facultad de informática

universidad politécnica de madrid

Automated Attribute Inference in Complex Service Workflows Based on Sharing Analysis

Dragan Ivanović
Manuel Carro
Manuel Hermenegildo

TR Number CLIP 5/2010.0

Automated Attribute Inference in Complex Service Workflows Based on Sharing Analysis

Technical Report Number: CLIP 5/2010.0
November, 2010

Authors

Dragan Ivanovic
idragan@clip.dia.fi.upm.es
Computer Science School
Universidad Politécnica de Madrid (UPM)

Manuel Carro
mcarro@fi.upm.es
Computer Science School
Universidad Politécnica de Madrid (UPM)

Manuel Hermenegildo
herme@fi.upm.es
Computer Science School
Universidad Politécnica de Madrid (UPM)
and IMDEA Software, Spain

Keywords

Workflow, Business Process, Service Composition, Horn Clause, Static Analysis

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme under the Network of Excellence S-Cube (Grant Agreement n° 215483). Manuel Carro and Manuel Hermenegildo were also partially supported by Spanish MEC project 2008-05624/TIN *DOVES* and CM project P2009/TIC/1465 (*PROMETIDOS*). Manuel Hermegildo was also partially supported by FET IST-231620 *HATS*.

Abstract

The properties of data and activities in workflows can be used to greatly facilitate several relevant tasks performed at design- and run-time, such as fragmentation, conformance checking, or top-down design. We present an approach to mechanically infer domain attributes of workflow components, such as data items, activities, or elements of sub-workflows, based on basic characteristics of workflow inputs, the structure of the workflow, and the results of sharing analysis applied to a Horn clause 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. As Service-Oriented Computing is quite often used to implement workflows, we relate the techniques we present with the technologies and approach taken by SOC.

Contents

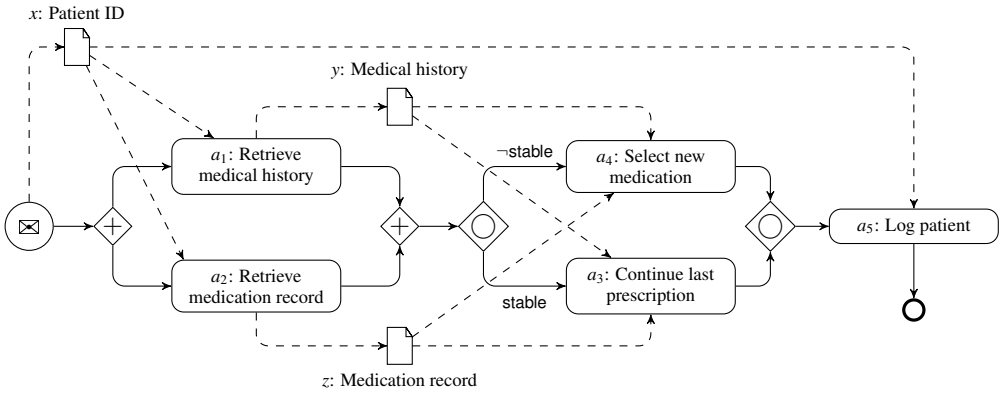


Figure 1: An example drug prescription workflow.

1 Introduction

Service-Oriented Computing is a paradigm that stresses interoperability between services, which are 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 (and potentially long running) business processes in an executable form, often across organizational boundaries. Compositions are usually described by specifying the sequencing of workflow activities and the routing of data using a language that allows process modelers and designers to capture the essential elements of business logic and processing requirements [Obj09, Jea07, vdAP06, vdAtH05].

Due to this relationship, inferring properties of workflows is valuable for several important design- and run-time activities in the service life cycle, such as driving the design and checking it, transforming and refactoring workflows, identifying patterns, facilitating run-time instrumentation, reshaping, or performing the distributed enactment of service compositions [MWL08]. Furthermore, it can serve as an input for verification of service systems based on their conversational patterns [BFHS03].

In this paper we focus on the inference of domain-specific attributes for data items and activities in service workflows. These attributes can be related to data contents, privacy and security, quality, or other interesting concepts. At the same time, we want to deal with workflows with a complex structure involving parallel flows, rich control and data dependencies between activities, complex control constructs (e.g. branches and loops), as well as component services with known structure.

