Towards a Goal-oriented Framework for the Automatic Synthesis of Underspecified Activities in Dynamic Processes

Andrea Marrella
Sapienza, University of Rome, Italy
Email: marrella@dis.uniroma1.it

Yves Lespérance
York University, Toronto, Canada
Email: lesperan@cse.yorku.ca

***Abstract*—It is difficult to produce a detailed model of a dynamic process ahead of time. Such processes may include some underspecified activities whose exact definition is not yet known at design-time, and may not be known until the time that an instance of the process has started execution, due to their context-dependent nature. In this paper, we propose a goal-oriented framework to model and specify dynamic processes that allows us to dynamically select and/or synthesize automatically at run-time the content of underspecified activities.**

I. Introduction

Business Process Management [1] (BPM) is a very active research area, because it is highly relevant from a practical point of view while at the same it offers many challenges for researchers. Once identified, a *business process* is formalized into a *process model* which captures every possible *process instance* to be executed at run-time through a *Process Management System* (PMS). Traditionally, PMSs have focused on the support of predictable and repetitive business processes, which can be fully pre-specified in terms of formal process models. This kind of highly-structured workflow includes mainly production and administrative processes (such as financial services, manufacturing, order handling, etc.) [2].

In recent years, the maturity of process management methodologies has led to the application of BPM approaches in new challenging knowledge-intensive scenarios [3], such as healthcare [4] and domotics [5]. The need to deal with *knowledge-intensive* and *dynamic processes* and provide support for flexible process management has emerged as a leading research topic in the BPM domain [6]. In a dynamic process, the sequence of tasks depends heavily on the specifics of the context (e.g., which resources are available and what particular options exist at the time), and it is often unpredictable how the process will unfold. During the run-time execution of a dynamic process, more execution paths could be incorporated in the range of the existing process model, when it becomes clear what needs to be done at a specific point in the process. Such processes do not have the same level of repeatability as classical business processes, and the execution changes on a case-by-case basis, generating instances that are different almost every time, depending on the context.

An interesting example comes from the *emergency management domain*. During the management of complex emergency scenarios, teams of first responders act in disaster locations with the objective of assisting potential victims, assessing and stabilizing the situation, etc. The set of activities and procedures that collectively define an emergency response plan are quite complex and involve teams with several members. In many cases, there exist established and standardized emergency plans and procedures, but they should be dynamically adjusted at run-time. In fact, the designer often lacks the needed knowledge to anticipate and incorporate all potential alternatives into the process model at design-time; as well this knowledge can become obsolete as process instances are executed and the context evolves.

Since a dynamic process is generally well-structured but does not have a detailed definition for every activity, building on our previous work [7], in this paper we propose a goal-oriented framework to model and specify dynamic processes that allows us to dynamically select and/or to synthesize automatically at run-time the content of those activities whose exact definition is not known at design-time, and may not be known until the time that an instance of the process is being executed. Following [8], we call these activities as *underspecified*. We allow the process designer to associate underspecified activities with an *abstract objective* at design-time, and to refine it at run-time into a *concrete goal condition* defined on the contextual data linked to the process, when the underspecified activity is going to be executed. Specifically, we are able to deal automatically at run-time with two types of underspecified activity enactment:

- *Run-time binding*: the content of an underspecified activity in a given context is selected from a set of available *templates* stored in a *process template library*; a template specifies a best-practice procedure for achieving a goal in contexts satisfying an associated condition; we select a template whose goal matches the one attached to the underspecified activity and whose condition is satisfied in the current context;
- *Run-time synthesis*: if the library does not contain any template matching the actual context/goal condition associated to the underspecified activity, a new template is generated using an external planner.

