Unified semantic model and reasoning techniques for mining and monitoring process-relevant data

Deliverable D2.2

FFG – IKT der Zukunft
SHAPE Project
2014 – 845638

Table 1 Document Information

Project acronym:	SHAPE
Project full title:	Safety-critical Human- & dAta-centric Process management in Engineering projects
Work package:	2
Document number:	2.2
Document title:	Unified semantic model and reasoning techniques for mining and monitoring process-relevant data
Version:	1
Delivery date:	01 October 2015 (M2)
Actual publication date:	————————
Dissemination level:	Public
Nature:	Report
Editor(s) / lead beneficiary:	WU Vienna
Author(s):	Cristina Cabanillas, Giray Havur, Jan Mendling, Axel Polleres
Reviewer(s):	Vadim Savenkov, Axel Polleres and Alois Haselboeck

Contents

1 Introduction

This document is part of work package 2 (WP2) on *Semantic Models for Mining & Monitoring Process Relevant Data* of the SHAPE project[1]. It reports work performed under Task 2.2 *Elicit and formalize the extended model in a unified formalism*.

Building upon the resource allocation example in the first deliverable, we investigate various reasoning tasks for checking constraints on business processes against (mined) historical data about process executions. We investigate possible formalisms that can perform reasoning tasks related to business processes. Namely; we look into incremental Answer Set Programming(iASP), Transaction Logic and Constraint Logic Programming in Finite Domains(CLP-FD).

In particular, the content of this deliverable is structured as follows: Section 2 provides an overview on semantic models for mining and monitoring process-relevant data. Section 3 describes models along with reasoning tasks for ensuring compliance and security requirements for business process. Section 4 describes resource allocation problem in three different formalisms: Incremental Answer Set Programming(iASP), Transaction Logic and CLP-FD. Section 5 gives an insight into scalability of the resource allocation problem in these formalisms and their performance comparisons. Section 6 suggests a novel method for improving feasibility and scalability for time-optimal resource allocation in business processes. Section 7 concludes the deliverable by remarking ongoing and future work.

2 Semantic Models for Mining and Monitoring Process-Relevant Data

Process mining is about extraction of useful and non-trivial information from event logs stored in an information system. It is a young research subject that has vastly evolved since the early work (i.e. [1, 13, 14]) on process logs.

Monitoring is considered an important building block to support business process compliance. A key application of monitoring business processes is revealing and pinpointing violations of imposed compliance rules that occur during process execution, which is crucial to SHAPE project [32]. An overview of process mining and monitoring is given in Figure 1.

Today, the majority of enterprise information systems are process-aware, using some mechanism to support, control, and monitor business processes [18]. This drives the trend in the business process management research towards semantic

[1] https://ai.wu.ac.at/shape-project/

technologies and targeted visualisation techniques that are more user-friendly to business experts and process owners [27].

There are different challenges that concerns mining and monitoring business process-relevant data. Some of these challenges are as follows:

1. Lack of common format of workflow logs
2. Incorrect identification of individual activities
3. Lack of means for proactive prevention of execution violations

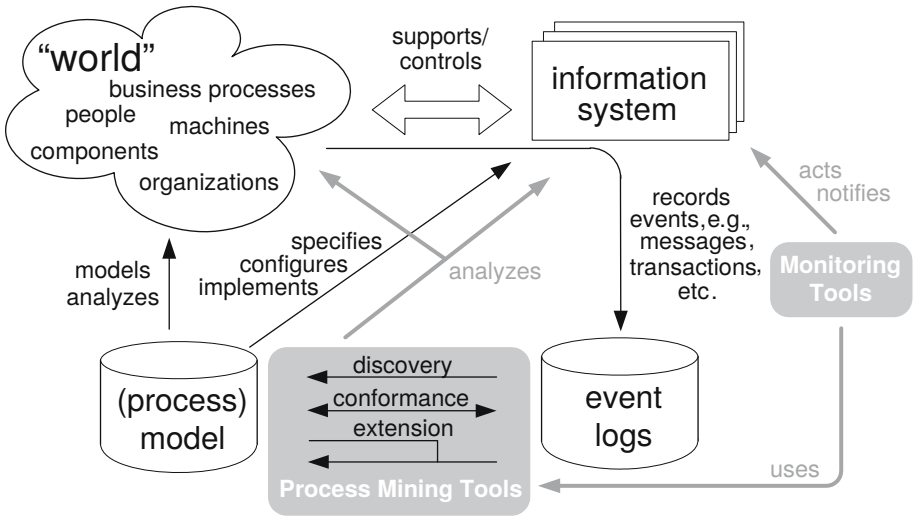

Figure 1 Overview of process mining and monitoring [15]

Semantic technologies [15, 38, 36], in particular ontologies [22, 33, 34], together with software tools such as repositories and reasoners, offer a suitable framework for business process mining and monitoring, catering for great interoperability and extensibility. There are two important qualities of semantic technologies that helps mining and monitoring business process-relevant data:

- Integration of heterogeneous sources of information (with regard to challenge 1)
- Formal definition ready for querying and automated reasoning (with regard to challenges 2 and 3)

The semantic models used in our project are further detailed in Deliverable 4.2-4.4 whereas herein we focus on encoding possible reasoning problems assuming that mined data witnessing process execution is already available.

3 Reasoning over Process-related Data for Compliance, Safety and Security

The fulfillment of compliance, safety and security requirements in business processes is indispensable for meeting all the governing regulations enforced on business operations [23]. Such regulations put restrictions and provide guidelines for businesses about performing operations to stay compliant. In case of a misalignment with these regulations, financial and criminal penalties are imposed. For this reason, there are many compliance management frameworks that help businesses to support specific compliance, safety and security requirements. [5] (e.g. [19] provides a literature surveys on the applicability of these frameworks). [10] studies different frameworks and modeling languages using a four point criteria. Semantic technologies, due to their benefits listed in Section 2, are applied in analyzing compliance, safety and security requirements [35, 25, 26].

There are different challenges at fulfillment of safety and security requirements which can benefit from automated reasoning methods and their applications. Some of them are namely

- detection of any non-compliant patterns at execution-time,
- detection of any non-compliant patterns in historical data (i.e. logs),
- planning the series of activities for ensuring compliant business processes execution, and
- documentation of compliance to safety and security regulations.

We will investigate the possibilities in some of the above mentioned directions concerning possible reasoning applications. The details of semantic models in this concern are further detailed in Deliverable 4.2-4.4.

4 Reasoning over Business Processes and Constraints

Various reasoning tasks over business processes can greatly benefit from historical business process execution data(i.e. activity execution times of resources). Such data about processes and resources can be obtained by applying data mining techniques on process logs(cf. Deliverable 3.1 [4], Deliverable 3.2 [3]). By using this data, we extend our automated resource allocation encoding [8] in iASP [21] and defined the resource allocation problem in Transaction Logic [6, 30] and CLP-FD [28].

