Making YAWL and SmartPM Interoperate: Managing Highly Dynamic Processes by Exploiting Automatic Adaptation Features

Andrea Marrella[1], Massimo Mecella[1], Alessandro Russo[1]
Arthur H.M. ter Hofstede[2,3], and Sebastian Sardina[4]

[1] Sapienza Università di Roma, Italy
{marrella|mecella|arusso}@dis.uniroma1.it
[2] Queensland University of Technology, Australia
[3] Eindhoven University of Technology, the Netherlands
a.terhofstede@qut.edu.au
[4] RMIT University, Australia
sebastian.sardina@rmit.edu.au

Abstract. In the last years, the trade-off between flexibility and support has become a leading issue in workflow technology. In this paper we show how an imperative modeling approach used to define stable and well-understood processes can be complemented by a modeling approach that enables automatic process adaptation and exploits planning techniques to deal with environmental changes and exceptions that may occur during process execution. To this end, we designed and implemented a Custom Service that allows the Yawl execution environment to delegate the execution of subprocesses and activities to the SmartPM execution environment, which is able to automatically adapt a process to deal with emerging changes and exceptions. We demonstrate the feasibility and validity of the approach by showing the design and execution of an emergency management process defined for train derailments.

1 Introduction

Process Management Systems (PMSs, a.k.a. Workflow Management Systems) are applied to support and automate the enactment of processes with the aim to increase their efficiency and effectiveness. Classical PMSs offer adequate process support as long as the processes are structured and do not require much flexibility. In the last years, the trade-off between flexibility and support has become a leading issue in workflow technology [9]. An important aspect of flexibility is the ability to react to exceptions that may occur during runtime by dynamically adapting the process.

A PMS that supports automatic adaptivity is able to automatically change the schema of affected instances in such a way that they can still be completed, according to the exceptions that have been raised. Process schemas are designed in order to cope with potential exceptions, i.e., for each kind of exception that is envisaged to occur, a specific contingency process (a.k.a. exception handler

or compensation flow) is defined, with the challenge that in many cases such a compensation cannot be performed by simply undoing and then redoing certain actions. Such an exception handler can be compared to the `try-catch` approach used in some programming languages such as `Java`; the `catch` is the definition of the possible exception and the specification, defined at design-time by the process engineer, of how to deal with it. An interesting example is provided by the exception handling capabilities of the YAWL system [8], that is among the most well-known PMSs coming from academia. For each exception that can be anticipated, it is possible to define an exception handling process, referred to as an *exlet*, which includes a number of exception handling primitives (for removing, suspending, continuing, completing, failing and restarting a workitem) and one or more compensatory processes in the form of further YAWL processes or in the form of *worklets* (i.e., self-contained YAWL specifications executed as a replacement for a workitem or as compensatory processes).

Taken YAWL's exception handling framework as a starting point, the innovation of this work is the automatic construction of exception handlers at run time, i.e. the automatic synthesis of the `catch` blocks. Currently, exception handlers for exceptions that have not occurred before and have not been anticipated beforehand have to be defined by a process administrator and this is a manual activity. The proposed demo shows how an imperative modeling approach used to define stable and well-understood processes (we will illustrate this by extending the YAWL environment as it provides strong support for plugging in external tools and services) can be complemented by a modeling approach that enables automatic process adaptation and exploits planning techniques to deal with environmental changes and exceptions that may occur during process execution. To this end, we designed and implemented a Custom Service that allows the YAWL execution environment to delegate the execution of subprocesses and activities to the SMARTPM execution environment [5, 4], which is able to automatically adapt a process to deal with emerging changes and exceptions. We demonstrate the feasibility and validity of the approach by showing the design and execution of an emergency management process defined for train derailments. The high dynamism of the operating environment requires some activities to be executed by the SMARTPM subsystem and we show how it is able to automatically build and execute recovery plans in response to environmental changes and external events.

2 The SmartPM Execution Environment

SMARTPM (Smart Process Management) [5, 4] is a model and a proof-of-concept PMS featuring a set of techniques providing support for automatic adaptation of processes. Such techniques are able to automatically adapt processes without explicitly defining handlers/policies to recover from exogenous events and without the intervention of domain experts. SMARTPM adopts a *service-based* approach to process management, that is, tasks are executed by services (that could be software applications, human actors, robots). The environment, services

and tasks are grounded in domain theories described in Situation Calculus [6]. Situation Calculus is specifically designed for representing dynamically changing worlds in which all changes are the result of the tasks' execution. Each task is described in terms of its preconditions and effects, and can be considered as a single step that consumes input data and produces output data. Data are represented through process variables whose definition depends strictly on the domain of interest of the process involved. The model allows to define logical constraints based on process variables that can be used to constrain task assignment (through task preconditions), to assess the outcome of a task (through task effects) and as guards in expressions at decision points (e.g., for cycles or conditional statements). Processes are represented as IndiGolog programs. IndiGolog [2] allows for 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. More details about the SmartPM general framework and the formalization of processes can be found in [4].