We apply techniques developed in the realm of static program analysis [NNH05] to perform attribute inference. Building on our previous work [ICH10], we use the notion of *sharing* between data items (and, by extension, activities). The application of general program analysis techniques allows us to produce a characterization of data items and activities in workflows, which can then be used to feed, for instance, fragmentation algorithms [Kha07]. The formalism we start with avoids the dependence on a particular business process description language some other analysis [KKL07] have, and significantly relaxes constraints on the shape of the workflows that can be treated [FYG09, YG07].

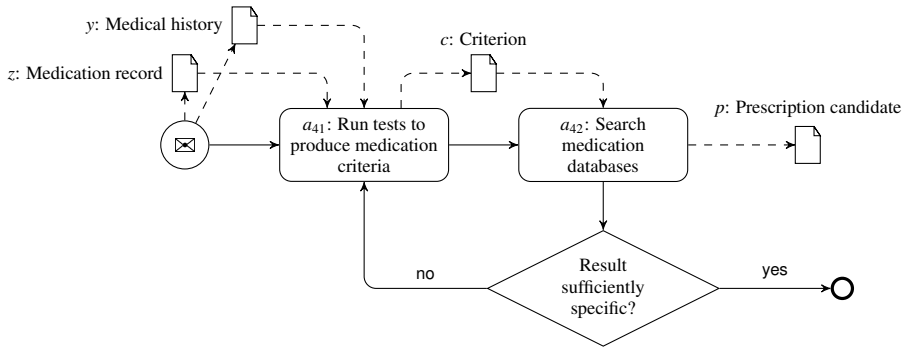


Figure 2: An example component service for selection of new medication.

2 Motivation

To motivate our approach, let us take a simplified example of a drug prescription workflow, given in standard BPMN notation [Obj09] in Fig. 1. The process is initiated by the arrival of a patient with an appropriate identification, shown as the data item x in the figure. Next, two parallel activities are run to retrieve the patient's medical history (activity a_1), and to retrieve the patient's medication record (activity a_2). The data items that result from these two activities are marked as y (medical history) and z (medication record), respectively. Additionally, while retrieving the medical history, activity a_4 indicates whether the health situation of the patient has been stable or not. Depending on that, either the last prescription is continued (activity a_3), or a new medication is selected (activity a_4). Finally, the treatment of the patient is logged (activity a_5).

From an engineering point of view, design-time analysis of data flow in the given workflow can be used to obtain interesting results useful for several tasks, including:

Fragmentation: In a distributed environment, a workflow can be broken into several fragments that can for example be farmed out for execution within different organizational (business) domains. Different criteria and constraints can be used to decide whether two activities can belong to a single fragment. In our example, a healthcare organization may wish to delegate some parts of the workflow to business partners that handle registry and medical checks. Confidentiality of information handled by individual activities may require that either the medical history or the medication record (or both) be hidden from some of the partners. In our case, that would mean separating the activities from the process in Fig. 1 into several swim-lanes corresponding to different organizational domains, and assigning activities accordingly.

Condition Checking: When putting several services together to form a business process (such as $a_1 \ldots a_5$ in our example), an organization needs to make sure that the information content used by a component service is adequate from the point of view of the desired behavior. This is related to the problem of verifying systems of Web services from the conversational point of view [BFHS03]. Syntactic compliance with the XML message formats is not sufficient. In our example, the patient ID (which may be a national identity card, a driving license, a passport, a health insurance card or something else) may or may not contain at runtime sufficient information required for the retrieval of, e.g., a medical history. Or, a particular combination of retrieved medical histories and medication records may not contain the necessary pieces of information required for the successful completion of the (possibly outsourced) service in charge of selecting new medication. Some

