Automated Attribute Inference in Complex Service Workflows Based on Sharing Analysis

Dragan Ivanović
idragan@clip.dia.fi.upm.es
Universidad Politécnica de Madrid (UPM)

Manuel Carro Manuel Hermenegildo
{mcarro,herme}@fi.upm.es
manuel.{carro,hermenegildo}@imdea.org
Universidad Politécnica de Madrid (UPM) and IMDEA Software Institute

***Abstract*—The properties of data and activities in business processes can be used to greatly facilitate several relevant tasks performed at design- and run-time, such as fragmentation, compliance checking, or top-down design. Business processes are often described using workflows. We present an approach for mechanically inferring business domain-specific attributes of workflow components (including data items, activities, and elements of sub-workflows), taking as starting point known attributes of workflow inputs and the structure of the workflow. We achieve this by modeling these components as concepts and applying sharing analysis to a Horn clause-based representation of the workflow. The analysis is applicable to workflows featuring complex control and data dependencies, embedded control constructs, such as loops and branches, and embedded component services.**

***Keywords*-workflow; business process; service composition; horn clause; static analysis;**

I. Introduction

Service-Oriented Computing stresses interoperability among services, i.e., among loosely coupled and platform-independent software components with standardized data type descriptions and interfaces. While back-end services essentially implement indivisible operations, service compositions express higher-level, potentially long-running business processes in an executable form, often across organizational boundaries. Compositions are often described by specifying workflows that describe links between activities and the routing of data. This is done using a language that allows process modelers and designers to capture the essential elements of business logic and processing requirements [17], [9], [21], [20].

Inferring properties of workflows is valuable for several important design- and run-time activities such as transforming and refactoring workflows, identifying patterns, facilitating run-time instrumentation, reshaping, or performing distributed enactment of service compositions [15].

In this paper we focus on the inference of user-defined business domain-specific attributes for data items and activities in service orchestration workflows featuring rich control structures (branching, loops, and-split-join, (x)or-split-join), data dependencies, and component services with a (partially) known structure. The user-defined attributes that describe input data are organized into contexts and concept lattices from Formal Concept Analysis (FCA) [5]. Inference based on static program analysis techniques (see [16] for a general introduction) allows us to further enrich attribute information by automatically deriving emerging properties which are implicit in, but not evident from, the input data attributes and the workflow structure. Static analysis results are mapped back to the FCA representation framework, and can then be used to feed, for instance, fragmentation algorithms [10], [11].

This paper is a natural continuation of earlier work [8] which presented the basis of the application of sharing analysis to workflows. However that work did not specify from where and how the entities subject to sharing were obtained, or where its results were applied.

Work related to our approach includes several proposals that address the problem of information control flow from the viewpoint of decentralized process/workflow execution in complex scientific [22] or business domains [4], [23]. In that work the attributes of data items and activities carry the particular semantics of privacy or confidentiality levels from the viewpoint of multiple participants. With respect to that work, this paper adds two dimensions of flexibility. First, we employ a type of static analysis, sharing analysis, based on abstract interpretation [2], [6], which is applicable to a generalized user-defined semantics of the data and activity attributes (e.g., describing data components or some form of "data quality" for the activities and the entire workflow). Second, we relax (sometimes significantly) the constraints on the control structure of workflows that can be treated, by allowing complex and nested control structures that are commonly found in practice (such as loops, branches, splits and joins). In that respect, the workflow representation we use avoids on purpose depending on a particular business process language unlike some other analyses [12]. References to related work connected to particular aspects of our approach are given throughout the subsequent sections.

The paper proceeds as follows. Section II motivates and outlines the approach, and introduces the medication prescription workflow example that is used throughout the text. Section III introduces the notion of concept lattices and describes how such lattices are derived from an assignment of user-defined attributes to a set of input data objects. Section IV deals with turning workflow definitions and the concept lattices

into a form amenable to sharing analysis, and addresses the analysis itself. Section V explains how the results from sharing analysis are interpreted in terms of contexts and the user-defined attributes. Finally, Section VI presents our conclusions.

II. Motivation and Sketch of the Approach

Fig. 1 depicts a simplified example of a drug prescription workflow in BPMN [17] . The process is initiated by the arrival of a patient with an appropriate identification (labeled as x in the figure). Next, two parallel activities (a_1 and a_2) are run to retrieve the patient's medical history and medication record. The data items resulting from these two activities are respectively marked y and z. Additionally, while retrieving the medical history, activity a_1 informs about the stability of the health of the patient. Depending on it, either the last prescription is continued (activity a_3) or new medication is selected (activity a_4). Finally, the treatment of the patient is logged (activity a_5).