SmartPM provides mechanisms for adapting process schemas that require no pre-defined handlers. To this end, we use a specialized version of the concept of adaptation from the field of agent-oriented programming [3]. Specifically, adaptation in SmartPM can be seen as reducing the gap between the *expected reality*, the (idealized) model of reality that is used by the PMS to reason, and the *physical reality*, the real world with the actual values of conditions and outcomes. Exogenous events and tasks' termination may lead to a deviation of the physical reality from the expected reality. An execution monitor is responsible for detecting whether the gap between the expected and physical realities is such that the original process δ_0 cannot progress its execution. In that case, the PMS has to find a recovery process δ_h that repairs δ_0 and removes the gap between the two kinds of reality. Currently, the adaptation algorithm deployed in SmartPM synthesizes a linear process δ_h (i.e., a process consisting of a sequence of tasks) and inserts it at a given point of the original process - specifically, that point of the process where the deviation was first noted. To provide more details, let us assume that the current process is $\delta_0 = (\delta_1; \delta_2)$ in which δ_1 is the part of the process already executed and δ_2 is the part of the process which remains to be executed when a deviation is identified. The adapted process is $\delta_0' = (\delta_1; \delta_h; \delta_2)$. However, whenever a process needs to be adapted, every running task is interrupted, since the "repair" sequence of tasks $\delta_h = [t_1, \ldots, t_n]$ is placed before them. Thus, active branches can only resume their execution after the repair sequence has been executed. This last requirement is fundamental to avoid the risk of introducing data inconsistencies during a repair.

3 The SmartPM Service

The service-oriented approach that characterizes the architecture of Yawl makes the system easily extendable and provides direct support for implementing the Flexibility as a Service approach [7]. In Yawl the engine manages running *cases*,

but is not directly responsible for task executions. Resources, entities and systems able to execute tasks are abstracted as *services* that interact with the YAWL engine via a set of interfaces. Specifically, the interaction between the engine and the Custom Services mainly occurs through Interface B, which provides an API defining endpoints for services to establish a session with the engine, launch process instances, check work items in and out of the engine, and retrieve process data and state information. At design-time each atomic task in a YAWL specification can be associated with a so-called Custom Service [8] (e.g., the Default Worklist Handler/Resource Service, the Worklet Service [1], the Declare Service [9], etc.) that at run-time is responsible for task execution according to its internal logic.

A complete integration between YAWL and SMARTPM has thus been achieved by designing and implementing a YAWL Custom Service, named SMARTPM Service, that enables the interaction between the two environments. The SMARTPM Service enables the decomposition of YAWL tasks into processes to be executed by SMARTPM. At design-time, the process designer is thus able to associate atomic tasks in the YAWL specification with the SMARTPM Service. The service requires as input variable a process to be performed by SMARTPM, defined according to the formalism introduced in the previous section. If the SMARTPM process is already available at design-time, it can be directly associated with the input variable required by the service. However, a process to be delegated to SMARTPM can also be built starting from an available template, which is configured and finalized by exploiting data produced as output by other tasks in the main YAWL process. In this case, the executable SMARTPM process has to be produced as output by a YAWL task that precedes the task associated with the SMARTPM Service. The domain-dependent configuration of a SMARTPM template can be done either manually by a process designer or automatically by a dedicated service. As any other Custom Service, the SMARTPM service implements the service-side of Interface B and is thus able to receive notifications from the YAWL engine when a new work-item is created and delegated to the service for execution. Invoking the specific methods provided by the engine-side of Interface B, the SMARTPM Service is then able to check-out a work-item (i.e., notify the engine that the work-item is going to be executed) and execute the corresponding SMARTPM input process. Once the execution of the subprocess has been completed, the service can check-in the work-item (i.e., notify the engine of execution completion), again via Interface B, along with the corresponding output. Specifically, the SMARTPM Service produces as output, among other possible values, a boolean one that indicates whether the input subprocess was successfully completed or not. Control is then passed back to the YAWL environment, which can continue to carry out the main process.