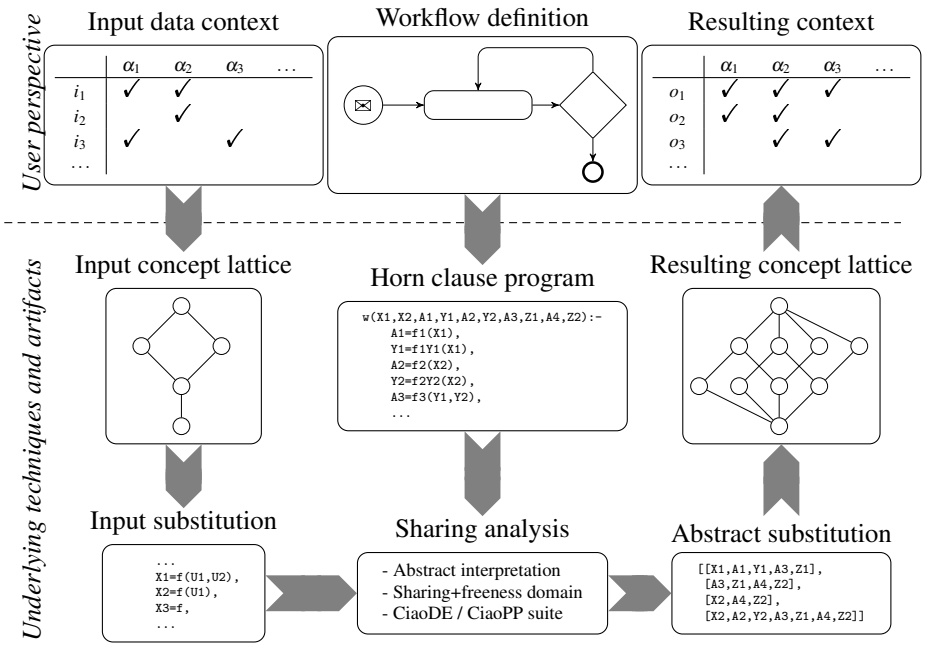


Figure 3: Overview of the approach.

of these potential problems can be detected at design time, by tracking the data flow from one activity to another, and analyzing which pieces of information are shared between activities. E.g., the retrieval of the medication record (activity a_2) may require the social security number, while running additional medical tests (activity a_{41}) may require the patient's address.

Top-Down Component Specification: Business process modeling can start at a high level, and gradually elaborate components in a top-down fashion. Some of the components may contain complex structured constructs, or be developed into separate, reusable services. Data flow analysis at the process level can help us identify features of inputs and outputs from such components, but additional results can be obtained for sub-activities of a component. For instance, if the workflow component a_4 is separated into a reusable service of its own with the structure given in Fig. 2, it would be interesting to derive from the process data flow attributes of the criterion c. The process can be repeatedly applied to the components a_{41} and a_{42} of the sub-workflow.

The goal of this paper is to address the problem of inferring domain specific attributes of relevant entities, such as data items and activities, within a workflow which possibly features complex control structures (branches, loops) and component sub-workflows. The inference is based on characteristics of the inputs to the workflow and the data flow between workflow activities. The inferred attributes can be used by design-time tools to achieve some of the above mentioned results.

3 Overview of the Approach

Fig. 3 depicts an overview of our approach. From the users' perspective, the starting point is a description of the workflow using an appropriate formalism (such as the BPMN notation used in our example),

and an input data context: a list of objects which constitute the input to the workflow, described using a repertoire of relevant attributes from the business domain of the workflow. The result is presented as a resulting context that lists all data items and the activities of the workflow, and assigns them the attributes that are inferred by the analysis. The user (or a tool) can then inspect the resulting context and use the descriptions of the workflow entities for further analysis of the workflow, transformation, or other design-time tasks.

The aforementioned initial context, which lists concepts and attributes, is used to set up a concept lattice (Sec. 4). Concept lattices are the main vehicle to prepare the input to the sharing analysis and to interpret the results of the analysis. The sharing analysis works on Horn clause program which results from translating the workflow definition into Horn clauses and the initial concept lattice into input logical variable substitutions (Sec. 5). Finally (Sec. 6), the abstract substitutions which result from the sharing analysis are used to produce the resulting context.

Note that all steps in Fig. 3 can be fully automated using already existing tools and technologies, with the possible exception of the translation of the workflow into a Horn clause program. The last one depends on the particular workflow description formalism used, and needs specific translations. A particular example for a rich workflow model was shown in [ICH10], and we reuse that notation here.

Automating the steps in Fig. 3 means that the end user does not have to understand concepts such as lattices, the structure and elements of the Horn clause program, abstract substitutions, or the mechanism of the sharing analysis per se. All of those technical details can be hidden "under the hood," and the user would be able to work at the level of the top row in Fig. 3, through a simple and intuitive interface which focuses on contexts. Modules that implement, for example, a particular refactoring or transformation task (such as fragmentation) can access the analysis results given in the simple form of the resulting context (Sec. 6).

4 From Contexts to Concept Lattices

To describe relevant properties of the data fed into the workflow, we will use Formal Concept Analysis (FCA) [DP02, GSW05], a mathematical apparatus for representing and analyzing data using formal contexts and concept lattices, which we will explain with depth just enough to follow this paper. The reader is kindly referred to the existing literature for more in-depth treatment of FCA.