II. Running Example

As an application scenario, we consider an emergency management process defined for train derailments, which is inspired by a real process used by Trenitalia. The corresponding

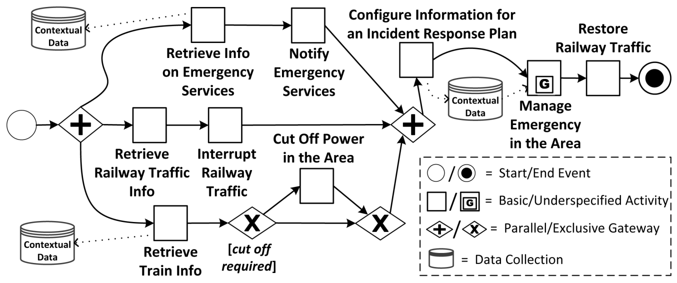
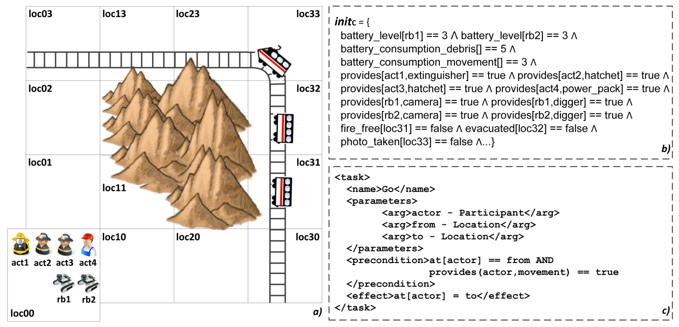

Contextual Data
Retrieve Info on Emergency Services
Notify Emergency Services
Configure Information for an Incident Response Plan
Restore Railway Traffic
Contextual Data
Manage Emergency in the Area
Retrieve Railway Traffic Info
Interrupt Railway Traffic
Cut Off Power in the Area
Contextual Data
Retrieve Train Info
[cut off required]
= Start/End Event
= Basic/Underspecified Activity
= Parallel/Exclusive Gateway
= Data Collection

Fig. 1. A process for managing derailments.



```
initc = {
  battery_level[rb1] == 3 ∧ battery_level[rb2] == 3 ∧
  battery_consumption_debris[] == 5 ∧
  battery_consumption_movement[] == 3 ∧
  provides[act1,extinguisher] == true ∧ provides[act2,hatchet] == true ∧
  provides[act3,hatchet] == true ∧ provides[act4,power_pack] == true ∧
  provides[rb1,camera] == true ∧ provides[rb1,digger] == true ∧
  provides[rb2,camera] == true ∧ provides[rb2,digger] == true ∧
  fire_free[loc31] == false ∧ evacuated[loc32] == false ∧
  photo_taken[loc33] == false ∧...}
```

```
<task>
  <name>Go</name>
  <parameters>
        <arg>actor - Participant</arg>
        <arg>from - Location</arg>
        <arg>to - Location</arg>
  </parameters>
  <precondition>at[actor] == from AND
                provides(actor,movement) == true
  </precondition>
  <effect>at[actor] = to</effect>
</task>
```

Fig. 2. Area and context of the intervention.

BPMN[1] [9] process is shown in Fig. 1. The process starts when the railway traffic control center receives an accident notification from the train driver and begins by collecting information about the train (e.g., the area where the train derailed, the number of affected coaches, etc.) and the emergency services available in the area. Then, it may need to cut off power in the area and interrupt railway traffic around the derailment scene. The cylinder shapes in Fig. 1 represent the collection of contextual data about the derailment that is manipulated and updated during the process enactment. For example, after the execution of the activity "Configure Information for an Incident Response Plan", we can assume to have contextual scenario information defined as in Fig. 2(a), where we depict a possible map of the area, represented as a 4x4 grid of locations. We suppose that for this specific emergency the train is composed only by a locomotive (located in *loc33*) and two passenger coaches (located in *loc32* and *loc31* respectively). The team is composed of four first responders (in the remainder, we refer to them as *actors*) and two robots, initially located in *loc00*. Actors are equipped with mobile devices (for picking up and executing tasks) and provide specific capabilities. For example, actor *act1* is able to extinguish fires, while *act2* and *act3* can evacuate people from train coaches. The two robots, instead, may take pictures and remove debris from specific locations. Each robot has a battery and each action consumes a given amount of battery charge. When a robot's battery is discharged, *act4* can charge it. Fig. 2(b) summarizes the above.