4 A Demonstration Scenario

As an application scenario, we consider an emergency management process defined for train derailments and inspired by a real process used by the main

Italian Railway Company (that is "Reti Ferroviarie Italiane"). The corresponding YAWL process to be executed is shown in Figure 1. The process starts when the railway traffic control center receives an accident notification from the train driver and collects some information about the derailment, including the train ID code, the GPS location and the number of wagons and passengers. Then it could be required to cut off the power in the area and to interrupt the railway traffic near the derailment scene. In parallel, after having collected additional information about the train (e.g., security equipment) and about emergency services available in the area, an emergency response team can be sent to the derailment scene. The information collected so far are then used for defining and configuring an incident response plan, which is defined by the set of activities to be executed directly on the field by first responders. Such activities can instruct first responders to act on location for evacuating people from train wagons, to take pictures and to assess the gravity of the accident. The high dynamism of the operating environment requires that such activities are executed by the SMARTPM subsystem. We assume that operators are equipped with mobile devices and coordinate themselves through the SMARTPM process execution environment. Here, context-awareness and managing frequent exogenous events (e.g., bad connections between devices and operators) is crucial. Let us suppose, for example, that a fire breaks out in one of the wagons. From the SMARTPM point of view, it means that an exogenous event changes asynchronously the value of the current physical reality, by turning the variable that represents the status of the fired wagon from "safe" to "fired". Since in the expected reality it is forecast that every wagon is safe after passengers evacuation, the SMARTPM execution monitor senses that the discrepancy between the two realities is relevant and starts reasoning on the basis of the available tasks and of the capabilities provided by first responders. The target is to find a recovery process that reduce the above gap. In this case, SMARTPM automatically builds a recovery process that concerns reaching the location of the fired wagon and starting extinguishing fire. Such a couple of tasks is assigned to that first responder that provides all the capabilities needed to execute them. When every wagon is evacuated and the situation is turned to the normality, the process control comes back again to YAWL, that can execute the last task about the restoring of the railway infrastructure. A screen-cast of the proposed demonstration is available at `http://www.dis.uniroma1.it/~marrella/public/DemoBPM2011.zip`.

5 Conclusions

In this demonstration paper we outlined how an imperative modeling approach used to define stable and well-understood processes can be complemented by a modeling approach that enables automatic process adaptation. To this end, we designed and implemented a Custom Service that allows the YAWL execution environment to delegate the execution of subprocesses and activities to the SMARTPM execution environment, which is able to automatically adapt a process to deal with emerging changes and exceptions. We demonstrated the

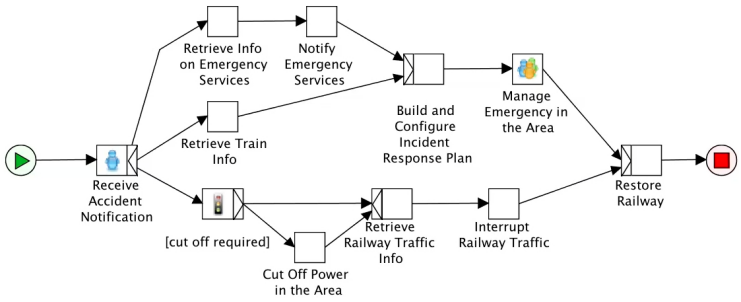

Fig. 1. The YAWL process defined for a train derailment scenario.

feasibility and validity of the approach by showing the design and execution of an emergency management process defined for train derailments.

Acknowledgments. The work of Andrea Marrella, Massimo Mecella and Alessandro Russo has been partly supported by the projects TESTMED, FARI 2010, SmartVortex and Greener Buildings.

References

1. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In: Proc. of the 14th Int. Conf. on Cooperative Information Systems (CoopIS), 291–308 (2006)
2. De Giacomo, G., Lespérance, Y., Levesque, H.J., Sardina, S.: IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents. In: Multi-Agent Programming: Languages, Platforms and Applications, 31–72. Springer (2009)
3. De Giacomo, G., Reiter, R., Soutchanski, M.: Execution Monitoring of High-Level Robot Programs. In: Proc. of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR), 453-465 (1998)
4. de Leoni, M., Marrella, A., Mecella, M., Sardina, S.: SmartPM – Featuring Automatic Adaptation to Unplanned Exceptions, SAPIENZA Università di Roma, DIS Technical Report 04-2011. `http://ojs.uniroma1.it/index.php/DIS_TechnicalReports/article/download/9221/9141` (2011)
5. de Leoni, M., Mecella, M., De Giacomo, G.: Highly Dynamic Adaptation in Process Management Systems Through Execution Monitoring. In: Proc. of the 5th International Conference of Business Process Management (BPM), 182–197 (2007)
6. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. The MIT Press (2001)
7. van der Aalst, W.M.P., Adams, M., ter Hofstede, A.H.M., Pesic, M., Schonenberg, H.: Flexibility as a Service. In: Proc. of Database Systems for Advanced Applications (DASFAA) International Workshops (2009)
8. van der Aalst, W.M.P., Aldred, L., Dumas, M., ter Hofstede, A.H.M.: Design and Implementation of the YAWL System. In: Proc. of the 16th International Conference on Advanced Information Systems Engineering (CAiSE), 142–159 (2004)
9. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between Flexibility and Support. In: Computer Science - R&D, 23, 99–113 (2009)