Checking Compliance of Execution Traces to Business Rules

Federico Chesani[1], Paola Mello[1], Marco Montali[1], Fabrizio Riguzzi[2], Maurizio Sebastianis[3], and Sergio Storari[2]

[1] DEIS – University of Bologna, V.le Risorgimento 2, 40136 Bologna, Italy
{federico.chesani,paola.mello,marco.montali}@unibo.it
[2] ENDIF – University of Ferrara, Via Saragat 1, 44100 Ferrara, Italy
{fabrizio.riguzzi,sergio.storari}@unife.it
[3] Think3 Inc., Via Ronzani 7/29, 40033 Casalecchio di Reno (BO), Italy
maurizio.sebastianis@think3.com

Abstract. Complex and flexible business processes are critical not only because they are difficult to handle, but also because they often tend to loose their intelligibility. Verifying compliance of complex and flexible processes becomes therefore a fundamental requirement. We propose a framework for performing compliance checking of process execution traces w.r.t. expressive reactive business rules, tailored to the MXML meta-model. Rules are mapped to Logic Programming, using Prolog to classify execution traces as compliant/non-compliant. We show how different rule templates, inspired by the ConDec language, can be easily specified and then customized in the context of a real industrial case study. We finally describe how the proposed language and its underlying a-posteriori reasoning technique have been concretely implemented as a ProM analysis plug-in.

1 Introduction

In the last years, Workflow Management Systems (WfMS) have been increasingly adopted by companies in order to efficiently implement their Business Processes. A plethora of tools, systems and notations have been proposed to cover all the phases of the Business Process Management life-cycle, from Process Design and Modeling to Execution and Monitoring/Analysis. To deal with needs and requirements of business users, two main dimensions have been recently tackled: *flexibility* and *expressiveness*. On one side, to be successfully employed WfMS should make a trade-off between controlling the way workers do their business and turning them loose to exploit their expertise during execution [1,2,3]; while constraining workers to follow a business process model, flexible WfMS support the possibility of deviating from its prescriptions and even changing it at run-time. On the other side, business processes are exploited to model complex problems and domains under different perspectives (e.g. the control flow perspective and the organizational one); to have an idea of the expressiveness needed

D. Ardagna et al. (Eds.): BPM 2008 Workshops, LNBIP 17, pp. 134–145, 2009.

to suitably face with this complexity, just take a look to the Worfklow Patterns [4] initiative[1].

Both dimensions are critical not only because they are difficult to handle, but also because they contribute to make the process less intelligible. Verifying compliance of complex and flexible processes becomes therefore a fundamental requirement. On the one hand, as claimed in [5] "deviations from the 'normal process' may be desiderable but may also point to inefficiencies or even fraud", and therefore flexibility could lead the organization to miss its strategic goals or even to violate regulations and governance directives. Among the different kinds of flexibility, *flexibility by change and by deviation* [1,3] enable the possibility of changing the process instance or deviating from the prescribed model during execution, making therefore impossible to assess compliance *before* the execution. On the other hand, as complexity increases it becomes important to provide support for a business analyst in the task of analyzing past process executions. This analysis can help the business manager in the process of assessing business trends and consequently making strategic decisions.

In this paper, we focus on this specific task, proposing a framework for performing compliance checking of process execution traces w.r.t. reactive business rules. Such rules are specified by means of a powerful declarative language, called CLIMB, which stems from Condec [6] constraints, extending their expressiveness not only as regards temporal aspects but also as regards event data, such as involved originators, event types and activity identifiers.

We sketch how the proposed language can be mapped to Logic Programming, enabling the possibility of exploiting Prolog in order to perform compliance checking. Such a reasoning technique has been exploited to implement a ProM [7] plug-in, called $\mathcal{S}$CIFFChecker, that classifies a set of MXML [8] execution traces as compliant/non-compliant w.r.t. a certain business rule, in the style of *LTL Checker* [5]. MXML is the format used in ProM to load the process execution traces.

The feasibility of our approach is assessed by considering a real case study involving Think3®[2], a company working in the Computer Aided Design (CAD) and Product Life-cycle Management (PLM) market.

