Supporting Dynamic Workflows with Automatic Extraction of Goals from BPMN

LUCA SABATUCCI, Italian National Research Council, ICAR, Italy
MASSIMO COSSENTINO, Italian National Research Council, ICAR, Italy

Organizations willing to employ workflow technology have to be prepared to undertake a significant investment of time and effort due to the exceptional dynamic nature of the business environment. Today, it is unlikely that processes are modeled once to be repeatedly executed without any changes. Goal-oriented dynamic workflows are a promising approach to provide flexibility to the execution of business processes. Many goal-oriented frameworks exist in literature to be used for the purpose. However, modeling goals is a burden for the business analyst.

This work proposes an automatic approach for extracting goals from a business process for supporting adaptive workflows. The approach consists of a static analysis of the global workflow state. Goals derive from individual BPMN elements and their interactions.

For validating the theory, we developed the BPMN2Goals tool, which has been used for supporting a middleware for self-adaptation.

CCS Concepts: • **Applied computing** → **Business process management**; • **Computing methodologies** → *Knowledge representation and reasoning*; • **Software and its engineering** → *Requirements analysis.*

Additional Key Words and Phrases: Dynamic Workflow, Self-Adaptive Systems, Business Process Goals

ACM Reference Format:
Luca Sabatucci and Massimo Cossentino. 2019. Supporting Dynamic Workflows with Automatic Extraction of Goals from BPMN. *ACM Trans. Autonom. Adapt. Syst.* 1, 1, Article 1 (January 2019), 38 pages. https://doi.org/10.1145/3355488

1 INTRODUCTION

Workflow management aims at modeling and controlling the execution of application processes according to predefined specifications [37, 72]. Traditionally, these specifications are given in the form of directed graphs whose nodes represent tasks and whose edges represent constraints between them, namely data flow and control flow [24, 51]. Task specification is the activity of decomposing the whole process in a set of logic units of work and specifying their types and interfaces.

Today, it is uncommon that processes are modeled once to be repeatedly executed without any changes [22]. The majority of them will face unplanned events and exceptions leading to ad hoc deviations from the preplanned flows of tasks. External events, service failures, incomplete or erroneous data, all these situations produce a mismatch between the real processes and the computerized counterpart [57].

Authors' addresses: Luca Sabatucci, Italian National Research Council, ICAR, Via Ugo La Malfa 253, Palermo, PA, 90100, Italy, luca.sabatucci@icar.cnr.it; Massimo Cossentino, Italian National Research Council, ICAR, Via Ugo La Malfa 253, Palermo, PA, 90100, Italy, massimo.cossentino@icar.cnr.it.

1556-4665/2019/1-ART1 $15.00
https://doi.org/10.1145/3355488

Dynamic workflows deal with changes during the workflow execution. Some of these changes are unpredictable, and it is hence necessary to allow flexible modeling and run-time maintenance. In the literature, two approaches exist to model and enact dynamic workflows: anticipating all the possible changes and failures [69] or operating contextual run-time modifications to the running workflow instance, so to fit the new conditions [41, 57].

Recently, the employment of declarative languages allowed breaking the design-time constraints typical of a static workflow definition and enlarging the space of solutions for workflow management. Among declarative approaches to dynamic workflow description we are interested in goal-based languages [13, 42] that, on the one hand, allow relaxing structural constraints, and, on the other hand, allow shifting the focus from 'how things must be executed' to 'which results must be addressed' [54]. In addition, goals are the base for accessing a wide literature about self-adaptive systems and implementing dynamisms in the enactment of workflows [15, 16, 34, 36, 41, 78].

The problems are: who will produce these goal-based workflows and how. The literature contains several approaches to drive and support business analysts in defining their business strategy through the explicit definition of goals [28, 28]. Unfortunately, this practice is not yet standard in the industrial setting, where modeling workflow as a control structure is largely preferred. The result is a traditional control flow structure in which goals are implicit and therefore not available for our purpose of dynamism.

In our vision, the BPMN [51] (or similar languages) must continue to be the primary instruments for defining a workflow.

The objective of this work is to automatically provide implicit goals that are embedded in a traditional workflow definition. We focus on the life cycles of information entities (i.e. documents, business objects, and artifacts) that are managed and/or exchanged during the process.

The first contribution of this paper is a semantics for identifying relevant states of the workflow. We decided to analyze data objects, messages and events to describe the evolution of the workflow state from the start to the end events. We propose a predicate logical semantics to characterize BPMN elements according to their input/output data objects, messages, and events.

The main contribution of the paper is an automated procedure to extract implicit goals from a BPMN workflow description. The proposed procedure works by relaxing some control flow constraints and focusing only on workflow states and gateways. Goals, generated in this way, are suitable for enabling the self-adaptation ability, and therefore enacting dynamic workflows.

Indeed, the self-adaptation ability is obtained by providing the extracted goals to an adaptive middleware, able to select and compose services at run-time. The result is a workflow management system that operates run-time changes to the workflow instance, as a consequence of failures or for the evolution of functionalities.

The paper is structured as follows: Section 2 sets the objectives of the work, explaining the general concept of using goals instead of tasks to enact a dynamic workflow. Section 3 introduces a semantics for identifying relevant states of a workflow and characterizing elements of a BPMN. Section 4 presents the theoretical basis for the proposed approach, whereas the concrete procedure is described in Section 5, where we also provide the analysis of the complexity. Section 6 provides a discussion of advantages, limitations concerning the proposed approach and it sets some future works. Section 7 compares the proposed approach with alternative/complementary works in the literature. Finally, some conclusions are drawn in Section 8.

2 MOTIVATION: GOAL-ORIENTED DYNAMIC WORKFLOWS (WITH MUSA)

This section introduces the general concept of goal-oriented adaptation, with particular emphasis to MUSA [65], the middleware for self-adaptation we use for implementing dynamic workflows. The peculiarity of MUSA is the adoption of run-time models for goals and capabilities and an engine (called *proactive means-end reasoning*) for generating a late service composition and binding them [61] in order to create many workflow solutions.

The aim of the sectionis to give a preliminary definition of goal and explains why goals are more flexible than traditional workflow provided as a flow of tasks.

2.1 Why Dynamic Workflows

Only a few business processes are statical. The majority of them will face unplanned events and exceptions leading to ad hoc deviations from the preplanned flows of tasks. Some of the most frequent causes of deviation are external events, failed tasks, incomplete or erroneous inputs and outputs, and, in general, situations in which a mismatch exists between the real processes and the computerized counterpart [57].

A taxonomy of motivations for dynamic workflow is provided in [36]. The main objective is to increase agility, flexibility, and adaptability of business processes. This objective is in conflict with the current trend of completely specifying a workflow at design-time, for example using BPMN. A basic step towards more flexibility [59] is the effective and efficient support of ad hoc modifications and well-aimed extensions of processes during their execution.

Example 2.1. Figure 1 reports the email-voting process, a simplified version of the business process available in [51]. It describes the process of mediating and coordinating some voting members who have to resolve issues in a participative way. The process reports the perspective of the manager of the issue list, whereas voting members are considered as external participants. The issue list manager reviews the list and figures out if there are issues that are ready for discussion. Then a discussion session starts: the participants will discuss via email for one week, proposing solutions to the issues. Then a voting session begins, and voting members are asked to vote. The last activity is assigned to the manager who prepares the results, checks that at least two-thirds of the members approved the solution and then communicates the results. If the issue has not been resolved, then a new discussion cycle starts. This example will be used as a running example in this paper.

The execution of this business process involves several automatic and manual tasks. Automatic tasks are performed through web service invocation, whereas manual tasks are performed by a human stakeholder with the support of computer applications. In Figure 1, an example of service task is the *Check Calendar for Conference*, whereas *Moderate Conference* is a manual task.

One of the crucial points of the email-voting process is the communication between the issue manager and the voting member. The basic way to implement this message exchange is via email. A `send_email_notification` is called every time the manager will notify a member.

This approach could fail when a member cannot read the inbox; however, other ways can be used in alternative or in combination: a `send_text_notification` uses phone texts, `send_im_notification` uses an instant messaging service; also cloud file-systems may be used to share some document.

A bit more complex situation occurs when the company, to reduce the manager burden, has implemented an automatic `issue_extration`. This service uses natural language processing to recognize the issues and produce the document. However, the algorithm fails in 10% of cases. The workflow management system has to recognize these cases and switch toward the manual extraction, in which the manager is supported during the task execution by a GUI (graphical user interface).

To cope with situations like these, changing the workflow instance during execution has become an important requirement. The use of a dynamic workflow management system allows run-time

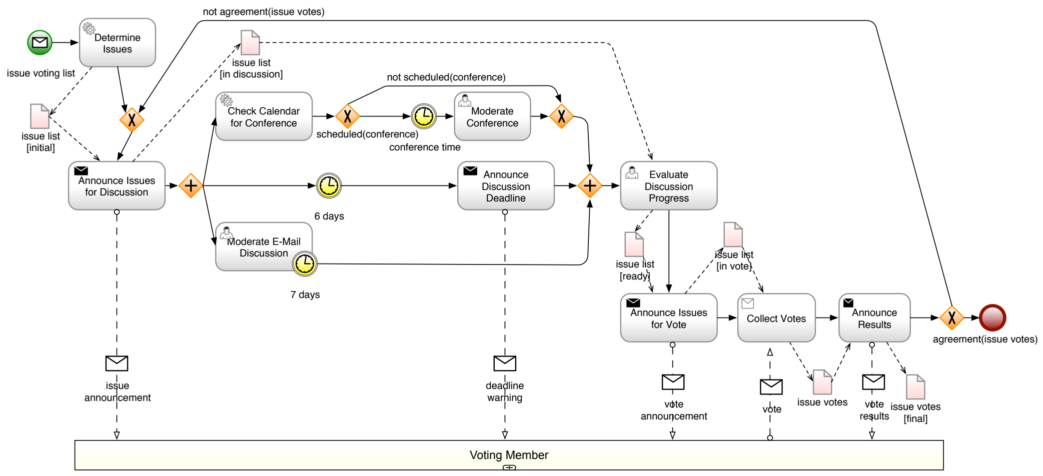

Fig. 1. Example of BPMN diagram representing the email voting process.

modification of the original business process according to the current execution context, the availability of resources and possible (unpredictable) failures happening during the execution.

Traditionally dynamic workflows have been implemented through Petri nets, decision points, and variants. The use of Petri nets in the area of workflow formalization has a long research tradition [71, 72]. Petri nets have also been used for increasing flexibility and failure recovery. Recovery Nets [35] are special Petri nets that include, for each type of failure, a recovery transition to handle specific recovery policies. In [25], authors describe a hierarchical Petri net approach that use refinement rules to decompose a workflow Petri net in regions that are associated to unexpected event handlers that restore the workflow execution within a region once an exception is raised. The main limit of Petri nets is that they focus only on the structure of the process, leaving out data-based elements. We consider goal extraction may benefit from the richness of such details (in particular events, messages, and data objects).

Another research trend consists in introducing decision/adaptation points inside the workflow model. A decision point is a special place, in which a human or an automated system decides what to do next according to predefined rules [40, 73]. A decision point may also be used to introduce conditional branches corresponding to the same (sub)process [5], thus choosing an alternative is a decision automatically taken according to contextual data and pre-defined rules [39]. There exist advanced construction techniques [47] and specific languages to define how to change the process at run-time [58].

2.2 Relaxing the Control Flow Constraint

Recently, the employment of declarative languages allowed breaking the design-time constraints typical of a static workflow definition and enlarging the space of solutions for workflow management.

Let us consider a simple sequence of two tasks (in Figure 1): *Determine Issue* and *Announce Issues for Discussion*. This static definition introduces a rigid constraint between the two tasks (*Determine Issue* is a predecessor of *Announce Issues for Discussion*). This indicates the second task will follow the first one. If *Determine Issue* fails, it is necessary to replace that with an equivalent service (if available).

However, by relaxing precedence constraints, and focusing on data/event/message entities of the model we can define a different kind of dependencies among tasks. Indeed, in Figure 1, *Announce Issues for Discussion* depends on the state of the issue list document (that is produced by Determine Issue). In our vision, *Announce Issues for Discussion* can start as soon as the data object is ready and in the expected state, even if *Determine Issue* is not terminated yet or even it never started. Relaxing control flow constraints allows re-organizing activities at run-time out of the pre-defined models. For instance, if two tasks wait for the same data object, and there is any dependency among them, according to this vision, they could play in parallel, even if the model prescribes a sequence.

Among declarative languages for dynamic workflow description we identified two main trends: rule based [49, 54, 77] and goal-based [13, 42]. In the former category, Milanovic et al. [49] use business rule patterns to enrich business process by possible cases.
Wang et al. [77] propose a local replanning strategy for repairing a failure in a workflow. They use dynamic description logic (combining description logic and action theory) to reason about available services and respond to declarative service requests (similarly to goals).

The approaches suggested in [2, 54] propose declarative languages that specify what should be done (without specifying how it should be done) and engine based on linear temporal logic and automata for generating executable workflows from a number of possible and reusable tasks at runtime [54].

On the other hand, goal-based specifications are driven by the idea that processes conceal goals, whose fulfillment is possible due to the actions of participants [43]; during the workflow execution, there are sub-goals to be addressed that denote milestones within the process [53]. To support this perspective, in May 2015, the OMG proposed the Business Motivation Model, standard notation for capturing business requirements across different dimensions to make explicit why the business wants to do something, what it is aiming to achieve, how it plans to get there, and how it assesses the result [6].

Almost all the approaches in this category are based on the common characteristic that goal models must be manually declared to enable the adaptation process.

Kazhamiakin et al. [42] adopt Tropos goal models [13] making explicit the objectives of the different actors involved in a choreography. In [45], authors adopt i^* [81] goal models with a design-time binding between leaf level tasks and the source code to execute them. A *situation calculus* planner interprets users' behavioral constraints and produces plans that best satisfy the user. Brown et al. [15] use formal KAOS [74] goal models to explicitly specify what should occur during adaptation.

It worths mentioning Baresi and Pasquale [10] who use fuzzy requirements for service compositions and suggest a strategy for building supervision models thus enforcing goals during workflow execution. The use of adaptive goals [78] allows setting requirements with special operators (such as 'as soon as' and 'as close as possible') granting more flexibility in how and when that functionality may be delivered.

This paper focuses on goals to capture data dependency between tasks, relaxing rigid structural constraints of a traditional workflow. In the following, we first illustrate the concept of goals and then we show an example demonstrating how goals enable high flexibility in run-time modifications.

2.3 Goal-Defined Workflow

If the workflow is specified through a set of goals, we shift the focus from 'how things must be executed' to 'which results must be addressed'. Indeed goals do not contain information about the flow of tasks. The workflow management system owns a degree of freedom in deciding how combining available services for addressing goals.

It is worth noting that in the rest of the paper we set the following convention: we use ***Tasks*** to refer to business tasks, i.e., the abstract units of work that appear in a business process; we use ***Services*** to refer to concrete software routines (with an external logic) the workflow management system invokes for addressing the Task. Consequently, we decouple tasks and services and set an additional hypothesis: there is no pre-defined link between services and tasks: the workflow management system will generate a late-binding between them.