At this point in the process, the information collected so far (mostly unknown at design time) may be used for defining and configuring at run-time an incident response plan, which includes the set of tasks to be executed directly on the field by first responders. Such tasks, abstracted into the underspecified activity "Manage Emergency in the Area", need to be contextually and dynamically selected (or generated) at run-time. In fact, it is infeasible to think that the process designer can pre-define at design-time all the possible processes required for coordinating first actors in the field, since the concrete goal condition associated to "Manage Emergency in the Area" depends on the kind of emergency that has to be faced (e.g., some coaches could be on fire or a landslide may have occurred), and may be different every time the process runs. Moreover the recovery procedure depends strictly on the current contextual information (the positions of operators, the battery level of robots, etc.). Finally, it is also difficult to define the ad-hoc emergency plan at run-time in a completely manual way, because the correctness of the process execution is highly constrained by the values (or combination of values) of each contextual data term.

III. On Modeling Dynamic Processes

Our approach to the definition of a dynamic process requires a fundamental shift in how one thinks about modeling business processes. A dynamic process depicts a *high-level procedure* to be enacted in a specific situation. It may include the execution of some *basic activities* and of some *underspecified activities*. The definition of the control flow of the process must be complemented with an explicit representation of the *contextual data* reflecting the state of the dynamic environment in which the process will be executed. Such data will act as a driver for automatically selecting/synthesizing the content of those activities that are underspecified at design-time. Basic activities may refer to human or automated tasks, and may be linked with an invokable application service. For example, in the process of Fig. 1, the activity "Retrieve Info on Emergency Services" is used for verifying the presence of an emergency team close to the area affected by the derailment. If so, contextual data associated to the process can be updated and, through the execution of the activity "Notify Emergency Services", the leader of the selected team is notified by phone to deal with the emergency. In general, we can assume that the contextual information summarized in Fig. 2 is collected step by step during the enactment of the process in Fig. 1.

To be more precise, we specify the context linked to a specific dynamic process in the form of a *Domain Theory* D (cf. [7]), that captures the contextual data and supporting information, such as the people/agents that may be involved in performing the process. Data are represented through some ground atomic terms $v_1[y_1], ..., v_m[y_m] \in \mathsf{V}$ that range over a set of tuples (i.e., unordered sets of zero or more attributes) $y_1, ..., y_m$ of *data objects*, belonging to some *data types*. In short, a data object depicts an entity of interest; for example, in our scenario we define data objects for representing participants (e.g., data type *Participant*=$\{$*act1, act2, act3, act4, rb1, rb2*$\}$) and locations in the area (e.g., data type *Location* = $\{$*loc00, ..., loc33*$\}$). 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 data types previously defined) represent the finite domains over which the term is interpreted. In our example, we may need boolean terms for expressing the presence of fire in a lo-

[1]BPMN (Business Process Modeling Notation) is a diagramming language designed to specify a process in a standardized way.

cation (e.g., $fire\_free[loc : Location] = (bool : Boolean)$), integer terms for representing the battery charge level of each robot (e.g., $battery\_level[prt : Participant] \in \mathbb{Z}$) and functional terms for recording the position of each actor in the area (e.g., $at[prt : Participant] = (loc : Location)$).

During process enactment, atomic terms defined in D may assume specific values, depending on the basic activities executed so far and the contextual information gathered. To this end, we introduce the concept of *State* S, that reflects an instantiation of the domain theory D at a specific point of the process. Basically, the state S records the evolution of the contextual data during the process enactment, and can be specified as a conjunction of atomic terms. Therefore, terms may be thought of as properties of the world whose values may vary across states. We do not assume complete information about S; this means we allow a process designer to instantiate only the terms necessary for representing what is known at a specific point of the process, i.e., $\mathsf{S} = \{v_1[y_1] == val_1 \wedge ... \wedge v_j[y_j] == val_j\}$, where val_j (with $j \in 1..m$) represents the j-th value assigned to the j-th atomic term. Fig. 2(b) shows the portion of S after the execution of the activity "Configure Information for an Incident Response Plan".