The paper is organized as follows. Section 2 grounds the compliance checking problem on the Think3 case study. Language and methodology for specifying and applying CLIMB rules are presented in Section 3. Section 4 illustrates the implementation of a-posteriori compliance checking inside ProM, reporting experiments made on the Think3 case study. Related works and conclusions follow.

2 An Industrial Case Study

An important current challenge in the manufactoring industry is to handle, verify and distribute the technical information produced by the design, development and production processes of the company. The adoption of a system supporting

[1] `http://www.workflowpatterns.com`
[2] `http://www.think3.com`

the management of technical data and the coordination of the people involved is of key importance, to improve productivity and competitiveness. The main issue is to provide solutions for managing all the technical information and documentation (such as CAD projects, test results, photos, revisions), which is mainly produced by workers during the design phase. Since an important part of the design process is spent by testing, modifying and improving previously released versions, the *traceability* of relevant information concerning an item is necessary.

Think3 is one of the leading global players in the field of CAD and PLM solutions: it provides an integrated software which bridges the gap between CAD modeling environments and other tools involved in the process of designing (and then manufacturing) products. All these tools are transparently combined with a non-intrusive information system which handles the underlying product workflow, recording all the relevant information and making it easily accessible to the workers involved, enabling its consultation, use and modification. Such an information system supplies a detailed, shared and constantly updated vision of the life-cycle of each product, providing a complete log of the executed activities.

The underlying Think3 workflow centers around the design of a manufacturing product. Different activities can be executed to affect the progress-status of an item, involving the modification and even the evolution of multiple co-existing versions of its corresponding project. Such a workflow can be adapted on each single Think3 client company in order to meet different specific requirements.

2.1 Compliance Checking and Decision Making Support: Think3 Requirements

To support a business manager in decision making, and in particular in the tasks of analyzing the life-cycle of different projects and pinpointing problems and bottlenecks, Think3 is investigating the development of a Business Intelligence dashboard. Within the TOCAI.IT FIRB Project[3], Think3 and the University of Bologna are collaborating to realize one of the main dashboard components: a tool supporting compliance verification (both on and off-line) of design processes w.r.t. configurable business rules. This will facilitate the manager in the identification of behavioural trends and non-compliances to regulations or internal policies. In this particular case study, we elicitated the following non-exaustive list of interesting properties:

(Br1) Evaluating the time relationship between the execution of two given activities (e.g. *Was a project committed by 18 days after its creation?*).
(Br2) Identifying which projects passed too many times through a certain activity (e.g., *Which projects have been modified at least twice?*).
(Br3) Analysing originators, i.e., workers involved in the process (e.g., *Was a project checked by a person different than the one who published it?*).

[3] `http://www.dis.uniroma1.it/~tocai/`

3 CLIMB Business Rules

We propose a language, inspired by the $\mathcal{S}$CIFF one [9], for specifying reactive business rules (called CLIMB business rules throughout the paper). Their structure resembles ECA (Event-Condition-Action) rules [10]; the main difference w.r.t. ECA rules is that, since CLIMB rules are used for checking, they envisage expectations about executions rather than actions to be executed. Expectations represent events that should (not) happen. Therefore, CLIMB rules are used to constrain the process execution when a given situation holds. Both positive and negative constraints can be imposed on the execution, i.e., it is possible to specify what is mandatory as well as forbidden in the process.

Rules follow an `IF` *Body* `having` *BodyConditions* `THEN` *Head* structure, where *Body* is a conjunction of occurred events, with zero or more associated conditions *BodyConditions*, and *Head* is a disjunction of positive and negative expectations (or *false*). Each head element can be subject to conditions as well.

The underlying intuitive semantics is that whenever a set of occurred events makes *Body* (and the corresponding conditions *BodyConditions*) true, then also *Head* must eventually be satisfied[4]. A positive (resp. negative) expectation is satisfied if a corresponding matching event indeed occurs (resp. does not occur) and the associated conditions are satisfied as well. Furthermore, it is possible to specify rules without the IF part: such rules are used to impose what the business manager expects (not) to find inside the process instances in any case.

The concept of event is tailored to the one of audit trail entry in the MXML meta-model [8]. Events are atomic and mainly characterized by: *(i)* the name of the *activity* it is associated to; *(ii)* an *event type*, according to the MXML transactional model [8], which models the life-cycle of each activity with event types like "start", "re-assignment", "completion"; *(iii)* an *originator*, identifying the worker who generated the event; *(iv)* an *execution time*, representing the time at which the event has been generated; *(v)* one or more *data* items[5].

