Towards Self-Adaptation and Evolution in Business Process

L. Sabatucci[1], C. Lodato[1], S. Lopes[1], and M. Cossentino[1]

ICAR-CNR, Consiglio Nazionale delle Ricerche, Palermo, Italy
{sabatucci,c.lodato,s.lopes,cossentino}@pa.icar.cnr.it

Abstract. Business process run-time evolution and adaptivity are two urgent objectives in the research agenda of dynamic workflow execution.
Traditional languages as BPMN or BPEL take an imperative style for defining the exact sequences of activities to execute. The imperative approach identifies a narrow space of solution that is generally optimized by the experience. However it does not provide enough freedom of action to bypass obstacles when something exceptional happens.
In the wake of declarative specification languages we propose a framework, based on the standard BPMN, in which both business goals and the operative context are monitored for changes during the execution time.
To enable a flexible adaptivity of the process to a changing environment we adopt the solution of relax some constraints of the rigid BPMN specification thus to give autonomous software agents the opportunity of exploring a wider space of solution, even when this space evolves unexpectedly or contains uncertainty.
The result is a multi-agent system that exploits its features (mainly autonomy and proactivity) in order to monitor the execution state of the process and to discover a distributed solution to unpredictable situations or to specifications' evolution.

1 Introduction

The business domain is a highly variable application context [1]. Business rules change frequently due to a couple of factors. The first aspect is connected to the knowledge of the domain that typically improves after the workflow deployment and execution. Analysts and stakeholders learn from scenarios execution [2]. This generally leads to corrections or extensions to increase the correctness or the coverage of exceptional cases that have not been considered yet. The second aspect is connected to the evolution of the society (with its laws and regulations) and/or of the company business strategies [1]. These changes imply a revision of the workflow model, and often it is necessary to include new goals or constraints.

Evolution and adaptivity are often strictly correlated. This is particularly true in the domain of dynamic workflows [3]. BPMN [4] provides basic mechanism to specify how the system will react to expected exceptions. Adaptation is more concerned to define how to react to unexpected exceptions, which are events that cannot be handled by following the workflow model. So far, BPMN also does not support a dynamic context. Every external change must be implemented into the workflow as a set of modifications. Thus, the workflow must be re-designed every time it is necessary implementing new

requirements. In addition, redesign often means to check the inter-dependencies and to verify the validity of the result.

With respect to adaptivity and evolution, the main limit of BPMN specifications is the use of an imperative style for defining the process [1]. A business process is generally described as the exact sequence of tasks to execute as a reaction to specified events. Moreover, exceptions are defined as special events that may occur during the main flow of execution. This modeling style limits the solution space to a unique solution or to a limited set of solutions and it reduces the capability of the system to recover from exceptions that were not considered in the business analysis phase. This choice is not adequate for designing self-adaptive systems because it does not aid systems to modify their behavior according to changes of the execution environment [5]. Indeed, a self-adaptive system should be able to autonomously search a solution in a wide solution space, in case the mainstream solution is not feasible [1, 2].

Whereas the imperative specification style limits the solution space, on the other hand, a declarative specification style would help in opening the solution space to many alternatives. In particular many authors indicate the goal-oriented approach may be used to define the expected results of a system, without forcing it to follow a pre-established plan in order to address these results. We adopted an ontology based goal oriented specification language as the grounding asset for our solution [6]. This is used for defining the business process in terms of goals to address rather than tasks to execute. The goal oriented description of the process breaks some rigid constraints that are typical of the imperative style of BPMN. With respect to the state of the art [7, 8] we also decided to completely decouple the description of what should be addressed by the plans used to accomplish these results. In this way there is not a pre-established set of tasks to execute for each specified goal. The responsibility of deciding which task to select is delayed at run-time, so goals may be dynamically injected into the system.

In addition, we designed a workflow engine as a multi-agent system in which each autonomous software agent is aware of its own capabilities and it is able to decide when and how to contribute to goal achievement. All together, the agents, self-organize in groups that proactively discover a distributed solution as the orchestration of many capabilities. In any case, we do not want to loose the flexibility and the expressiveness of BPMN as well as its important feature of being targeted to humans. Indeed we consider important, in a realistic application context, maintaining the BPMN as an interface for business analysts to model their processes. For this purpose we defined an algorithm to automatically extract business goals, starting just from the BPMN definition of the process.