Some relevant questions can be raised. For example, is the medical history (y) available to activity a_4? This will depend on what activities a_3 and a_4 do with y (note that a_4 internally executes a loop) and on whether a_3 or a_4 is executed. If a_5 needs that information and it is not available, we have a correctness problem which may lead the workflow to failure. If a_5 should not have that information (for, e.g., privacy reasons) but it can be leaked, there is a potential problem too. More generally, design-time analysis of the characteristics of data and activities in the given workflow can be used to obtain results that are useful for several tasks, of which we highlight just a few:

a) Fragmentation: A workflow can be split into several fragments that can be executed within different organizations. The assignment of activities to fragments can be done according to different criteria. In our example, a healthcare organization may wish to delegate parts of the workflow to business partners. Confidentiality requirements may require that either the medical history or the medication record (or both) be hidden from some of the partners. That would mean separating the activities in Fig. 1 into several swimlanes corresponding to different organizational domains, and assigning activities accordingly.

b) Data Compliance: When services are composed into a business process (such as $a_1 \ldots a_5$ in our example), organizations need to ensure that the information content used by a component service is adequate to implement the desired behavior, and often syntactic compliance with XML message formats is not enough. For example, the Patient ID (which may be a national identity card, a driving license, a passport, etc.) may or may not contain at runtime information enough to retrieve, e.g., a medical history. Some potential problems can be detected at design time by tracking data flow between activities and analyzing which pieces of information are shared between activities and which are needed by them.

c) Robust Top-Down Development: Business process modeling can start at a high level, and elaborate components in a top-down fashion. Some components may contain complex structured constructs, or be developed into separate, reusable services. Data flow analysis at the process level can help us identify required features of inputs and outputs from such components. Additional results can be obtained for sub-activities of a component. For instance, in Fig. 2, which exposes a possible structure for workflow component a_4, it would be interesting to derive the attributes of criterion c. The process can be repeatedly applied to the components a_{41} and a_{42} of the sub-workflow.

Fig. 3 depicts our approach for inferring domain-specific attributes for entities in a workflow which can later be used by design-time tools. From the users' perspective, the starting point is a description of the workflow using an appropriate formalism (BPMN, in our example), and an input data *context* (see Fig. 4 and Section III-A): a list of business domain objects which are input to the workflow, described using relevant attributes from the domain. The result is presented as a context that assigns attributes inferred by the analysis to data items and activities. This context can then be inspected by a user or a tool and be used for further analysis of the workflow, transformation, or other design-time tasks.

The initial context is used to set up a concept lattice (Sec. III), which is the main vehicle to prepare the input to the sharing analysis and to interpret its results. The sharing analysis works on a logic program generated from the translation of the workflow definition and the initial concept lattice into Horn clauses and logical variable substitutions, respectively (Sec. IV). Finally (Sec. V), the abstract substitutions which result from the sharing analysis are used to produce a result lattice which in turn is used to generate the final context.

III. From Contexts to Concept Lattices

We will use FCA [3], [5], a mathematical formalism used to represent and analyze data using contexts and concept lattices, to describe properties of the data and activities in the workflow. Due to space constraints, the reader is kindly referred to the existing literature for a more in-depth introduction to FCA.

A. Contexts and Concept Lattices

In a given domain, a *context* is a relationship between (finite) sets of relevant (user-defined) *objects* and *attributes*, and is usually represented as a table (Fig. 4). If O is a set of objects and A a set of attributes, the context is a relation $\rho \subseteq O \times A$. For any object $o \in O$, o' denotes the set of all attributes $a \in A$ such that $o \rho a$. By extension, for any $B \subseteq O$, $B' = \cap_{o \in B} o'$ is the set of all attributes common to all objects in B. Conversely, for any $D \subseteq A$, $D' = \{o \in O : D \subseteq o'\}$ is the set of all objects that have all attributes from D.

In FCA, a set of objects $B \subseteq O$ is said to be a *concept* if $(B')' = B'' = B$. If $D = B'$, it follows that $(D')' = D'' = D$. The operator $(\cdot)''$ is a closure that, when applied to a subset of objects, gives the concept that includes these objects. Speaking intuitively, it ensures that a concept includes all objects that share the same set of attributes. In particular, $(\emptyset)''$ gives the most general concept that contains all objects, including possibly those with no known attributes.

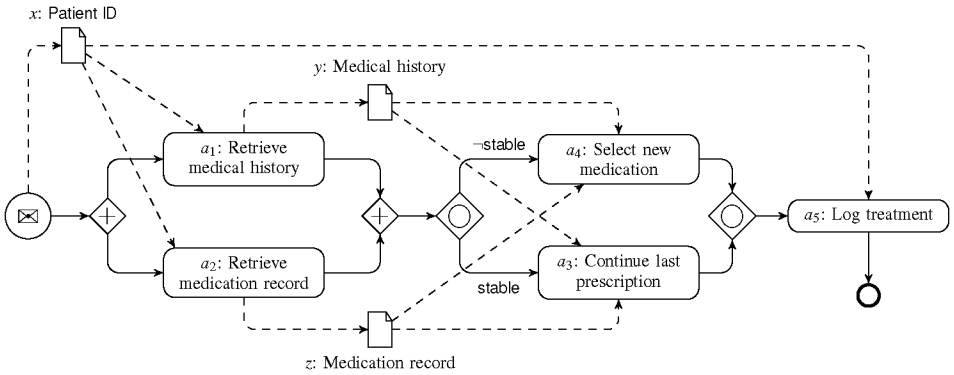


Fig. 1. An example drug prescription workflow.