While basic activities may be enacted independently from the values of contextual data, the final specification of *underspecified activities* is not known at design-time. The process in Fig. 1 contains a single underspecified activity "Manage Emergency in the Area", which involves coordinating the first responders' actions on the field. Clearly, it is difficult to foresee at design-time the final content of this activity, since much of the contextual information associated to the derailment is discovered during the process execution. Each underspecified activity is labeled at design-time with an *abstract objective*, whose purpose is to drive the process designer in the definition of more *concrete goal conditions*, defined over the contextual data at run-time. An abstract objective may be specified in a descriptive way (e.g., in natural language, and it may also corresponds to the name of the activity itself, cf. Fig. 1), or by writing a first-order formula defined over the domain theory. For example, to express at design-time that at the end of the execution of the activity "Manage Emergency in the Area" every person in the train must have been evacuated, the process designer may write the following formula: $\forall\ (loc)\ Location(loc) \rightarrow evacuated[loc] == true$. Clearly, the choice to label an underspecified activity in a descriptive way or with a first-order logic formula depends on the amount of information available at design-time. When the underspecified activity is going to be executed, a concrete goal condition has to be provided in place of the abstract objective. Since the concrete goal may vary depending on the evolution of the state S, we can characterize completely an underspecified activity with a *Case* C, that reflects an instantiation of the domain theory D with a starting state $init_{\mathsf{C}}$ and a goal condition $goal_{\mathsf{C}}$. Specifically, we denote by $init_{\mathsf{C}}$ the starting state needed for executing the underspecified activity, and this state will be equal to the state S just before we execute the underspecified activity (cf. Fig. 2(b)). Furthermore, we define a concrete goal condition $goal_{\mathsf{C}}$ as a conjunction of terms we want to make true through the execution of the underspecified activity. For example, in the scenario shown in Section II, the goal can be represented as: $goal_{\mathsf{C}} = \{fire\_free[loc31] == true \wedge evacuated[loc32] == true \wedge\ photo\_taken[loc33] == true\}$. It reflects the objective of evacuating people from the coach located in *loc32*, extinguishing a fire in the coach in *loc31* and finally taking pictures for evaluating possible damages to the locomotive in *loc33*. Note that the abstract objectives specified at design-time may be not sufficient for capturing the concrete goal condition that will be associated to an underspecified activity at run-time. In fact, the evolution of the contextual data during the process enactment may require that the process designer specifies the concrete goal condition in a way that reflects the current knowledge about the contextual scenario.

The concrete realization of an underspecified activity is finally provided in the form of a *process template*. A process template PT captures a partially ordered set of *planner tasks* whose successful execution (i.e., without exceptions) leads from $init_{\mathsf{C}}$ to $goal_{\mathsf{C}}$. Formally, we define a template as a directed graph consisting of planner tasks, parallel gateways, events and transitions between them. Planner tasks $t_i \in \mathsf{T}$ are collected in a specific repository, and each planner task can be considered as a single step that consumes input data and produces output data. In our case study, the repository contains a set of emergency management (annotated) planner tasks, that range from the simple activity of taking pictures to the more complex activity of extinguishing a fire. Each planner task is annotated with *preconditions* and *effects*. Preconditions can be used to constrain the task assignment and must be satisfied before the task is applied, while effects establish the outcome of a task after its execution. For example, the task *Go* described in Fig. 2(c) involves two parameters $from$ and *to* of type $Location$, and a parameter *actor* of type $Participant$, that is the first responder that will execute the task. An instance of *Go* can be executed only if *actor* is currently at the starting location $from$ and has the required capabilities for executing the task. As a consequence of task execution, the actor moves from the starting to the destination location, and this is reflected by assigning to the term $at[actor]$ the value *to* in the effect.

