok\}$

To re-align the physical reality with the expected one, SmartPM has to first 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, SmartPM may first instruct the first robot to move to cell *loc*03 (cf. Fig. 2.8a) in order to re-establish network connection to actor *act*1, and then instruct the second robot to reach location *loc*23 in order to extend the network range to cover the locomotive's location *loc*33. Finally, task GO(*loc*03, *loc*33) is reassigned to actor *act*1 (cf. Fig. 2.8b). The corresponding updated process is shown in Fig. 2.9a, with the encircled section being the recovery (adaptation) procedure. The two realities are updated as follows:

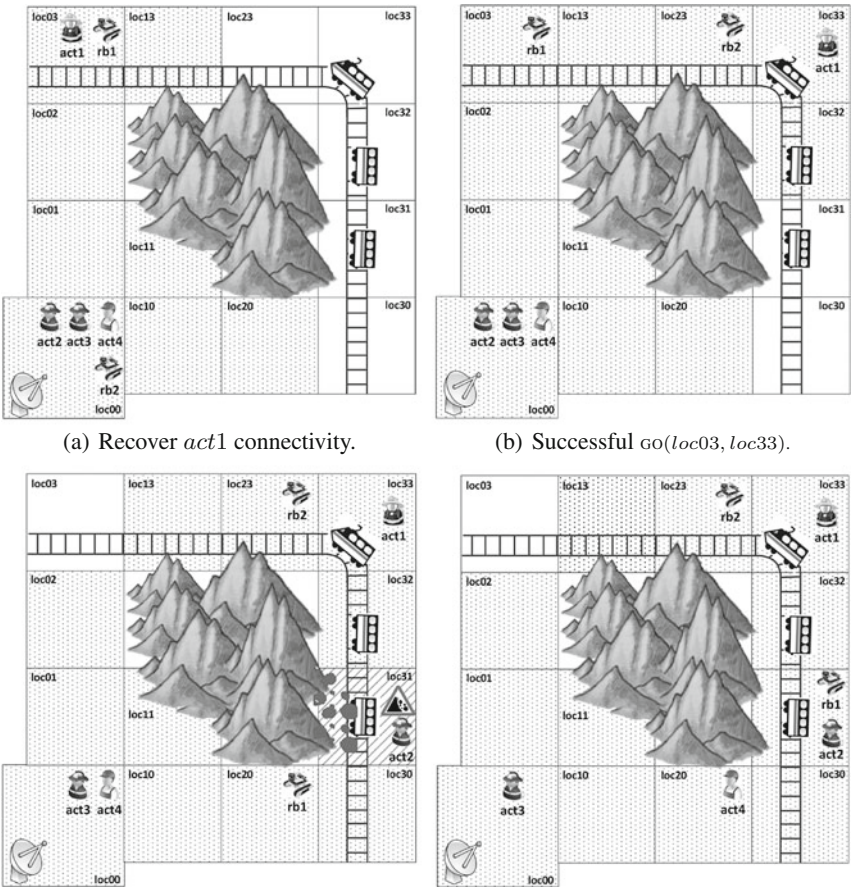
- $\phi(S_2) = \{At[act1]=loc33, \ldots, Connected[act1]=\texttt{true}, \ldots, Status[loc31]=ok\}$
- $\psi(S_2) = \{At[act1]=loc33, \ldots, Connected[act1]=\texttt{true}, \ldots, Status[loc31]=ok\}$

Notice that after the recovery procedure, the enactment of the original process can be resumed to its normal flow. For example, in the third parallel branch, actor *act*2 can now be instructed to reach *loc*31. However, even if *act*2 completes its task as expected (cf. Fig. 2.8c), a further exception is thrown. In fact, *act*2 is out of the network connectivity range and, again, the PMS may instruct the first robot to move from cell *loc*03 to cell *loc*20 in order to re-establish network connection to actor *act*2 (cf. top of Fig. 2.9b). At this point, *act*2 may start evacuating people from *loc*31.