The aim of this paper is to provide an overview of the whole framework. It is organized as follows: Section 2 describes the architecture of the framework and provides a scenario from our case study. Section 3 briefly presents the language for specifying a process and how it is extracted from a standard BPMN process. Section 4 describes the multi-agent system main characteristics: self-awareness, self-organization and the ability to discover distributed solutions. Some final notes are provided in Section 5.

2 Overview of the adaptive workflow system

Self-adaptive software is a response to the demand for management complexity reduction, management automation, robustness, and achieving all of the desired quality requirements within a reasonable cost and time range during operation [2, 9]. Many challenges arise when designing a self-adaptive system: monitoring itself and its context, detecting significant changes, deciding how to react, and acting to execute such decisions [1]. We propose a framework for self-adaptive workflow execution.

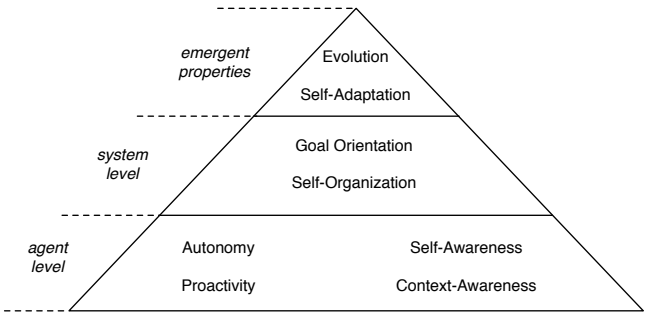

Fig. 1. Hierarchy of properties of the proposed framework (revised from [10])

In traditional workflow design, BPMN (or BPEL) diagrams specify systems requirements. Moreover, executable business processes also include details on the behavior of the system. Indeed specifications often include instructions on how to invoke the associated web-services or local applications.

The framework we present in this paper grounds on one leading idea: decoupling *what* should be addressed from *how* this result can be achieved; this allows to make the system and the goal set evolve independently. Thus, being dynamic, system requirements may be considered as part of the execution context.

We adopted a declarative style for specifying system requirements because it grants high freedom in exploring the solution space. More precisely, we focused on goals because they provide the proper level of abstraction for our purposes. With respect to many goal-oriented approaches in literature we preferred to completely decouple goals and tasks. In our specifications, what should be addressed has no explicit links to how this result can be achieved.

We define GoalSPEC [6] (Section 3) a language to specify system goals. A goal in our language only specifies what is the expected result, but it lacks of any information about which operation is good to reach the result. It is up to the system to elaborate this missing link at run-time, when the execution context is better known. Discovering a solution depends on three dynamic factors that influence one the others: i) which system goals are injected into the system at a given time (the expected result to address), ii) which capacities has the system to solve the problem, (variability in how to address goals) and iii) which resources are available for the system (dynamic execution context).

The software system we present in this paper is an autonomic multi-agent system able to adjust its behavior according to changes of goals, capabilities and resources, during its execution. Figure 1 summarizes the main properties of the system. Evolution and self-adaption grounds on two underlying levels: i) the system level that uses goals to describe the expected behavior and self-organization for addressing them; ii) the agent level that exploits autonomy, proactivity, self-awareness and context-awareness as the pillars for the whole system.

A possible execution scenario is the following. When the goal set is ready, it is injected into the running multi-agent system for the workflow enactment. This action triggers system agents to self-organize themselves in order to modify the whole behavior. Each agent is aware of its own capabilities and of the execution context (Section 4.1). Thus when the agent reads the goal set, it can evaluate whether its capabilities are adequate to address the specified goals or not.

Generally an agent is not able to execute the whole workflow alone. Anyway agents may collaborate to discover a distributed plan. In our approach agents are all peers in the system, thus each of them may locally explore possible sub-solutions to the workflow execution. A distributed algorithm discovers, in parallel, many alternative solutions, whereas only the most promising is finally selected. When this happens all the involved agents commit to address the expected result (Section 4.2).