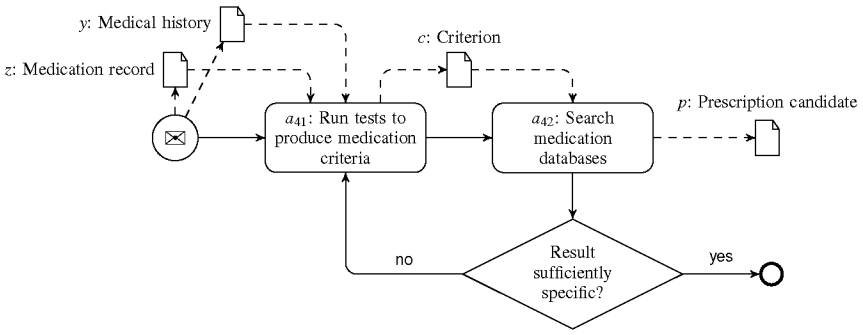


Fig. 2. Selection of new medication.

In order to make concepts useful for analysis, we need to organize them into *concept lattices*. A *lattice* is a mathematical structure $(L, \leq, \vee, \wedge)$ built around a set L (in our case containing concepts from a context), a partial order relation $\leq$, the *least upper bound* (LUB) operation $\vee$, and the *greatest lower bound* (GLB) operation $\wedge$. For arbitrary $x, y \in L$, the element $x \vee y = z$ has the property $x \leq z$ and $y \leq z$, but it is also the least such element, i.e., for any other $w \in L$ such that $x \leq w$ and $y \leq w$, we have $z \leq w$. The case for the greatest lower bound operation $\wedge$ is symmetric. In this paper, we deal only with finite and *complete* lattices, where for any arbitrary non-empty subset of lattice elements the LUB and the GLB exist in L; such lattices have unique greatest ($\top$) and least ($\bot$) elements.

For concept lattices, the ordering relation $\leq$ between two concepts B_1 and B_2 holds iff $B_1 \subseteq B_2$, or, equivalently, iff $B_2' \subseteq B_1'$: a higher concept includes all objects from a lower (or derived) one; lower concepts are derived from higher ones by adding attributes. Consequently, the LUB is obtained using $B_1 \cap B_2$, and the GLB using $B_1 \cup B_2$.

Context lattices are usually represented using a variant of Hasse diagrams (Fig. 5). Nodes correspond to concepts, with the top concept visually on the top, and the bottom concept placed accordingly. The annotations associated with a concept (using dashed lines) show the attributes introduced by the concept (besides the derived attributes from the higher concepts) above the line, and the objects that belong to that concept, but not to any of its derived concepts, below the line. Concepts may have one or both parts of the annotation empty; in the latter case, the annotation is not shown.

Fig. 5 presents the concept lattices for the medical database contexts from Fig. 4. The most general concepts are shown on top of the lattices, and the most specific (empty in both cases) at the bottom.

B. Describing Data with Concept Lattices

The data items that are input to the workflow need to be mapped to the appropriate objects in the input concept lattice. In the case of our example (Fig. 1), we would need to map the Patient ID input data item to either Passport, National ID, Driving License, or Social Security Card. In our example, each of those objects maps to a different concept in the lattice, but in general several objects can map to the same concept.

The prerequisite in order to use concept lattices is to create an adequate context at a level of abstraction that captures enough information to represent all relevant concepts and their attributes. Complex models can always be simplified by keeping only those attributes that really discriminate between different concepts. Existing tools (e.g., ConExp, Lattice Miner, Colibri and others [1]) facilitate the process of eliciting and exploring knowledge using FCA.

A relevant point is that some data sources may not appear explicitly as workflow inputs. In Fig. 1, activities a_1 and a_2 need to access some external source to extract records using the input Patient ID. The attributes of the retrieved records depend on properties of these data sources and therefore, they