As a further example, let us suppose now that a rock slide collapses in location *loc*31 (cf. Fig. 2.8c) while *act*2 is evaluating the damages in that area (i.e., *act*2 is executing the UPDATESTATUS(*loc*31) task). Such an exogenous event, which corresponds to ROCKSLIDE(*loc*31), changes in asynchronous manner only the physical reality, as follows:

- $\phi(S_3) = \{At[act1]=loc33, Connected[act1]=\texttt{true}, \ldots, Status[loc31]=debris\}$
- $\psi(S_3) = \{At[act1]=loc33, Connected[act1]=\texttt{true}, \ldots, Status[loc31]=ok\}$

In such a case, SmartPM needs first to abort the running task UPDATESTATUS(*loc*31) (the presence of a rock slide may possibly prevent the correct execution of the task), and then to find a recovery procedure that allows to remove the rock slide from *loc*31 by maintaining all the process participants inter-connected to the network. A possible solution is shown in Fig. 2.8d, and consists of instructing *act*4 to reach *loc*20 for recharging the battery of *rb*1, of moving the robot *rb*1 in *loc*31 in order to remove debris, and finally of reassigning the UPDATESTATUS(*loc*31) task to *act*2. The corresponding adapted process is shown in the bottom of Fig. 2.9b, and the two realities are updated as follows:



(a) Recover $act1$ connectivity.

(b) Successful $\text{GO}(loc03, loc33)$.

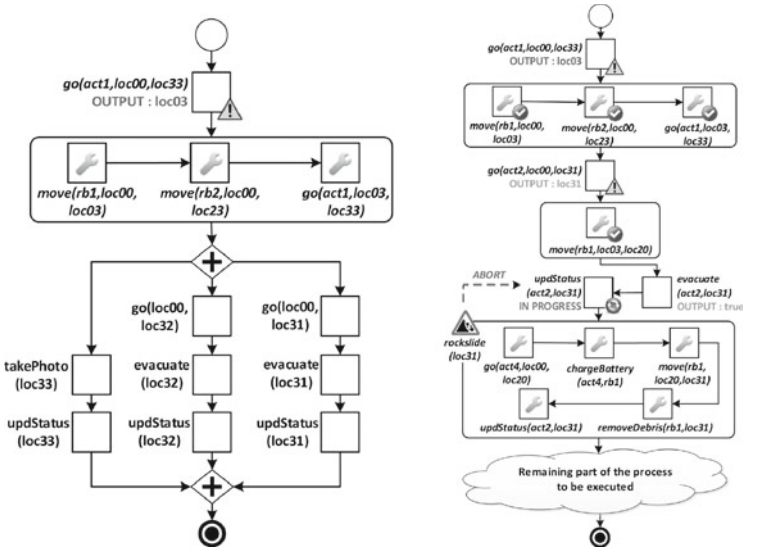
(c) Recover $act2$ connectivity.

(d) Final stage.

Fig. 2.8 Evolution of the contextual scenario introduced in Sect. 2.3.1

- $\phi(S_4)$ = {At[$act1$]=$loc33$, $Connected$[$act1$]=`true` , ..., $Status$[$loc31$]=ok}
- $\psi(S_4)$ = {At[$act1$]=$loc33$, $Connected$[$act1$]=`true` , ..., $Status$[$loc31$]=ok}

It is worth noting that we validated the SmartPM approach with a case study based on real processes coming from the emergency management domain. Specifically, we first performed empirical experiments on synthetic data by enacting several emergency management processes, and they confirm the feasibility of the planning-based approach provided by SmartPM for adapting processes in medium-sized cyber-physical domains from the timing performance perspective. Then, we tested the SmartPM System with 3600 different process models having control flows with different structures (and different domain theories associated to them) to measure the effectiveness of SmartPM in adapting processes. We define the *effectiveness* of a PMS as the *ability of a PMS to complete the execution of a process model (i.e., to*



(a) Adapted process after a task failure

(b) Adapted process after an exogenous event

Fig. 2.9 Recovery procedures for the emergency response plan introduced in Sect. 2.3.1

execute all the tasks involved in a path from the start event to the end event) by adapting automatically its running process instance if some failure arises, without the need of any manual intervention of the process designer at run-time. To evaluate the effectiveness of SmartPM, we simulated processes execution by introducing task failures and exogenous events during the process enactment according to a given probability. As an instance, if the percentage of tasks failures was equal to 70 % and the process model to be executed was composed of 10 tasks, we had that 7 tasks of its running process instance completed with some physical outcome different from the one expected, thus requiring the process to be adapted. To sum up, 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, we invite the interested reader to refer to [58].