4.1 Contexts and Concepts

A *context* is defined as an assignment of a set of *attributes* to a set of *objects*. We will assume that both the set of attributes and the set of objects are finite. These may, but need not necessarily, be objects and attributes in the object-oriented design sense. In our case, objects represent data items given as inputs to the workflow, data items produced by the workflow activities, and the activities themselves. Attributes are predicates that may be true or not for an object: we then say that the object has the attribute (or not).

The purpose of contexts is to express properties of objects that are relevant in some domain of discourse. A simple and intuitive way of presenting contexts is in the form of a table, with rows for objects and columns for attributes (Fig. 4). Checkmarks are equivalent to logical truth, and absence of checkmarks denotes logical falsity. Fig. 4(a) shows a simple context describing the content that is retrieved from the medical history and the medication record databases, in terms of (among many other possible) attributes denoting the content of retrieved records: the observed / treated symptoms, the medical

	Symptoms	Tests	Coverage
Medical history DB	✓	✓	
Medication record DB	✓		✓

(a) Characteristics of medical databases.

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

(b) Types of identity documents.

Figure 4: Two examples of contexts.

tests run on the patient, and the insurance coverage for the medication. Likewise, Fig. 4(b) shows a context describing different types of documents used to identify the patient in terms of the information they contain: name, address, the personal identification number (PIN), and the social security number (SSN). Contexts can be easily combined or split by grouping and projecting over objects and attributes, respectively.

A *concept* in FCA is a pair (O,A) of a subset of objects O and a subset of attributes A such that (1) all attributes that objects from O have in common are in A, and (2) all objects that have all attributes from A are in O. This way of relating the two subsets is usually called a Galois connection. If with O' we denote the set of attributes common to all objects in O, and with A' we denote the set of all objects that have all of the attributes from A, then for concept (O,A) we have $O' = A$, $A' = O$, and thus $O'' = O$ and $A'' = A$. Therefore, we will often keep track of just the attributes of a concept, since the objects can be derived using $(')$.

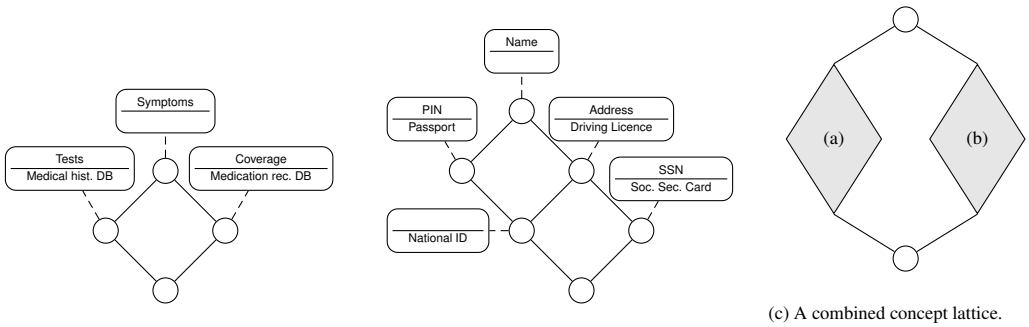
4.2 Concept Lattices

While contexts are a convenient and intuitive mechanism for knowledge representation, to make them useful for analysis they need to be transformed 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* operation $\vee$, and the *greatest lower bound* 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 least upper and the greatest lower bounds exist in L; such lattices have unique greatest ($\top$) and least ($\bot$) elements.

For concept lattices, the ordering relation $\leq$ between two concepts (O_1,A_1) and (O_2,A_2) holds if and only if $O_1 \subseteq O_2$, or, equivalently, if and only if $A_2 \subseteq A_1$; in other words, a higher concept includes all objects from a lower (or derived) one, and lower concepts are derived from higher ones by adding attributes. Consequently, the least upper bound is obtained using $A_1 \cap A_2$, and the greatest lower bound using $A_1 \cup A_2$.

The usual way of depicting context lattices is a variant of Hasse diagrams given in Fig. 5. Nodes in the lattice correspond to concepts, with the top concept visually on the top, and the bottom concept placed accordingly. 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 directly belong to the concept below the line. Concepts may have either the part of the annotation empty, or both parts – for instance when they just introduce new attributes, or combine attributes from several higher concepts. When both parts are empty, the annotation is not shown; that is the case with the bottom elements in all cases in the figure.



(a) Concept lattice for medical databases.

(b) Concept lattice for identity documents.

(c) A combined concept lattice.