Concluding, one of the peculiarities of this framework is to maintain the BPMN as the main interface to model workflow in order to reduce any additional burden for business analysts. We developed *BPMN2Goal*, a component that is responsible to take a BPMN 2.0 XML file in input and to generate as output a set of *GoalSPEC* goals. Section 3.1 briefly introduces the algorithm for this transformation.

3 From business process to goals

GoalSPEC is a language suitably created for supporting evolution and self-adaptation. The language has been presented in details in [6]. Here we summarize its salient characteristics on purpose.

Basically the GoalSPEC productions allow to specify system goals, under the hypothesis the domain is described as a set of states. In our vision, by addressing a goal, the actor operates a *state transition* from a initial state to an final state. In GoalSPEC every goal describes three elements: (i) a *triggering condition* that describes the initial state of the domain, (ii) a desired *final state* of the domain and (iii) a list of *actors* that are involved in the transition.

Process goals and Activity goals. GoalSPEC considers two categories of goals:

- a *process goal* is a goal that derives from a Process or a Subprocess in a workflow and describes the triggering condition and the final state of a group of related activities.
- an *activity goal* is related to a specific outcome in the workflow instance; it derives from a Task in a workflow, and it can not be further decomposed into sub goals. Addressing an activity goal produces an advancement for the achievement of the whole workflow.

The domain as a set of states. Our assumption is that of considering the domain as a finite set of states. This assumption holds for a range of application domains in which relevant features are (at least) qualitatively measurable.

In order to describe any state of the domain, GoalSPEC adopts an ontological description that exploits logic predicates to specify properties and relationships between elements of the domain. Logic predicates are explained below by few examples taken from the domain of an e-commerce website:

- *atoms* are ontology objects; for instance *productA* is a concrete item for sale in the website;
- *variables* are ontology categories; for instance *Product* is an abstract object that may range among many concrete objects (for instance productA, productB, etc),
- *predicates* are used to describe properties; for instance *product_for_sale(productB)* specifies productB is a product in the catalogue,
- *structures* can specify more complex relationships between elements or categories; for instance *contains(my_shopcart,Product)* means that object my_shopcart contains any kind of Product.

In goalSPEC both triggering condition and final state are described my means of states.

- A *triggering condition* is the initial state of the goal transition. A goal is generally inactive until the *state* defined in the triggering condition becomes true. Only after that, actors may address it.
- A *final state* is the final state of the goal transition. It is the desired *state* that must be true in order to claim the goal is satisfied.

Categories of actors. The *actors* section answers the question 'who is responsible to address the given goal'. According to BPMN standard, GoalSPEC includes two categories of workflow actor: the human and the system. When the actor is *the system* then the goal may be automatically addressed without human supervision. Conversely when *humans* are listed among actors then they are responsible of addressing the goal, whereas the system will be up to support and monitor human activities.

Listing 1 reports an example of production of GoalSPEC extracted from the domain of digital document management.

Listing 1. Example of goal in GoalSPEC

```
1 WHEN available(Doc) AND WHEN unclassified(Doc)
2 THE SYSTEM SHALL ADDRESS classified(Doc)
```

In this example the only actor is *the system*; the goal is active when the *a new document is available* (trigger condition); on the other hand, the desired result is the *document is classified* (final state).

A concluding remark about GoalSPEC concerns the absence of goal of type *maintain*. This category of goal prescribes the desired state must be addressed and then maintained during time. The current application context in which we applied GoalSPEC did not require goal of this kind. Future versions of the language will be extended for supporting this category of goals.

3.1 Extracting Goals from Activities

In our intentions BPMN is maintained as the main interface for modeling business processes and we developed an algorithm that is able to extract a preliminary set of goals directly from the BPMN diagram.

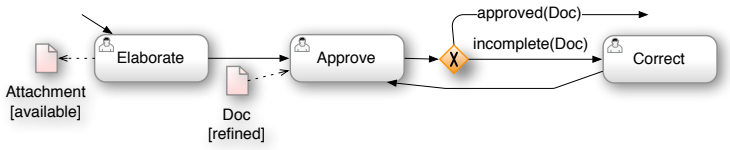

Fig. 2. A slice of the digital document management focusing on the supervision of documents.