Consequently, the workflow management system must be aware of the kind of services are available to enact the workflow and to address intermediate and final results. Clearly, it is required a specific language for service description that enables automatic selection and manipulation. The workflow management system must be able to select and composing services according to the goals to be addressed. The extensive literature on dynamic workflow is often based on late service binding [12, 23, 34] and composition [3, 7, 19, 26, 55]. In [23], authors present eFlow, a composite service engine that runs on top of E-services platforms, i.e. platforms allow service provider to register descriptions of the services they offer. Systems like ADEPT [57] develop very complex workflow engines that are able to handle inserting, deleting and moving tasks at run-time. They advocate the need for a separation of the application's control structures from the implementation of its tasks.

Another approach for enacting goal-defined workflow is MUSA [65], a middleware specifically conceived for service composition and adaptation. MUSA proposes an adaptation of type 3 [66], that consists of systems with no pre-defined strategies but aware of their objectives. It assemblies ad-hoc functionalities according to the execution context. Consequently, in MUSA the adaptation ability depends on the number and kind of services available in the repository and on the possibility to compose them. Moreover, the redundancy of services in the repository is fundamental to the aim of proposing alternative solutions to the same set of goals.

MUSA is compliant with the two operative hypotheses illustrated below. Hereafter, we refer to MUSA as the reference adaptive system for demonstrating how goal-oriented workflows extend the flexibility of run-time modifications.

Let us consider the abstract workflow in Figure 2 provided as a sequence of tasks (*taskA -> taskB -> taskC ...*). A classical workflow management system must execute these tasks in the given order. Let us suppose the same workflow is provided as a set of goals [G1], [G2], [G3]... When MUSA processes these goals, according to the current service repository (shown in the top-right side of Figure 2), it performs a run-time means-end analysis [61] and it proposes –among many alternative solutions– the following ones:

(W_1) s1 -> s2 -> s3...
(W_2) s1 -> s2' -> s3...
(W_3) s1' -> s3...
(W_4) s1" -> s1"'-> s2 -> s3...
(W_5) s1 -> (s4' || s4")...

where the -> operator indicates a sequence of tasks, and the || operator a parallel of tasks. When selecting one of these alternative solutions, the others are stored for future adaptation purpose. For example, suppose after the execution of s1, service s2 fails, MUSA may resolve by:

- replacing s2 with s2' (i.e.. switching to W2)
- replacing s2 with the parallel execution of s4' and s4" (i.e. switching to W4)

The presented approach to dynamic workflows has been already presented in [60, 64, 65]. In these academic projects, requirement engineers worked for manually producing the goal specifications, respectively for, a smart travel system, a cloud B2B application, and a document management system.

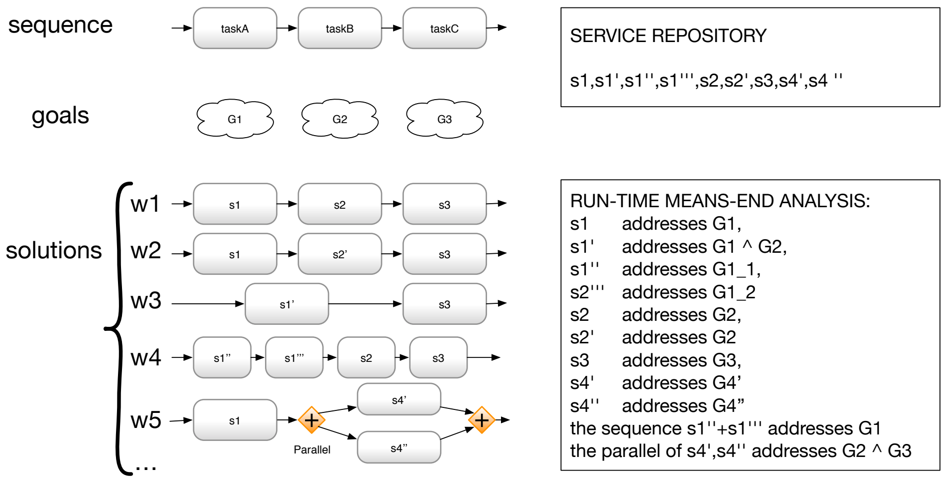

Fig. 2. Elements of dynamic workflow with MUSA. If the workflow is given as a sequence of tasks, the workflow management system must execute *taskA*, *taskB*, and *taskC* in the given order. If the same workflow is given as a set of goals, then –according to the available services– many different workflows are generated by composing services at run-time. The system may execute one of these and switch to another solution when a failure occurs.

This paper faces the problem that, even if it is advantageous, the goal-based description of a workflow is rarely available in the industrial setting where BPMN represents the de facto standard for specifying a business process. The objective of this work is to provide an instrument for automatically extracting implicit goals to be used for feeding the dynamic execution of the workflows specified through the BPMN.

3 SEMANTICS FOR DESCRIBING THE STATE OF A BUSINESS PROCESSES

The ultimate objective of workflow management is executing and managing a business process, assuring that the right person executes the proper activities at the right time [71]. Business process modeling plays a major role in workflow automation, and BPMN represents the de-facto standard for designing workflows.

This section illustrates one of the contributions of this paper: the identification of relevant states of a business process. Firstly, it gives an overview of the principal BPMN constructs. Then, in line with [18, 32, 44], the following subsections propose semantics for describing relevant states of a workflow. Given the focus of the whole paper is on relaxing structural constraints, relevant states are identified by looking at data items, messages, events, and conditions.

3.1 Business Process Modeling Notation

Modeling business processes [76] consists in defining the order of execution of a set of activities which collectively realize a business objective within the context of an organizational structure.

The Business Process Model and Notation (BPMN) [24, 51] is a very expressive graphical notation for representing business processes of diverse natures. A graphical notation is the de-facto standard choice to express a representation of a process which should be syntactically valid: examining a BPMN diagram allows analysts to discover inefficiencies, inconsistencies, deadlocks, non-terminating conditions and so forth [24].

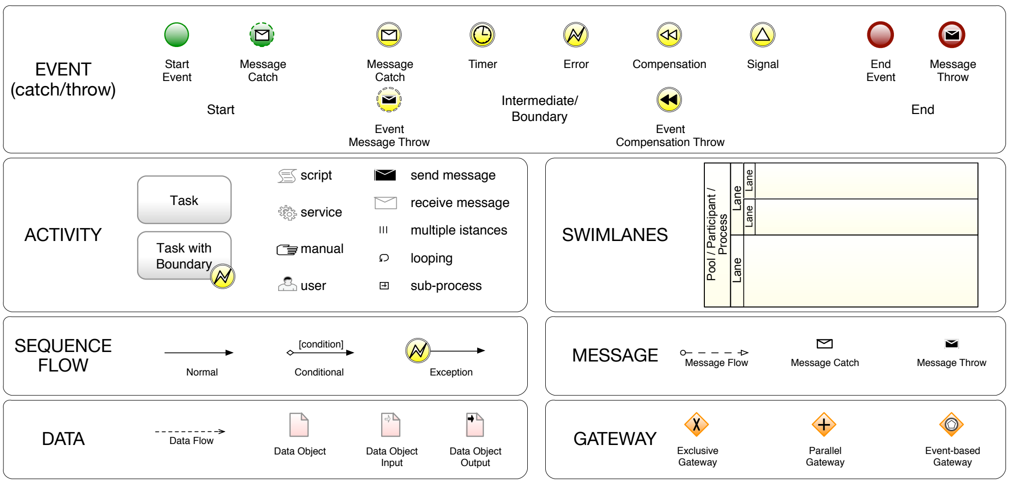

Fig. 3. Overview of the main BPMN elements.

Figure 3 presents a summary of the elements of BPMN 2.0. This paper focuses on collaboration diagrams composed of seven categories of objects: events, activities, swimlanes, sequence flows, messages, data objects, and gateways. The standard also includes elements for conversations and choreographies [24, 51].

The core elements are tasks, events, and gateways. An event triggers the start and the end of a process and it may also occur during the process (intermediate or boundary events). A message event is used to specify an incoming or throwing message. A timer event triggers at a given instant of time. Error, compensation, and signal events are special events used to manage failures.

An activity can be a task or a sub-process. A task is an atomic activity of six types: service, receive, send, user, manual and script. It may also optionally have attributes such as multiple instances, looping, and sub-process. A sub-process is a compound defined by a sub-flow of activities.

A gateway is a routing construct that specifies either parallel or exclusive forks/joins. The event-based gateway is a special gateway driven by events. An event may also be set in the boundary of a task thus specifying exceptional termination when that event occurs.

A sequence flow is used to indicate the execution order of activities, events, and gateways. When a condition is attached to the sequence flow, it is executed when the condition occurs. A message flow is used to represent the exchange of messages between two interacting processes.

Data objects are used to represent items produced/consumed during the process explicitly. A data object may have a lifecycle. It is identified by a state, put between square brackets after the name. Input/output objects are external data used within a process or a task. Data flow is used to associate data elements to the corresponding activities that consume or produces them.

3.2 Input/Output Data Objects

Task activation may be subject to the availability of their input. Moreover, they often produce specific data output.

The BPMN specification [51] provides several constructs to model data within the workflow – primarily DataStores, DataObjects, DataInputs, and DataOutputs. These BPMN elements could be associated with activities and events via DataInputAssociations and DataOutputAssociations to specify their requirements and their outcome in terms of data and resources.

We use the predicate 'available' for indicating the condition of availability of data. The result is a predicate in the form $available(\langle Data \rangle)$, where $\langle Data \rangle$ is a placeholder for the name of the input/output artifact.

If the state of a data object is specified, then we use a predicate in which the functor is the string corresponding to object state. Examples: notified(document) or filled(questionnaire).

Example 3.1. The *Announce Result* task, in Figure 4.a, has been extracted by the email-voting process. The issue_votes data object is a Data Input since it is connected to an input port of the task. The predicate used for representing this fact is $available(issues\_votes)$.

The same data object, but with a state (ready), is shown in Figure 4.b. The result of *Evaluate Discussion Progress* is a change of state for the output data object; this fact is modeled via the predicate $ready(issue\_list)$.

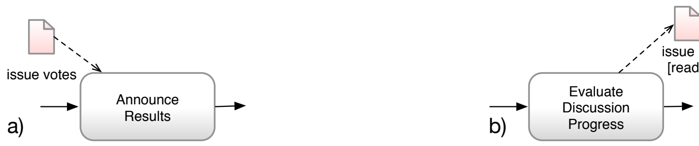

Fig. 4. On the left a stateless data is the input of a task. On the right the same data object is enriched with a state.

3.3 Received/Sent Messages

BPMN processes are often composed of several collaborating parties, each modeled as a separate sub-processes but communicating each other via messages exchange [51].

An activity or an event which is the destination of messageFlows activates when it receives the prescribed message. The convention uses the predicate 'message_received' for indicating this condition holds.

The result is a fact in the form $message\_received(\langle Message \rangle, \langle Actor \rangle)$ where $\langle Message \rangle$ is the placeholder for message content and $\langle Actor \rangle$ is the optional message sender.

Conversely, an activity or an event may be the source of messageFlows. This means it produces a new state of the world when messages are produced. We use the predicate 'message_sent' for indicating this condition. The result is a fact in the form $message\_sent(\langle Message \rangle, \langle Actor \rangle)$ where $\langle Message \rangle$ is the placeholder for message content and $\langle Actor \rangle$ is the optional name of the recipient.

Example 3.2. Continuing the example of the email-voting process, the *Determine Issues* is a task that receives messages of type (Figure 5.a), whereas the *Announce Issues for Vote* task produces the vote_announcement message for the voting member (Figure 5.b).

The predicate $received(issue\_voting\_list, customer)$, or $received(issue\_voting\_list)$, may be used for representing the fact, whereas the predicate used for describing this outgoing message is $sent(vote\_announcement, member)$ or, simply, $sent(vote\_announcement)$.

3.4 Caught / Thrown Events

BPMN events are extremely numerous and varied [51]. Every event is associated with a specific 'state of affairs' which persists for a limited duration. Generalizing, the convention is to use a set of predicates in the form $caught(\langle event \rangle)$ or $thrown(\langle event \rangle)$, depending if the event is captured or

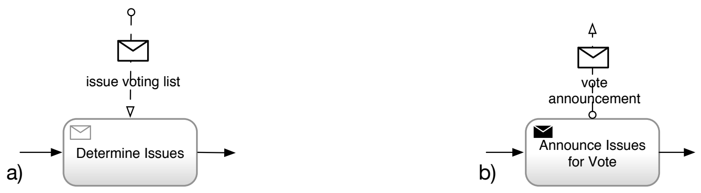

Fig. 5. The predicate used for representing this incoming message is: $received(issue\_voting\_list, customer)$
.

generated by the workflow management system. Both predicates use $\langle event \rangle$ as a placeholder for the specific category of event. A couple of examples are $caught(error)$ and $thrown(compensate)$.

A specific notation is used for the Timer Event. Timer Events wait for a time-based condition to trigger the catch Event. The BPMN specification [51] proposes two kinds of time definition: absolute and relative. An absolute time condition triggers at a specific date-time. The predicate $at(\langle DateTime \rangle)$ is used to specify the event condition. A relative time condition acts as a delay mechanism: it triggers after a lapse since a previous condition. The predicate to be used is $after(\langle Delay \rangle, \langle PreviousEvent \rangle)$.

Example 3.3. In Figure 1, both absolute and relative timer events exist. The first, reported in Figure 6.a is characterized by the label 'conference time'. The predicate for representing the first event is $at(conference\_time)$ because the time is a specific date.

The second one is characterized by the label '6 days', reported in Figure 6.b. In order to identify the event condition, the analyst has to consider Timer predecessors' elements. In the example, this event is the announcement of the issues to discuss (modeled as $done(announce\_issue\_for\_discussion)$. Therefore the predicate $after(days(6), done(announce\_issue\_for\_discussion)$ represents the second events.

a)
conference time
b)
6 days

Fig. 6. Two kinds of events in the email-voting process.

3.5 Termination and Boundary Events

The BPMN specification provides many instruments for dealing with task termination. It allows modeling that activities may fail, be canceled before its completion or event compensated after their execution. The modeling tool for these exceptional situations is to add interrupting events as task boundaries [51].

This approach deals with most of the boundary events in the specification. Basically the predicates 'done' and 'error' are used to indicate, respectively, a normal or an abnormal termination. The result is a fact $done(\langle action\_name \rangle)$ or $error(\langle action\_name \rangle)$ – where $\langle action\_name \rangle$ is the placeholder for the label of the task.

If the activity is interrupted because of a generic boundary event, then it is possible to use a set of predicates of the form $\langle boundary\_event \rangle\_termination\ (\langle action\_name \rangle)$, where $\langle boundary\_event \rangle$ is a placeholder for the kind of boundary event has triggered.
Example: $compensate\_termination(flight\_booking)$.