4.1 Resource Allocation Problem

Automated resource allocation problem in this section comprise the following elements:

- A timed Petri net[2] PN describing a set of activities A, their estimated durations D_A and precedence constraints P_A
- An organizational model describing a set of resources R and their roles in the organizational hierarchy L_R
- A set of role-activity constraints $C_{R \times A}$
- A set of temporal constraints $C_{(R \cup L) \times A \times \mathbb{Z}}$

We describe an example scenario for clarifying the encodings of the problem. Fig. 2 and Fig. 3. Fig. 2 depicts a model representing the process of publishing a book from the point of view of a publishing entity. In particular, when the publishing entity receives a new textbook manuscript from an author, it must be proofread. If changes are required, the modifications suggested must be applied on text and figures, which can be done in parallel. This review-and-improvement procedure is repeated until there are no more changes to apply, and the improved manuscript is then sent back to the author for double-checking. In Fig. 2, the numbers above the activities indicate their (default maximum) duration in generic time units (TU)[3]. The organisational model depicted in Fig. 3 shows the hierarchy of roles of a publishing entity. Specifically, it has four roles and five resources assigned to them. The following relation specifies how long it takes to each role and resource to complete the process activities: $(Role \cup Resource) \times Activity \times TU \supset$ {(Copy Editor, Proofread, 2), (Glen, Proofread, 5), (Drew, Proofread, 2), (Drew, Revise Text, 2)}. For resource allocation purposes, the duration associated with a specific resource is used in first place followed by the duration associated with roles and finally, the duration of activities (cf. Fig. 2). Resources are assigned to activities according to their roles. In particular, the relation activity-role in this case is as follows: $Role \times Activity \supset$ {(Publisher, Receive Manuscript), (Copy Editor, Proofread), (Copy Editor, Revise Text), (Graphic Artist, Revise Visual), (Admin. Asst., Send Press Release)}.

[2] Timed Petri nets are detailed in Section 4 of [8]
[3] Please, note that events are instantaneous, and hence, they take zero time units.

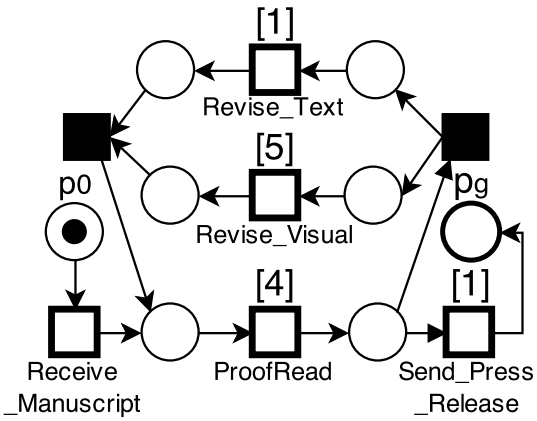

Figure 2 Process to publish a book

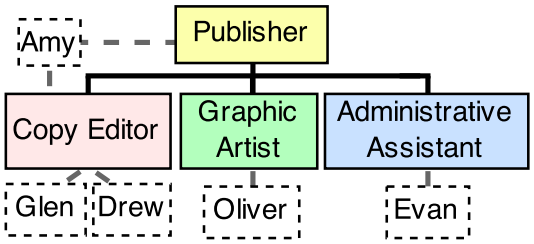

Figure 3 Organisational model of `AmyPublishingHouse`

4.2 Resource Allocation in iASP

In the previous report, we provided a detailed explanation of an automated resource allocation encoding in iASP [8]. We have improved this encoding with additional concepts and constraints related to time and resources.

4.2.1 Resource Allocation for Multiple Processes

We add the capability of resource allocation for multiple processes with multiple instances by identifying each process instance with both a business process ID, `B`, and an instance ID, `I`. Appendix B provides the updated version of the ASP encoding.

4.2.2 Break Calendars

As an additional temporal constraint, adding the following rule to our program gives us the possibility of not assigning any activities to a resource between a given time `C1` and `C2`. In the program, this interval is defined with the predicate `noAssignmentBetween(C1,C2)`.

```
:- noAssignmentBetween(C1,C2),assign(_, _,C3,C4, _, _, _),C1>=C3,
   C2<=C4.
```

This rule allows us to define breaks such as holidays in the assignment schedule.

4.2.3 Separation of Duties

Separation of duties is a requirement of having more than one person required to complete a task. This separation of more than one individual on one single task is an internal control intended to prevent fraud or error. The predicates `strictlySeparated` and `weaklySeparated` introduce this concept at two different levels.

The following rules are related to strict separation of activities, which separate the assignee of the activity `A` of business process `B` from the the assignee of the activity `A1` of business process `B1` for the overall execution of the process.

```
notAssignStrictly(R,A1,B1,C2,s) :- strictlySeparated(A,B,A1,B1),
                                   assign(R,A,_,C2,s-1,B,I).
:- assign(R,A,C1,C2,S,B,I), notAssignStrictly(R,A,B,C,S1), C1>=C.
```

The following rules are related to weak separation of activities, which separate the assignee of the activity `A` of business process `B` from the assignee of the activity `A1` of business process `B1` for one consecutive execution of `A` and `A1`.

```
notAssignWeakly(R,A1,B1,C2,s) :- weaklySeparated(A,B,A1,B1),
                                 assign(R,A,_,C2,s-1,B,I).
notAssignWeakly(R,A,B,C,s) :- notAssignWeakly(R,A,B,C,s-1),
                              not assign(_,A,_,_,s-1,B,_).
:- assign(R,A,C1,C2,s,B,I), notAssignWeakly(R,A,B,C,s), C1>=C.
```

4.2.4 Binding of Duties

A binding of duty ensures that if a particular resource is assigned to perform a certain activity, then the same user must also be assigned to perform a certain other activity. The predicates `strictlyBinded` and `weaklyBinded` introduce this concept in two different levels.

The following rules are related to strict binding of activities, which bind the assignee of the activity `A` of business process `B` with the assignee of the activity `A1` of business process `B1` for the overall execution of the process.

```
mustAssignStrictly(R,A1,B1,C2,s) :- strictlyBinded(A,B,A1,B1),
                                    assign(R,A,_,C2,s-1,B,I).
:- assign(R1,A,C1,C2,S,B,I), mustAssignStrictly(R,A,B,C,S1), C1>=C,
   R1!=R.
```

The following rules are related to weak binding of activities, which bind the assignee of the activity `A` of business process `B` with the assignee of the activity `A1` of business process `B1` for one consecutive execution of `A` and `A1`.

```
mustAssignWeakly(R,A1,B1,C2,s) :- weaklyBinded(A,B,A1,B1),
                                  assign(R,A,_,C2,s-1,B,I).
mustAssignWeakly(R,A,B,C,s) :- mustAssignWeakly(R,A,B,C,s-1),
                               not assign(R,A,_,_,s-1,B,_).
:- assign(R1,A,C1,C2,s,B,I), mustAssignWeakly(R,A,B,C,s), C1>=C,
   R1!=R.
```

4.2.5 Mapping of RAL Expressions to ASP Rules

Resource Assignment Language (RAL) is a domain-specific language that enables the definition of conditions to select the candidates to participate in a process activity. Listing 1 describes all possible expressions that can be formed in RAL. We refer the reader to [9] for further information about RAL.