2.5 Discussion

The analysis performed in this chapter underlines that processes enacted in cyber-physical domains demand a more flexible approach to process management, recognizing the fact that in real-world environments process models quickly become outdated and hence require closer interweaving of modeling and execution. The fact is that the common strategy used by the adaptive PMSs to deal with exceptions is to

manually or *semi-automatically* define recovery procedures at run-time. However, in 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 any specific adaptation policy or exception handler at design-time. Furthermore, if compared with the existing techniques on process adaptation coming from the field of AI, the SmartPM approach provides unique features that make it particularly suitable for managing processes in cyber-physical domains. For example, if compared with the works [47, 48] (discussed in Sect. 2.2.3), the SmartPM approach adapts a running process instance by modifying only those parts of the process that need to be changed/adapted and keeps other parts stable. This is particularly important, as processes executed in cyber-physical domains often involve real human participants, and to completely re-define the process specification at run-time for adaptation purposes would mean to revolutionize the work-list of tasks assigned to the process participants. Finally, while closely related to works [49, 50], the SmartPM approach deals with changes in a more abstract and domain-independent way, by just checking misalignment between expected/physical realities. Conversely, the work [49] requires that the process designer explicitly defines the policies for detecting the exceptions at design-time, while the work [50] requires specification of a (domain-dependent) adaptation policy, based on volatile variables and when changes to them become relevant.

From a general perspective, the planning-based automated exception handling approach of SmartPM should be considered as complementary with respect to existing techniques, acting as a "bridge" between pre-planned approaches and unplanned approaches. When an exception occurs and is detected, the run-time engine may first check the availability of a predefined exception handler, and if no handler was defined, it can rely on an automated synthesis of the recovery process. In the case that a planning-based approach fails in synthesizing a suitable handler (or a handler is generated but its execution does not solve the exception), a human participant can be involved, leaving her/him the task of manually adapting the process instance.

The use of classical planning techniques for the synthesis of the recovery procedure has a twofold consequence. On the one hand, we can exploit the good performance of current state-of-the-art planners to solve medium-sized real-world problems as used in practice (cf. [58]). On the other hand, classical planning imposes some restrictions for addressing more expressive problems, including incomplete information, preferences and multiple task effects. To sum up, specific requirements frame the scope of applicability of the approach, which basically relies on the following assumptions:

1. process structure can be completely captured in a procedural predefined process model that explicitly defines the tasks and their execution constraints;
2. process execution context can be fully captured as part of the process model, i.e., complete information about a fully observable domain is available;
3. domain objects and contextual properties representing the state of the world can be reconducted to a finite set of finite-domain variables;
4. process tasks can be completely specified in term of I/O data elements, preconditions and deterministic effects.

Moreover, in addition to the full observability assumption, the approach relies on a high degree of *controllability* over the environment: when process execution deviates from the prespecified expected behavior (i.e., the physical reality deviates from the expected one), it should be possible to synthesize a recovery process whose execution modifies the environment (as reflected in the physical reality) so that the process instance can progress as expected, according to the prespecified model (basically, the physical reality is reconducted to the expected reality). When the operational environment and process state cannot be reconducted to their expected representation, we are back to the case where a process cannot be recovered to progress according to the predefined model, and it is the process itself that has to be (manually or semi-automatically) adapted to the changed environment.

The above assumptions result from the need of balancing between modeling complexity and expressive power, and the practical requirements that enable exploiting classical planning tools. Although the need to explicitly model process execution context and annotate tasks with preconditions and effects may require some extra modeling effort at design-time (also considering that process modeling efforts are often mainly directed to the sole control flow perspective), the overhead is compensated at run-time by the possibility of automating exception handling procedures.