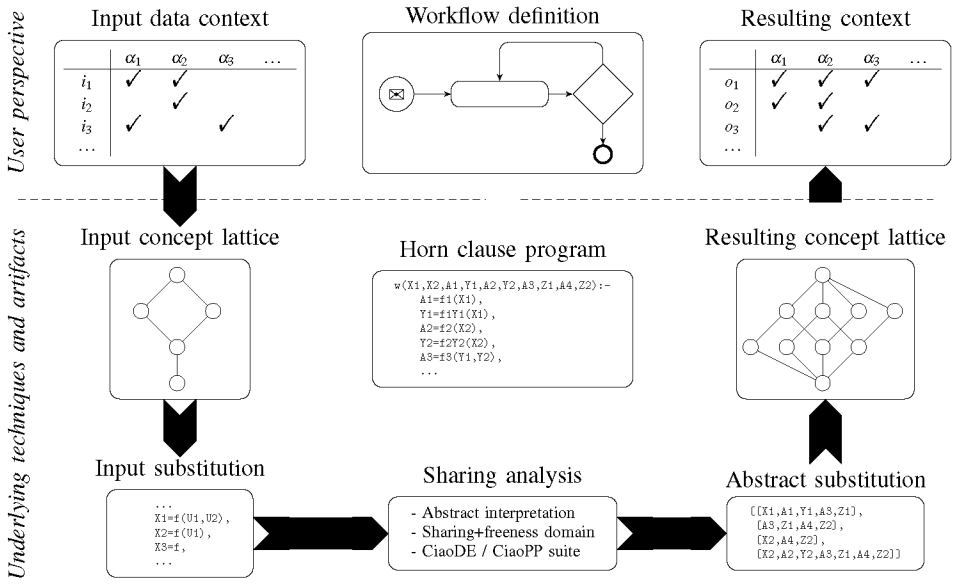


Fig. 3. Overview of the approach.

need to be mapped to appropriate objects (in this case the Medical history and the Medication record from Fig. 5(a)).

IV. Applying Sharing Analysis

Our application of sharing analysis to elicit new knowledge about attributes of the workflow entities is based on three points: (a) representing the control structures of the workflow in a language amenable to analysis, (b) representing data links *and activities* in the workflow as explicit variables, and (c) representing attributes of these entities as additional *hidden variables* which can share with the variables set up in (b). Two variables *share* if there is some object which is reachable from both, maybe following a reference chain. By inferring how runtime variables can share in the programming language representation of the workflow we deduce the runtime attributes of data items and activities in the workflow.

A. Workflows as Horn Clauses

The sharing analysis tools we will use [7], [6] work on logic programs, and therefore the workflow under consideration needs to be represented in the form of a logic program [14]: a series of logical implications which can be operationally understood as stating which subgoals are needed to accomplish a given goal. Note that the translation into a logic program does not need to be operationally equivalent to the initial workflow; it only needs to represent the data flow and data aliasing correctly. We translate data flow into parameter passing and data aliasing into unification of logical variables. Here we build on [8], where examples of translations from an abstract notation for workflows into Horn clauses are given. Due to size constraints we cannot reproduce here the description of the translation. Instead, we kindly direct the reader to [8] for more details. A key ingredient of the translation is representing, for each activity in the workflow, the sets of data items read and written. This vantage point in workflow modeling is shared with the existing approaches to the analysis of soundness of Workflow Nets with Data (WFD nets) [18], as well as with the approaches to verifying validity of business process specifications using data-flow matrices [19]. However, unlike those higher-level conceptual views that are mainly concerned with various aspects of business process management, in our case we aim at inferring properties on a more technical level that takes into account details of (possibly complex and nested) control flow and data operations. For that purpose, WFD nets or UML activity diagrams are not sufficiently informative, while Horn clauses provide an adequate computation paradigm that has been extensively studied.

	Symptoms	Tests	Coverage
Medical history	✓	✓	
Medication record	✓		✓

(a) Characteristics of medical databases.

	Name	Address	PIN	SSN
Passport	✓		✓	
National Id Card	✓	✓	✓	
Driving License	✓	✓		
Social Security Card	✓	✓		✓

(b) Types of identity documents.

Fig. 4. Two examples of contexts.

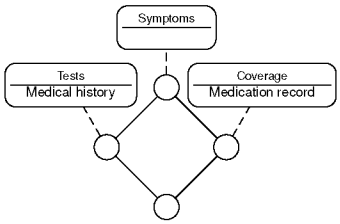
As an illustration, we give here a commented translation of our workflow written in BPMN (Figs. 1 and 2) into Horn clauses. The translation for this case is given in Fig. 6 using Prolog syntax, and will be explained in the following text.

Lines 1-8 are a Horn clause that defines the predicate `w` for the workflow with a list of comma-separated *goals* in the body (lines 2-8) following the definition symbol "`:-`". Character "`%`"

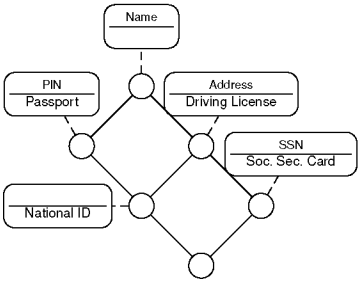
introduces a comment line and helps relate parts of the body with activities from Fig. 1. Arguments to `w` are listed inside parentheses in line 1; following the Prolog syntax convention, *variable names* start with an uppercase letter. These variables correspond to the names of activities and data items from the BPMN diagrams, including those data sources which are not explicit in the diagram: in this case the databases from which the medical history and medical record are retrieved from. These databases have to be characterized in the result lattice, and in order to do so they are assigned variables `D` and `E`. To expose results for the sub-workflow from Fig. 2, the arguments to `w` include all activities and data items from the sub-workflow.