The main distinctive feature of our rules is that all these parameters are treated, by default, as variables. To specify that a generic activity A has been subject to a whatsoever event, the rule body will simply contain a string like: `activity` A `is` *performed* `by` O_A `at time` T_A, where A stands for the activity's name, O_A and T_A represent the involved originator and execution time respectively, and *performed* is a keyword denoting any event type. To facilitate readibility, the part concerning originator and execution time can be omitted if the corresponding variables are not involved in any condition.

Such a generic sentence will match with any kind of event, because all the involved variables (A, O_A and T_A) are completely free, and the event type is not specified. The sentence can then be configured in many different ways. In particular, the involved variables can be grounded to specific values or constrained by means of explicit conditions. The Event type can be instead fixed by simply

[4] Therefore, rules having *false* in the head are used to express denials.

[5] For simplicity, in the paper we will not take into account this aspect, but it can be seamlessly treated in our framework.

substituting the generic *performed* keyword with one of the specific types envisaged in the MXML transactional model.

Positive (negative) expectations are represented similarly to occurred events, by only changing the `is` part with `should (not) be`.

3.1 A Methodology for Building Rules

To clarify our methodology, let us consider a completely configured rule, namely the specification of the (Br3) rule of Think3:

$$\begin{aligned} &\texttt{IF activity } A \texttt{ is } performed \texttt{ by } O_A \texttt{ having } A \texttt{ equal to } Check \\ &\texttt{THEN activity } B \texttt{ should NOT be } performed \texttt{ by } O_B \\ &\quad \texttt{having } B \texttt{ equal to } Publish \texttt{ and } O_B \texttt{ equal to } O_A \end{aligned} \qquad \text{(CLIMB-Br3)}$$

By analyzing this rule, we can easily recognize two different aspects: on the one hand, the rule contains generic elements, free variables and constraints, whereas on the other hand it specifically refers to concrete activities. The former aspect captures re-usable patterns: in this case, the fact that the same person cannot perform two different activities A and B, which is known as the *four-eyes principle*. The latter aspect instantiates the rules in a specific domain, in this case grounding the four-eyes principle in the context of Think3's workflow. To reflect such a separation, we foresee a three-step methodology to build, configure and apply business rules: *(i)* a set of re-usable rules, called *rule templates*, are developed and organized into groups by a technical expert (i.e., someone having a deep knowledge of rules syntax and semantics); *(ii)* rule templates are further configured, constrained and customized by a business manager to deal with her specific requirements and needs; *(iii)* configured rules are exploited to perform compliance checking of company's execution traces.

3.2 Specification of Conditions

Conditions are exploited to constrain variables associated to event occurences and expectations inside business rules (namely activity names, originators and execution times). As shown in Figure 1 two main families of conditions are currently envisaged: string and time conditions. String conditions are used to constrain an activity/originator by specifying that it is equal to or different from another activity/originator, either variable or constant. An example of a string condition constraining two originator variables is the "O_B `equal to` O_A" part in rule (CLIMB-Br3).

Time conditions are used instead to relate execution times, in particular for specifying ordering among events or imposing quantitative constraints, such as deadlines and delays. The semantics of constraints is determined by time operators, which intuitively capture basic time relationships (such as $before$ or at). Absolute time conditions constrain a time variable w.r.t. a certain time/date, whereas relative time conditions define orderings and constraints between two

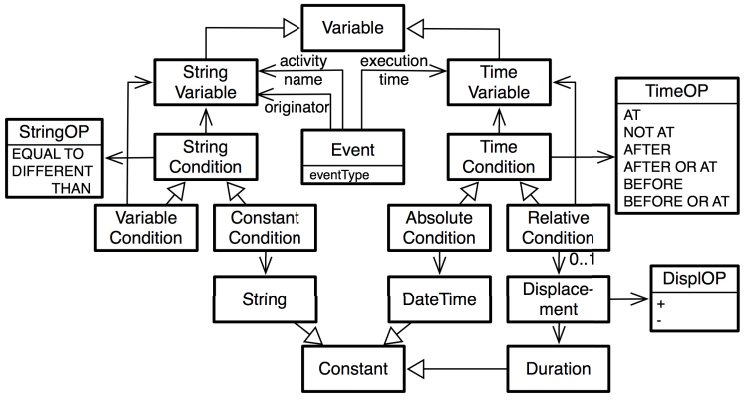

Fig. 1. Basic hierarchy of string and time constraints

variables. Relative conditions can optionally attach a displacement to the target time variable as well. For example, to specify that the time variable T_B must be within a *displacement* of 2 days *after* T_A, we simply write T_B `BEFORE` T_A+2*days*.