2.6 Conclusion

In this chapter we have introduced a general approach, a concrete framework and a PMS, called SmartPM, for automated adaptation of processes enacted in cyber-physical domains in case of unanticipated exceptions. The approach is based on declarative task specifications and planning techniques, and relies on the ability of automatically synthesizing recovery procedures at run-time. No predefined exception detection and handling logic is thus required. The current prototype of SmartPM is developed to be effectively used by process designers and practitioners. Users define processes in the well-known BPMN language, enriched with semantic annotations for expressing properties of tasks, which allow our interpreter to derive the IndiGolog program representing the process. Interfaces with human actors (as specific graphical user applications in Java) and software services (through Web service technologies) allow the core system to be effectively used for enacting processes.

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*. In fact, in the last years, the current widespread availability of wireless network technology for mass consumption has triggered the appearance of plenty of wireless and/or mobile devices providing applications able to enhance the visitors' experience in cultural sites. The "pre-fixed" and static visits of physical spaces have been turned into interactive dynamic experiences customized to the human visitors' behaviours and needs. In this context, a process can be used to personalize the visit of an individual into a smart space.

In addition, we aim at turning the *centralized control* provided by SmartPM (in which the reasoning is performed by a single entity, which subsequently instructs the process participants what to do) into a *decentralized control*, in which each participant will be provided with her/his mobile device with the SmartPM system installed into it. The challenge is to provide each SmartPM system with the ability to adapt the single processes of individual process participants by considering not only the local knowledge collected by the single participant, but also the knowledge produced by the other visitors of the smart space and the global knowledge provided by the smart space as a whole (e.g., the knowledge produced by the sensors installed in the smart space). As shown in [71, 72], our research is already going in this direction.

Acknowledgements This work has been partly supported over the years by the following projects: EU FP-6 WORKPAD, EU FP-7 SM4All, Italian Sapienza grant TESTMED, Italian Sapienza grant SUPER, Italian Sapienza award SPIRITLETS, Italian cluster Social Museum and Smart Tourism, Italian project NEPTIS, Italian project RoMA. The authors would like to thanks the many persons involved over the years in the SmartPM conception and development, namely Giuseppe De Giacomo, Massimiliano de Leoni, Patris Halapuu, Arthur H.M. ter Hofstede, Alessandro Russo, Sebastian Sardina, Paola Tucceri, Stefano Valentini.

References

1. Lee, E.A.: Cyber Physical Systems: Design Challenges. In: 11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC). IEEE Computer Society, pp. 363–369 (2008)
2. Weske, M.: Business Process Management—Concepts, Languages, Architectures (2nd edn). Springer (2012)
3. Allweyer, T.: BPMN 2.0: Introduction to the Standard for Business Process Modeling. BoD–Books on Demand (2010)
4. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., et al.: Business Process Execution Language for Web Services. Version 1.1. (2003)
5. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches. J. Data Semant. **4**(1), 29–57 (2015)
6. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive processes: an overview of contemporary approaches. In: 1st International Workshop on Knowledge-intensive Business Processes (KiBP). CEUR Workshop Proceedings, vol. 861. CEUR-WS.org (2012)