Listing 1 The Extended Backus–Naur Form (EBNF) syntax of RAL [9]

```
Expr = PersonExpr / GroupResourceExpr / CommonalityExpr
       / CapabilityExpr / HierarchyExpr / ReportExpr /
       DelegateExpr / DenyExpr / CompoundExpr

CompoundExpr = (Expr "OR"  Expr )/
               (Expr "AND" Expr )

PersonExpr = "IS" PersonConstraint

GroupResourceExpr = ("HAS" RoleConstraint ("IN" UnitConstraint)?)/
                    ("HAS" PositionConstraint)/
                    ("HAS" UnitConstraint)

CommonalityExpr = ("SHARES" Amount "POSITION WITH" PersonConstraint)/
                  ("SHARES" Amount "UNIT WITH" PersonConstraint)/
                  ("SHARES" Amount "ROLE" ("IN" UnitConstraint)?
                   "WITH" PersonConstraint)

CapabilityExpr = ("HAS CAPABILITY" capability)
HierarchyExpr = (ReportExpr) / (DelegateExpr)
ReportExpr = (Depth "REPORT TO" PositionRef) /
             ("IS" Depth "REPORTED BY" PositionRef)

DelegateExpr = ("CAN DELEGATE WORK TO" PositionRef) /
               ("CAN HAVE WORK DELEGATED BY" PositionRef)

DenyExpr = ("NOT" DeniableExpr)
DeniableExpr = PersonExpr / GroupResourceExpr / CommonalityExpr /
               CapabilityExpr
```

```
PersonConstraint = (personName) / ("PERSON IN DATA FIELD"
                                   dataObject.fieldID)/
                   ("ANY PERSON" taskDuty "ACTIVITY" activityID)

PositionConstraint = ("POSITION" positionName) /
                     ("POSITION IN DATA FIELD" dataObject.fieldID)

UnitConstraint = ("UNIT IN DATA FIELD" dataObject.fieldID)/
                 ("UNIT" unitName)

RoleConstraint = ("ROLE IN DATA FIELD" dataObject)/
                 ("ROLE" roleName !"IN")

PositionRef = ("POSITION OF" PersonConstraint) / PositionConstraint

Depth = "DIRECTLY"?
Amount = "SOME" / "ALL"
```

We make use of RAL expressions for describing resource selection constraints in our ASP program. For this reason, we map these expressions into ASP rules. For instance, if the activity `Receive_Manuscript` in Fig. 2 has the following statement:

`HAS ROLE Copy_Editor IN AmyPublishingHouse`

the corresponding ASP rule with respect to the encoding in Appendix B would be

`canExecute(Copy_Editor,Receive_Manuscript).`

4.3 Resource Allocation in Transaction Logic

Transaction Logic has been proved to be useful in a vast range of applications: from databases to robot action planning to reasoning about actions to workflow analysis. FLORA-2 integrates Transaction Logic [7, 29] with other formalisms such as F-logic [31] and HiLog [12] with new extensions. In this section, we encode resource allocation problem in FLORA-2.

4.3.1 FLORA-2

FLORA-2 is a knowledge base engine and a complete environment for developing knowledge-intensive applications using Transaction Logic formalism [30].

4.3.1.1 F-Logic

F-logic (frame logic) is a knowledge representation and ontology language [37]. It combines the advantages of a declarative approach with object-oriented concep-

tual modeling. Some of its features are namely object identity, complex objects, inheritance, polymorphism, query methods and encapsulation. first-order variable-free terms to represent *object identity*(OID). Objects can have single-valued, set-valued or Boolean attributes. These formulas are called *F-logic molecules*. For instance:

$$\texttt{billing\_process}[\texttt{biller} \rightarrow \texttt{john}, \texttt{customers} \twoheadrightarrow \{\texttt{alice}, \texttt{nancy}\}].$$
$$\texttt{billing\_process}[\texttt{customers} \twoheadrightarrow \{\texttt{jack}\}, \texttt{valid\_voucher}].$$

Note that each formula above asserts more than one fact simultaneously. In the first formula $\texttt{biller} \rightarrow \texttt{john}$ says that object `billing_process` has a single-valued attribute `biller`, whose value is OID `john`, while $\texttt{customers} \rightarrow \{\texttt{alice}, \texttt{nancy}\}$ in the same object description says that the value of the set-valued attribute `customers` is a set that contains two OIDs: `alice` and `nancy`. We emphasize "contains" because sets do not need to be specified all at once. For instance, the second formula above says that `billing_process` has another customer `jack`. The attribute `valid_voucher` in the second formula is Boolean: its value is true in the above example.

In the standard convention, uppercase symbols denote variables while symbols beginning with a lowercase letter denote constants. For example,

$$\texttt{X}[\texttt{customer} \twoheadrightarrow \{\texttt{C}\}] : -\texttt{Y}[\texttt{biller} \rightarrow \texttt{X}, \texttt{customer} \twoheadrightarrow \{\texttt{C}\}].$$

derives $\texttt{john}[\texttt{customers} \twoheadrightarrow \{\texttt{alice}, \texttt{nancy}, \texttt{jack}\}]$.

There are also functions that take arguments in F-logic, called *methods*. For instance,

$$\texttt{john}[\texttt{salary}(\texttt{january}, 2015) \rightarrow 1000, \texttt{on\_leave}(2015) \twoheadrightarrow \{\texttt{june}, \texttt{july}\}].$$

says that john has a single-valued method, `salary`, whose value on the arguments `january` and 2015 is 1000; it also has a set-valued method `on_leave`, whose value on the argument 2015 is a set of OIDs that contains `june` and `july`. Like attributes, methods can be defined using inference rules.

The F-logic syntax for class membership is $\texttt{john} : \texttt{employee}$ and for subclass relationship it is $\texttt{employee} :: \texttt{person}$. In addition, F-logic supports specification of schema information. For instance, $\texttt{person}[\texttt{name} \Rightarrow \texttt{string}, \texttt{colleague} \Rightarrow\!\!\!\!\Rightarrow \texttt{person}]$ says that the signature of class `person` has two attributes, a single-valued attribute `name` and a set-valued attribute `colleague`. Moreover, the first attribute returns objects of type string and the second returns sets of objects such that each object in the set is of type person. F-logic is an extension of first-order logic and thus it integrates relational and object-oriented paradigms. We refer to [31] for further details of F-logic.