The leading idea for the transformation algorithm is that a BPMN diagram already contains some hints of business goals. Indeed the whole workflow can be considered as a state transition from a start event to an end event. All the *FlowNodes* (Activities, Events and Gateways) in a BPMN exist to address a given intermediate state. We are interested to analyze how the state evolves during the process enactment. Figure 2 shows a slice of BPMN that we use as running example.

Let us consider a BPMN diagram as a directed graph $G(N, E)$ where *FlowNodes* are the nodes and *SequenceFlows* are the directed edges. Given a node n, we denote $Pre(n)$ the set of all node $p : (p, n) \in E$ and $Succ(n)$ the set of all node $p : (n, p) \in E$.

For each node a, denoting an Activity in the BPMN, we extract the goal

$$Goal(a) = (Actor(a), TrigCond(a), FinState(a)). \tag{1}$$

Whereas the Actor(a) is easily extracted directly by the BPMN data, for building the trigger condition and the final state we use a two-steps strategy, commented in the rest of the section.

Step 1. Firstly, we consider the node a as it were disconnected from all its $Pre(a)$ and $Succ(a)$. Therefore the Activity is characterized only by its internal features: input/output data, incoming/outgoing messages, catching/throwing boundary events. In this circumstance the state transition due to the target activity would be:

$$\begin{bmatrix} input\_data \\ incoming\_messages \\ catching\_boundary\_events \end{bmatrix} -> \begin{bmatrix} output\_data \\ outgoing\_messages \\ throwing\_boundary\_events \end{bmatrix} \tag{2}$$

We call the two components of the state transition in (2): Ready Condition (on the left) and Done Condition (on the right).

Def. 3: *The Ready Condition for a FlowNode a, denoted by $ReadyCond(a)$, is the internal condition whose satisfaction is necessary in order for a to be ready for the activation.*

Def. 4: *The Done Condition for a FlowNode a, denoted by $DoneCond(a)$, is the condition internally produced by a as the consequence of its activation.*

For instance, the activity *Approve* in Figure 2 has *refined(doc)* as ready condition, whereas we have no information about the done condition (there is no outputs), thus, we conventionally set it as *done(approve)*.

Step 2. Let us consider a couple of generic nodes n_1,n_2 where $e = (n_1, n_2) \in E$. In order the edge e is a valid BPMN connection, then the state generated by n_1 must be coherent with the state processed by n_2. As a consequence, when connecting a target element a to all its $Pre(a)$ and $Succ(a)$, the latter consideration is applied twice, generating: 1) the backward coherence: from $Pre(a)$ to a, and 2) the forward coherence: from a to $Succ(a)$.

Def. 5: *The Backward Coherence of a FlowNode a, denoted by $BackCoher(a)$ is the condition that must be true in order to properly connect $Pre(a)$ with a.*

Def. 6: *The Forward Coherence for a FlowNode a, denoted by $ForwCoher(a)$, is the condition that must be true in order to properly connect a with $Succ(a)$.*

As a conclusion, we omit to prove that:

$$\begin{aligned} TrigCond(A) &= ReadyCond(A) \wedge BackCoher(A) \\ FinState(A) &= DoneCond(A) \wedge ForwCoher(A) \end{aligned} \tag{3}$$

Equations in (3) may be used in (1) to calculate $Goal(a)$ for the FlowNode a. For instance Listing 2 reports the complete goal of the activity *Approve* in Figure 2.

Listing 2. Goal for the Approve activity

```
1 WHEN refined(Doc) AND ( (WHEN available(Attachment) AND WHEN done(elaborate))
      OR WHEN done(correct) )
2 THE manager ROLE SHALL ADDRESS
3 done(approve) AND (approved(Doc) OR incomplete(Doc))
```

4 Distributed and Context-Adaptive planning

The workflow enactment is possible because every single agent of the system owns special capabilities for addressing the activities specified in the business plan. Anyway agents have not a priori knowledge about which capabilities is good for a given BPMN activity, nor which is the right order for executing the capabilities.