Example 3.4. The *Moderate E-Mail Discussion* task of Figure 7 contains an example of boundary event. The meaning of the notation is the task may either terminate normally or after seven days pass. In the example of Figure 1, the 'normal' termination is not used; the unique outgoing flow is via the boundary condition: the task terminates after a week.

The predicate $timer\_termination(moderate\_email\_discussion)$ indicates this termination condition.

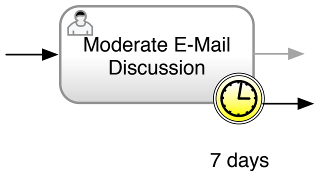

Fig. 7. A task with a timer event boundary condition. The 'normal' outgoing flow is in gray because it is not used in Figure 7

Table 1 recaps all the semantics we will use to identify states of the workflow in the remaining sections of this paper.

Moreover, we provide a denotation for some of the relevant characteristics of a BPMN element (activity, event and gateway), namely e.

- Data Objects: depending if the data object appears as an input or an output of e, we refer to either the Input Data Conditions, denoted by $data\_in(e)$, or the Output Data Condition, indicated by $data\_out(e)$.
- Messages: depending if the message arrives to or departs from the BPMN element e, we use $mess\_in(e)$ in order to refer to the Received Message Condition and $mess\_out(e)$ to refer to the Sent Message Condition.
- Events: depending if the event is caught or thrown by the BPMN element e, we refer to either the Catch Event Condition, denoted by $event\_in(e)$ or the Throw Event Condition $event\_out(e)$.

Example 3.5. Let us consider the *Collect Votes* task of Figure 8 – that receives a document and a message as input and it produces a document as output.

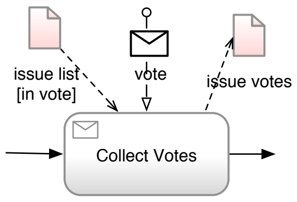

Fig. 8. A task, represented with input and output, of the email-voting process.

We can observe that:

$$\begin{cases} data\_in(e) & = in\_vote(issue\_list) \\ data\_out(e) & = available(issue\_votes) \\ mess\_in(e) & = received(vote, member) \\ mess\_out(e) & =< unspecified > \end{cases} \tag{1}$$

Table 1. Summary of the semantic introduced in this work to describe the state of a workflow instance during its exection.

$available(\langle Data \rangle)$	it specifies the availability of a given input/output artifact;
$\langle state \rangle(\langle Data \rangle)$	it specifies the input/output artifact assumes a given state;
$received(\langle Message \rangle, \langle Actor \rangle)$	it is true when a given message incomes from an external participant;
$sent(\langle Message \rangle, \langle Actor \rangle)$	it is true when a message is sent to an external participant;
$caught(\langle Event \rangle)$	it is true when a specified event is captured by the system;
$thrown(\langle Event \rangle)$	it is true when the system generates a specified event;
$at(\langle DateTime \rangle)$	it becomes true when the current date is equals (or pretty close to) the specified one;
$after(\langle Delay \rangle, \langle Event \rangle)$	it becomes true after a time lapse after since when the specified event has occurred;
$done(\langle Activity \rangle)$	it specifies a generic activity has been executed with success;
$error(\langle Activity \rangle)$	it is true when the given activity has failed;
$\langle boundary \rangle\_termination(\langle Activity \rangle)$	it specifies the activity has been interrupted for given a boundary event;

According to the semantic introduced in this secion, the task's required input is the issue_list object when the state is 'in vote': $data\_in(e) = in\_vote(issue\_list)$. The task's output is the production of a new issue_vote object: $data\_out(e) = available(issue\_votes)$. Finally the task also waits for incoming vote messages: $mess\_in(e) = received(vote, member)$, whereas no output message is sent.

4 FROM BUSINESS PROCESSES TO GOALS

This section presents the main novelty of the paper. In Section 2 we provided an example to show to which extent goals are more flexible than flows of tasks. Goals may be used to relax the rigid structural constraints of a traditional workflow and, conversely, emphasize other types of dependency. However, very often, real goals are not available in the workflow model, especially in industrial settings. We face the problem of automatically deriving implicit goals from a traditional workflow definition by exploiting the identification of relevant states of Section 3. This section describes a framework for extracting implicit goals from a business process with the purpose of enabling and automatizing the execution of dynamic workflows through the MUSA middleware.

For introducing the definition of goal, let us consider a workflow instance is a running instance of a workflow schema that is characterized by an execution state. Activities, events, and gateways of the process have an impact on this state. We suppose to be able to describe the state evolution of this state from the initial state to the final state with the logic of possible worlds to study state dynamics along with a temporal line. The execution of a workflow may be explicitly represented as a finite sequence of states of the world.

Definition 4.1 (State of the World). The state of the world in a given time t is a set $W_t \subset S$ where S is the set of all the (non-negated) first order variable-free statements (facts) $s_1, s_2 \dots s_n$ that can be used in a given domain. All the facts that are in W_t are considered to be true at time t.

For the definition of goal, we have been inspired by Zambonelli et al. [1]: a goal is the eventual achievement of a condition over the state of the world. The formalization is aligned with most of the literature about goal models [13, 50, 81]. In the context of workflow management, a goal may not necessarily be always active. Rather, it can be the case that a goal will only get activated when specific conditions occur. When these conditions happen, then the goal prescribes a *change* in the state of the world:

Definition 4.2 (Goal). A goal is a pair: $\langle tc, fs\rangle$ where *tc* and *fs* are conditions (i.e. logical formula of type $\varphi : W \longrightarrow \mathbb{B}$ – built through the standard set of logical connectives: ($\neg$, $\wedge$, $\vee$[1]). Respectively, the *trigger condition* (tc) describes when the goal may be pursued and the *final state* (fs) describes the desired state of the world.

In practice, a goal is *addressed* if and only if, after the trigger condition becomes true, then eventually the final state must be true.

In the previous section, we illustrated how to identify and describe relevant states by looking at data, events, and messages. Now we study the evolution of the state due to a generic BPMN element *e* (activity, event, or gateway): $evolution(e)$ is a function defined in $W \times W$. It describes the change in the state of the world when the element *e* is disconnected from the rest of the workflow.

To deal with activities, events, and gateways with the same approach, we introduce the following conventions:

Definition 4.3 (Waited State). The Waited State (WS) for *e*, denoted by $ws(e)$, is the condition that is expected to be true before the element is ready for the activation.

Definition 4.4 (Generated State). The Generated State (GS) for *e*, denoted by $gs(e)$, is the condition that is expected to be true after the execution of *e*.

The waited state and the generated state are *internal factors* of an element. In the way they are defined, WS and CS describe the individual contribution of a generic BPMN element to the evolution of the state of the world: by looking at input/output ports of a BPMN element, it is possible to derive which kind of impact has towards the global state. Indeed, Sections 4.1, 4.2, 4.3 analyze, respectively, the impact of activities, events and gateways, in terms of state-change.

However, WS and GS are not enough to formulate the goals we aspire to because they study the element when it is disconnected from the rest of the workflow. The waited state does not correspond to the goal's trigger condition and the generated state is not the goal's final state.

Indeed, when we reconnect the element to the workflow, the state is influenced by its neighbors. Therefore, it is necessary to study the mutual incidence of predecessor and successor elements.

Example 4.5. Let us consider the simple sequence of two tasks, shown in Figure 9.a. In this case, the state generated by the first task necessarily coincides with the state waited by the second task, $initial(issue\_list)$, because there are not other factors. However, when considering the second example, shown in Figure 9.b, then the previous consideration does not hold. Indeed the generated state is $initial(issue\_list) \wedge available(priority)$ whereas the waited state is only $initial(issue\_list)$. In this case $GS \neq WS$; the question is how to calculate the state between the two tasks?

Things are even more complicated when the structure contains some gateways because in this case it is necessary to consider multiple predecessors and successors. Intuitively, the state, at the input of *Announce Issues for Discussion* (Figure 9.c), is the xor of two branches: $(initial(issue\_list) \wedge available(priority)) \oplus (final(issue\_votes) \wedge \neg majority(issue\_votes))$. Indeed, the task starts either 1) when a new issue_list incomes (*Determine Issues*) or 2) after the *Discussion and Vote* phase terminates with no agreement.

[1]In the rest of the paper we frequently use the xor logical connective. The common notations for xor are: $\oplus$, $\veebar$ and sometimes $||$. The xor is a derived logical connective that can be expressed in terms of the other logical connectives as follows: $a \oplus b = (a \vee b) \wedge \neg(a \wedge b)$).

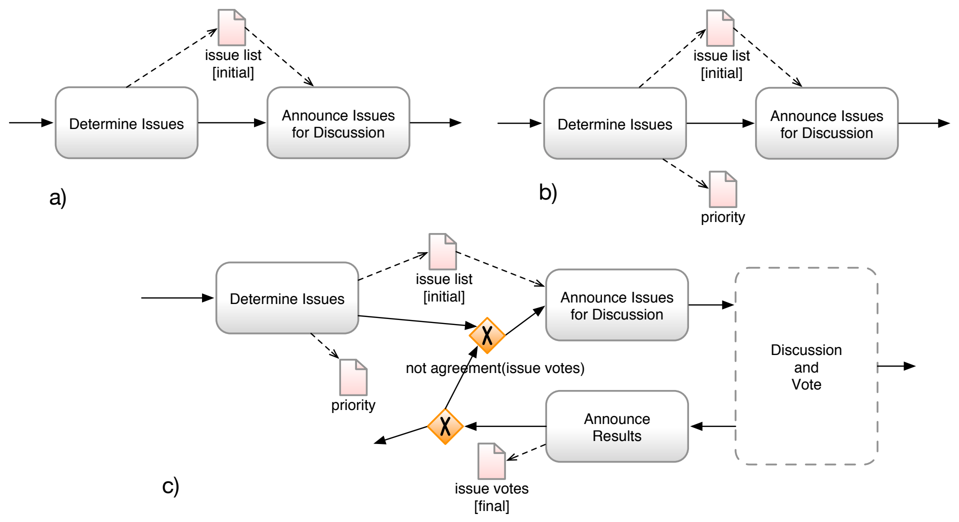

Fig. 9. Different ways of connecting two tasks. a) a simple sequence of two tasks that exchange a data; b) the same sequence but the first task also generates additional data; c) slice extracted from Figure 1 in which there is also the influence of a couple of gateways.

To provide a general method for deriving the goals, we introduce the *external factors*, that characterize the position of the element within the workflow structure:

Definition 4.6 (Predecessors Influence). The Predecessors Influence for an element e, denoted by $pre\_inf(e)$, is the condition that must be true to make the element e compatible with its predecessor elements in the workflow.

Definition 4.7 (Successors Influence). The Successors Influence for an element e, denoted by $succ\_inf(e)$, is the condition that must be true to make the element e compatible with its successor elements in the workflow.

Figure 10 explains the idea: the BPMN element is ruled by internal conditions, but, when connected with other elements, it must undergo to a 'balance of states': the *waited state* must balance the *predecessors influence* and the *generated state* must balance the *successors influence*. Balancing these forces, it is possible to 'measure' the state transition, before and after, and thus formulating the corresponding goal.

The operative hypothesis. For each task e in the workflow, the corresponding goal $G = \langle tc, fs \rangle$ can be calculated by solving the boolean expression due to the following rules:

$$\begin{cases} \text{if } \exists M_{tc} : M_{tc} \models tc & \Rightarrow M_{tc} \models ws \land M_{tc} \models pre\_inf \\ \text{if } \exists M_{fs} : M_{fs} \models fs & \Rightarrow M_{fs} \models gs \land M_{fs} \models succ\_inf \end{cases} \tag{2}$$

where M_{tc} and M_{fs} are possible intepretations of the model.

Formula 2 may be read as: if an intepretation exists, such that tc is true, then according to the same intepretation also ws and pre_inf will be true; similarly, if another intepretation exists, such that fs is true, then gs and $succ\_inf$ will be true too. Therefore, by solving this equation system, i.e. finding M_{tc} and M_{fs}, we will derive tc and fs of the goal. In other words, it is necessary to find an interpretation suitable for both the internal factors and the external factors.

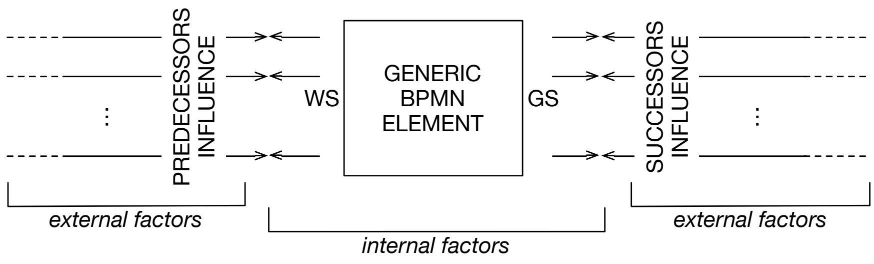

Fig. 10. Balancing the state conditions due to the connection of a BPMN element into a workflow structure.

4.1 Analysis of the Impact of Tasks

In BPMN, Task is a class of elements of different nature (the most commons are service task, send/receive task, and manual task). Besides the specific semantics, all these elements model a state-change due to an internal transition, i.e., they produce an endogenous evolution of the state of the world.

Concerning the internal factors, we are interested in observing the state immediately before and soon after their execution. The nature of a task characterizes who is responsible for performing the evolution (the software system, rather than a human or an organization). A task usually consumes or produces data or resources. This information is provided by connecting Data Objects to the task with Data Flow relationships. Moreover, send and receive tasks provide helpful information for identifying the state before and after their execution. The state is also influenced by the kind of expected termination: a normal termination versus an exceptional one due to some boundary condition (timer, escalation, error, cancel, compensation and signal).

If e is a Task, the analysis has to consider possible Input and Output Data Objects, Incoming and Outgoing Messages and Boundary Conditions. The general model is:

$$\begin{cases} ws(e) = data\_in(e) \land mess\_in(e) \\ gs(e) = (data\_out(e) \land mess\_out(e)) \oplus \langle boundary\rangle\_termination(e) \end{cases} \tag{3}$$

the waiting state is the logical and of expected data input and incoming messages, whereas the final state is an exclusive or between two cases: the normal termination ($(data\_out(e) \land mess\_out(e))$) and the exceptional termination ($\langle boundary\rangle\_termination$), expression that consider all the alternative termination conditions of the task.

Example 4.8. The *Collect Votes* task of Figure 8 takes the issue_list as input and it waits for votes from members thus to produce an issues_votes artifact. Let us suppose to mofify Figure 8 to add a boundary error event for capturing network problems: in this case $termination(collect\_votes) = error(collect\_votes)$. The waited state and generated state would be:

$$ws(collect\_votes) = in\_vote(issue\_list) \land received(vote)$$

$$gs(collect\_votes) = available(issue\_votes) \oplus error\_termination(collect\_votes)$$

It is worth noting that, frequently, tasks are modeled without explicitly specifying input/output data and messages. Even if this style of modelling violates the principle of information-centric process models [44], sometimes this may be an obligatory choice. An example is the *Moderate Conference* task in Figure 1, because moderating a conference is a task that does not produce any kind of artifact. Therefore, even if this modeling style is not in line with our approach (see discussion in Section 6), this approach is able to deal with underspecified tasks by using the normal termination predicate ($fs(e) = done(\langle activity \rangle)$). In the specific example, $fs(moderate\_conference) = done(moderate\_conference)$.