4.3.1.2 Hi-Log

HiLog was introduced in [12] in order to extend logic programming with higher-order syntax in first-order semantics. The main goal was to enable flexible and natural querying of term structures and to support reification of atomic formulas. For instance, the following statement,

$$\texttt{call(X)} : -\texttt{X}.$$

means that HiLog does not distinguish between function terms and atomic formulas. Combined with F-logic, HiLog improves the meta-features of the language. For example, in the combined language, one can write:

$$\texttt{X[methods} \twoheadrightarrow \texttt{\{M\}]} : -\texttt{X[M(\_,\_)} \rightarrow \texttt{\_]}.$$

Thus, a query of the form

$$?- \texttt{john[methods} \twoheadrightarrow \texttt{M]}.$$

will return the set of all 2-argument set-valued methods defined for the object `john`.

4.3.1.3 Transaction Logic

For updating the database part of the program, Prolog introduced `assert` and `retract` operators. However, these operators could never provide a truly logical database update system, because in case the execution of a Prolog program fails, all the changes made by `assert` and `retract` would stay in the database, which may easily lead an inconsistent state. Transaction Logic [7, 6] provides a comprehensive theory of logical updates in logic programming, which does not suffer from the drawbacks of Prolog style updates that previously detailed. In Transaction Logic, both *actions* (transactions) and *queries* are represented as predicates. In FLORA-2, transactions are represented as object methods with a prefixed symbol "#".

4.3.2 Problem Encoding in FLORA-2

The input for the running example is provided in Figure 2 and Figure 3. We encode this static knowledge in FLORA-2 as follows:

```
// petri net

publish_book:business_process.

publish_book[inPlace(receive_manuscript)->{p0}].
publish_book[inPlace(proofread)->{p1}].
publish_book[inPlace(send_press_release)->{p2}].
```

```
publish_book[inPlace(and_split)->{p2}].
publish_book[inPlace(revise_text)->{p3}].
publish_book[inPlace(revise_visual)->{p5}].
publish_book[inPlace(and_join)->{p4,p6}].

publish_book[outPlace(receive_manuscript)->{p1}].
publish_book[outPlace(proofread)->{p2}].
publish_book[outPlace(send_press_release)->{pg}].
publish_book[outPlace(and_split)->{p3,p5}].
publish_book[outPlace(revise_text)->{p4}].
publish_book[outPlace(revise_visual)->{p6}].
publish_book[outPlace(and_join)->{p1}].

publish_book[duration(proofread)->4].
publish_book[duration(revise_text)->1].
publish_book[duration(revise_visual)->5].
publish_book[duration(send_press_release)->1].

// organizational model

publisher_om:organisational_model.

publisher_om[role(publisher)->{amy}].
publisher_om[role(copy_editor)->{amy,glen,drew}].
publisher_om[role(graphic_artist)->{oliver}].
publisher_om[role(admin_asst)->{evan}].
```

In the next step, we will encode the resource assignment rules in FLORA-2.

4.4 Resource Allocation in CLP-FD

The CLP-FD solver is an instance of the general Constraint Logic Programming scheme introduced in [28]. This approach is useful for modeling discrete optimization and verification problems such as scheduling, planning, packing, timetabling etc. where the values for certain variables are picked from some pre-defined domains so that the given constraints on the variables are all satisfied. We refer to [11] for further details of CLP-FD.

The solver is available as a library module and can be loaded with a query

```
:- use_module(library(clpfd)).
```

4.4.1 Problem Encoding in CLP-FD

In the next step, we will provide the resource assignment rules in CLP-FD.

4.5 Elicitation of a Unified Formalism to Solve the Reasoning Tasks in SHAPE

In SHAPE project we formalize process models and regulations on safety-critical systems by developing semantic models for processes, resources and compliance rules (cf. Deliverable 4.1 and 4.2). These semantic models combined with the process-relevant data mined from process logs (cf. Deliverable 3.1 and 3.2) allow us to perform automated reasoning tasks such as resource allocation and recovery planning. We plan to use ASP for these reasoning tasks after our comparison to other two formalisms that we investigate in this section. These advantages are mainly as follows

- Compact, declarative and intuitive problem encoding
- Rapid prototyping and easy maintenance (e.g., no need to define heuristics)
- Complex reasoning modes (e.g., weight optimization)
- Ability to model effectively incomplete specifications
- Efficient solvers (e.g., *clingo*)

In the literature, ASP is preferable when the size of the problem does not explode the grounding of the program [16, 2, 20]. The experiments in Section 5 show that our resource allocation encoding in ASP is applicable to the problems of business processes at a real-world scale. We will further investigate options for enhancing the performance of our ASP programs by applying symmetry breaking [17] and configuring *clingo*.

5 Performance Evaluation of Formalisms

This section investigates the performance comparisons of our automated resource allocation encodings in iASP, Transaction Logic and CLP-FD. The problems are encoded in Section 4.

5.1 Benchmark Design for Resource Allocation Task

We design a benchmark for evaluating time and memory requirements of our automated resource allocation encoding in ASP [24] with respect to the size of given problems. The benchmark consists of 3 steps:

Steps:

1. Translation of PNML files into the input language of the ASP solver
2. Creating problem instances for the benchmark
3. Running the generated problem instances and collecting statistics

5.1.1 Translating PNML into Predicates

We translated PNML files representing a Petri net in XML (i.e. Listing 2), into predicates. In order to achieve this, we used Python programming language by taking advantage of its ElementTree module for parsing the PNML file and its regular expression module for isolating structured strings.

Listing 2 PNML file example

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<pnml>
   <net id="Net-One" type="Petri net">
      <place id="2_in"/>
      <transition id="a2">
      <name>
         <value >bill of exchange payable posted</value>
         <text >bill of exchange payable posted</text>
      </name>
   </transition>
   <arc id="from_2_in_to_a2" source="2_in" target="a2">
      <inscription>
         <value >1 </value>
         <text >1</text>
      </inscription>
   </arc>
   <arc id="from_a2_to_2_out" source="a2" target="2_out">
      <inscription>
         <value >1 </value>
         <text >1</text>
      </inscription>
   </arc>
   </net>
</pnml>
```

Output of the Listing 2 through the python script:

```
activity(a2).
inPlace(a2_in,a2).
outPlace(a2_out,a2).
```

5.1.2 Creating Problem Instances

The variables identified for creating the problem instances for the benchmark are as follows:

- A business process

- An organizational model
  - n_l : number of roles
  - n_r : number of persons
- Constraints related to the organizational model
- Constraints related to time
  - n_{tc} : number of activities that have duration
  - n_{mtc} : maximum value for estimated duration of an activity
  - n_{ml} : number of roles with temporal constraints
  - n_{mlc} : maximum value for a role-activity time constraint
  - n_{mr} : number of resources with temporal constraints
  - n_{mrc} : maximum value for a resource-activity time constraint
- Benchmark parameters
  - n_{rep} : number of repetitions of each problem instance
  - n_f : number of firings for the shortest solution in the given business process
  - n_{ri} : number of running instances in each problem instance
  - i_{start}, $i2_{start}$,... : starting time for each running instance

First, we need to invent an organizational model for each problem instance (resources and their roles). Then, temporal constraints are generated by complying with the given minimum and maximum values. Afterwards, activity-role and activity-time constraints should be assigned with respect to the following rules:

- Each activity can be executed by the members of one role
  - Using the predicate `canExecute(Role,Resource)`
- Each resource must have at least one role
  - Using the predicate `hasRole(Resource,Role)`
- Activities may have default execution times
  - Using the predicate `timeActivity(Activity,Duration)`
- Roles estimated execution times of activities
  - Using the predicate `timeActivityRole(Activity,Role,Duration)`
- Resources may have estimated execution times of activities
  - Using the predicate `timeActivityResource(Activity,Resource,Duration)`

Then, resources, roles, and temporal constraints are generated according to given parameters. An example problem instance is as follows:

```
person(r_UCUV).
person(r_OTNK).
hasRole(r_UCUV,l_TVNA).
hasRole(r_OTNK,l_MQFG).
canExecute(l_MQFG,a2).
timeActivity(a2,3).
timeActivityRole(a2,l_MQFG,6).
```

```
timeActivityResource(a2,r_OTNK,6).
```

The parameters can be provided in two different ways. If the values are given as exact values, one problem instance is generated. An example set of parameters for generating one problem instance is provided in Table 2.

n_l	n_{mtc}	n_{rep}	n_{ml}	n_{tc}	n_{mr}	n_{mlc}	n_f	n_{mrc}	n_r
5	10	3	3	3	3	10	10	10	10

Table 2 Static variables example

There is a second way for generating more than one problem instance at a time. Each parameter can be set in a range of values when an initial value, an incremental value and a step size is provided. For instance, the dynamic parameter assignment to *n_ri* and c_i by using the values in Table 3 assigns the values $\{2, 4, 6\}$ to *n_ri*, and the values $\{\{0, 5\}, \{0, 5, 10, 15\}, \{0, 5, 10, 15, 20, 25\}\}$ to c_i.

n_{ri}			c_i	
initial value	increment	step	initial value	increment
2	2	3	0	5

Table 3 Dynamic variables example

5.1.3 Collecting Statistics

After generating problem instances, we run the iASP solver *clingo*, the Transaction Logic program solver *Flora-2* and the CLP-FD solver *SWI-Prolog* using the same problem instance files. We collect the following values as output:

- Performance statistics (i.e. time and memory requirement of each solution)
- Solver statistics describing the problem size (i.e. number of atoms)

Using the iASP encoding in Appendix B, we ran two sets of problems. The results of the first set is provided in Table 4. We expected that the length of the business process has a direct effect on the difficulty of the respective allocation problem. The two columns, the number of steps(`#step`) and `CPUtime` verifies our prediction.

The results of the second set of experiments are provided in Table 5. These results show that as the number of parallel instances increases, finding a solution to the problem instance gets harder. The columns the number of parallel instances(*#ins*) and `CPU time` illustrates this phenomenon.

We will provide the performance results of resource allocation encodings in Transaction Logic and CLP-FD once we finalize the problem definitions in Section 4.

#step	#atom	#rule	#var	#constr.	CPU time	real time	mem
8	1068	11298	1010	7859	0.06	1.12	6.6
8	1281	14896	1634	11977	0.06	0.78	6.6
8	1461	14574	1655	11786	0.08	1.03	6.3
12	2353	37844	2402	24251	0.17	1.09	7.9
12	2435	37859	2411	24276	0.17	1.15	7.7
12	6319	255347	26604	232400	0.98	4.95	20.9
16	10156	395231	12544	225873	2.14	10.89	19.4
16	10445	663806	18449	383532	3.22	16.12	25.7
16	8461	396414	15657	248491	1.74	8.31	19.2
20	15934	2186743	19728	1147794	11.13	39.02	50.7
20	13058	730681	24984	447554	3.79	12.4	26.9
20	18939	1562029	31821	870997	10.11	32.84	41.6
24	19502	1548062	41059	913440	9.78	34.02	45.4
24	30686	6645835	115880	3820071	45.79	140.28	141.5
24	23813	1439256	65554	896135	14.95	43.77	51.2

Table 4 Process length vs solution difficulty (All times in seconds, memory usage in MB)

6 Preprocessing Business Processes for Improving Feasibility and Scalability

Solving NP-hard reasoning problems related to business processes, such as time-optimal resource allocation problem, becomes computationally challenging and time consuming when the problem size is big. For this reason, such problems become inapplicable in real-life scenarios which require fast computation of solutions, as in SHAPE project. In this section we will describe a method for dividing a business process into sub-processes and apply our resource allocation method on the resulting sub-processes. The expected performance gain of this novel divide-and-conquer approach will be presented by a quantitative experiment and a qualitative discussion in the next version of this deliverable.

7 Summary and Future Work

In this deliverable we provided an overview on semantic models for mining and monitoring process-relevant data (cf. Section 2) and identified possible reasoning

#ins.	#atom	#rule	#var	#constr.	cputime	real time	mem
1	1446	21220	2173	17190	0.09	0.63	7.4
1	1395	17986	1359	12031	0.09	0.75	6.8
1	1918	29472	2057	19595	0.12	1.04	7.6
2	4309	56182	5246	41567	0.24	1.83	9.7
2	4467	85804	4043	51039	0.36	1.73	10