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

Fig. 5(a) presents the concept lattice for the medical database context from Fig. 4(a). The top (i.e., the most general) concept here introduces the attribute Symptoms, since it is shared by all medical database objects. However, no object has *only* that attribute, so the top concept has no directly associated objects. Two descendant concepts introduce the attributes Test and Coverage, respectively, which discriminate between the Medical history database and the Medication record database.

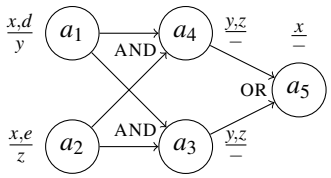
Fig. 5(b) shows a slightly more complex concept lattice corresponding to the context for identity documents from Fig. 4(b). All identity documents contain the patient's Name (introduced by the top concept), while, e.g., the National ID inherits all attributes from the top concept, the Passport concept, and the Driving License concept. Two concept lattices that have disjoint sets of attributes and objects (which is the case in our example) can be combined as shown in Fig. 5(c).

4.3 Describing Data with Concept Lattices

In our analysis, the data items that are fed into the workflow as inputs need to mapped to the appropriate objects in the input concept lattice. In the case of the medication prescription workflow 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, and especially when we project over attributes, several objects used in building the context can map to the same concept in the lattice.

The prerequisite for using concept lattices is to create an adequate context on a level of abstraction that captures a sufficient amount of information in order to represent all relevant concepts and their attributes. A more complex model can always be simplified by projecting over attributes, and thus keeping only those that discriminate between different concepts. There are several available tools that facilitate the process of elicitation and exploration of knowledge using the FCA mathematical apparatus, such as ConExp, Lattice Miner, Colibri and others [CR04]. Such tools usually offer functionality to:

- Check consistency and completeness of the context and ask for counterexamples. For instance, the user can be asked to verify that all identification documents contain the person's name (which is true), or that all medical databases of interest contain data on symptoms, which may or may not be true, and in the latter case the user is asked to introduce a counter-example.
- Investigate Boolean and probabilistic associations that exist between the attributes.
- Discretize attributes that range over numeric values.



$$
\begin{aligned}
W &= \langle A, C, D \rangle \\
A &= \{a_1, a_2, a_3, a_4, a_5\} \\
C &= \{\mathsf{pre}\text{–}a_4 \equiv \mathsf{done}\text{–}a_1 \wedge \neg \mathsf{succ}\text{–}a_1 \wedge \mathsf{done}\text{–}a_2, \\
&\quad\ \mathsf{pre}\text{–}a_3 \equiv \mathsf{done}\text{–}a_1 \wedge \mathsf{succ}\text{–}a_1 \wedge \mathsf{done}\text{–}a_2, \\
&\quad\ \mathsf{pre}\text{–}a_5 \equiv \mathsf{done}\text{–}a_3 \vee \mathsf{done}\text{–}a_4\} \\
D &= \{y, z\},\ y = \langle a_1, \{a_3, a_4\}\rangle,\ z = \langle a_2, \{a_3, a_4\}\rangle
\end{aligned}
$$

Figure 6: The abstract notation for the medical workflow.

- Visually explore the resulting Context Lattice, run queries, and engage in reasoning on the knowledge contained in it.

Another important point is that some data sources may not be explicitly shown on a workflow. In our example from Fig. 1, the retrieval activities (a_1 and a_2) both need to access some external data sources from which they extract the records using the input Patient ID. Obviously, the characteristics of the retrieved records depend on properties of these implicit data sources. Therefore, both need to be mapped to appropriate objects, in this case the Medical history DB and the Medication record DB from Fig. 5(a).

5 Applying Sharing Analysis

5.1 Representing Workflows as Horn Clause Programs

In order to apply sharing analysis, the workflow under consideration needs to be represented in the form of a Horn clause program [Llo87]: a series of logical implications which can be understood as stating which subgoals are needed to accomplish a given goal. The translation from the original form in which the workflow is designed into a Horn clause program does not need to preserve the operational semantics of the original formalism. It instead needs to focus on representing the data flow correctly. In our previous work [ICH10], we have given an example of translation from an abstract notation for workflows with logical link dependencies between activities expressed as logic formulas, explicit description of input and output data items for each activity, and complex constructs such as branches and loops. That abstract notation has been purposefully crafted to reflect the most common elements used in business process design using languages such as BPMN [Obj09], BPEL [Jea07], or XPDL [Wor08], and can be used as an intermediate representation in the process of translation from the original formalism. Our abstract workflow notation does not support the full range of workflow patterns [vdAtHKB03], and it does not preclude the possibility of (or need) for advanced direct or customized translation from specific workflow languages when necessary.