4.2 Analysis of the Impact of Events

An Event is something that "happens" during the course of a Process. Even if events are instantaneous, our semantics allow modeling an event has occurred. Indeed, catching or throwing an event is a change of the global state of the workflow allowing goals to deal with messages, signals and timers.

Beside start/end events, the BPMN notation mainly distinguishes two categories of intermediate events: elements that catch and elements that throw events. Messages are a specific category of events: an incoming message is captured by a catch message event, whereas sending a message may be executed through a throw message event. Other kinds of event are internal signals, errors, compensation, escalation and so on.

The Event Catch element corresponds to the action of catching a trigger, i.e. waiting for a signal or specific state of affairs occurs. This kind of elements produces a state change: 'a message is produced' or 'a trigger is thrown', respectively represented as $mess\_out(e)$ and $event\_out(e)$. Therefore, for these elements, we can calculate fs.

Conversely, the Event Throw produces an event, i.e. it raises a signal that is, generally, consumed within the scope of the same workflow for internal purposes (like signaling an error, requesting a compensation action, starting the escalation of the process, and so on). This kind of elements produces an (instantaneous) state change: 'a message is arrived' or 'a trigger has been caught', respectively represented as $mess\_in(e)$ and $event\_in(e)$. Therefore, for these elements, we can calculate ws.

Generalizing, if e is an event element, the analysis has to consider possible Incoming and Outgoing Messages and generic Caught/Thrown Events. The general model is:

$$\begin{cases} ws(e) = mess\_in(e) \land event\_in(e) \\ gs(e) = mess\_out(e) \land event\_out(e) \end{cases} \tag{4}$$

The result will appear different according to the kind of event element we are modeling. For instance:

- Message Catching: $ws(e) = message\_received(\ldots)$ and $gs(e) =< unspecified >$
- Timer Catching: $ws(e) = timer\_event(\ldots)$ and $gs(e) =< unspecified >$
- Error Catching: $ws(e) = error(\ldots)$ and $gs(e) =< unspecified >$
- Message Throwing: $ws(e) =< unspecified >$ and $gs(e) = message\_sent(\ldots)$
- Escalation Throwing: $ws(e) =< unspecified >$ and $gs(e) = escalation(\ldots)$

Example 4.9. Continuing the example of the email-voting process, Table 2 summarizes the waited state and the generated state for all the tasks and the events of the process.

4.3 Analysis of the Impact of Gateways

Gateways are used to control how Sequence Flows interact as they converge and diverge within a process. The BPMN specification proposes several categories of gateways, but for what concerns the internal factors, they do not directly affect the state of the world. Indeed, managing the control flows has not a direct impact on data, messages, and events, thus, according to the definition of waited state and generated state, if e is a gateway, we may conclude $ws(e) = gs(e) =< unspecified >$.

Conversely, the different categories of gateways with significant differences to the external factors, as demonstarted in the next section.

Table 2. Email Voting: waited and generated states.

Element Name	Waited State	Generated State
Determine Issues	$received(issue\_voting\_request)$	$initial(issue\_list)$
Announce Issues for Discussion	$initial(issue\_list)$	$in\_discussion(issue\_list) \wedge sent(issue\_announcement)$
Check Calendar for Conference	$< unspecified >$	$done(check\_calendar\_for\_conference)$
Conference Time (timer event)	$< unspecified >$	$at(conference\_time)$
Moderate Conference	$< unspecified >$	$done(moderate\_conference)$
6 days (timer event)	$< unspecified >$	$after(days(6), done(announce\_issues\_for\_discussion \wedge sent(issue\_announcement))$
Announce Discussion Deadline	$< unspecified >$	$sent(dedline\_warning)$
Moderate e-Mail Discussion	$< unspecified >$	$done(moderate\_email\_discussion) \oplus after(days(7), done(announce\_issues\_for\_discussion \wedge sent(issue\_announcement))$
Evaluate Discussion Progress	$in\_discussion(issue\_list)$	$ready(issue\_list)$
Announce Issues for Vote	$ready(issue\_list)$	$in\_vote(issue\_list) \wedge sent(vote\_announcement)$
Collect Votes	$in\_vote(issue\_list) \wedge received(vote)$	$available(issue\_votes)$
Announce Results	$available(issue\_votes)$	$final(issue\_votes) \wedge sent(vote\_results)$

4.4 Analysis of Predecessors

Predecessors and successors influence the state at input/out ports of a BPMN element. Activities and events directly affect the state of the world, whereas gateways act by combining/propagating the state (from input to output and vice-versa).

Calculating the predecessors' influence ($pre\_inf(e)$) consists in observing **which state is expected at the input ports of** e.

The analysis consists of observing each of the predecessors $Pre(e)$, thus evaluating how it propagates the state towards one of e input ports. If the predecessor is a task or an event, then its contribution is directly the generated state (see element k_1 in Figure 11). Differently, a gateway (that is a state-propagating element) produces a contribution that depends on the state at the input ports. This is graphically represented in Figure 11.

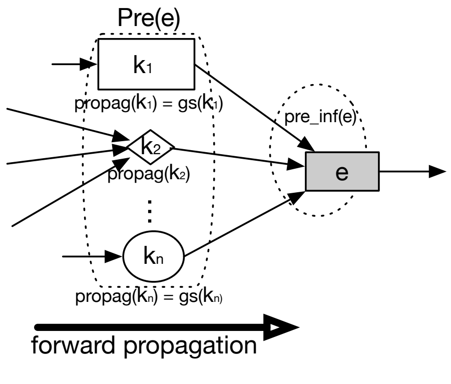

Fig. 11. Example of Back to Forward Propagation in which $pre\_inf(e) = gs(k_1) \oplus propag(k_2) \oplus \ldots \oplus gs(k_n)$.

Generalizing, given a generic predecessor k, then $propag(k)$ describes the state observed at the output port, compared to the state observed at all the input ports. Denoting as $flow(k, e)$ the (optional) flow condition attached to the sequenceFlow that connects k to e, we can set:

$$pre\_inf(e) = \bigoplus_{\forall k \in Pre(e)} (propag(k) \wedge flow(k, e)) \tag{5}$$

where $propag(k)$ is defined recursively as follows:

$$propag(k) = \begin{cases} gs(k) & \text{activity, event} \\ \bigoplus_{\forall r \in Pre(k)} (propag(r) \wedge flow(r, k)) & \text{converging exclusive gateway} \\ \bigwedge_{\forall r \in Pre(k)} (propag(r) \wedge flow(r, k)) & \text{converging parallel gateway} \\ \bigvee_{\forall r \in Pre(k)} (propag(r) \wedge flow(r, k)) & \text{converging inclusive gateway} \\ propag(pre) \wedge flow(pre, k) & \text{diverging gateway} \end{cases} \tag{6}$$

It is worth noting that if k is a converging gateway, (i.e. a gateway used to merge and synchronize the control flow, previously forked by a diverging gateway), different types of gateway work differently.

- the Exclusive Gateway is used to create alternative paths within a process flow: only one of the paths can be taken. The decision depends on conditions attached to outgoing sequence flows. It is typically followed by a converging exclusive gateway used to merge the alternative paths. For the purpose of our analysis, only one of the branches has been active, therefore the state at the output is the XOR of the state at input ports.
- the Parallel Gateway is used to create parallel flows: all the paths must be taken, without checking any conditions. It is typically followed by a converging parallel gateway used to synchronize diverging paths. For the purpose of our analysis, all the branches are active, consequently, we use the AND operator.
- the Inclusive Gateway can be used to create an alternative but also parallel paths within a Process flow. In this case, conditions attached to outgoing sequence flows may overlap: the true evaluation of one condition does not exclude the evaluation of other conditions. Since each path is considered to be independent, all combinations of the paths may be taken. For the purpose of our analysis, it acts as an OR logical operator.
- the Event-based Gateway represents a branching point where the alternative paths that follow are based on events that occur. Similarly to the exclusive gateway, only one branch could be chosen, but the decision is made by another participant (via an external event). The first occurring event decides the path to follow. We follow the BPMN specification, in which the events are drawn separately from the gateway through some intermediate events. Figure 12 provides a fragment of a BPMN diagram for this kind of gateway. This pattern can be interpreted as follows: "if *Event A* happens, then either execute *Task 1* if *Event B* happens or *Task 2* if *Event C* happens." To the purpose of this analysis, we simulate the gateway's *external factors* by using the formula for a converging exclusive gateway. External events' conditions are rendered as outgoing flow conditions: *flow(r,k)*.
- the Parallel Event-based Gateway is similar to the event-bases gateway but, when the event in the configuration is triggered, the other ones are still valid. For the purpose of our analysis, we simulate this element as a combination of a converging parallel gateway where flows'conditions come from event elements.

If k is a diverging gateway (one input, many output ports), we are interested only in analyzing what happens when the port that is connected with e is active: consequently, the gateway works as a passthrough: the state at the target output is the same as the unique input, whatever the type (i.e.. exclusive, parallel, . . .). Therefore, the type of diverging gateway (i.e.. exclusive, parallel, . . .) has no impact on this kind of propagation.

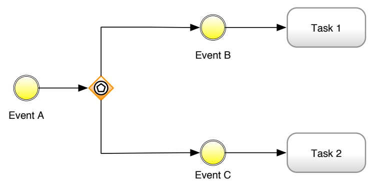

Fig. 12. Example of Event-Based Gateway. After Event A occurs, only one of Task1 and Task 2 will be executed. Task 1 will be selected if Event B occurs, Task 2 will be selected if Event C occurs.

Example 4.10. Going back to the examples of Figure 9, let us rename tasks for convenience:

$$\begin{cases} determine\_issues & \leftarrow A \\ announce\_issues\_for\_discussion & \leftarrow B \\ announce\_result & \leftarrow C \end{cases} \tag{7}$$

The task sequence of Figure 9.a is the trivial case in which $gs(A) = ws(B) = initial(issue\_list)$. There is no propagation of state. Therefore it is possible to assert

$$\begin{cases} succ\_inf(A) = ws(B) = initial(issue\_list) \\ pre\_inf(B) = gs(A) = initial(issue\_list) \end{cases} \tag{8}$$

The task sequence of Figure 9.b is a bit more complicated because $gs(determine\_issues) \neq ws(announce\_issues\_for\_discussion)$.

In this case the value of $pre\_inf(announce\_issues\_for\_discussion)$ is affected by the previous task:

$$\begin{cases} succ\_inf(A) = ws(B) = initial(issue\_list) \wedge available(priority) \\ pre\_inf(B) = gs(A) = initial(issue\_list) \wedge available(priority) \end{cases} \tag{9}$$

This third case (Figure 9.c) includes a gateway, and therefore it requires to inspect more predecessors when calculating the predecessors' influence to the *Announce Issues for Discussion* task. Applying formula 5, we consider both the two input sequenceFlows. Then, the presence of a gateway suggests continuing looking backward until the *Announce Results* task. Therefore, applying the appropriate formula in 6, the result is:

$$\begin{aligned} pre\_inf(B) = gs(A) \oplus gs(C) = & \\ & = (initial(issue\_list) \wedge available(priority)) \\ & \oplus final(issue\_votes) \end{aligned} \tag{10}$$

4.5 Analysis of Successors

The successors' influence ($succ\_inf$) is the result of observing **which state is expected at the output ports of** e. Focusing on e, it is necessary to observe each of the successors $Succ(e)$ and to evaluate the inverse state propagation (see Figure 13).

Given a generic BPMN element k, the $inv\_prop(k)$ describes the state observed at the input port (compared to to the state observed at all their output ports):

$$succ\_inf(e) = \bigoplus_{\forall k \in Succ(e))} (inv\_prop(k) \wedge flow(e,k)) \tag{11}$$

where *inv_prop* is defined as follows:

$$
inv\_prop(k) = \begin{cases} ws(k) & \text{activity, event} \\ \bigoplus_{\forall r \in Succ(k)} (inv\_prop(r) \wedge flow(k,r)) & \text{diverging exclusive gateway} \\ \bigvee_{\forall r \in Succ(k)} (inv\_prop(r) \wedge flow(k,r)) & \text{diverging inclusive gateway} \\ \bigwedge_{\forall r \in Succ(k)} (inv\_prop(r) \wedge flow(k,r)) & \text{diverging parallel gateway} \\ propag(pre) \wedge flow(pre,k) & \text{converging gateway} \end{cases} \tag{12}
$$

Example 4.11. Continuing the example of the email-voting process, Table 3 summarizes predecessor, successor influences of the elements of the process.

Table 3. Email Voting: predecessors and successors influence.

Element Name	**Predecessors' Influence**	**Successors' Influence**
Determine Issues	$< unspecified >$	$initial(issue\_list)$
Announce Issues for Discussion	$((final(issue\_votes) \wedge sent(vote\_results) \wedge \neg agreement(issue\_votes)) \oplus initial(issue\_list)$	$< unspecified >$
Check Calendar for Conference	$sent(issue\_announcement) \wedge in\_discussion(issue\_list)$	$scheduled(conference) \oplus (\neg scheduled(conference) \wedge in\_discussion(issue\_list))$
Moderate Conference	$at(conference\_time)$	$in\_discussion(issue\_list)$
Announce Discussion Deadline	$after(days(6), in\_discussion(issue\_list) \wedge sent(issue\_announcement))$	$in\_discussion(issue\_list)$
Moderate e-Mail Discussion	$sent(issue\_announcement) \wedge in\_discussion(issue\_list)$	$< unspecified >$
Evaluate Discussion Progress	$(done(moderate\_conference) \oplus (done(check\_calendar\_for\_conference) \wedge \neg scheduled(conference))) \wedge after(days(7), in\_discussion(issue\_list) \wedge sent(issue\_announcement)) \wedge sent(deadline\_warning)$	$ready(issue\_list)$
Announce Issues for Vote	$ready(issue\_list)$	$in\_vote(issue\_list) \wedge received(vote)$
Collected Votes	$in\_vote(issue\_list) \wedge sent(vote\_announcement)$	$available(issue\_votes)$
Announce Results	$available(issue\_votes)$	$agreement(issue\_votes) \oplus (initial(issue\_list) \wedge \neg agreement(issue\_votes))$

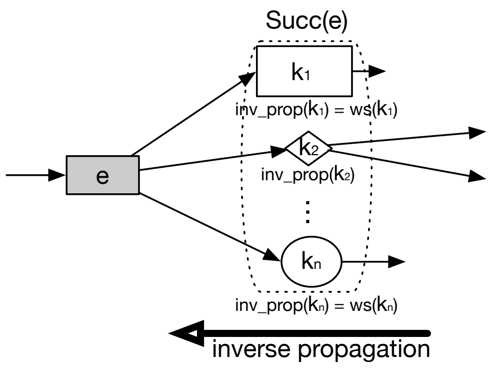

Fig. 13. Example of Inverse Propagation: $succ\_inf(e) = ws(k_1) \oplus inv\_prop(k_2) \oplus \ldots \oplus ws(k_n)$.