7. Lenz, R., Reichert, M.: IT support for healthcare processes—premises, challenges, perspectives. Data Knowl. Eng. **61**(1), 39–58 (2007)
8. Cossu, F., Marrella, A., Mecella, M., Russo, A., Bertazzoni, G., Suppa, M., Grasso, F.: Improving operational support in hospital wards through vocal interfaces and process-awareness. In: 25th International Symposium on Computer-Based Medical Systems (CBMS), pp. 1–6. IEEE Computer Society (2012)
9. Cossu, F., Marrella, A., Mecella, M., Russo, A., Kimani, S., Bertazzoni, G., Colabianchi, A., Corona, A., Luise, A.D., Grasso, F., Suppa, M.: Supporting doctors through mobile multimodal interaction and process-aware execution of clinical guidelines. In: 7th International Conference on Service-Oriented Computing and Applications (SOCA), pp. 183–190. IEEE (2014)
10. Marrella, A., Mecella, M.: Continuous planning for solving business process adaptivity. In: 12th International Working Conference on Business Process Modeling, Development and Support (BPMDS). Lecture Notes in Business Information Processing, vol. 81, pp. 118–132. Springer (2011)
11. Marrella, A., Russo, A., Mecella, M.: Planlets: automatically recovering dynamic processes in YAWL. In: 20th International Conference on Cooperative Information Systems (CoopIS). Lecture Notes in Computer Science, vol. 7565, pp. 268–286. Springer (2012)
12. Seiger, R., Keller, C., Niebling, F., Schlegel, T.: Modelling complex and flexible processes for smart cyber-physical environments. J. Comput. Sci. (2014)
13. Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., Jansen, E.: The gator tech smart house: a programmable pervasive space. IEEE Comput. **38**(3), 50–60 (2005)
14. Rajkumar, R.R., Lee, I., Sha, L., Stankovic, J.: Cyber-physical systems: the next computing revolution. In: 47th Design Automation Conference (DAC), pp. 731–736. ACM (2010)
15. Klein, M., Dellarocas, C.: A knowledge-based approach to handling exceptions in workflow systems. Comput. Support. Coop. Work (CSCW) **9**(3–4), 399–412 (2000)
16. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer (2012)
17. Marrella, A., Mecella, M., Russo, A.: Featuring automatic adaptivity through workflow enactment and planning. In: 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), pp. 372–381. ICST/IEEE (2011)
18. Klein, M., Dellarocas, C., Bernstein, A.: Introduction to the special issue on adaptive workflow systems. Comput. Support. Coop. Work (CSCW) 9(3–4), 265–267 (2000)
19. Sadiq, S., Orlowska, M.: On capturing exceptions in workflow process models. In: 3rd International Conference on Business Information Systems (BIS), pp. 3–19. Springer London (2000)
20. Casati, F., Ceri, S., Paraboschi, S., Pozzi, G.: Specification and implementation of exceptions in workflow management systems. ACM Trans. Database Syst. (TODS) **24**(3), 405–451 (1999)
21. Casati, F., Cugola, G.: Error handling in process support systems. In: Advances in Exception Handling Techniques (ECOOP). Lecture Notes in Computer Science, vol. 2022, pp. 251–270. Springer (2001)
22. Eder, J., Liebhart, W.: The workflow activity model WAMO. In: Third International Conference on Cooperative Information Systems (CoopIS), pp. 87–98 (1995)
23. Eder, J., Liebhart, W.: Workflow recovery. In: First IFCIS International Conference on Cooperative Information Systems (CoopIS), pp. 124–134. IEEE Computer Society (1996)
24. Hagen, C., Alonso, G.: Exception handling in workflow management systems. IEEE Trans. Softw. Eng. **26**(10), 943–958 (2000)
25. Luo, Z., Sheth, A., Kochut, K., Miller, J.: Exception handling in workflow systems. Appl. Intell. **13**(2), 125–147 (2000)
26. Adams, M.J.: Facilitating Dynamic Flexibility and Exception Handling for Workflows. Ph.D. thesis, Queensland University of Technology Brisbane, Australia (2007)
27. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Workflow exception patterns. Advanced Information Systems Engineering. Lecture Notes in Computer Science, vol. 4001, pp. 288–302. Springer, Berlin Heidelberg (2006)
28. Lerner, B.S., Christov, S., Osterweil, L.J., Bendraou, R., Kannengiesser, U., Wise, A.: Exception handling patterns for process modeling. IEEE Trans. Softw. Eng. **36**(2), 162–183 (2010)