Simple activities (including external service invocation) are translated as goals of the shape $\alpha = \varphi(\Gamma)$, where α stands for an activity, Γ is a sequence of all data items used as inputs by the activity, and φ is an uninterpreted function symbol (whose particular name is not relevant for sharing analysis, and has been chosen to recall the activity name). This is followed by goals of the same shape where the left-hand side of "=" stands for data item produced by the activity, and the Γ part on the right hand side includes data items used in the computation of the data item. For instance, goals `A1=f1(X,D)` and `Y=f1_Y(X,D)` in lines 2 and 3 represent the fact that a_1 uses data items x and d as inputs, to produce data item y. The only exception in `w` is the goal for sub-workflow a_4 (line 7) to be discussed below.

The ordering of activities in the body of a clause must respect data dependencies, in the sense that data items should appear as arguments in a goal only if they are produced by a preceding activity. The ordering also needs to respect control dependencies arising from explicit sequences and joins (AND and OR). Otherwise, as in the AND-split case, the relative order of activities as goals in the body of a Horn clause is



(a) Concept lattice for medical databases.



(b) Concept lattice for identity documents.

Fig. 5. Concept lattices for contexts from Fig. 4.

```
1  w(X,D,E,A1,Y,A2,Z,A3,A4,A41,C,A42,P,A5):-
       A1=f1(X,D), % a_1
3      Y=f1_Y(X,D),
       A2=f2(X,E), % a_2
5      Z=f2_Z(X,E),
       A3=f3(Y,Z), % a_3
7      a_4(Y,Z,A4,A41,C,A42,P), % a_4
       A5=f5(X). % a_5
9
   a_4(Y,Z,A4,A41,C,A42,P):-
11     w2(Y,Z,A41,C2,A42,P2),
       A4=f(P2),
13     a_4x(Y,Z,C2,P2,C,P,A4,A41,A42).

15 a_4x(_,_,C,P,C,P,_,_,_).
   a_4x(X,Z,_,_,C,P,A4,A41,A42):-
17     a_4(X,Z,A4,A41,C,A42,P).

19 w2(Y,Z,A41,C,A42,P):-
       A41=f41(Y,Z), % a_41
21     C=f41_C(Y),
       A42=f42(C), % a_42
23     P=f42_P(C).
```

Fig. 6. Horn clause program encoding for the medication prescription workflow.

not significant from the sharing analysis point of view [8], and one such ordering can always be found, unless there is a race condition between potentially parallelized activities that try to read/write the same data item. This is not the case in our example and the possibility of this happening can be statically detected from the structure of the workflow. Also note that we include both branches of the XOR-split, since the data in a_5 can be affected by either one of them. The workflow for the component activity a_4 is effectively a *repeat-until* loop, and its body (activities a_{41} and a_{42}) is translated in lines 19-23 in the same manner as `w`.

The goal for a_4 in the definition of `w` (line 7) is a call to a predicate a_4 defined in lines 10-13. Its loop structure is translated by introducing auxiliary clauses in lines 15-17 that represent the case of loop exit (line 15) and the loop iteration by means of a recursive call. The call to the body of the loop (`w2` in line 11) is translated before the call to the auxiliary predicate `a_4x`.

B. Input Substitutions

An input substitution sets up the initial sharing (and therefore which attributes are shared) between the input top-level variables. It is a mapping from the variables that represent the data items given as input to the workflow to subsets of the "hidden" variables which represent attributes.

Variable sharing can be represented as a lattice where nodes represent variable sets which share a unique, *hidden* variable. The structure of the sharing lattice can be directly derived from the input concept lattice by assigning a hidden variable to each attribute in the input context. For clarity, hidden variables are named after the corresponding attributes. Next, the top-level variables are mapped to objects from the input context, and,

```
1  init1(X,D,E):-
       X=[Name,PIN],
3      D=[Symptoms,Tests],
       E=[Symptoms,Coverage].
5
   init2(X,D,E):-
7      X=[Name,Address,SSN],
       D=[Symptoms,Tests],
9      E=[Symptoms,Coverage].
```

Fig. 7. Initial substitution for the two cases.

therefore, to subsets of the hidden input variables (which are nodes in the sharing lattice). The ordering $a \sqsubseteq b$ between top-level variables a and b in the sharing lattice holds iff $A \subseteq B$, where A and B are the corresponding subsets of the associated hidden input variables. It directly follows that $\sqsubseteq$ in the sharing lattice is the exact opposite of $\leq$ in the concept lattice.

In the text that follows, we will use two cases for input substitutions:

Case 1 Patient ID (item x) maps to the Passport object (has the attributes Name and PIN).

Case 2 Patient ID (item x) maps to the Social Security Card object (has the attributes Name, Address and SSN).

In both cases, the data source d for medical histories maps to the Medical history DB object (attributes Symptoms and Tests), and the data source e for medication records maps to the Medication record DB (attributes Symptoms and Coverage).