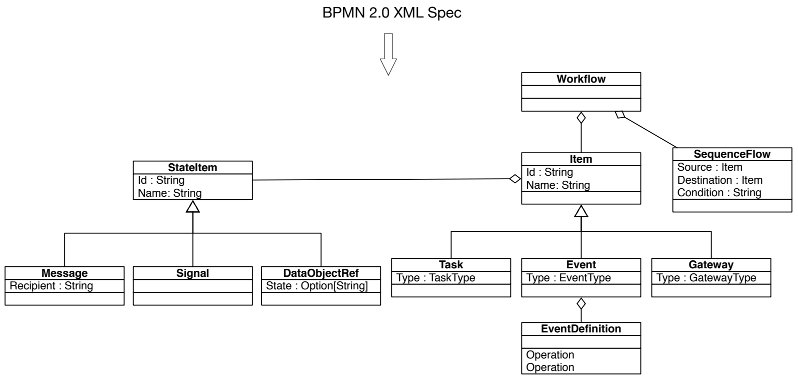

Fig. 14. First step of the procedure: from a BPMN 2.0 XML file, a parser generates objects according to the shown class diagram.

5 PROCEDURE FOR GOALS EXTRACTION

This section describes BPMN2Goals, a tool developed for automatically generating GoalSPEC from a BPMN workflow. The tool has been implemented in Java and Scala [52], and it is available as a web service[2]. The tool exploits an external library, Tweety [70], for building first-order logic expressions. This library allows instantiating Predicate objects and combining them by conjunction and disjunction operators.

The procedure for extracting goals works in two steps.

First step: Parsing (shown in Figure 14). From the BPMN file, the parser generates a preliminary graph where nodes represent generic BPMN elements, i.e., tasks, events, and gateways. Each node is annotated with an id and with all the related data objects and messages. Arcs represent sequenceFlows and are annotated with flow-conditions, when available.

The **Second step** – Goal Extraction – consists of visiting the graph and, for each node, calculating waited state, generated state, successor influence, and predecessor influence according to formulas provided in the previous section. It is worth noting that the procedure for calculating the forward/backward propagation for gateways requires recursion.

Given Formulas 3-4, and the considerations of Section 4.3, the procedures for generating the waited state and the generated state are straightforward. Algorithm 1 is an example of how to calculate the generated state for a task. It is composed of two steps: the first one uses output data objects and outgoing messages to build the 'normal termination condition'. For instance, looking at the *Announce Discussion Deadline* task, $normal\_termination = sent(deadline\_warning)$. When the task does not explicitly produce any data or message, the *done* predicate is used. For instance, for the *Moderate Conference* task, $normal\_termination = done(moderate\_conference)$.

The second step of Algorithm 1 looks at possible boundary terminations. If the task is provided with boundary events, it produces the task generated state (gs) as a xor disjunction between the normal termination and all the boundary terminations. For instance, the Moderate eMail Discussion task, has a timer boundary event, therefore Algorithm 1 has to consider a special termination $after(days(7), in\_discussion(issue\_list)$. In this specific case, the boundary

[2]The web service is available, with a front page, at http://aose.pa.icar.cnr.it:8080/BPMN2Goal/. It accepts BPMN files in the XMI format, according to the OMG specification. So far, the tool is not able to accept BPMN containing sub-processes. The source-code is available on GitHub, at https://github.com/icar-aose/bpmn2goalspec

Algorithm 1: generated_state(t) - Calculating the Generated State of a Task

```
Data: $t$ is a BPMN Task
Result: $gs$ is the generated state for $t$
begin
    $terms = \emptyset$ ;
    foreach $data\_object$ in outputDataObjects($t$) do
        if state($data\_object$) is defined then
            $terms$ += Predicate(state($data\_object$),Term(name($data\_object$))) ;
        else
            $terms$ += Predicate("available",Term(name($data\_object$))) ;
        end
    end
    foreach $message$ in outgoingMessages($t$) do
        $terms$ += Predicate("sent",Term(content($message$))) ;
    end
    if $terms != \emptyset$ then
        $normal\_termination$ = conjunction($terms$) ;
    else
        $normal\_termination$ = Predicate("done",Term(name($t$))) ;
    end
    $boundary\_terminations = \emptyset$ ;
    foreach $boundary\_event$ in boundaryTerminations($t$) do
        $boundary\_terminations = generated\_state(boundary\_event)$
    end
    $gs = xor\_disjunction(normal\_termination|boundary\_terminations)$ ;
end
```

termination is the only possible termination (there is not an outgoing sequence flow), hence $done(moderate\_email\_discussion)$ may be ignored: $gs = after(days(7), in\_discussion(issue\_list)$.

Conversely, the procedures for calculating the predecessor influence and the successor influence require visiting a portion of the graph by recursion (see Formulas 6 and 12).

Algorithm 2: predecessor_influence(e) - Calculating the Predecessors' Influence.

```
Data: $e$ is a generic BPMN element
Result: $predecessor\_influence(e)$ is the influence of $Prec(e)$ to input ports of $e$
begin
    $terms = \emptyset$ ;
    foreach $flow$ in incomingSeqFlow(e) do
        if condition($flow$) is defined then
            $terms+ = conjunction(direct\_propagation(source(flow)), Predicate(condition(flow)))$ ;
        else
            $terms+ = direct\_propagation(source(flow))$ ;
        end
    end
    $predecessor\_influence(e) = xor\_disjunction(terms)$
end
```

In practice, predecessor/successor influence is calculated by recursion where tasks and events represent the base cases and gateways the recursive case. An example of execution is represented in Figure 15 where the successors' influence of the Evaluate Discussion Progress task is calculated. Algorithm 2 starts by looking back to the unique task input flow (conducting to a converging parallel gateway). The flow has no attached condition, therefore the algorithm invokes Algorithm 3 to follow the tasks' predecessor.

When Algorithm 3 encounters a gateway, it recursively calls itself for each input flow. Conversely, when it encounters either a task or an event, it exits by returning gs, i.e. the generated state. For instance, in Figure 15, gs(check_calendar) = done(check_calendar), gs(moderate_conference) =

Algorithm 3: direct_propagation(g) - Calculating the Direct Propagation of a Gateway.

```
Data: $g$ is a gateway
Result: $direct\_propagation(g)$ is the direct propagation of $g$
begin
    $terms = \emptyset$ ;
    foreach $flow$ in incomingSeqFlow(g) do
        if condition($flow$) is defined then
            $terms$ += conjunction(direct_propagation($flow$),Predicate(condition($flow$))) ;
        else
            $terms$ += direct_propagation($flow$) ;
        end
    end
    if $type(g)$ is Exclusive then
        $direct\_propagation(f) = xor\_disjunction(terms)$ ;
    else if $type(g)$ is Parallel then
        $direct\_propagation(f) = conjunction(terms)$ ;
    else if $type(g)$ is Inclusive then
        $direct\_propagation(f) = disjunction(terms)$ ;
    end
end
```

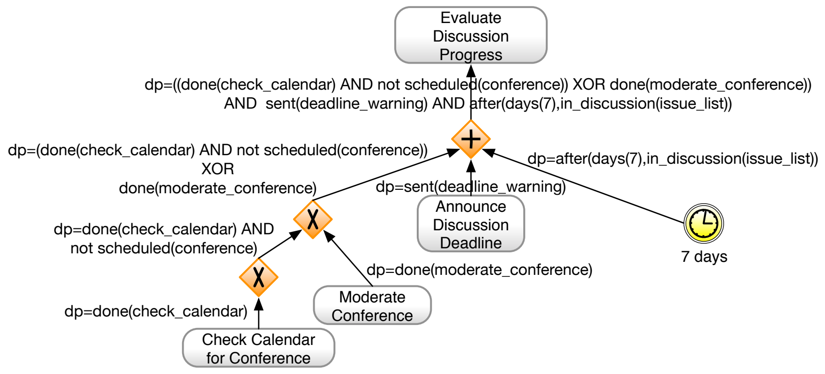
Evaluate Discussion Progress
dp=((done(check_calendar) AND not scheduled(conference)) XOR done(moderate_conference)) AND sent(deadline_warning) AND after(days(7),in_discussion(issue_list))
dp=(done(check_calendar) AND not scheduled(conference)) XOR done(moderate_conference)
dp=after(days(7),in_discussion(issue_list))
dp=sent(deadline_warning)
Announce Discussion Deadline
dp=done(check_calendar) AND not scheduled(conference)
7 days
dp=done(moderate_conference)
dp=done(check_calendar)
Moderate Conference
Check Calendar for Conference

Fig. 15. An example of successor influence calculated for the *Evaluate Discussion Progress* task of the email-voting example. The figure is annotated with values of dp, i.e. the direct_propagation calculated for the nearby element.

done(moderate_conference), gs(announce_discussion_deadline) = sent(deadline_warning) and gs(seven_days_timer) = after(days(7),in_discussion(issue_list)).

Going back to the recursion, the direct propagation of gateways may be calculated opportunely as a xor disjunction (for exclusive gateways) and conjunction (for the parallel gateway) as reported in Figure 15. Algorithm 2 terminates by returning: *((done(check_calendar) AND not scheduled(conference)) XOR done(moderate_conference)) AND sent(deadline_warning) AND after(days(7),in_discussion(issue_list))*.

Finally, Algorithm 4 puts togheter the pieces and assembly tasks' goals. The strategy is to adopt the simplest model that solves Equations 2, i.e. :

$$\begin{aligned} tc &= ws \wedge pre\_inf \\ fs &= gs \wedge succ\_inf \end{aligned} \tag{13}$$

Table 4 summarizes goals extracted for the email-voting example.

Algorithm 4: goal(task) - Calculating the goal conditions of a BPMN task.

```
Data: $task$ is a BPMNTask
Result: $goal(task)$ is the corresponding goal
1 begin
2     $tc = conjunction(genetrate\_ws(e), pre\_inf(e))$ ;
3     $fs = conjunction(genetrate\_gs(e), succ\_inf(e))$ ;
4     $goal(task) = \langle tc, fs \rangle$ ;
5 end
```

Table 4. Email Voting: goals triggering condition and final states.

Goal	Triggering Condition	Final State
Determine Issues	$received(issue\_vote\_list)$	$initial(issue\_list)$
Announce Issues for Discussion	$(((final(issue\_votes) \land sent(vote\_results)) \land \neg agreement(issue\_votes)) \oplus initial(issue\_list)) \land initial(issue\_list)$	$in\_discussion(issue\_list) \land sent(issue\_announcement)$
Check Calendar for Conference	$sent(issue\_announcement) \land in\_discussion(issue\_list)$	$done(check\_calendar\_for\_conference) \land (scheduled(conference) \oplus (notscheduled(conference) \land in\_discussion(issue\_list)))$
Moderate Conference	$at(conference\_time)$	$done(moderate\_conference) \land in\_discussion(issue\_list)$
Announce Discussion Deadline	$after(days(6), done(announce\_issues\_for\_discussion \land sent(issue\_announcement))$	$in\_discussion(issue\_list) \land sent(deadline\_warning)$
Moderate e-Mail Discussion	$sent(issue\_announcement) \land in\_discussion(issue\_list)$	$done(moderate\_email\_discussion) \oplus after(days(7), in\_discussion(issue\_list) \land sent(issue\_announcement))$
Evaluate Discussion Progress	$((done(moderate\_conference) \oplus (done(check\_calendar\_for\_conference) \land \neg scheduled(conference))) \land after(days(7), in\_discussion(issue\_list) \land sent(issue\_announcement)) \land sent(deadline\_warning)) \land in\_discussion(issue\_list)$	$ready(issue\_list)$
Announce Issues for Vote	$ready(issue\_list)$	$(in\_vote(issue\_list) \land sent(vote\_announcement)) \land (received(vote) \land in\_vote(issue\_list))$
Collected Votes	$(in\_vote(issue\_list) \land sent(vote\_announcement)) \land (received(vote) \land in\_vote(issue\_list))$	$available(issue\_votes)$
Announce Results	$available(issue\_votes)$	$(final(issue\_votes) \land sent(vote\_results)) \land (agreement(issue\_votes) \oplus (initial(issue\_list) \land \neg agreement(issue\_votes)))$

5.1 Analysis of Complexity

Here, we illustrate an empirical analysis of the complexity of the presented algorithms.

Experiment 1 Setup. We selected a set of processes from common repositories, by considering workflows of increasing size and intricacy. For each of these processes, we:

(1) selected six processes freely available as XML files[3] (their size ranges from 8 to 80 elements)
(2) measured the time-to-complete for calculating process goals,
(3) calculated the cyclomatic complexity by using ad-hoc metrics for business processes [20],
(4) used the regression to correlate time-to-complete and cyclomatic complexity,
(5) validated the linear model by calculating the p-value.

Experiment 1 Results: Figure 16 is the scatter plot highlighting the linear dependency between execution time and workflow complexity. The analysis shows high positive correlation between

[3]The XML description of the processes used for the analysis of complexity is available online at https://github.com/icar-aose/bpmn2goalspec/tree/scala_java/sanet-tool/process/analysis_of_complexity

these two variables ($cor = 0.932$). Moreover, the linear regression revealed a $p{-}value = 0.006 < 0.05$, i.e. a good statistical significance.

We can conclude the computational complexity of Algorithms 1-3 grows linearly with the complexity of the input workflow.

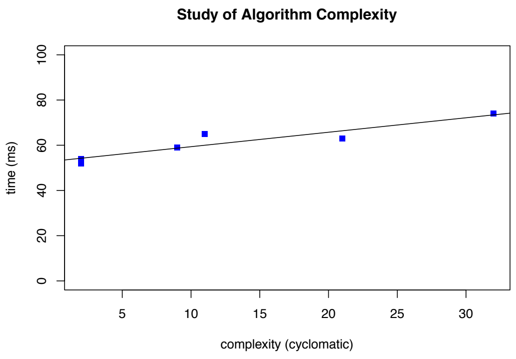

Fig. 16. The analysis of complexity reveals a linear dependency between the time-to-complete and the cyclomatic complexity of the workflow.

5.2 Reliability of the Generated Goals

We empirically evaluated the output of the algorithm with respect to the input BPMN, to understand if the extracted goals are suitable for describing the workflow from which they are generated.

Experiment 2 Setup. The rationale of the experiment is to check if the process of goal extraction produces loss of information. The first step for the evaluation is the selection of a subset of the processes to be used. We picked some workflows from those available online at the website www.bpmn.org:

- travel-booking, is a typical example used in many related works [31, 56, 60];
- order-fulfillment, selected because it presents a simple BPMN scheme, but its execution requires efficiency, flexibility, robustness, and adaptability [28, 46, 65];
- email-voting (also used as running example along the paper) selected because of its complexity.

These processes ensure coverage of different domains, different degrees of human involvement (from automatized entirely to human-centered processes), various kinds of failures (human delays, network or service errors, resource unavailability).