IV. The General Framework

The starting point for the creation of a new dynamic process is a *BPMN graphical editor* used to define the control flow of the process and to label underspecified activities with abstract objectives. We also provide a *modeling tool* for specifying the domain theory D, for building the repository of planner tasks T and for updating the state S associated to the specific process under execution. If during the process execution an underspecified activity is reached, a new case C for the activity is created; the starting state $init_{\mathsf{C}}$ is derived from the current instantiation of the state S, while the concrete goal condition $goal_{\mathsf{C}}$ is inferred from the abstract objective associated to the activity (if the objective is specified as a first-order formula) or manually refined by the process designer.

Our approach (cf. Fig. 3) allows us to generate the content of an underspecified activity with two different techniques, named respectively *run-time binding* and *run-time synthesis*. To this end, we rely on a *planning-based tool* that includes a *library of process templates* and an *external planner* that are used respectively for selecting and for synthesizing complex concurrent process templates. First of all, a specific software module is in charge of converting the domain theory D and the set of planner tasks T into a Planning Domain PD, and

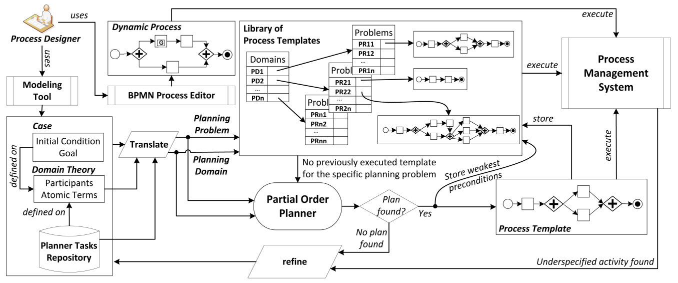


Fig. 3. The Overall Framework.

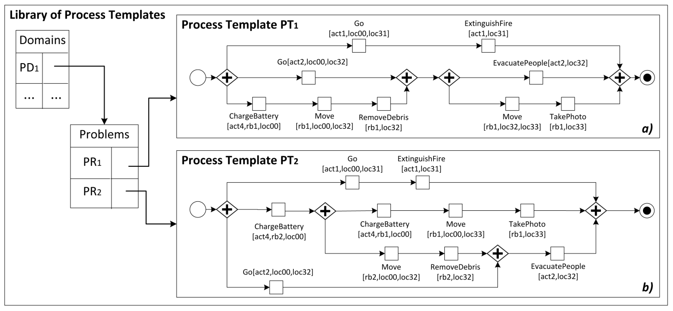


Fig. 4. Templates dealing with the scenario in Fig. 2.

the starting state $init_{\mathsf{C}}$ and the goal condition $goal_{\mathsf{C}}$ into a Planning Problem PR. The planning domain PD and the planning problem PR are both specified in PDDL version 2.1 [10]. PDDL (known as the Planning Domain Definition Language) is the standard representation language for classical planners. A planner that works on such inputs generates a partially ordered set of planner tasks (the *plan*) that leads from the initial state to a state meeting the goal condition.

For dealing with run-time binding, at the heart of our framework lies a library of process templates built for specific planning domains and problems. These are reusable processes that achieve specified goals of interest in any context that satisfies the template's required preconditions. Specifically, for each template we keep their *weakest preconditions*. This means that many library templates may be compatible with the values of PD and PR. In fact, given a planning domain PD, a template can be executed in several starting states, since it (usually) requires a fragment of the contextual knowledge of the starting state to successfully achieve its objectives. For example, as shown in Fig. 4, two different process templates match with the current values of PD and PR. This means that the process designer can choose which template is the best for her/his purposes: one with less concurrency in the tasks enactment but with fewer participants (i.e., PT_1), or one with more concurrency but requiring more resources for being executed (i.e., PT_2). In fact, PT_2 requires the presence of one more robot (i.e., robot rb2) and more contextual information for being executed, but it provides an higher degree of concurrency in the execution of its tasks. After an appropriate template is selected, it can be executed through an external PMS.