29. Chiu, D.K.W., Li, Q., Karlapalem, K.: A logical framework for exception handling in ADOME workflow management system. In: 12th International Conference on Advanced Information Systems Engineering (CAiSE). Lecture Notes in Computer Science, vol. 1789, pp. 110–125. Springer (2000)
30. ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its Support Environment. Springer (2009)
31. Weske, M.: Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In: 34th Annual Hawaii International Conference on System Sciences (HICSS). IEEE Computer Society (2001)
32. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features—enhancing flexibility in process-aware information systems. Data Knowl. Eng. **66**(3), 438–466 (2008)
33. Reichert, M., Weber, B.: Process Change Patterns: Recent Research, Use Cases, Research Directions. In: Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE, pp. 397–404. Springer (2013)
34. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems—a survey. Data Knowl. Eng. **50**(1), 9–34 (2004)
35. Rinderle, S., Weber, B., Reichert, M., Wild, W.: Integrating process learning and process evolution—a semantics based approach. In: 3rd International Conference on Business Process Management (BPM). Springer (2005)
36. Weber, B., Wild, W., Breu, R.: CBRFlow: enabling adaptive workflow management through conversational case-based reasoning. Lecture Notes in Computer Science, vol. 3155, pp. 434–448. Springer (2004)
37. Minor, M., Bergmann, R., Görg, S.: Case-based adaptation of workflows. Inf. Syst. **40**, 142–152 (2014)
38. Reichert, M., Dadam, P.: ADEPTflex—supporting dynamic changes of workflows without losing control. J. Intell. Inf. Syst. **10**(2), 93–129 (1998)
39. Reichert, M., Rinderle, S., Dadam, P.: ADEPT workflow management system. In: Business Process Management (BPM). Lecture Notes in Computer Science, vol. 2678, pp. 370–379. Springer (2003)
40. Reichert, M., Rinderle, S., Kreher, U., Dadam, P.: Adaptive process management with ADEPT2. In: 21st International Conference on Data Engineering (ICDE), pp. 1113–1114. IEEE Computer Society (2005)
41. Lanz, A., Reichert, M., Dadam, P.: Robust and flexible error handling in the AristaFlow BPM suite. In: Information Systems Evolution—CAiSE Forum 2010, Selected Extended Paper. Lecture Notes in Business Information Processing, vol. 72, pp. 174–189. Springer (2011)
42. Müller, R., Greiner, U., Rahm, E.: AGENT WORK: a workflow system supporting rule-based workflow adaptation. Data Knowl. Eng. **51**(2) (2004)
43. Myers, K., Berry, P.: Workflow management systems: an AI perspective. AIC-SRI report (1998)
44. Beckstein, C., Klausner, J.: A meta level architecture for workflow management. J. Integr. Des. Process Sci. **3**(1), 15–26 (1999)
45. Jarvis, P., Moore, J., Stader, J., Macintosh, A., du Mont, A.C., Chung, P.: Exploiting AI technologies to realise adaptive workflow systems. AAAI Workshop on Agent-Based Systems in the Business Context (1999)
46. R-Moreno, M.D., Kearney, P.: Integrating AI planning techniques with workflow management system. Knowl. Based Syst. **15**(5–6), 285–291 (2002)
47. Gajewski, M., Meyer, H., Momotko, M., Schuschel, H., Weske, M.: Dynamic failure recovery of generated workflows. In: 16th International Workshop on Database and Expert Systems Applications (DEXA), pp. 982–986. IEEE Computer Society Press (2005)
48. Ferreira, H., Ferreira, D.: An integrated life cycle for workflow management based on learning and planning. Int. J. Coop. Inf. Syst. **15**(4), 485–505 (2006)
49. Bucchiarone, A., Pistore, M., Raik, H., Kazhamiakin, R.: Adaptation of service-based business processes by context-aware replanning. In: 4th International Conference on Service-Oriented Computing and Applications (SOCA), pp. 1–8 (2011)