3.3 Rule Templates

The first step in our methodology envisages the creation of rule templates, i.e. re-usable partially constrained rules. They typically fix the rule structure, e.g. deciding how many events are contained in the body, and use variable string conditions and/or relative time conditions. They do not involve absolute time and constant conditions, which are exploited to ground rules on a specific domain.

We have developed a hierarchy of rules which strictly resembles the one proposed for ConDec. Three basic groups are defined: *existence* rules, *IF... THEN* rules and *IF... THEN NOT* rules. The hierarchy is not fixed: it can be adapted or even replaced by writing other rules and by organizing them differently.

Existence Rules impose the presence/absence of some events in the execution trace, independently from the occurrence of other events. The *presence* (*absence*) template simply state that a certain event is expected to (not) occur, and is simply formalized as: `activity` A `should (NOT) be` *performed*. *Choice* extends *presence* by introducing disjunction of expectations. The *at least* N (*at most* N) rule extends the presence (absence) one by stating that the specified event should (not) be repeated N times.

Such rules are useful for modeling the presence/absence of multiple instances of a certain event in the execution trace, as in (Br2), but they are rather difficult to be represented, especially when N increases. For this reason, we have extended the syntax of the language for supporting repetitions as first-class entities. To specify that activity A should be performed at least 3 times, we will then write: `activity` A `should be` *performed* `3 times between` T_{sA} `and` T_{cA}. The two involved time variables extend the concept of execution time when dealing with multiple events,

by identifying the two time points at which the repetition starts and completes. To express that A must be performed at most 3 times, we can simply state: IF activity A is *performed* 4 times THEN $false$.

IF...THEN Rules are positive relationships which specify that when certain events happen, then also other events should occur, satisfying the imposed time orderings. The simplest rule belonging to this group is the *responded existence* one, which simply states that when a certain event happens, then another event should happen too, either before or afterward. Starting from this rule, *response* and *precedence* templates extend it by adding respectively an *after* and *before* relative time condition among the involved execution times.

Response and precedence rules can then be specialized to express more complex event patterns, e.g. introducing conjunctions and disjunctions of events. For example, the following template represents a *synchronized response*, i.e. a response triggered by the occurrence of two events:

$$\begin{aligned} &\texttt{IF activity } A \texttt{ is } performed \texttt{ at time } T_A \\ &\texttt{and activity } B \texttt{ is } performed \texttt{ at time } T_B \\ &\texttt{THEN activity } C \texttt{ should be } performed \texttt{ at time } T_C \\ &\texttt{having } T_C \; after \; T_A \texttt{ and } T_C \; after \; T_B. \end{aligned}$$

IF...THEN NOT Rules express events to be forbidden when other events happen. Roughly speaking, they substitute positive expectations with negative ones; for example, the *responded absence* states that IF activity A is *performed* THEN activity B should not be *performed*.

This templates is a good example to illustrate how conditions about originators can be used. Indeed, by adding an *equal to* constraint between the originators of A and B, it actually models the already cited four-eyes principle.

3.4 From Templates to Customized Business Rules

In a second phase, rule templates are configured by a business manager to deal with her specific requirements. This step exploits constant string conditions, absolute time conditions and relative time conditions with displacements, specifically referring to the company's domain. For example, the CLIMB rule

$$\begin{aligned} &\texttt{IF activity } A \texttt{ is } performed \texttt{ at time } T_A \\ &\texttt{having } A \texttt{ equal to } Creation \\ &\texttt{THEN activity } B \texttt{ should be } performed \texttt{ at time } T_B \text{ (CLIMB-Br1)} \\ &\texttt{having } B \texttt{ equal to } Commit \\ &\quad \texttt{and } T_B \texttt{ after } T_A \texttt{ and } T_B \texttt{ before } T_A + 18 days. \end{aligned}$$

models (Br1) by extending *response* with a *before* relative time constraint having a 18 days-displacement (used to express a deadline).