2	9975	414389	10100	230077	1.95	6.13	20.9
3	11346	230521	7710	129330	0.95	3.9	15.9
3	9614	484246	10781	271238	1.74	5.49	21.6
3	13287	337662	14057	199740	1.91	6.03	20.6
4	20070	258373	15884	160442	1.67	3.79	21.6
4	32540	1029609	33796	582319	6.79	17.9	46.6
5	20876	1120307	25641	620692	6.28	17.18	43.7
5	19238	931543	23605	520943	5.02	12.93	36
5	28867	686832	24870	379018	4.96	13.22	33.5
6	52870	23499944	82876	11036880	174.94	391.43	372.4
6	66338	36491275	92908	18782619	256.1	259.47	554.6

Table 5 Number of running instances vs solution difficulty (All times in seconds, memory usage in MB)

related tasks that can be useful in the scope of SHAPE project (cf. Section 3). Resource allocation encodings are defined in different formalisms(cf. Section 4). We aim at a publication on qualitative and quantitative comparison of three formalisms (iASP, Transaction Logic and CLP-FD), which we outlined in Section 4 and Section 5. This document also frames a novel method for increasing applicability of reasoning tasks in business process-related data (cf. Section 6) which can be considered for another publication. Some of the possible venues for these publication are as follows:

- International Conference on Automated Planning and Scheduling (ICAPS)
  - June 12-17, 2016, London, UK
  - 22 November 2015 – submission deadline
- International Joint Conference on Artificial Intelligence (IJCAI)
  - 12-15 July 2016, New York, USA
  - 2 February 2016 – submission deadline
- European Conference on Artificial Intelligence (ECAI)
  - August 29-September 2, 2016, The Hague, Netherlands
  - 15 April 2016 – submission deadline
- International Conference on Advanced Information Systems Engineering (CAISE)
  - 13-17 June 2016, Ljubljana, Slovenia
  - 30 November 2015 – submission deadline
- International Conference on Business Information Systems (BIS)

- 6-8 July 2016, Leipzig, Germany
- 8 January 2016 – submission deadline

- International Conference on Business Process Management (BPM)
  - 18-22 September 2016, Rio de Janeiro, Brazil
  - 14 March 2016 – submission deadline

The document will be updated as we make progress on the ongoing work defined in respective sections.

Appendices

A Visualization of Automated Resource Allocation Solutions

In this part, we produce Gantt charts from the output of the resource allocation program. Basically, there are two goals:

- Obtaining the allocation results in a graphics format, and
- Seeing the effects of changes in the allocation encoding instantly

The output of the ASP program is parsed using Python and translated into the input language of GNUPlot, which is a free, command-driven, interactive, function and data plotting program. The results in Table 6 are translated into Gantt charts in Figure 4, Figure 5 and Figure 6:

In Figure 4, each lane shows the resource allocation for different instances. In every lane, the duration of activity executions are represented by the rectangles while the activities are encoded in colors and described in the margin.

Figure 5 shows the overall busy time of resources. Each lane corresponds to instances while the colors represents allocated resources.

Resource	*Activity*	*TimeStart*	*TimeEnd*	*Instance*
amy	t_m	0	0	i1
amy	t_PR	0	2	i1
drew	t_RT	2	4	i1
oliver	t_RV	2	7	i1
drew	t_PR	7	9	i1
evan	t_SPR	9	10	i1
amy	t_m	6	6	i2
amy	t_PR	6	8	i2
amy	t_RT	8	9	i2
oliver	t_RV	8	13	i2
glen	t_PR	13	18	i2
drew	t_RT	18	20	i2
oliver	t_RV	18	23	i2
amy	t_PR	23	25	i2
drew	t_RT	25	27	i2
oliver	t_RV	25	30	i2
amy	t_PR	30	32	i2
evan	t_SPR	32	33	i2
amy	t_m	11	11	i3
drew	t_PR	11	13	i3
drew	t_RT	13	15	i3
oliver	t_RV	13	18	i3
amy	t_PR	18	20	i3
evan	t_SPR	20	21	i3

Table 6 Example output of allocation program

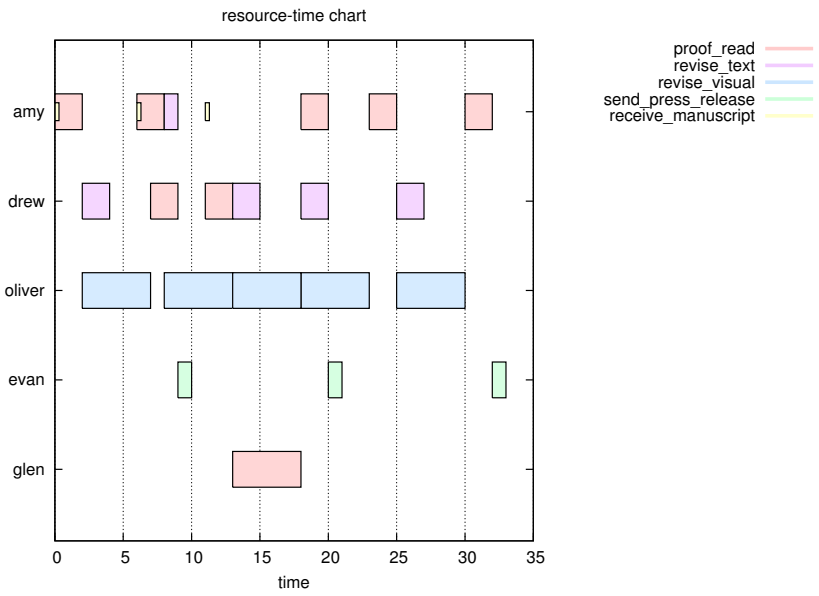

Figure 6 Chart summarizing the availability of resources

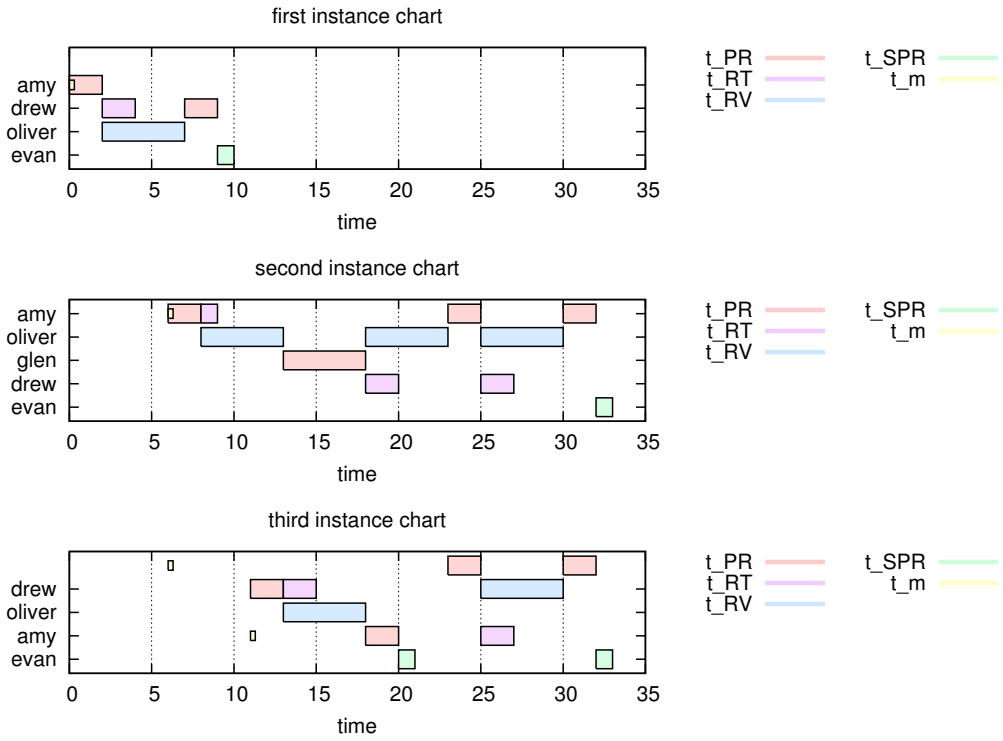

Figure 4 Chart showing the resource activity for all instances

B iASP Encoding for Automated Resource Allocation

We provide the improved and enhanced iASP encoding version for automated resource allocation.

Listing 3 Static knowledge

```
#program base.

% default duration of a transition
firingDelay(A,B,0) :- not activityDuration(A,B,_),
                      activityTransition(A,B).
firingDelay(A,B,D) :- activityDuration(A,B,D).