The input substitution is easy to produce: the input variables are made equal to (i.e., made to share with) terms that contain *exactly* the associated hidden variables from the input sharing lattice (Fig. 7). The actual shape of the terms is not significant, and therefore we just use lists of variables associated to the attributes.

C. Obtaining Sharing Analysis Results

The sharing analysis is applied to the program resulting from the translation of the workflow (Fig. 6) and the code that sets up the initial substitutions (Fig. 7). The underlying theoretical framework we use is abstract interpretation [2], which interprets a program by mapping concrete, possibly infinite sets of values onto (usually finite) abstract domains and reinterprets the operations of the language in a way that respects the original semantics of the language. The abstract approximations of the concrete behavior are *safe* in the sense that properties proved in the abstract domain necessarily hold in the concrete case. However, its precision depends in general on the problem and on the choice of the abstract domain.

The analysis is run using CiaoPP [7], [6], a tool for the analysis and transformation of logic programs featuring, among others, a powerful sharing analysis. While the sharing analysis we used is, in pathological cases, exponential in the number of variables in a clause, in our experience it exhibits a reasonable speed in practice.[1]

```
1  [[X,D,E,A1,Y,A2,Z,A3,A4,A41,C,A42,P,A5],
2   [X,D,A1,Y,A2,Z,A3,A4,A41,C,A42,P,A5],
3   [X,E,A1,Y,A2,Z,A3,A4,A41,C,A42,P,A5],
4   [X,A1,Y,A2,Z,A3,A4,A41,C,A42,P,A5],
5   [D,E,A1,Y,A2,Z,A3,A4,A41,C,A42,P],
6   [D,A1,Y,A3,A4,A41,C,A42,P],
7   [E,A2,Z,A3,A41]]
```

(a) The resulting substitution

Top-level variables	Recovered hidden variables
X, A5	$\{u_1, u_2, u_3, u_4\}$
E	$\{u_1, u_3, u_5, u_7\}$
D	$\{u_1, u_2, u_5, u_6\}$
A2, Z	$\{u_1, u_2, u_3, u_4, u_5, u_7\}$
A1, Y, A42, C, P	$\{u_1, u_2, u_3, u_4, u_5, u_6\}$
A3, A4, A41	$\{u_1, u_2, u_3, u_4, u_5, u_6, u_7\}$

(b) Points in the resulting sharing lattice.

Fig. 8. Abstract substitution and the recovered hidden variables.

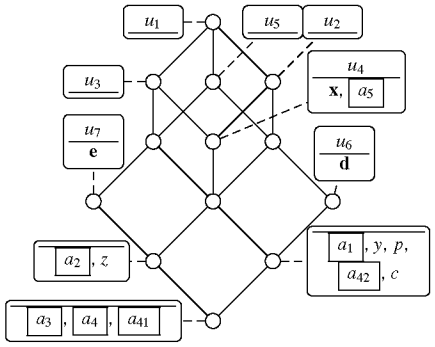


Fig. 9. The resulting concept lattice.

The output of the analysis is an *abstract substitution* (Fig. 8(a)), which is common to both cases of input data mapping. The difference will arise in the interpretation, as described in the next section. Each row (1-7) contains a subset of the top-level variables (representing items and activities in the workflow) that share at least one unique hidden variable. The minimal set of hidden variables which can explain that abstract substitution can be easily recovered [8]: for each line $i = 1 \ldots 7$ in Fig. 8(a), we introduce a new output hidden variable u_i (arbitrarily but uniquely named), and we assign to each top-level variable all hidden variables corresponding to the rows in which it appears. The result is shown in Fig. 8(b), where each row shows the top-level variables associated to a set of the output hidden variables.

V. From Sharing back to Concepts

The mapping of intermediate variables (those which are not initial top-level variables) to subsets of the hidden output variables carries the information on the relationship between these variables that the sharing analysis inferred. However, the

[1] The results presented here were obtained in 1.192ms using CiaoPP running on an Apple MacBook computer with Intel Core Duo processor, 2GB of RAM and MacOS X 10.6.5.

Item	Name	PIN	Symp.	Tests	Cover.
x	✓	✓			
d			✓	✓	
e			✓		✓
a_2, z	✓	✓	✓		✓
a_1, y, p, a_{42}, c	✓	✓	✓	✓	
a_3, a_4, a_{41}	✓	✓	✓	✓	✓
a_5	✓	✓			

Item	Name	Address	SSN	Symp.	Tests	Cover.
x	✓	✓	✓			
d				✓	✓	
e				✓		✓
a_2, z	✓	✓	✓	✓		✓
a_1, y, p, a_{42}, c	✓	✓	✓	✓	✓	
a_3, a_4, a_{41}	✓	✓	✓	✓	✓	✓
a_5	✓	✓	✓			

Fig. 10. The resulting context for the two analysis cases.

meaning of these output hidden variables has to be interpreted in terms of the original attributes — starting with those of the input data items. The sharing analysis of course preserves the original relationship among the input top-level variables [8]: if two variables a and b were associated in the input sharing lattice to subsets of attributes A and B, respectively, such that $A \subseteq B$, then for the corresponding subsets A_1 and B_1 to which a and b map in the resulting sharing lattice, it is the case that $A_1 \subseteq B_1$.