Finally, Think3's Rule (Br2) can be easily modeled by grounding the *at least* N template on the *Modify* activity.

3.5 Compliance Verification with Logic-Programming

Compliance verification is concretely carried out by mapping each execution trace and the CLIMB rule to Logic Programming, exploiting Prolog for reasoning. An execution trace is treated as a knowledge base storing each audit trail entry as a fact of the type

$$happened(event(EventType, ActivityName, Originator), ExecutionTime))).$$

The rule used for checking is instead transformed into a Prolog query by computing the negation of the implication represented by the CLIMB rule. So, if the CLIMB rule is represented by the implication $B \rightarrow H$, then the query would be $B \wedge \neg H$. Such a query tries to find a set of occurred events in the execution trace that satisfy the rule body but violate the rule head. For example, rule (Br3) is translated to the following query:

$$?{-}A = \texttt{'Check'}, happened(event(\_, A, O_A), T_A),$$
$$not(B = \texttt{'Publish'}, O_B \neq O_A, not(happened(event(\_, B, O_B), T_B))).$$

Since the analysis is performed a-posteriori, positive expectations are flattened to occurred events, and negative expectations to the absence of events. If the query succeeds, then a counter-example which violates the rule has been found in the execution trace. The trace is then evaluated as non-compliant.

4 $\mathcal{S}$CIFFChecker: Compliance Checking in ProM

Drawing inspiration from the *LTL Checker* [5], we have embedded such a compliance reasoning technique into a ProM [7] analysis plug-in, called $\mathcal{S}$CIFFChecker, for the classification of MXML execution traces w.r.t. CLIMB business rules. $\mathcal{S}$CIFFChecker relies on the three-steps methodology described in Section 3.1, providing a user-friendly GUI for the customization of rule templates.

At start-up, templates are loaded from an XML-based template file. At the moment, different templates are already available, following the structure proposed in Section 3.3. In order to extend or modify the template hierarchy, the technical expert has simply to change this file. As shown in Figure 2, templates are displayed exploiting a tree-like component; clicking on a template description causes its corresponding CLIMB representation to appear in the center panel. By clicking on a "configuration" button, the different variables and customizable elements of the rule become highlighted. When selecting an highlighted element, a specific customization panel appears, supporting the user in setting the parameters (such as event types and repetitions) and in the specification of conditions. When the chosen rule has been customized, it can be either saved to a special group containing all the user-defined rules or used for compliance checking. In the latter case, the user has first to choose a *granularity*, which ranges from milliseconds to months and defines the time unit for converting time quantities into integer values. For each execution trace contained in the considered MXML

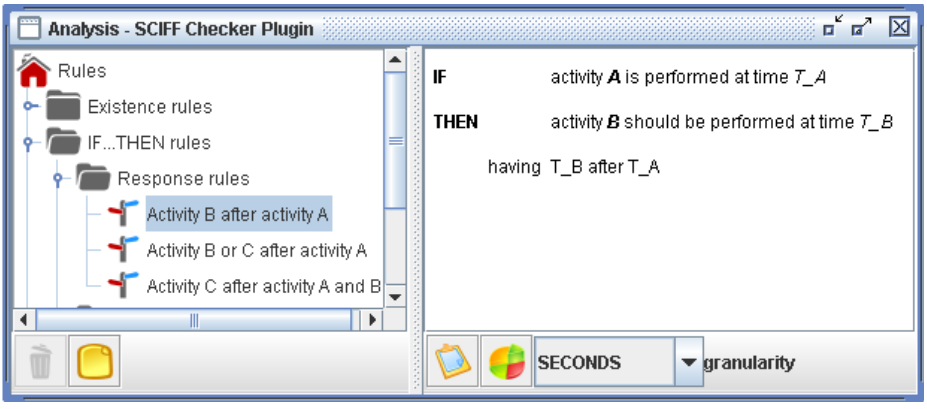

Fig. 2. A screenshot of the main $\mathcal{S}$CIFFChecker window

log, three steps are then performed transparently to the user: *(i)* the execution trace is translated into a Prolog knowledge base, converting involved execution times; *(ii)* the CLIMB business rule is mapped to a Prolog query, following the proof-of-concept shown in Section 3.5; *(iii)* a Prolog engine based on SWI[6] is exploited to verify whether the execution trace complies to the CLIMB rule.