The second step is to revise the workflows to make the data-objects explicitly appearing in the BPMN scheme, and adequately connected to tasks with data-flows. This was a low-profile activity in which the involved researchers often recovered missing information directly from diagram annotations and the documentation.

Therefore, we generate a set of goals for the process.

The third step is to execute MUSA with these goals and a minimal set of services. In practice, we provide only a basic set of services with a one-to-one correspondence with the tasks of the original workflow: each task is realized by only one service.

Experiment 2 Results: in all the cases, MUSA was able to produce the same flow structure of the original workflow, where tasks are replaced by the corresponding services. We conclude that the extracted goals are consistent with the input workflow definition.

In addition, providing other services, MUSA generated many other solutions, allowing to change the operative workflow at run-time. This aspect represents the advantage of using goals rather than control flows, and it is discussed in the next section.

6 DISCUSSION

This section presents the strengths and limitations of automatically extracting goals from a BPMN.

The first part discusses the advantages of enacting a workflow by employing the extracted goals. The email-voting example is exploited here to demonstrate three scenarios of adaptation.

The second part summarizes the level of granularity for the proposed adaptation, by analyzing four categories of task replacement.

Finally, the third part of the section discusses some limitations and future works.

6.1 Goal-Defined Workflows Support Adaptation

This subsection analyzes some consequences of adopting a goal-based workflow description, relaxing the rigid constraints of the control flow structure.

In Section 4, goals are mainly generated 1) by considering the state of dataobjects, messages and events, and 2) by combining these states with logic connectives derived by gateways. The main consequence of relaxing the control flow structure is that the workflow management system has a certain degree of freedom in recombing available services to address the set of goals.

An example of this freedom is evident in the trivial sequence pattern shown in the upper side of Figure 17. According to the goal extraction procedure, the taskB triggering condition depends only on data1, whereas the taskC triggering condition depends on data2. Therefore, ignoring the original control flow structure, the sequence may be re-organized as a parallel between taskB and taskC (bottom side of Figure 17).

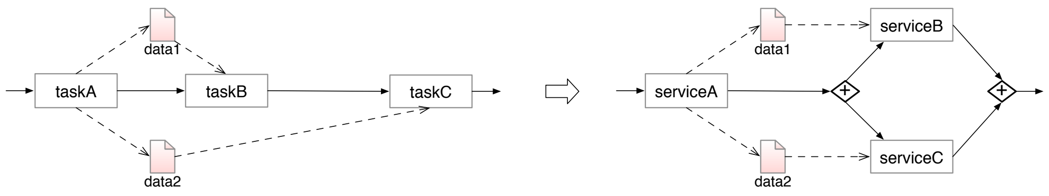

Fig. 17. Example of how the structure of the workflow may change due to the goal-based description.

The freedom of re-organizing the original workflow, according to available services is the base for enacting self-adaptive workflows. In the following, we exploit the email-voting example to discuss to which extent the generated goals support the self-adaptation ability.

Table 5. Email Voting: example of repository of services. The syntax is: [identifier]:[service_name]

SERVICE REPOSITORY	
s1: manual_issue_list	s9: share_perm_link
s2: collaborative_issue_extraction_via_CMS	s10: CMS_notification
s3: automatic_extraction	s11: moderate_CMS_discussion
s4: set_issue_list_state	s12: check_calendar
s5: send_email_notification	s13: moderate_conference
s6: send_text_notification	s14: moderate_email_discussion
s7: share_doc_via_email	s15: evaluate_discussion
s8: upload_on_cloud	s16: collect_votes

Available services are listed in Table 5. In practice, besides basic services, there are some redundancies. For instance, whereas the basic notification mechanism is to send an email (`send_email_` notification), the send_text_notification uses an instant messaging provider. There are some cloud-based services (upload_on_cloud and share_perm_link), and other ones based on a content management system (CMS). Finally, there is an automatic_extractor that uses NLP algorithms.

Table 6. Email Voting: example of run time association between services and goals (+ is the sequence operator, x is the exclusive choice operator, | is the parallel operator, => means 'addresses').

RUN-TIME MEANS-END ANALYSIS	
s1+s4 => Determine Issues	s5 => Announce Issues for Vote
s2 => Determine Issues AND Announce Issues for Discussion	s5 => Announce Results
s3 x (s1+s4) => Determine Issues	s5 | s6 => Announce Discussion Deadline
s7 => Announce Issues for Discussion	s5 | s6 => Announce Issues for Vote
s8+s9 => Announce Issues for Discussion	s5 | s6 => Announce Results
s12 => Check Calendar for Conference	s10 => Announce Discussion Deadline
s13 => Moderate Conference	s10 => Announce Issues for Vote
s14 => Moderate Discussion	s10 => Announce Results
s11 => Moderate Discussion	s15 => Evaluate Discussion
s5 => Announce Discussion Deadline	s16 => Collected Votes

When providing MUSA with the set of goals in Table 4 and the repository of services in Table 5, the middleware will discover some associations between services (atomic services or combination of services) and the goals. The result of this run-time means-end analysis is shown in Table 6. An example of binding is between the share_doc_via_email service (s7) and the [Announce Issues for Discussion] goal (see Table 6, first column, fourth row). There are also more interesting cases, such as i) the sequence of s1 and s4 for addressing the [Determine Issues] goal; ii) s2 addressing a couple of goals ([Determine Issues AND Announce Issues for Discussion]) and the parallel of s5 and s6, addressing [Announce Discussion Deadline].

The consequence of this analysis is the generation of a number of 'solutions' to the original set of goals. Five of these output workflows are represented in Figure 18.

The solution labeled as w1 is obtained by choosing basic services, i.e. the services with a one-to-one correspondence with original tasks. These services have been implemented to directly realize the related goal. For this reason, the workflow structure is maintained and the result is directly replacing original tasks with implementing services.

Adaptation case 1: unreachable service. Let us consider the workflow w1 is selected for the enactment. A typical situation of failure occurs when some of the services in a workflow is temporarily unavailable. For example, the share_doc_via_email service could appear to be offline when starting a new process instance. In this case, the workflow management system can notice this problem, by contacting all service providers involved in a solution, before choosing that. In the case of unavailability of service, the workflow management system will select another solution with the lowest risk of failures. In this specific example, the solution w2 is a good alternative to w1 because it does not contain any unavailable service. In this specific case, the strategy to share documents via email is replaced by adopting a collaborative platform (CMS) where to upload all the documents.

Adaptation case 2: workflow failure. Let us consider the workflow w1 is selected for the enactment, and an error happens during the execution of the share_doc_via_email service. In the repository, there is no direct replacement of this service. However, the other available solutions allow the workflow management system to react to the failure and to fix the workflow. The adaptation must, clearly, consider the current state of execution, including the history of services that have been successfully executed. For this reason, some alternative solutions are not suitable for continuing the process. It is the case of w2, adopting a CMS to share and manage the issue_list. Indeed, the

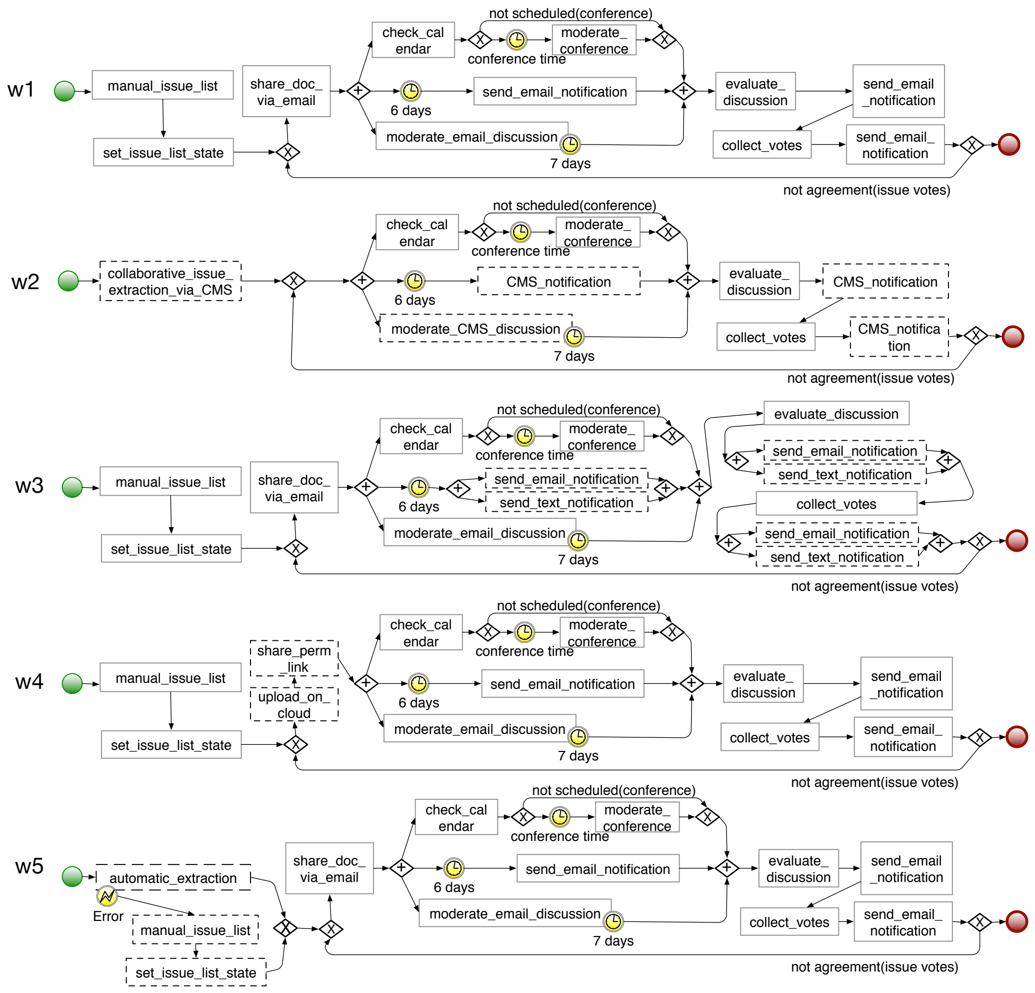

Fig. 18. The most relevant output workflows generated by MUSA for the email-voting example. Many other combinations –for instance automatic extraction and CMS based services — have been omitted for sake of space.

collaborative_issue_extraction_via_CMS service is not compatible with the current state (before the failure) in which the issue_list is a document in a file system. Conversely, this is exactly the case of w4, in which the file is uploaded on a shared cloud file system and then its permanent link is shared to all members.

Adaptation case 3: workflow evolution. A further case of adaptation occurs when new services enter the repository, later, providing some improved functionality. It is the case of the automatic_extraction service that uses NLP algorithms to automatically extract issues from a text. In w5, there is an example of that usage: the automatic_extraction has a boundary condition due to the high risk the NLP algorithms cannot recognize the content. In case of error, the manual_issue_list is selected. Clearly, after introducing the automatic_extraction service, w5 will always be preferable in place of w1. Indeed, even if the goal Determine Issues is satisfied in both cases, the quality of service of automatically extracting a text is higher than the one related to the manual extraction. So far, considering non-functional aspects is out the scope of this paper. A brief discussion is introduced, later, as future works.

6.2 Type of Task Replacements

Along with this paper, we stated goals are more flexible than flows of tasks, indeed goals allow breaking rigid structural constraints whereas the real objective is to produce some result. Breaking and reorganizing the structure, at run-time, is the core of the proposed approach to dynamic workflows.

In order to discuss the level of granularity in repairing a workflow, we have classified run-time task replacement in four categories. In the following, we discuss each of them by referring to the email-voting example and therefore considering the workflows w1-w5 from Figure 18.

We consider w1 is the current executing workflow, whereas the other ones could be selected as the consequence of a service failure or a change in the repository (as shown in Section 6.1).

Type A: 1-1 replacement.

Let us consider the [To Announce Issues for Discussion] goal. The basic workflow solution (w1 in Figure 18), uses the `send_email_notification` service (exploiting the standard SMTP protocol to send a message to a list of participants). Let us consider this service sometimes fails due to network problems or the recipient does not open the email timely. For these reasons, we have conveniently put another service in the repository: `send_text_notification` uses an instant messaging protocol (the precondition is the user provided her phone number). It is the direct alternative to be used when `send_email_notification` fails (as shown in Figure 19).

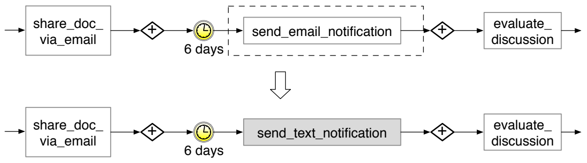

Fig. 19. Example of 1-1 replacement: on the upper side, the 'send_email_notification' service, that is directly replaced by the 'send_text_notification' service, highlighted in grey on the bottom side.

This situation is classified as *1-1 replacement*, because the malfunctioning is solved by replacing a single service A with an alternative service B. In literature, it is considered an ordinary situation of adaptation for service-oriented workflows.

Type B: 1-n replacement. Focusing on the `share_doc_via_email` service, used to share documents as email attachments, let us consider the case the customer has attached a video clip where to better explain some issues. This is embedded in the document to share with all voting members. In this case, it is possible the mail server will refuse to deliver the email because of restrictions about the size. Fortunately, the repository contains other two services, useful in cases like this:

- `upload_on_cloud` uses a cloud file storage where to upload a file;
- `share_perm_link` generates a permanent URL for the resource, easily sharable with cloud users.

The generated solution w4 (Figure 18) uses a sequence of these two services in order to replace the `share_doc_via_email`: it uses i) `upload_on_cloud` to make the video clip available from everywhere, and ii) the share_perm_link for sharing the permanent link to all the members (see Figure 20).

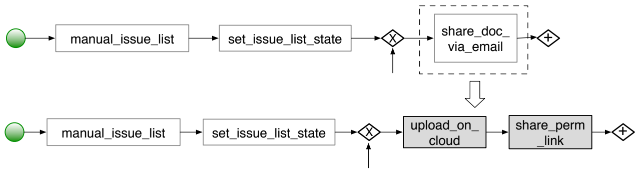

Fig. 20. Example of 1-n replacement: the share_doc_via_email service, on the upper side, is replaced by the sequence of upload_on_cloud and share_perm_link, on the bottom side.

This solution is classified as *1-n replacement* because the malfunctioning is solved by replacing a single service A with a composition of many services $B_1, B_2, \ldots B_n$.

Type C: m-1 replacement.

The [Determine Issue] goal, in w1 (see Figure 18) is addressed by two services: manual_issue_list supports the manager at manually extracting a list of issues from an informal document; and set_issue_list_state service generates a document and stores it in the file system of the voting manager to be shared with other members.

To reduce the burden of this manual activity, another strategy has been implemented that consists of exploiting a content management system where all the voting members can collaboratively help extracting issue_list. For this reason we have included a suite of services such as: collaborative_issue_extraction_via_CMS, CMS_notification and moderate_CMS_discussion.

In the solution w2 of Figure 18, the CMS is exploited rather than broadcasting emails or texts. Voting members are registered into the platform and are automatically notified by the content management system when new content is shared or an existing one is modified.