The next step is to construct a result concept lattice (Fig. 9) based on the sharing analysis results where data items and activities are considered as objects and the hidden variables in the result are considered as a new set of attributes. The activities are highlighted and framed, and the input data items from the input concept lattice are set in boldface. In this lattice we first assign the original attributes to the input data items, and then pass them down to all the lower-level concepts. We then obtain the resulting contexts (Fig. 10) for the two initial cases aforementioned. Note that only the attributes that are associated with some input data item may appear.

It should be noted that the construction of the resulting concept lattice can be done in polynomial time with respect to the number of objects (data items, activities) and attributes [13]. Different algorithms for construction of concept lattices differ in performance over different types of sparse contexts.

We want to note that in the most general case sharing analysis is undecidable, and the results of the analyzer can be a *safe over-approximation* which can indicate sharing between variables when it could not be proved that there is definitely no sharing. However, when it indicates no sharing, then this is definitely the case. The assignment of attributes to the workflow elements should be interpreted accordingly: the absence of an attribute is always certain, but its presence is not guaranteed.

We can now go back to the application cases mentioned in Section II and illustrate how the information in the contexts in Fig. 10 can be applied.

d) Fragmentation: The organization responsible for medicine prescription may want to split the workflow among several partners, based on what kind of information they are allowed to handle. The basis for fragmentation is the resulting contexts from Fig. 10. An example of fragmentation is shown in Fig. 11. The swim lanes correspond to the health organization and its partners. *Registry and Archive* cannot handle Symptoms, Tests, or Coverage data, and is therefore assigned activity a_5. *Medical examiners* can at most see Symptoms and Tests, and are thus assigned the activities a_1 and a_{42}. *Medication providers* can only take care of Symptoms and Coverage, and are assigned activity a_2. All other activities (a_3, a_4 and a_{41}) need full access and remain centrally handled by the health organization.

e) Data Compliance: It may be known that a particular kind of information identifying a patient, such as his/her SSN, is required for retrieving the patient's medication record (activity a_2), and that the patient's address is required for sending the results of tests (activity a_{42}). It can therefore be detected at design time that unless the patient is identified with a Social Security Card, these activities will fail. The designer may either restrict the use of the workflow by requiring the card, or select implementations of the mentioned activities with weaker success preconditions.

f) Robust Top-Down Development: Based on the characterization of the input data items, designers can derive the attributes of the data items in nested workflows. For instance, the attributes of the medicine search criterion (c) and the prescription candidate (p) are inferred in Fig. 10 in a safe way.

VI. Conclusions

We have shown how an FCA-based characterization of input data to a workflow can be enriched to include intermediate data items and internal activities. These are annotated with attributes which are inferred from emergent properties of the workflow which stem from the workflow structure and relationships between input data. We have shown how this task can be automated by translating (a) an initial FCA into a lattice from which sharing conditions are derived and (b) the workflow structure into a logic program. Then, (a) and (b) are subjected to a sharing analysis, and the results are mapped back to a resulting lattice and that to a resulting context, whose information can be used as a starting point for a number of other tasks. We have illustrated this methodology with a worked example.

As future work, we plan to address the development of automatic translations from common business process specification languages (BPEL, XPDL, YAWL, etc.) into logic programs amenable to sharing analysis in order to further test and refine the techniques proposed herein. Besides, we plan to explore other applications of the concept of sharing to services, aiming not only at (local) data sharing between activities, but also looking towards the representation of stateful service conversations and quality aspects of services.