Therefore it is up to the system i) to generate, at run-time, the missing link between the goal set to address and the agent activities to execute, and ii) to plan the right order for executing these actions. The advantage to elaborate this plan at run-time on the occurrence is to optimize the plan according to contextual knowledge. Indeed, the execution plan may depend on the current execution context and on the availability of capabilities and services to be invoked.

We named this mechanism as *goal injection*. When goals are injected into the system, they generate a perturbation of the internal equilibrium among the agents of the system. Indeed the agents may re-organize their priorities in order to achieve the new objectives together with the old ones. The receipt to obtain this behavior is the following:

– the agents are able to interpret the injected goals and to reason on the trigger condition and the desired final state;
– the agents are aware of their own capabilities and can check if they are able to commit to a system goal or a portion of it;
– the agents monitor the current state of the context in order to identify when an expected condition holds (trigger condition or final state) and to check if something wrong happens.

We subdivide the description of the adopted solution in two topics: the agent level section specifies the characteristics of the internal architecture of agents, whereas the system level section specifies how agents interact and collaborate.

4.1 Agent Level: Self-awareness and Context-awareness

Many works in literature agree that self-adaptive software grounds on the capability of the agent to take autonomous decisions [5, 9]. Whereas autonomy and proactivity are native in the agent paradigm, we designed agents in order to include other two fundamental features for supporting self-adaptation and evolution.

Self-awareness is the first of these features. This term has a range of meanings in literature. In our framework we implemented self-awareness as the ability of an agent to know its capabilities and its current state of execution. This knowledge is necessary for creating the bridge between agent capabilities and system goals (what an agent is able to do and what it is expected). The agent also may reason on the cost of its commitment to a goal, by considering, for example, of its current queue of tasks and the availability of resources.

In order to implement self-awareness, the knowledge about capabilities is obtained by agent believes. The agent_capability belief describes the existence of a plan the agent can select to operate over the environment. This belief comes together with a set of capability_input_state and capability_output_state believes. The former describes a state or a resource that is required to execute the capability, whereas the latter describes a state or a resource that will be generated if the capability is correctly executed.

Listing 3 reports an example of beliefs for self-awareness:

Listing 3. Example of belief-set for a capability

```
1 agent_capability(classify_document).
2 capability_input_state(classify_document,property(available,[doc])).
3 capability_output_state(classify_document,property(classified,[doc])).
```

The second feature is *context-awareness* that is the knowledge of the global state of the execution context. The context is the portion of the environment in which the agent operates. It may include both software abstractions (web-services, software resources, database, etc) and physical resources (hardware device/resources, physical environment and also human participants).

Typically an agent is able to access only a portion of the context, and often it should be able to deal with uncertainty. For instance properties and predicates that are defined in the previous example (e.g. $property(available, [doc])$, $property(classified, [doc])$)

describe properties of elements of the context. The agent must reason on the current state of the execution context in order to select and to execute a given capability. Moreover, when an agent is executing a given capability, it must be aware whether the expected result is actually addressed, otherwise it must take proper recovery actions.

We implemented a proactive monitor loop based on a set of *perception* agent capabilities. These are specifically designed for monitoring a portion of the environment. To avoid to consume too much processor resource, perceptions are activated only when the agent is actually interested to update its believes about a portion of the environment. For example this is necessary when the agent is actively committed to address a goal and it is necessary to know if the goal trigger condition holds.

Sometimes agents deal with *human participants*. The BPMN standard allows to set a human role as the responsible of an activity. When the goal is up to the system, agents get the responsibility to address the desired state transition. Conversely when the goal is up to a human role, the role of the agent, in these situations, may be twofold:

- when the task is semi-manual, the agent may supports the human in the task, by proposing user-interfaces and verifying the task progress
- when the task is totally manual, the agent can only monitor that the desired final state is correctly addressed

4.2 System Level: Self-organization

The injection of new goals into the system generates a perturbation of the internal equilibrium of the agent society. Agents must (re)organize in order to provide a solution to the problem by considering their capabilities, their current execution context and the state of environment.