All verification outcomes are finally collected and a summarizing pie chart is shown, together with the explicit list of compliant/non compliant traces. The user can then start a new classification by considering the whole log or only the (non) compliant execution traces. In this way, a conjunction of CLIMB rules can be verified by performing a sequence of tests, each one dealing with a single rule, and selecting at each step only the compliant subset for the next verification.

$\mathcal{S}$CIFFChecker has been concretely applied to analyze the execution traces of a Think3 client. We have first exploited the ProM Import tool[7] in order to convert the relevant information from the client database into an MXML format, by considering the project name as case identifier.

In particular, we extracted a portion of 9000 execution traces, ranging from 4 to 15 events. Then we used, together with a Think3 business manager, the plug-in to express and test the business rules of interest described in Section 3.4. The average time for performing compliance checking have been assessed to be around 10-12 seconds. The verification outcomes have been finally analyzed with the business manager. For example, considering rules (Br2) and (Br3), we discovered that, fortunately, only 2% of the execution traces involved more than two project revisions, and that in 3,5% of the cases only the same person was responsible for both publishing and checking the project.

The verification of rules like (Br1) was found interesting especially by varying the deadline involved. Indeed, the business manager wanted to detect projects

[6] http://www.swi-prolog.com/
[7] http://promimport.sourceforge.net

taking too much time as well as projects released too soon, to point out both possible bottlenecks and potential inaccuracies.

5 Related Work

A huge amount of work has been (and is being) carried out in order to deal with flexibility and adaptivity in Process Aware Information Systems [2]. DECLARE [1] is a constraint-based WfMS which adopt ConDec as a declarative graphical specification language. Flexibility is tackled at design-time, supporting the user in the specification of a minimal set of constraints that should be met during execution rather than focusing on a specific procedural solution, and at run-time, supporting the dynamic removal and insertion of constraints. In [3] ADEPT2 is proposed as a Workflow Management System capable to support the change of running instances (*flexibility by change*). The authors also suggest that Adaptive Process Management System should store, besides the enactment log, also the log of sequences of changes applied to a process model during execution.

$\mathcal{S}$CIFFChecker could be considered as complementary to these systems: it can be used to assess whether executed instances met the desired requirements/ regulations. An interesting future work concerns the verification of process logs containing also the sequence of changes; in this setting, it would be possible to express requirements involving also such changes, and to investigate the relationship between non-compliances and changes.

The closest work to the one here presented is the ProM *LTL-Checker* [5], that shares with our approach motivation and purposes. While *LTL-Checker* exploits Linear Temporal Logic (LTL) for the formalization of properties, our approach belongs to the Logic Programming family. The main technical difference between the two approaches is that while LTL formulae employ temporal modalities to qualitatively deal with time and must be completely grounded before the verification, CLIMB rules support variables and an explicit notion of time. The impact of this difference is twofold: *(i)* CLIMB rules are more expressive than LTL formulas, being e.g. able to constrain execution times and model delays/deadlines; *(ii)* the configuration of templates inside the *LTL-Checker* mainly consists of associating a ground value to the involved parameters, while $\mathcal{S}$CIFFChecker supports the specification of many different conditions on the variables, enabling the possibility of modeling a variety of business rules starting from a single template.

The task of verifying compliance with regulations and rules can also be carried out before the execution. In [11], the compliance of processes to regulations and standards is enforced by design rather than being checked a posteriori. In order to identify obligations that an enterprise has to fulfill for the process to be compliant, the authors adopt a deontic logic language. Obviously, a static approach is not suited to deal with flexibility by change or deviation. Furthermore, it cannot deal with situations where the outcome of the compliance test inherently depends on the actual configuration (resources and data) of the process instance, e.g. like in the case of the *four-eyes principle*, where the focus of the constraint is about the actual originators.