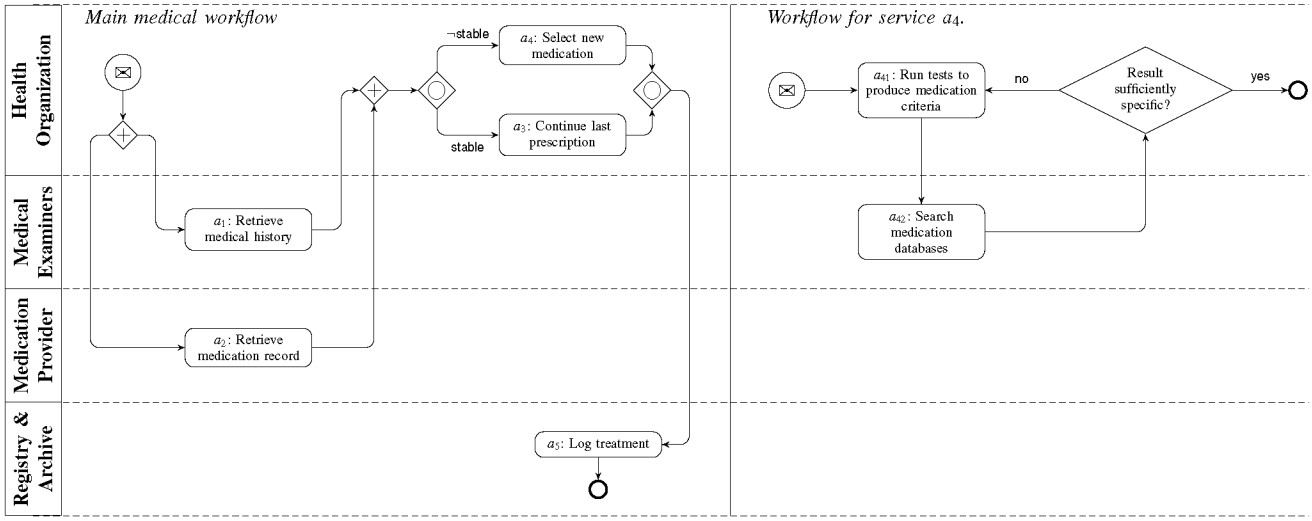


Fig. 11. An example fragmentation for the drug prescription workflow.

REFERENCES

[1] Claudio Carpineto and Giovanni Romano. *Concept Data Analysis: Theory and Applications*. Wiley, 2004.

[2] P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *ACM Symposium on Principles of Programming Languages (POPL'77)*. ACM Press, 1977.

[3] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2nd ed. edition, 2002.

[4] Walid Fdhila, Ustun Yildiz, and Claude Godart. A Flexible Approach for Automatic Process Decentralization Using Dependency Tables. In *ICWS*, pages 847–855, 2009.

[5] Bernhard Ganter, Gerd Stumme, and Rudolf Wille, editors. *Formal Concept Analysis, Foundations and Applications*, volume 3626 of *Lecture Notes in Computer Science*. Springer, 2005.

[6] M. Hermenegildo, G. Puebla, F. Bueno, and P. López García. Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor). *Science of Computer Programming*, 58(1–2), 2005.

[7] M. V. Hermenegildo, F. Bueno, M. Carro, P. López, E. Mera, J.F. Morales, and G. Puebla. An Overview of Ciao and its Design Philosophy. *Theory and Practice of Logic Programming*, 2011. http://arxiv.org/abs/1102.5497.

[8] Dragan Ivanović, Manuel Carro, and Manuel Hermenegildo. Automatic Fragment Identification in Workflows Based on Sharing Analysis. In Mathias Weske, Jian Yang, Paul Maglio, and Marcelo Fantinato, editors, *Service-Oriented Computing – ICSOC 2010*, number 6470 in LNCS. Springer Verlag, 2010.

[9] D. Jordan and et. al. Web Services Business Process Execution Language Version 2.0. Technical report, IBM, Microsoft, et. al, 2007.

[10] R. Khalaf and F. Leymann. E Role-based Decomposition of Business Processes using BPEL. In *IEEE International Conference on Web Services (ICWS'06)*, 2006.

[11] Rania Khalaf. Note on Syntactic Details of Split BPEL-D Business Processes. Technical Report 2007/2, IAAS, U. Stuttgart, July 2007.

[12] Oliver Kopp, Rania Khalaf, and Frank Leymann. Deriving Explicit Data Links in WS-BPEL Processes. In *International Conference on Services Computing (SCC)*, 2008.

[13] Sergei O. Kuznetsov and Sergei A. Obiedkov. Comparing performance of algorithms for generating concept lattices. *J. Exp. Theor. Artif. Intell.*, 14(2-3):189–216, 2002.

[14] J.W. Lloyd. *Foundations of Logic Programming*. Springer, 2nd Ext. Ed., 1987.

[15] Daniel Martin, Daniel Wutke, and Frank Leymann. A Novel Approach to Decentralized Workflow Enactment. In *EDOC '08: Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pages 127–136, Washington, DC, USA, 2008. IEEE Computer Society.

[16] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 2005. Second Ed.

[17] Object Management Group. *Business Process Modeling Notation (BPMN), Version 1.2*, January 2009.

[18] Natalia Sidorova, Christian Stahl, and Nikola Trcka. Workflow soundness revisited: Checking correctness in the presence of data while staying conceptual. In *CAiSE*, pages 530–544, 2010.

[19] Sherry X. Sun, J. Leon Zhao, Jay F. Nunamaker, and Olivia R. Liu Sheng. Formulating the data-flow perspective for business process management. *Information Systems Research*, 17(4):374–391, 2006.

[20] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, June 2005.

[21] Wil van der Aalst and Maja Pesic. DecSerFlow: Towards a Truly Declarative Service Flow Language. In *The Role of Business Processes in Service Oriented Architectures*, number 06291 in Dagstuhl Seminar Proceedings, 2006.

[22] Ping Yang, Shiyong Lu, Mikhail I. Gofman, and Zijiang Yang. Information flow analysis of scientific workflows. *J. Comput. Syst. Sci.*, 76(6):390–402, 2010.

[23] Ustun Yildiz and Claude Godart. Information Flow Control with Decentralized Service Compositions. In *ICWS*, pages 9–17, 2007.