50. van Beest, N., Kaldeli, E., Bulanov, P., Wortmann, J., Lazovik, A.: Automated runtime repair of business processes. Inf. Syst. **39**, 45–79 (2014)
51. Baheti, R., Gill, H.: Cyber-Physical Systems. The Impact of Control Technology. Technical Report (2011)
52. Neyem, A., Franco, D., Ochoa, S.F., Pino, J.A.: An approach to enable workflow in mobile work scenarios. In: 11th International Conference on Computer Supported Cooperative Work in Design IV (CSCWD), Lecture Notes in Computer Science, vol. 5236, pp. 498–509. Springer (2007)
53. Catarci, T., de Leoni, M., Marrella, A., Mecella, M., Salvatore, B., Vetere, G., Dustdar, S., Juszczyk, L., Manzoor, A., Truong, H.L.: Pervasive software environments for supporting disaster responses. IEEE Internet Comput. **12**(1), 26–37 (2008)
54. Humayoun, S.R., Catarci, T., de Leoni, M., Marrella, A., Mecella, M., Bortenschlager, M., Steinmann, R.: The WORKPAD user interface and methodology: developing smart and effective mobile applications for emergency operators. In: 5th International Conference on Universal Access in Human-Computer Interaction. Applications and Services (UAHCI). Lecture Notes in Computer Science, vol. 5616, pp. 343–352. Springer (2009)
55. Catarci, T., de Leoni, M., Marrella, A., Mecella, M., Russo, A., Steinmann, R., Bortenschlager, M.: WORKPAD: process management and geo-collaboration help disaster response. Int. J. Inf. Syst. Crisis Response Manag. (IJISCRAM) **3**(1), 32–49 (2011)
56. Marrella, A., Mecella, M., Russo, A.: Collaboration on-the-field: suggestions and beyond. In: 8th International Conference on Information Systems for Crisis Response and Management (ISCRAM) (2011)
57. van der Aalst, W.M.P.: Business process management: a comprehensive survey. ISRN Software Engineering (2013)
58. Marrella, A., Mecella, M., Sardina, S.: SmartPM: An adaptive process management system through situation calculus, indigolog, and classical planning. In: 14th International Conference on Principles of Knowledge Representation and Reasoning (KR). AAAI Press (2014)
59. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
60. De Giacomo, G., Lespérance, Y., Levesque, H., Sardina, S.: IndiGolog: a high-level programming language for embedded reasoning agents. In: Multi-Agent Programming, pp. 31–72. Springer, US (2009)
61. Nau, D., Ghallab, M., Traverso, P.: Autom. Plan. Theory Pract. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004)
62. van Der Aalst, W.M.P.: Three good reasons for using a petri-net-based workflow management system. In: International Working Conference on Information and Process Integration in Enterprises (IPIC'96), pp. 179–201. Cambridge, Massachusetts (1996)
63. Jensen, K.: Coloured Petri Nets. In: Petri Nets: Central Models and their Properties, pp. 248–299. Springer (1987)
64. van der Aalst, W.M.P.: The application of petri nets to workflow management. J. Circuits Syst. Comput. **8**(01), 21–66 (1998)
65. Puhlmann, F., Weske, M.: Using the π-calculus for formalizing workflow patterns. Business Process Management (BPM). Lecture Notes in Computer Science, vol. 3649, pp. 153–168. Springer, Berlin Heidelberg (2005)
66. Meyer, A., Smirnov, S., Weske, M.: Data in Business Processes. No. 50, Universitätsverlag Potsdam (2011)
67. Reichgelt, H.: Knowledge representation: an AI perspective. Ablex (1991)
68. Brachman, R., Levesque, H.: Knowledge Representation and Reasoning. Morgan Kaufmann Publishers Inc. (2004)
69. De Giacomo, G., Reiter, R., Soutchanski, M.: Execution monitoring of high-level robot programs. In: 6th International Conference on Principles of Knowledge Representation and Reasoning (KR), pp. 453–465. Morgan Kaufmann (1998)
70. Gerevini, A., Saetti, A., Serina, I., Toninelli, P.: LPG-TD: a fully automated planner for PDDL2.2 domains. In: 14th International Conference on Automated Planning and Scheduling (ICAPS). International Planning Competition abstracts (2004)

71. Marrella, A., Lespérance, Y.: Towards a Goal-oriented framework for the automatic synthesis of underspecified activities in dynamic processes. In: 6th International Conference on Service-Oriented Computing and Applications (SOCA), pp. 361–365. IEEE (2013)
72. Marrella, A., Lespérance, Y.: Synthesizing a library of process templates through partial-order planning algorithms. In: 14th International Working Conference on Business Process Modeling, Development and Support (BPMDS). Lecture Notes in Business Information Processing, vol. 147, pp. 277–291. Springer (2013)

http://www.springer.com/978-3-319-52179-4

Advances in Intelligent Process-Aware Information Systems
Concepts, Methods, and Technologies
Grambow, G.; Oberhauser, R.; Reichert, M. (Eds.)
2017, XI, 249 p. 99 illus., Hardcover
ISBN: 978-3-319-52179-4