Fig. 6 shows the abstract notation for the medical workflow (Fig. 1), given in a condensed form and stripped of activity/data descriptions. Details of the abstract notation can be found in our previous work [ICH10], while here we focus on the key elements of the notation and its correspondence with the original BPMN diagram. In the abstract notation, an activity can start executing as soon as its preconditions are met, and any number of activities can execute in parallel. As the activities a_1 and a_2 do not have preconditions, a parallel AND-split (↼◈⇀) from Fig. 1 is assumed. Activities are annotated with the symbolic names of their inputs (above the line) and output (below the line) data items; those are the same symbolic names from Fig. 1: x stands for the Patient ID, y for the Medical history, z for the Medication record, etc. Note that two new data items, d and e, appear in Fig. 6. They stand here for the

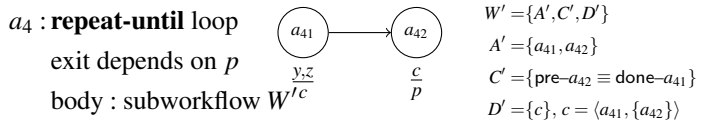


Figure 7: The abstract notation for the complex activity a_4.

```
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),
12     A4=f(P2),
       a_4x(Y,Z,C2,P2,C,P,A4,A41,A42).
14
   a_4x(_,_,C,P,C,P,_,_,_).
16 a_4x(X,Z,_,_,C,P,A4,A41,A42):-
       a_4(X,Z,A4,A41,C,A42,P).
18
   w2(Y,Z,A41,C,A42,P):-
20     A41=f41(Y,Z), % a_41
       C=f41_C(Y),
22     A42=f42(C), % a_42
       P=f42_P(C).
```

Figure 8: Horn clause program encoding for the medication prescription workflow.

implicit data sources from which activities a_1 and a_2, respectively, retrieve the medical history and the medication record for the patient. In the abstract workflow notation, these are also treated as inputs to the workflow.

The AND-join (→◇←) and the XOR-split (←◇→) from Fig. 1 are fused into preconditions for the activities a_3 and a_4, marked as pre–a_3 and pre–a_4 in the set C of preconditions in Fig. 6. Both preconditions are conjunctions involving completion of a_1 and a_2 (denoted by the symbols done–a_1 and done–a_2, respectively), while the options are discriminated on the basis of succ–a_1, which stands for the logical result of a_1, in this case the indication of a stable health indicator. Finally, the OR-join (→◇←) from Fig. 1 is represented with the precondition for a_5. Data dependencies D for the internally generated data items show the activity that creates the data item, and the set of activities that use it as input.

The sub-workflow for the complex activity a_4 from Fig. 2 is similarly translated into the abstract notation in Fig. 7. Note that the abstract notation is *goto*-free, and therefore a_4 is explicitly annotated as a *repeat-until* loop with the test depending on the internally generated data item p. The body of the loop constitutes the sub-workflow W'.

The translation of our example workflow from the abstract notation into a Horn clause program is shown in Fig. 8. For the details of the Horn clause program notation and the translation process the reader is again kindly referred to our previous work [ICH10]. The input and the intermediate data items, as well as the workflow activities, are represented with the "top-level" variables (visible as the formal arguments to `w` on line 1, and given in capital letters). The formal arguments to predicate `a4` (line 10) for complex activity a_4, and the arguments to `w2`, which represents the sub-workflow in the body of the *repeat-until* loop, play an analogous role.

5.2 Setting Up an Input Substitution

An input substitution sets up sharing between the input top-level variables in the Horn clause program. It is a mapping from the top-level variables that represent the data items given as input to the workflow to subsets of the "hidden" or "tagging" variables that do not appear in the logic program. That relationship

```
1 init1(X,D,E):-
2         X=[Name,PIN],
          D=[Symptoms,Tests],
4         E=[Symptoms,Coverage].
```

```
1 init2(X,D,E):-
2         X=[Name,Address,SSN],
          D=[Symptoms,Tests],
4         E=[Symptoms,Coverage].
```

Figure 9: Setup of the initial substitution for the two cases.

is based on the notion of abstract substitution [MH91], and it means that the given top-level variable is bound to a term that contains the associated subset of hidden variables.