For this reason, the collaborative_issue_extraction_via_CMS service replaces the manual extraction, the document creation, and the document sharing: everything is done through the platform. Figure 21 highlights this specific kind of replacement.

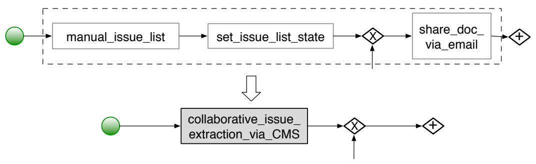

Fig. 21. Example of m-1 replacement: the triple 'manual_issue_list' and 'set_issue_list_state' and 'share_doc_via_email' (on the upper side) is replaced by the 'collaborative_issue_extraction_via_CMS' service, on the bottom side.

This solution is classified as *1-n replacement* because the malfunctioning (or the evolution) of the services $A_1, A_2 \ldots A_n$ is recovered by using an alternative service B.

Type D: m-n replacement. The automatic_extraction service uses an experimental algorithm based on the recognition of some keywords in the text, to automatically generate the issue_list document. However, this kind of services is risky to be used in a process, because the NLP algorithm fails in analyzing some kind of sentences. However, given this error is defined in the service interface, it is dealt as a boundary termination (in particular it is marked as error termination). The solution w5 in Figure 18 uses the manual extraction to compensate for these errors.

In this case, the goal based description of the workflow allowed a couple of services (see the left side of Figure 22) are replaced by three services (right side of Figure 22).

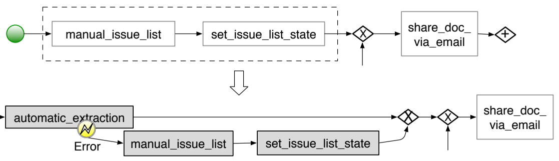

Fig. 22. Example of m-n replacement: the couple of services, in the upper side, are replaced by a triple of services, in the bottom side.

This case is classified as *m-n replacement* because the malfunctioning is solved by replacing a set of services $A_1, A_2 \ldots A_n$ with an alternative set $B_1, B_2, \ldots, B_m$.

6.3 Limitations and Future Works

In the last years, the approach has been used in several research projects and application domains [29, 60, 62–65]. Here, we discuss some limits we identified and propose some future works to improve the approach.

Implicit vs Explicit Goals. The literature contains several approaches to drive and support business analysts when defining business strategy through goal and organizational models [28, 28]. Goal analysis, even if supported by powerful methodologies [13] and tools[10, 74], remains a human activity in which rationality and creativity are fundamental abilities. Unfortunately, this practice is not yet standard in the industrial setting, where modeling workflow as a control structure is largely preferred. This work focuses on extracting goals, i.e. declarative entities, from the structure of a workflow. As shown in Sections 3–4, goals derive from explicitly modeled data objects and messages (that combined via logic connectives that derive from the control structure).

It is important to highlight the differences between goals from goal analysis (explicit goals) and automatically extracted goals (implicit goals).

Explicit goals may be expressed in either (semi-)formal [74] or natural English languages [13]; implicit goals are necessarily formal, expressed according to a precise formalism.

Explicit goals are often functional and non-functional (for example, in i* [81], there are, respectively, hard and soft goals); implicit goals basically define functional aspects. The unique exception is for implicit goals deriving from timer-events; these contain a proto-version of non-functional goals.

Explicit goals are often organized in hierarchies or networks [13, 81] (goal models) that highlight different level of abstractions and dependencies of various nature; implicit goals are flat, and, all of them are at the same level of abstraction, that is actually very operative, indeed, their objective is to enable changes in dynamic workflows.

Information-Centric Workflow vs Procedure Information-centric workflow [44] are business processes modeled with specific attention to the life cycles of information entities (documents, business objects, and artifacts.

In the proposed approach, explicitly modeling information entities (and their relationships with activities) is the key to discover implicit goals. To this purpose, the workflow model shall contain the most information about the resources to be used and how they are exchanged. This is necessary for enabling the algorithm to properly generate significant milestones and constraints to rule process execution. In general, this formal way to model workflows, including ontologies for specifying input and output data objects, is often encountered in literature [33].

Actually, the approach of Section 4 is robust with respect to some activities that are unrelated to any data objects and messages. Indeed, a done(<activity_name>) is always generated as the ultimate solution to generate a non-null state. For instance, the Moderate Conference task, in the email voting process, does not manipulate any data or produce any message: its generated state is done(moderate_conference), as reported in Table 2.

However, goals containing this kind of states are less interesting for adaptation, because they reduce the degree of freedom when replacing a malfunctioning service. Intuitively, to solve malfunctioning of a service producing a done(<activity>), the workflow management system will search for another service producing the same state: in other words, only 1-1 task replacements are possible in these cases.

A deeper analysis of this aspect is out the scope of this paper, but we are aware there is a direct relation between the quality of information entities described in the input workflow and the adaptation flexibility to recover failures and meet changes.

A future work concerns the automatic identification of relationships among data objects (for instance the concept of Data Dominance [44]). This could improve the granularity of goal description, for instance identifying implicit relationships among the extracted goals.

QoS and Temporal Factors Another weakness of the presented approach is that, focusing only on the functional dimension, it does not generate goals concerning quality aspects like accuracy, conformance to specification, and reliability.

Whereas almost all the goal-oriented frameworks support the difference between functional and non-functional goals [81], so far, the interrelation between control flow and quality is little understood [38].

This is a recognized limit of the current approach. In Section 6.1, we have analyzed a situation in which the same set of extracted goals would produce alternative workflow instances, that are functionally equivalent but different from a non-functional point of view. In order to allow the workflow management system to make a QoS-aware choice, it is necessary to include information about non-functional aspects.

A solution could be to integrate the classic BPMN model with other models describing quality aspects. An instance is the Process Root Cause Analysis [38] that combines goal-oriented and activity-oriented process modeling for an explicit description of quality aspects of a process.

Moreover, services should be annotated too with attributes about their individual impact towards these QoS factors. For instance, it is necessary to specify the automatic_extraction service has a better impact over personal productivity than the manual_issue_list service.

Among all the quality aspects, the management of time is one of the most important. Indeed, time is one of the most critical aspects of business processes. BPMN has provided specific elements to handle timer events, but the management of time, during the adaptation, remains an open challenge.

The proposed approach is to handle time as a relationship between predicates. The timer event is translated into a second-order predicate like after(<delay>,<event>). This is a clear weakness of the current approach in dealing with time expressions.

In particular, this breaks the first order logic structure we have chosen for all the goals. We are studying a temporal extension based on some metric temporal language [14]. This kind of languages has the strength to be highly expressive for dealing with durations, time distance between tasks, and deadlines.

However, introducing temporal goals would introduce additional aspects to be considered during the adaptation process. Extending the language to the temporal asset may add a further dimension of complexity to the problem of workflow modification [67]. Time-related decision making is highly dynamic and complex. So far, we have been studying how to support temporal logic and time management during the reconfiguration.

7 RELATED WORKS

It is often assumed that a business process or application is associated with some explicit business goal definition that, unfortunately, is usually not available from an industrial perspective [68]. Very often, real goals remain implicit in the workflow model. This work faces the problem of automatically deriving implicit goals from a traditional workflow definition by the analysis of data objects manifested in persistent documents. Such goal extraction would enable and automatize the dynamic workflow generation process presented in previous sections.

This section illustrates works about the extraction of information from an existing business process model. We start by looking at the extraction of states, by comparing them with our semantic described in Section 3. In the second part, we mention some works about the extraction of business rules and goals, and compare them with the core of our approach, described in Section 4.

7.1 Extracting States from a Business Process

An extensive literature exists on *extracting states* from a workflow model with the main purpose of verification and simulation. There are two main streams: i) focus on control flow and ii) consideration of the whole workflow (control and data flow).

The most used instrument to formalize states of a workflow is the Petri net because it is based on a strong mathematical ground. They are mainly used to check workflow properties like soundness and related issues such as liveness, boundedness, safeness, deadlock, livelock, and dead activity. A pioneer in this research area is [72] that introduces the concept of workflow-net. These can be directly derived through a direct mapping between control-flow structures and Petri net patterns [30]. Places represent the state of the execution of a part of the workflow, and transitions provide a description of how the state of the workflow may evolve [25]. Petri net can be complemented with logical formulas [11] and process algebra [80] for increasing their expressiveness. All the works, in this category, basically exploit the structure of the business process, not considering important elements such as events, messages and, above all, data objects. States, extracted with these approaches, describe only precedence and control flow constraints, that are not particularly suitable for the adaptation purposes we intend to pursue in our work.

Conversely, a contribution to our objective comes from [8] that suggests incorporating data access into the formalization of the state, thus enabling data-aware verification. They adopt linear temporal logic to formalize and checking past, present and future states of the Petri net. In Section 3 we adopt a similar approach for identifying relevant states of the workflow. However, we relax control flow constraints and we adopt a simpler representation of data states. Temporal reasoning is briefly discussed on Section 6.

In [4], authors use net-theoretical concepts to derive a statechart in which each node abstracts the global state of the workflow. The main limitation in the use of this artifact in our work would be they do not provide a semantic description of the state contained in each node. In practice, the focus is on state transitions and not on the description of the state itself.

Another contribution to our work is [75], where authors propose an approach to detect data-flow errors in BPMN 2.0 process models through the use of data-flow anti-patterns to describe anomalies of the data flow. To this aim, they identify which constructs in a BPMN must be explicit for a correct data description. Authors highlight the graphical representation may contain ambiguities, thus other elements must be specified in the XML model, for example, task's DataInput/Output or information about mandatory/optional data.

Kumaran et al. [44] highlight the importance of information-centric process modeling, where the key is to discover the right information entities that describe the business process. They also formalize the relation of domination (e1 dominates e2 if, for every activity that uses e2 as an input/output, e1 is also used as an input/output): a dominant entity can be treated as a container for the dominated ones. Including the analysis of dominance may be relevant, in our work, to start identification of goal relationships.

Cabanillas et al. [18] define an algorithm to derive object life cycles from business process models with object flows. The algorithm only deals with sequential process models and generates object life cycles by adopting Petri nets and the reachability graph. Similarly, Eshuis and Van Gorp [32] define an approach for synthesizing hierarchical state machines also considering parallelism in the business process model. The approach of making explicit the life cycle contained in the process is close to the semantic definition that we propose in Section 3. With respect to their state-chart generation algorithms, we also look at received/sent messages, caught/thrown events, termination events and boundary conditions, they do not consider.

7.2 Extracting Rules from a Business Process

The research area concerning the *extraction of active rules* puts emphasis on workflow events rather than on states, moving a step towards goal-based workflows. Active rules follow the event-condition-action (ECA) paradigm and represent the reactive engine of a workflow management system. Events refer to changes of the database content, conditions are queries whereas actions are similar to workflow tasks.

Several works face the problem of automatically deriving rules that frequently are embedded within the business process itself. This research area has the main objective to integrate workflow and database systems coping with the workflow scalability problem [27, 79] and improving task distribution and parallelization [17]. Approaches like [9, 48] derive rules directly from workflow control structure patterns. This produces trivial rules like 'when EndOf(taskA) then Start.Activity(taskB)'. In [21], authors propose an approach for deriving rules by identifying a relevant set of events in the workflow (tasks with preconditions, exceptions, ...) and then instantiating the corresponding rule template. In [48], the authors propose to automatically derive rules, by inferring them by an underlying context-aware system, exploiting available knowledge about the user's situation in order to adapt their functionalities and to provide an automated service personalization.

The common format of an active rule is '*on event if condition do action*' that is similar to the concept of goals. However, the action part contains a sequence of commands in imperative format. In practice, a rule may be seen as a design-time tie-up of a goal and a task/service. In our approach, we want to overcome this design-time relationship and, above all, we want to apply a certain degree of flexibility that rules like '*when EndOf(taskA) then Start.Activity(taskB)*' cannot provide.

7.3 Extracting Goals from a Business Process

In this paper, we are specifically interested in *extracting goals*. Indeed, despite the general agreement, a business process is associated with some explicit business goal definition, unfortunately, they are usually not available from an industrial perspective.

In [82], the authors propose to explicitly model the intentional dimension of workflow through the i^* notation [81]. When this is not possible, they suggest using some reverse engineering algorithm, supported with plan recognition techniques, that could be used to uncover underlying intentions.

This extraction of goals has been faced in [28], in which authors describe guidelines to (manually) derive requirements models from organizational models through goal modeling. The links between the domains are in the BPMN notation, which can be understood by all the stakeholders, and in a goal tree that is derived from business process models.

We agree with the claim, in [68], that real goals of a business often remain implicit in the workflow models. Our focus is rather in deducting some implicit goals of a business process.

Therefore, our focus is on the extraction of implicit goals by looking at states of persistent data objects and messages in the workflow model. For example, in a travel reservation workflow where the explicit goal is to perfectly organize the travel, we can realize an implicit goal addressing a final state about the creation of the required travel documents. The extraction of i^*-like goals [81] (i.e. the real goals of the workflow) is still an open challenge.

8 CONCLUSION

The unpredictability of business processes requires that workflow systems own the ability to adapt to the changing environment dynamically. To the purpose, a research community is working in enhancing run-time flexibility to the process by enabling run-time modifications. Given the effectiveness of an information system is judged according to how well it meets the needs of the organization it serves, we focused on initiatives that use goals for specifying operative milestones and constraints.

This work has proposed an automatic approach for extracting implicit goals from a BPMN to be used for supporting the enactment of a dynamic workflow. The idea is to extract goals by looking at how the state of the workflow changes and combining these states according to logical connectives deriving from the workflow gateways.

In concrete, a tool has been developed for testing the approach. It has been used together with MUSA, a middleware for self-adaptation to test the reliability of the extracted goals. The results are promising: goals support the adaptation at run-time of the workflow in case of service failure, workflow failure or workflow evolution. The study of the granularity of adaptation revealed the workflow management system could totally change the original structure of the workflow, for instance converting a sequence of tasks into the parallel of those. Moreover, the approach covers four types of run-time task replacements: 1-1, 1-n, m-1, m-n.

In the future, we will consider how to extend the expressiveness of the language used to represent conditions and goals. A promising candidate seems the metric interval temporal logic, perfect for representing task durations, time distance between tasks, and deadlines.

REFERENCES

[1] Dhaminda B Abeywickrama, Nicola Bicocchi, and Franco Zambonelli. 2012. SOTA: Towards a general model for self-adaptive systems. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on.* IEEE, 48–53.

[2] Michael Adams. 2010. Dynamic workflow. In *Modern Business Process Automation.* Springer, 123–145.

[3] Rohit Aggarwal, Kunal Verma, John Miller, and William Milnor. 2004. Constraint driven web service composition in METEOR-S. In *Services Computing, 2004.(SCC 2004). Proceedings. 2004 IEEE International Conference on.* IEEE, 23–30.

[4] Alessandra Agostini and Giorgio De Michelis. 2000. Improving flexibility of workflow management systems. In *Business Process Management.* Springer, 218–234.