% default resource-activity duration preference handling
defaultRAD(R,A,B,D) :- resourceActivityDuration(R,A,B,D).
defaultRAD(R,A,B,D) :- roleActivityDuration(L,A,B,D), hasRole(R,L),
                       not resourceActivityDuration(R,A,B,_),
                       canExecute(L,A,B).
defaultRAD(R,A,B,D) :- firingDelay(A,B,D), hasRole(R,L),
                       not resourceActivityDuration(R,A,B,_),
                       not roleActivityDuration(L,A,B,_),
                       canExecute(L,A,B).
```

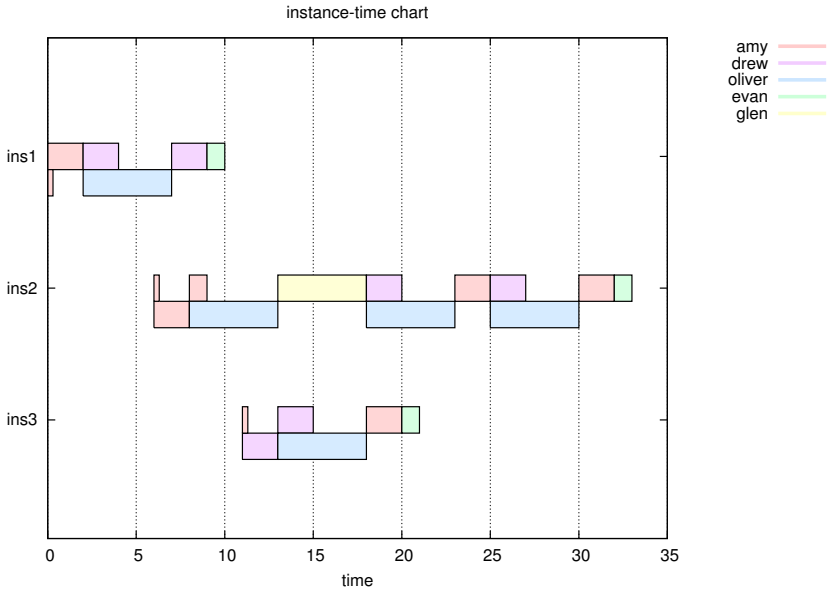

Figure 5 Chart summarizing the activities in all instances

Listing 4 Cumulative knowledge

```
#program cumulative(s).

%%% PETRI NET DYNAMINCS %%%

% generate action fire
{fire(T,s,B,I) : inPlace(P,T,B), bpInstance(B,I)}.

% fire precondition: if no token at preceding P, then can't fire
:- fire(T,s,B,I), bpInstance(B,I), inPlace(P,T,B),
   not tokenAt(P,s,B,I).

% fire effect: if fire at s-1, token at next place
tokenAt(P,s,B,I) :- fire(T,s-1,B,I), outPlace(P,T,B), bpInstance(B,I).

% fire constraint only 1 succeeding transition can fire
:- inPlace(P,T1,B), inPlace(P,T2,B), T1!=T2, fire(T1,s,B,I),
   fire(T2,s,B,I), bpInstance(B,I).

% inertia: tokenAt(P,s): if not fire token remains at its place
consumeToken(P,s,B,I) :- inPlace(P,T,B), fire(T,s,B,I),
                         bpInstance(B,I).
tokenAt(P,s,B,I) :- tokenAt(P,s-1,B,I), not consumeToken(P,s-1,B,I).

%%% TIME MANAGEMENT %%%

% Max time
```

```
maxTimeAtInPlace(P,T,s,B,I) :- inPlace(P,T,B),
                               not greaterTimeExistsAtInPlace(P,T,s,B,I),
                               fire(T,s,B,I), bpInstance(B,I).
greaterTimeExistsAtInPlace(P1,T,s,B,I) :- inPlace(P1,T,B),
         inPlace(P2,T,B), fire(T,s,B,I), timeAt(P1,C1,s,B,I),
         timeAt(P2,C2,s,B,I), P1!=P2, C1<C2, bpInstance(B,I).

% fire effect on time (T is not an activity)
timeAt(P2,X,s,B,I) :- not activityTransition(T,B), fire(T,s-1,B,I),
                      maxTimeAtInPlace(P,T,s-1,B,I), timeAt(P,X,s-1,B,I),
                      outPlace(P2,T,B), bpInstance(B,I).

% fire effect on time (T is an activity)
timeAt(P2,C2,s,B,I) :- activityTransition(T,B),
                       assign(R,T,C1,C2,s-1,B,I), fire(T,s-1,B,I),
                       outPlace(P2,T,B), bpInstance(B,I).

% no fire wait
timeAt(P,C,s,B,I) :- timeAt(P,C,s-1,B,I), inPlace(P,T,B),
                     not activityTransition(T,B),
                     not consumeToken(P,s-1,B,I), bpInstance(B,I).

% time relaxation: if activity : relaxation is possible
timeAt(P,C+1,s,B,I) :- timeAt(P,C,s-1,B,I), inPlace(P,T,B),
                       activityTransition(T,B),
                       not consumeToken(P,s-1,B,I), bpInstance(B,I).

%%% RESOURCE ASSIGNMENT %%%

% assign each activity to a person
{assign(R,A,C,C+D,s,B,I): defaultRAD(R,A,B,D)} :- inPlace(P1,A,B),
      timeAt(P1,C,s,B,I), activityTransition(A,B), bpInstance(B,I).

% an activity can not be fired before assigned
:- not assign(_,A,_,_,s,B,I), fire(A,s,B,I), activityTransition(A,B),
   bpInstance(B,I).

% can not assign same task to another person (task identifier: T-I-S)
:- assign(R,A,C1,C2,S1,B,I), assign(R1,A,C1,C3,S2,B,I), R!=R1.

% can not assign same person to another activity at the same time
:- assign(R,A1,C1,C2,S1,B,I), assign(R,A2,C1,C3,S2,B,I), C1<C2, C1<C3,
   A1!=A2.
:- assign(R,A1,C1,C2,S1,B,I1), assign(R,A2,C1,C3,S2,B,I2), C1<C2,
   C1<C3, A1!=A2, I1!=I2.
:- assign(R,A1,C1,C2,S1,B1,I1), assign(R,A2,C1,C3,S2,B2,I2), C1<C2,
   C1<C3, B1!=B2.
```

```
% can not assign same person to same activity at the same time in
% another instance
:- assign(R,A,C1,C2,S1,B,I1), assign(R,A,C1,C3,S2,B,I2), C1<C2, C1<C3,
   I1!=I2.
:- assign(R,A,C1,C2,S1,B1,I1), assign(R,A,C1,C3,S2,B2,I2), C1<C2,
   C1<C3, B1!=B2.