However, if no template exists for the current values of PD and PR, we can invoke an external partial-order planner (a.k.a. POP [11]) on these same inputs. We exploit the idea behind POP of representing flexible plans that enables deferring decisions. Instead of committing prematurely to a complete, totally ordered sequence of actions, plans are represented as a partially ordered set, and only the required ordering decisions are recorded. The planner will try to generate a plan fulfilling the goal condition $goal_{\mathsf{C}}$. When the POP planner is able to find a partially ordered plan P consistent with the actual contextual information, it is translated into a template PT that preserves the ordering constraints imposed by the plan (i.e., *run-time synthesis*). For example, the templates PT_1 and PT_2 can be generated by our planning-based tool. A process template generated in this way guarantees some interesting properties, such as the *executability* of the template with respect to the information available in the case C, and the property of *sound concurrency*, meaning that concurrent tasks of a template are proven to be independent from each other. Moreover, as anticipated above, we are able to identify the weakest preconditions of process templates, and all the states satisfying such preconditions are good candidates for executing them. To show the feasibility of our approach, we ran some experiments and measured the time required for synthesizing a plan for some variants of our running example. Interested readers may refer to [7] for the detailed experiments performed and algorithms used for generating the library and synthesizing the templates. If the tool fails to generate a process template or the generated processes are of insufficient quality (e.g., they are too time consuming, unreliable, or lack concurrency), the designer can try to *refine* the domain theory D or the case C and add information so that it becomes possible to generate a satisfactory plan. There are many ways to strengthen a problem description, such as adding to the starting state $init_{\mathsf{C}}$ some terms initially ignored (e.g., to specify the position of every participant), or adding new objects in D or new activities in T (e.g., if a task for extinguish fire is missing), or changing the goal condition. Once a satisfactory template has been obtained, it is used to execute the underspecified activity. The template may also be added to the library so it can be re-used whenever a case that matches the template's preconditions arises.

V. Related Work

The concept of "flexibility by underspecification" has been introduced in [8], as the "ability to execute an incomplete process model at run-time". An incomplete process is one which does not contain sufficient information for execution as it may contain several "placeholders", i.e., underspecified activities whose content is to be discovered during the execution of the process. The topic of process incompleteness has been handled from different perspectives in the research literature. The YAWL system [12], a well-known PMS coming from academia, allows one to define a number of different placeholders within a well-structured business process. YAWL is able to provide both the run-time binding and the run-time specification of the placeholders' content, but such content has to be pre-defined in advance as a complete stand-alone business process and stored in a repository, or can be defined manually directly at run-time. The Case Management and Modeling Notation (CMMN 1.0) [13], recently released by the Object Management Group (OMG), enables run-time binding/modelling by introducing discretionary elements which can be manually selected, composed and added to a case at run-time by the case worker. Clearly, the above approaches work for processes where procedures are well known, repeatable and the content of placeholders can be specified in advance with some level of detail. Some adaptive PMSs like ADEPT2 [14] and SmartPM [15] recognize that dynamic processes are incomplete "by nature", since they are executed in changing environments. For this reason, they support the adaption of the process instance by dynamically adding/implementing process fragments that were not specified in the original process model, on a case-by-case basis. However, process adaptation in [14] and [15] is mainly seen as a solution to exception handling; i.e., when some exception arises during the process enactment by changing the context in an undesirable way, the running process instance is dynamically adapted into a new instance that may re-set the "wrong" context on the right track. Furthermore, process mining [16], in some sense, deals with process incompleteness. It allows the extraction of process descriptions, stemming from a set of recorded executions of real processes. Such executions are stored in so called *event logs*; each recorded *event* reports the execution of a task (i.e., a well-defined step in the process) in a specific instance of the process. Starting from several event logs, mining techniques aim to infer a structured model of the executed workflow. However, a dynamic process may be different every time it is executed, making it impossible to build a process model that represents every possible instance of the dynamic process.