[5] G. H. Alférez, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz. 2014. Dynamic Adaptation of Service Compositions with Variability Models. *J. Syst. Softw.* 91 (May 2014), 24–47. https://doi.org/10.1016/j.jss.2013.06.034

[6] Jim Amsden. 2012. Capturing requirements with Business Motivation Model, IBM Rational RequisitePro, and IBM Rational Software Modeler.

[7] Danilo Ardagna, Marco Comuzzi, Enrico Mussi, Barbara Pernici, and Pierluigi Plebani. 2007. Paws: A framework for executing adaptive web-service processes. *IEEE software* 24, 6 (2007), 39.

[8] Ahmed Awad, Matthias Weidlich, and Mathias Weske. 2009. Specification, verification and explanation of violation for data aware compliance rules. In *Service-Oriented Computing*. Springer, 500–515.

[9] Joonsoo Bae, Hyerim Bae, Suk-Ho Kang, and Yeongho Kim. 2004. Automatic control of workflow processes using ECA rules. *IEEE transactions on knowledge and data engineering* 16, 8 (2004), 1010–1023.

[10] Luciano Baresi and Liliana Pasquale. 2010. Adaptive goals for self-adaptive service compositions. In *Web Services (ICWS), 2010 IEEE international conference on*. IEEE, 353–360.

[11] Henry H Bi and J Leon Zhao. 2004. Applying propositional logic to workflow verification. *Information Technology and Management* 5, 3-4 (2004), 293–318.

[12] Julien Bidot, Christos Goumopoulos, and Ioannis Calemis. 2011. Using ai planning and late binding for managing service workflows in intelligent environments. In *2011 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 156–163.

[13] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. 2004. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems* 8, 3 (2004), 203–236.

[14] Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, Arthur Milchior, and Benjamin Monmege. 2018. Efficient algorithms and tools for MITL model-checking and synthesis. In *2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 180–184.

[15] Greg Brown, Betty HC Cheng, Heather Goldsby, and Ji Zhang. 2006. Goal-oriented specification of adaptation requirements engineering in adaptive systems. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*. ACM, 23–29.

[16] PA Buhler and JM Vidal. 2005. Towards adaptive workflow enactment using multiagent systems. *Information Technology and Management* 6, 1 (2005), 61–87.

[17] Christoph Bussler and Stefan Jablonski. 1994. Implementing agent coordination for workflow management systems using active database systems. In *Research Issues in Data Engineering, 1994. Active Database Systems. Proceedings Fourth International Workshop on*. IEEE, 53–59.

[18] Cristina Cabanillas, Manuel Resinas, Antonio Ruiz-Cortés, and Ahmed Awad. 2011. Automatic generation of a data-centered view of business processes. In *International Conference on Advanced Information Systems Engineering*. Springer, 352–366.

[19] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. 2005. QoS-aware replanning of composite web services. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 121–129.

[20] Jorge Cardoso. 2005. How to measure the control-flow complexity of web processes and workflows. *Workflow handbook* 2005 (2005), 199–212.

[21] Fabio Casati, Stefano Ceri, Barbara Pernici, and Giuseppe Pozzi. 1996. Deriving active rules for workflow enactment. In *International Conference on Database and Expert Systems Applications*. Springer, 94–115.

[22] Fabio Casati, Stefano Ceri, Barbara Pernici, and Giuseppe Pozzi. 1998. Workflow evolution. *Data & Knowledge Engineering* 24, 3 (1998), 211–238.

[23] Fabio Casati, Ski Ilnicki, LiJie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. 2000. Adaptive and dynamic service composition in eFlow. In *International Conference on Advanced Information Systems Engineering*. Springer, 13–31.

[24] Michele Chinosi and Alberto Trombetta. 2012. BPMN: An introduction to the standard. *Computer Standards & Interfaces* 34, 1 (2012), 124–134.

[25] Piotr Chrzastowski-Wachtel, Boualem Benatallah, Rachid Hamadi, Milton O'Dell, and Adi Susanto. 2003. A top-down petri net-based approach for dynamic workflow modeling. In *International Conference on Business Process Management*. Springer, 336–353.

[26] Massimiliano Colombo, Elisabetta Di Nitto, and Marco Mauri. 2006. Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In *International Conference on Service-Oriented Computing*. Springer, 191–202.

[27] Umeshwar Dayal, Meichun Hsu, and Rivka Ladin. 1990. Organizing long-running activities with triggers and transactions. In *ACM SIGMOD Record*, Vol. 19. ACM, 204–214.

[28] Jose Luis De la Vara González and J Sanchez Diaz. 2007. Business process-driven requirements engineering: a goal-based approach. In *Proceedings of the 8th Workshop on Business Process Modeling*. Citeseer.

[29] Claudia Di Napoli, Marco Valentino, Luca Sabatucci, and Massimo Cossentino. 2018. Adaptive Workflows of Home-Care Services. In *2018 IEEE 27th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE, 3–8.

[30] Remco M Dijkman, Marlon Dumas, and Chun Ouyang. 2008. Semantics and analysis of business process models in BPMN. *Information and Software Technology* 50, 12 (2008), 1281–1294.

[31] Ahmed Elkhodary, Naeem Esfahani, and Sam Malek. 2010. FUSION: a framework for engineering self-tuning self-adaptive software systems. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 7–16.

[32] Rik Eshuis and Pieter Van Gorp. 2012. Synthesizing object life cycles from business process models. In *International Conference on Conceptual Modeling*. Springer, 307–320.

[33] Chiara Di Francescomarino, Chiara Ghidini, Marco Rospocher, Luciano Serafini, and Paolo Tonella. 2011. A framework for the collaborative specification of semantically annotated business processes. *Journal of Software Maintenance and Evolution: Research and Practice* 23, 4 (2011), 261–295.

[34] Christos Goumopoulos, Ioannis Calemis, and Achilles Kameas. 2010. Deployment of adaptive workflows in intelligent environments. In *Intelligent Environments (IE), 2010 Sixth International Conference on*. IEEE, 197–202.

[35] Rachid Hamadi and Boualem Benatallah. 2004. Recovery nets: Towards self-adaptive workflow systems. In *International Conference on Web Information Systems Engineering*. Springer, 439–453.

[36] Yanbo Han, Amit Sheth, and Christoph Bussler. 1998. A taxonomy of adaptive workflow management. In *Workshop of the 1998 ACM Conference on Computer Supported Cooperative Work*. 1–11.

[37] WfMC Workflow Handbook. 2000. Workflow Management Coalition. *Lighthouse Point, Florida, USA* (2000).

[38] Mitra Heravizadeh, Jan Mendling, and Michael Rosemann. 2008. Dimensions of business processes quality (QoBP). In *International Conference on Business Process Management*. Springer, 80–91.

[39] Gabriel Hermosillo, Lionel Seinturier, and Laurence Duchien. 2010. Creating context-adaptive business processes. In *International Conference on Service-Oriented Computing*. Springer, 228–242.

[40] Thomas T Hildebrandt and Raghava Rao Mukkamala. 2011. Declarative event-based workflow as distributed dynamic condition response graphs. *arXiv preprint arXiv:1110.4161* (2011).

[41] Peter J Kammer, Gregory Alan Bolcer, Richard N Taylor, Arthur S Hitomi, and Mark Bergman. 2000. Techniques for supporting dynamic and adaptive workflow. *Computer Supported Cooperative Work (CSCW)* 9, 3-4 (2000), 269–292.

[42] Raman Kazhamiakin, Marco Pistore, and Marco Roveri. 2004. A framework for integrating business processes and business requirements. In *Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International*. IEEE, 9–20.

[43] Peter Kueng and Peter Kawalek. 1997. Goal-based business process models: creation and evaluation. *Business Process Management Journal* 3, 1 (1997), 17–38.

[44] Santhosh Kumaran, Rong Liu, and Frederick Y Wu. 2008. On the duality of information-centric and activity-centric models of business processes. In *International Conference on Advanced Information Systems Engineering*. Springer, 32–47.

[45] Sotirios Liaskos, Shakil M Khan, Marin Litoiu, Marina Daoud Jungblut, Vyacheslav Rogozhkin, and John Mylopoulos. 2012. Behavioral adaptation of information systems through goal models. *Information Systems* 37, 8 (2012), 767–783.

[46] Fu-Ren Lin and Michael J Shaw. 1998. Reengineering the order fulfillment process in supply chain networks. *International Journal of Flexible Manufacturing Systems* 10, 3 (1998), 197–229.

[47] Ruopeng Lu, Shazia Sadiq, and Guido Governatori. 2009. On managing business processes variants. *Data & Knowledge Engineering* 68, 7 (2009), 642–664.

[48] Jose F Mejia Bernal, Paolo Falcarin, Maurizio Morisio, and Jia Dai. 2010. Dynamic context-aware business process: a rule-based approach supported by pattern identification. In *Proceedings of the 2010 ACM Symposium on Applied Computing*. ACM, 470–474.

[49] Milan Milanovic, Dragan Gasevic, and Luis Rocha. 2011. Modeling flexible business processes with business rule patterns. In *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*. IEEE, 65–74.

[50] M Morandini, L Penserini, and Anna Perini. 2008. 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.

[51] Object Management Group (OMG). 2010. Business Process Model and Notation (BPMN 2.0) by Example. Available online at https://www.omg.org/cgi-bin/doc?dtc/10-06-02.pdf.

[52] Martin Odersky, Lex Spoon, and Bill Venners. 2008. *Programming in scala*. Artima Inc.

[53] Martyn A Ould and MA Ould. 1995. *Business Processes: Modelling and analysis for re-engineering and improvement*. Vol. 598. Wiley Chichester.

[54] Maja Pesic and Wil MP Van der Aalst. 2006. A declarative approach for flexible business processes management. In *International conference on business process management*. Springer, 169–180.

[55] Marco Pistore, Annapaola Marconi, Piergiorgio Bertoli, and Paolo Traverso. 2005. Automated composition of web services by planning at the knowledge level. In *IJCAI*, Vol. 19. 1252–1259.

[56] Nauman A Qureshi, Anna Perini, Neil A Ernst, and John Mylopoulos. 2010. Towards a continuous requirements engineering framework for self-adaptive systems. In *Requirements@ Run. Time (RE@ RunTime), 2010 First International Workshop on*. IEEE, 9–16.

[57] Manfred Reichert and Peter Dadam. 1998. ADEPT flex—supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems* 10, 2 (1998), 93–129.

[58] Manfred Reichert, Alena Hallerbach, and Thomas Bauer. 2015. Lifecycle management of business process variants. In *Handbook on Business Process Management 1*. Springer, 251–278.

[59] Manfred Reichert and Barbara Weber. 2012. *Enabling flexibility in process-aware information systems: challenges, methods, technologies*. Springer Science & Business Media.

[60] L Sabatucci, A Cavaleri, and M Cossentino. 2016. Adopting a Middleware for Self-adaptation in the Development of a Smart Travel System. In *Intelligent Interactive Multimedia Systems and Services 2016*. Springer, 671–681.

[61] Luca Sabatucci and Massimo Cossentino. 2015. From means-end analysis to proactive means-end reasoning. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 2–12.

[62] Luca Sabatucci and Massimo Cossentino. 2017. Self-adaptive smart spaces by proactive means–end reasoning. *Journal of Reliable Intelligent Environments* 3, 3 (2017), 159–175.

[63] Luca Sabatucci, Massimo Cossentino, Giada De Simone, and Salvatore Lopes. 2018. Self-Reconfiguration of Shipboard Power Systems. In *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE, 124–129.

[64] Luca Sabatucci, Carmelo Lodato, Salvatore Lopes, and Massimo Cossentino. 2013. Towards Self-Adaptation and Evolution in Business Process.. In *AIBP@ AI* IA*. Citeseer, 1–10.

[65] Luca Sabatucci, Salvatore Lopes, and Massimo Cossentino. 2017. Self-configuring cloud application mashup with goals and capabilities. *Cluster Computing* (2017), 1–17.

[66] Luca Sabatucci, Valeria Seidita, and Massimo Cossentino. 2017. The Four Types of Self-adaptive Systems: A Metamodel. In *International Conference on Intelligent Interactive Multimedia Systems and Services*. Springer, 440–450.

[67] Shazia W Sadiq, Olivera Marjanovic, and Maria E Orlowska. 2000. Managing change and time in dynamic workflow processes. *International Journal of Cooperative Information Systems* 9, 01n02 (2000), 93–116.

[68] Biplav Srivastava and Jana Koehler. 2003. Web service composition-current solutions and open problems. In *ICAPS 2003 workshop on Planning for Web Services*, Vol. 35. 28–35.

[69] Diane M Strong and Steven M Miller. 1995. Exceptions and exception handling in computerized information processes. *ACM Transactions on Information Systems (TOIS)* 13, 2 (1995), 206–233.

[70] Matthias Thimm. 2014. Tweety: A Comprehensive Collection of Java Libraries for Logical Aspects of Artificial Intelligence and Knowledge Representation. *KR* 14 (2014), 528–537.

[71] Wil MP Van der Aalst. 1998. The application of Petri nets to workflow management. *Journal of circuits, systems, and computers* 8, 01 (1998), 21–66.

[72] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. 2000. Verification of workflow task structures: A petri-net-baset approach. *Information systems* 25, 1 (2000), 43–69.

[73] Tim Van Eijndhoven, Maria-Eugenia Iacob, and María Laura Ponisio. 2008. Achieving business process flexibility with business rules. In *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE*. IEEE, 95–104.

[74] Axel Van Lamsweerde, Robert Darimont, and Emmanuel Letier. 1998. Managing conflicts in goal-driven requirements engineering. *IEEE transactions on Software engineering* 24, 11 (1998), 908–926.

[75] Silvia Von Stackelberg, Susanne Putze, Jutta Mülle, and Klemens Böhm. 2014. Detecting data-flow errors in BPMN 2.0. *Open Journal of Information Systems (OJIS)* 1, 2 (2014), 1–19.

[76] Ivo Vondrák. 2007. Business Process Modeling. *Frontiers in Artificial Intelligence and Applications* 154 (2007), 223.

[77] Xianghui Wang, Zhiyong Feng, and Keman Huang. 2018. D3L-based Service Runtime Self-adaptation Using Replanning. *IEEE Access* (2018).

[78] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty HC Cheng, and Jean-Michel Bruel. 2010. RELAX: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering* 15, 2 (2010), 177–196.

[79] Jennifer Widom and Stefano Ceri. 1996. *Active database systems: Triggers and rules for advanced database processing*. Morgan Kaufmann.

[80] Peter YH Wong and Jeremy Gibbons. 2008. A process semantics for BPMN. In *Formal Methods and Software Engineering*. Springer, 355–374.

[81] Eric Yu. 2011. Modelling strategic relationships for process reengineering. *Social Modeling for Requirements Engineering* 11 (2011), 2011.

[82] Eric SK Yu and John Mylopoulus. 1996. Using goals, rules and methods to support reasoning in business process reengineering. *Intelligent Systems in Accounting, Finance & Management* 5, 1 (1996), 1–13.