% can not assign when assignments overlap
:- assign(R,T,Y1,Y2,S1,B1,I1), assign(R,T2,X1,X2,S2,B2,I2), X1>Y1,
   X1<Y2.
:- assign(R,T,Y1,Y2,S1,B1,I1), assign(R,T2,X1,X2,S2,B2,I2), X2<Y2,
   X2>Y1.
```

References

1 Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. *Mining process models from workflow logs*. Springer, 1998.

2 Markus Aschinger, Conrad Drescher, Gerhard Friedrich, Georg Gottlob, Peter Jeavons, Anna Ryabokon, and Evgenij Thorstensen. Optimization methods for the partner units problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 4–19. Springer, 2011.

3 Saimir Bala, Cristina Cabanillas, Jan Mendling, and Axel Polleres. Mining processes, resource consumption and witnesses for task completion from logs. 2015.

4 Saimir Bala, Cristina Cabanillas, Jan Mendling, and Axel Polleres. Requirements for process, resource and compliance rules extraction from text. 2015.

5 Jörg Becker, Patrick Delfmann, Mathias Eggert, and Sebastian Schwittay. Generalizability and applicability of model-based business process compliance-checking approaches—a state-of-the-art analysis and research roadmap. *BuR-Business Research*, 5(2):221–247, 2012.

6 AJ Bonner and M Kifer. Results on reasoning about action in transaction logic. *Submitted for Publication*, 1998.

7 Anthony J Bonner and Michael Kifer. A logic for programming database transactions. In *Logics for databases and information systems*, pages 117–166. Springer, 1998.

8 Cristina Cabanillas, Giray Havur, Jan Mendling, and Axel Polleres. State-of-the art on existing models for processes, resources, constraints and security, and their underlying formalisms. 2015.

9 Cristina Cabanillas, Manuel Resinas, and Antonio Ruiz Cortés. Specification and Automated Design-Time Analysis of the Business Process Human Resource Perspective. *Inf. Syst.*, page In press., 2015.

10 Cristina Cabanillas, Manuel Resinas, and Antonio Ruiz-Cortés. Hints on how to face business process compliance. *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos*, 4(4):26–32, 2010.

11 Mats Carlsson. *SICStus Prolog User's Manual 4.3: Core reference documentation*. BoD–Books on Demand, 2014.

12 Weidong Chen, Michael Kifer, and David S Warren. Hilog: A foundation for higher-order logic programming. *The Journal of Logic Programming*, 15(3):187–230, 1993.

13 Jonathan E Cook and Alexander L Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7(3):215–249, 1998.

14 Anindya Datta. Automating the discovery of as-is business process models: Probabilistic and algorithmic approaches. *Information Systems Research*, 9(3):275–301, 1998.

15 AK Alves De Medeiros, Carlos Pedrinaci, Wil MP Van der Aalst, John Domingue, Minseok Song, Anne Rozinat, Barry Norton, and Liliana Cabral. An outlook on semantic business process mining and monitoring. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, pages 1244–1255. Springer, 2007.

16 Agostino Dovier, Andrea Formisano, and Enrico Pontelli. A comparison of clp (fd) and asp solutions to np-complete problems. In *Logic Programming*, pages 67–82. Springer, 2005.

17 Christian Drescher, Oana Tifrea, and Toby Walsh. Symmetry-breaking answer set solving. *CoRR*, abs/1008.1809, 2010.

18 Marlon Dumas, Wil M Van der Aalst, and Arthur H Ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, 2005.

19 Amal Elgammal, Oktay Turetken, Willem-Jan van den Heuvel, and Mike Papazoglou. On the formal specification of regulatory compliance: a comparative analysis. In *Service-Oriented Computing*, pages 27–38. Springer, 2011.

20 Marco Gavanelli, Maddalena Nonato, Andrea Peano, Stefano Alvisi, and Marco Franchini. An asp approach for the valves positioning optimization in a water distribution system. In *CILC*, pages 134–148, 2012.

21 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. Engineering an incremental ASP solver. In *Logic Programming*, pages 190–205. Springer, 2008.

22 Armin Haller, Mateusz Marmolowski, Eyal Oren, and Walid Gaaloul. oxpdl: a process model exchange ontology. Technical report, Citeseer, 2007.

23 Mustafa Hashmi and Guido Governatori. A methodological evaluation of business process compliance management frameworks. In *Asia Pacific Business Process Management*, pages 106–115. Springer, 2013.

24 Giray Havur, Cristina Cabanillas, Axel Polleres, and Jan Mendling. Automated Resource Allocation in Business Processes with Answer Set Programming. *Business Process Management*, submitted to BPM 2015 (on 2015.03.22).

25 Thorsten Humberg, Christian Wessel, Daniel Poggenpohl, Sven Wenzel, Thomas Ruhroth, and Jan Jürjens. Ontology-based analysis of compliance and regulatory requirements of business processes. In *CLOSER*, pages 553–561, 2013.

26 Thorsten Humberg, Christian Wessel, Daniel Poggenpohl, Sven Wenzel, Thomas Ruhroth, and Jan Jürjens. Using ontologies to analyze compliance requirements of cloud-based processes. In *Cloud Computing and Services Science*, pages 36–51. Springer, 2014.

27 Jon Espen Ingvaldsen and Jon Atle Gulla. Industrial application of semantic process mining. *Enterprise Information Systems*, 6(2):139–163, 2012.

28 Joxan Jaffar and Michael J Maher. Constraint logic programming: A survey. *The journal of logic programming*, 19:503–581, 1994.

29 Michael Kifer. Deductive and object data languages: a quest for integration. In *Deductive and Object-Oriented Databases*, pages 187–212. Springer, 1995.

30 Michael Kifer. Nonmonotonic reasoning in flora-2. In *Logic Programming and Nonmonotonic Reasoning*, pages 1–12. Springer, 2005.

31 Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM (JACM)*, 42(4):741–843, 1995.

32 Linh Thao Ly, Stefanie Rinderle-Ma, David Knuplesch, and Peter Dadam. Monitoring business process compliance using compliance rule graphs. In *On the move to meaningful internet systems: OTM 2011*, pages 82–99. Springer, 2011.

33 Carlos Pedrinaci and John Domingue. Towards an ontology for process monitoring and mining. In *CEUR Workshop Proceedings*, volume 251, pages 76–87, 2007.

34 Carlos Pedrinaci, John Domingue, and Ana Karla Alves de Medeiros. *A core ontology for business process analysis*. Springer, 2008.

35 Bill Tsoumas and Dimitris Gritzalis. Towards an ontology-based security management. In *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, volume 1, pages 985–992. IEEE, 2006.

36 Branimir Wetzstein, Zhilei Ma, Agata Filipowska, Monika Kaczmarek, Sami Bhiri, Silvestre Losada, Jose-Manuel Lopez-Cob, and Laurent Cicurel. Semantic business process management: A lifecycle based requirements analysis. In *SBPM*, 2007.

37 Guizhen Yang, Michael Kifer, and Chang Zhao. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pages 671–688. Springer, 2003.

38 Michael Zur Muehlen. Process-driven management information systems combining data warehouses and workflow technology. In *Proceedings of the International Conference on Electronic Commerce Research (ICECR-4)*, pages 550–566. Citeseer, 2001.