Variable sharing can be represented as a lattice where nodes represent variable sets which share a unique, *hidden* variable. The structure of such sharing lattice can be directly derived from the input concept lattice by assuming hidden variables that correspond one to one to the input context attributes. For clarity, we name those hidden variables as the corresponding attributes. Next, we map the top-level variables to objects from the input context, and, therefore, to subsets of the hidden input variables, which are the points in the sharing lattice. The ordering $a \sqsubseteq b$ in the input sharing lattice between arbitrary top-level variables a and b holds if and only if $A \subseteq B$, 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 take 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).

Producing an input substitution from a sharing lattice is straightforward. What we need to do is to equate the top-level input variables to terms that contain exactly the associated hidden variables from the input sharing lattice (Fig. 9). Since for the sharing analysis the actual shape of the terms is not significant, we use lists of variables associated to the attributes.

5.3 Obtaining the Sharing Results

Sharing analysis is applied to the Horn clause program that consists of the translation of the workflow in Fig. 8 and the code that sets up the initial substitutions for the cases of interest (Fig. 9). The underlying technology for sharing analysis used here is abstract interpretation [CC77], a static analysis technique that interprets the program by mapping concrete, possibly infinite sets of variable values onto (usually finite) abstract domains, together with data operations, in a way that respects the original semantics of the programming 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.