Together all the agents try to find a distributed plan to address the injected goals. The algorithm is distributed and recursive. Each agent in the system checks locally its own capabilities and verifies whether these are useful to solve a the problem (or a portion). Two cases may occur: (i) the agent is able to address a whole process goal; or (ii) the agent can contribute positively to a process goal goal by addressing an activity goal.

The case (i) is the base case for terminating the algorithm. The case (ii) implies the agent calls for collaboration in discovering a distributed solution. In this case the solution is given by a *Team* of agents, each of which produces a partial activity, but together they address the whole goal. Given a process goal G with triggering condition TC and final state FS, denoted by $PG : TC \rightarrow FS$, then the agent is not able to address PG, but it claims to be able to solve an activity goal $SG : S1 \rightarrow S2$. Therefore the agent proposes a Team in which itself is the *Coordinator*. Thus, it decomposes PG into three goals: $PG_{left} : TC \rightarrow S1$, $SG : S1 \rightarrow S2$ and $PG_{right} : S2 \rightarrow FS$, where PG_{left} and PG_{right} are process goals. Therefore it publishes PG_{left} and PG_{right} and ask other agent if they are able to solve them (recursion). PG_{left} and PG_{right} may be further decomposed until case (i) occurs or until no agents reply to the call. When both PG_{left} and PG_{right} are covered by a solution, then the Team is complete and it is compared to other ones emerged in parallel according an utility function. In our case study, we use the cost of the involved resources and an estimation of time to accomplish the result. Anyway many different parameters may be on the field.

5 Concusions

In this paper we presented the overview of a framework for adaptive workflow. The work grounds on two decisions: 1) to use a declarative specification languages, compatible with BPMN, to represent business goal that may be delegated to the system; 2) to implement an autonomous multi-agent system, in which agents are self-aware and context-aware.

Self-Adaption is pushed by the possibility to inject goal into the system at run-time and the ability of agents to create a bridge between their capabilities and the expected results. In particular the process for discovering a solution is based on a distributed and recursive self-organization algorithm.

A beta version of the described framework is currently under a test phase. This is conducted at some SMEs with a set of scenarios that come from the local industrial context within a research project funded by the Autonomous Region of Sicily (PO FESR Sicilia 2007-2013).

References

1. Van der Aalst, W., Basten, T., Verbeek, H., Verkoulen, P., Voorhoeve, M.: Adaptive workflow. Enterprise Information Systems. Kluwer Academic Publishers (1999)
2. Han, Y., Sheth, A., Bussler, C.: A taxonomy of adaptive workflow management. In: Workshop of the 1998 ACM Conference on Computer Supported Cooperative Work. (1998)
3. Jarke, M., Mylopoulos, J., Schmidt, J.W., Vassiliou, Y.: Daida: An environment for evolving information systems. ACM Transactions on Information Systems (TOIS) **10**(1) (1992) 1–50
4. BPMN, O.: Business process model and notation (bpmn). www.omg.org/spec/BPMN/2.0/ (2009)
5. Cheng, B., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., et al.: Software engineering for self-adaptive systems: A research roadmap. Software Engineering for Self-Adaptive Systems (2009) 1–26
6. Sabatucci, L., Ribino, P., Lodato, C., Lopes, S., Cossentino, M.: GoalSPEC: a Goal Specification Language supporting Adaptivity and Evolution. In: EMAS 2013, LNAI 8245. (2013) p.237–256
7. Morandini, M., Penserini, L., Perini, A.: Towards goal-oriented development of self-adaptive systems. Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems (2008) 9–16
8. Liaskos, S., Khan, S.M., Litoiu, M., Daoud Jungblut, M., Rogozhkin, V., Mylopoulos, J.: Behavioral adaptation of information systems through goal models. Information Systems (2012)
9. Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM Transactions on Autonomous and Adaptive Systems (TAAS) **4**(2) (2009) 14
10. Sadiq, S., Marjanovic, O., Orlowska, M.: Managing change and time in dynamic workflow processes. International Journal of Cooperative Information Systems **9**(1-2) (2000) 93–116