An interesting research topic concerns the integration of $\mathcal{S}$CIFFChecker with the process mining algorithm described in [12], which follows the opposite direction: it aims at discovering a set of declarative business rules, specified in the $\mathcal{S}$CIFF language, starting from execution traces previously classified as correct or wrong, s.t. the mined specification evaluates as compliant the correct subset and as non compliant the wrong one. We could first mine a set of CLIMB business rules, use them to classify new traces, or exploit $\mathcal{S}$CIFFChecker to split a given MXML log into the wrong and correct subsets required as input for the mining algorithm. The latter approach could be exploited to discover a declarative model giving an *explanation* of the $\mathcal{S}$CIFFChecker classification.

6 Conclusions and Future Work

We have described a framework for checking the compliance of process execution traces to declarative reactive business rules, proposing a three-steps methodology for developing and applying rules. The approach has been tested on a real industrial case study, identifying what kind of rules the Think3 company would be able to check and showing how they can be easily expressed by customizing rule templates (re-usable patterns resembling ConDec constraints). In order to effectively use such rules for reasoning, we have sketched how they can be mapped to Logic Programming, making possible to adopt Prolog for verification. A ProM plug-in that classifies MXML execution traces w.r.t. CLIMB business rules, helping a business manager in the assessment of business trends and providing decision making support, has been implemented on top of this reasoning technique.

We are applying our approach in other domains, such as a regional health screening protocol and in a chemo-physical process of wastewater treatment plants [13]. In the future, we will continue to develop the framework, introducing the possibility of dealing also with case and event data (which is seamlessly supported by the underlying reasoning technique), and investigating relationships between CLIMB rules and other languages.

In particular, we will study to what extent CLIMB business rules can be expressed as SQL queries in temporal databases. We have chosen Prolog as underlying formal framework because of its great expressiveness. Thanks to Prolog, tested rules can be complemented with background knowledge composed of (possibly recursive) rules that allow to infer new facts about the analyzed traces (e.g., an organizational perspective comprising roles and groups can be easily expressed). Anyway, only further investigation will reveal if such an expressiveness will be effectively exploited by business analysts.

Acknowledgments. This work has been partially supported by the FIRB project *TOCAI.IT*. The authors would like to thank Wil van der Aalst and Maja Pesic for their valuable comments and suggestions.

References

1. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-based workflow models: Change made easy. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 77–94. Springer, Heidelberg (2007)
2. Weber, B., Rinderle, S., Reichert, M.: Change patterns and change support features in process-aware information systems. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 574–588. Springer, Heidelberg (2007)
3. Reichert, M., Rinderle, S., Kreher, U., Dadam, P.: Adaptive process management with ADEPT2. In: Proceedings of the 21st International Conference on Data Engineering (ICDE 2005), pp. 1113–1114. IEEE Computer Society, Los Alamitos (2005)
4. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
5. van der Aalst, W., de Beer, H., van Dongen, B.: Process Mining and Verification of Properties: An Approach based on Temporal Logic. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 130–147. Springer, Heidelberg (2005)
6. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006)
7. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Mans, R.S., de Medeiros, A.A., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W., Weijters, A.J.M.M.: ProM 4.0: Comprehensive Support for Real Process Analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)
8. van Dongen, B.F., van der Aalst, W.M.P.: A Meta Model for Process Mining Data. In: Casto, J., Teniente, E. (eds.) Proceedings of the CAiSE 2005 Workshops (EMOI-INTEROP Workshop), FEUP, Porto, Portugal, vol. 2, pp. 309–320 (2005)
9. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. ACM Transactions on Computational Logic 9(4) (to appear) (2008)
10. Pissinou, N., Snodgrass, R.T., Elmasri, R., Mumick, I.S., Özsu, T., Pernici, B., Segev, A., Theodoulidis, B., Dayal, U.: Towards an infrastructure for temporal databases. SIGMOD Rec 23(1), 35–51 (1994)
11. Sadiq, S.W., Governatori, G., Namiri, K.: Modeling control objectives for business process compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 149–164. Springer, Heidelberg (2007)
12. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing declarative logic-based models from labeled traces. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 344–359. Springer, Heidelberg (2007)
13. Luccarini, L., Bragadin, G.L., Mancini, M., Mello, P., Montali, M., Sottara, D.: Process quality assessment in automatic management of wastewater treatment plants using formal verification. In: Proceedings of Simposio Internazionale di Ingegneria Sanitaria Ambientale (SIDISA 2008) (to appear) (2008)