An interesting goal-oriented approach for dealing with incomplete processes is presented in [17]. This work relies on Agent Technology, especially on BDI-Agents features [18] to obtain agile business process behavior. In the BDI (belief-desire-intention) agent architecture, an agent is described by its *beliefs* (i.e., the information an agent has about itself and its environment), its *desires* (i.e. motivations of the agent that drive its course of action) and finally its *intentions* (i.e. the short-term plans that the agent wants to execute), derived from its desires and external events, to which the agent reacts. To achieve its goals/intentions an agent has certain plans on how the goals can be achieved. Different plans are designed for different situations, which are described by the plan's context condition. A plan consists of certain actions/steps that have to be executed to achieve the corresponding goal. The agent has to select which goals it wants to pursue next and which plan can be used to achieve the goal. In [17], based on the BDI agent architecture, the idea is to have a final model of the business process consisting of one or more goal hierarchies, a list of context variables and a set of plans with conditions linked to subgoals. The process can be executed by considering the current goal and the context to determine the next step of the process, and the agent can be seen as an assistant of the user who is responsible for driving a task through the process, whose real structure is discovered only during the process enactment. Unfortunately, in [17] plans for dealing with subgoals have to be defined in advance. So that approach is not particularly useful for dynamic processes, where the content of underspecified activities is unknown at design-time.

VI. Conclusion

To face the challenges posed by today's highly dynamic business contexts, traditional BPM approaches should be complemented with new techniques that support flexible execution of underspecified business processes. Towards this objective, in this paper we have presented a goal-oriented framework for modeling and executing incompletely-specified dynamic processes, which relies on a library of process templates and on planning techniques to obtain at run-time process templates that can be used to realize underspecified activities. This allows the dynamic process to be elaborated automatically at run-time and then executed, providing the required flexibility. In general, elements of the process may have to be specified interactively with the system or manually, but we have focused here on automatic generation. Our framework is an extension of the one described in [7]. The novel aspect of this paper is that we address dynamic processes with underspecified activities, and the run-time elaboration of such activities. A future direction for this work is to generate and support hierarchical process templates, with high-level templates achieving more general goals that can invoke simpler templates to achieve some of their subgoals. It seems that agent-technology can provide promising approaches and methods to address this challenge.

References

[1] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[2] Leymann, Frank and Roller, Dieter, *Production workflow: concepts and techniques*. Prentice Hall, 2000.

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

[4] R. Lenz and M. Reichert, "IT support for healthcare processes - premises, challenges, perspectives," *Data Knowl. Eng.*, vol. 61, 2007.

[5] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen, "The Gator Tech Smart House: A Programmable Pervasive Space," *Computer*, vol. 38, no. 3, 2005.

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

[7] A. Marrella and Y. Lespérance, "Synthesizing a Library of Process Templates through Partial-Order Planning Algorithms," in *14th International Conference on Business Process Modeling, Development, and Support, BPMDS*, 2013.

[8] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. M. P. van der Aalst, "Process Flexibility: A Survey of Contemporary Approaches," in *CIAO! / EOMAS*, 2008.

[9] OMG, "Business Process Modeling Notation - Final Specification Ver.2.0." http://www.omg.org/spec/BPMN/2.0/PDF/, 2011.

[10] M. Fox and D. Long, "PDDL2.1: an Extension to PDDL for Expressing Temporal Planning Domains," *J. Artif. Int. Res.*, vol. 20, no. 1, 2003.

[11] D. S. Weld, "An Introduction to Least Commitment Planning," *AI Magazine*, vol. 15, no. 4, 1994.

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

[13] Object Management Group (OMG), "Case Management Model and Notation (CMMN)," http://www.omg.org/spec/CMMN/1.0/Beta1/, 2013.

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

[15] A. Marrella and M. Mecella, "Continuous Planning for Solving Business Process Adaptivity," in *12th International Conference on Business Process Modeling, Development, and Support, BPMDS*, 2011.

[16] W. M. Van der Aalst, *Process mining: discovery, conformance and enhancement of business processes*. Springer, 2011.

[17] B. Burmeister, M. Arnold, F. Copaciu, and G. Rimassa, "BDI-agents for agile goal-oriented business processes," in *AAMAS*, 2008.

[18] A. S. Rao, M. P. Georgeff *et al.*, "BDI Agents: From Theory to Practice," in *ICMAS*, vol. 95, 1995.