We run the analysis using CiaoPP [HBC$^+$10, HPBG05], one of the analysis tools that have been developed for logic programs, which gives users the possibility of selecting different analysis algorithms on input programs. We use the *sharing, freeness and groundness* analysis available in CiaoPP. While the

```
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.

Figure 10: Abstract substitution and the recovered hidden variables.

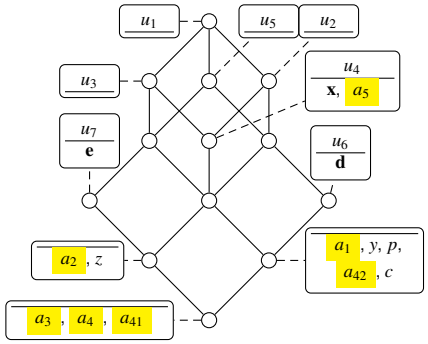


Figure 11: The resulting concept lattice.

applied sharing analysis of a logic program is, in the worst case, exponential in the number of variables, in our experience in most cases it exhibits a reasonable speed.[1]

The sharing analyzer outputs its results in a form known as an abstract substitution, shown in Fig. 10(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 (that represent the items and activities in the workflow) that share at least one unique hidden variable. To the resulting abstract substitution, we can apply the algorithm for recovery of a minimal set of the hidden output variables [ICH10]: for each line $i = 1 \ldots 7$ in Fig. 10(a), we introduce a new output hidden variable u_i (that is arbitrarily, but uniquely, named), and to each top-level variable we assign all resulting hidden variables for the rows in which it appears. The result is shown in Fig. 10(b). Each row shows the top-level variables associated to a set of the output hidden variables.

6 Interpreting Sharing Results as Concepts

The mapping of intermediate variables (those which are not initial top-level variables) to subsets of the hidden output variables represents the new information on the relationship between the variables based on the sharing analysis. The meaning of these output hidden variables has to be interpreted in terms of the original attributes, starting from the known original attributes of the input data items. It should be noted [ICH10] that the sharing analysis preserves the original relationship between the input top-level variables: if two variables a and b were associated to subsets A and B, respectively, of attributes in the input sharing lattice, such that $A \subseteq B$, then for the corresponding subsets A_1 and B_1 to which a and b, respectively, map in the resulting sharing lattice, we also have the property $A_1 \subseteq B_1$.

[1] The results presented here were obtained in 1.192ms using CiaoPP running on an Apple MacBook computer with Intel Core

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

Figure 12: The resulting context for the two analysis cases.

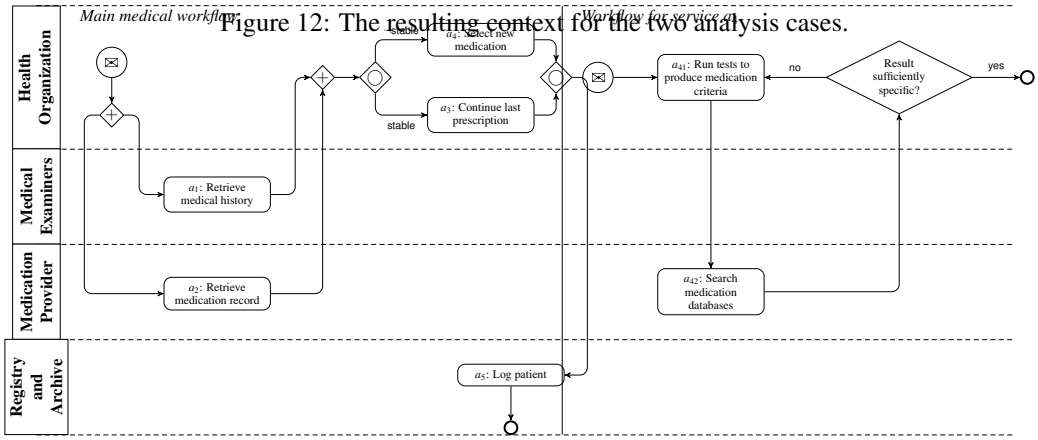


Figure 13: An example fragmentation for the drug prescription workflow.

In the next step, we construct a resulting concept lattice where the data items and activities are considered as objects, and the resulting hidden variables are considered as a new set of attributes. Such lattice is shown in Fig. 11. The activities are highlighted, and the input data items from the input concept lattice are set in boldface. In the resulting concept lattice, we first assign the original attributes to the input data items, and then pass them down to all lower-level concepts. In that way, we obtain the resulting contexts for the two analysis cases shown in Fig. 12. Note that only the attributes that are associated with some input data item may appear.

Note that this may introduce a degree of *over-approximation*: sharing analysis is conservative and indicates sharing between variables also in the case where the analysis could not prove that there is definitely no sharing. Therefore, although the analyzer aims at being as accurate as possible, if it cannot be completely precise, it always errs on the *safe side*, by assuming potential sharing, rather than ignoring it. Thus, the assignment of attributes to the workflow elements should be interpreted accordingly: absence of an attribute means that the element certainly does not have the attribute, but its presence may be either certain or potential.

We can illustrate the application of the information contained in the resulting context in the cases mentioned in the motivation part:

Fragmentation: The health organization responsible for medicine prescription may want to fragment and share the workflow with several partners, based on what kind of information about patients they are allowed to see. Medical examiners, who run labs and outpatient care centers, need to access the information on tests applied to a patient, but should not access the insurance coverage data. Medication providers, on the other hand, do not need to see the details of the patient's medical tests, but need access to insurance coverage data in order to select the most effective

Duo processor, 2GB of RAM and MacOS X 10.6.5.

medication. Any activity that needs access to data on both the tests and the coverage has to be centrally executed by the home health organization. Finally, registrars have access only to the patient's identity. The four types of partners are shown as swimlines in Fig. 13.

Condition Checking: 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 may fail. Therefore, the designer may either restrict the use of the workflow by requiring the card, or try to choose among implementations of the mentioned activities that have weaker success preconditions.

Top-Down Component Specification: Based on the characterization of the input data items, the designer can derive the attributes of the data items nested into complex structures inside the workflow. For instance, the properties of the medicine search criterion (c) and the prescription candidate (p) can be inferred. As mentioned above, such inference is safe in that it may assign more attributes than is strictly needed, due to loss of precision, but will never erroneously omit an attribute that is actually shared.

7 Conclusions

Sharing analysis can be applied in an automated manner by workflow designers to infer business-domain specific attributes of data items and activities inside (potentially complexly structured) service workflows. Starting with business domain knowledge expressed in the form of object-attribute grids or contexts, and by mapping workflow inputs to the appropriate objects with business meaning, the sharing analysis produces results that are interpreted to provide inferred attributes for the rest of workflow data items and activities. The inferred attributes can be then used at design time to drive, e.g., workflow fragmentation, condition checking, and top-down design.

Our future work is planned to address the development of automatic translations from common business process specification languages such as BPEL, XPDL, YAWL, into Horn clause programs amenable to sharing analysis, with the objective of achieving full automation of the analysis process as well as increasing the accuracy of the sharing analysis. Besides, we plan to explore other applications of the sharing concept 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.

References

[BFHS03] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation Specification: A New Approach to Design and Analysis of E-Service Composition. In *WWW*, 2003.

[CC77] 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.

[CR04] Claudio Carpineto and Giovanni Romano. *Concept Data Analysis: Theory and Applications*. Wiley, 2004.

[DP02] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2nd ed. edition, 2002.

[FYG09] Walid Fdhila, Ustun Yildiz, and Claude Godart. A Flexible Approach for Automatic Process Decentralization Using Dependency Tables. In *ICWS*, pages 847–855, 2009.

[GSW05] Bernhard Ganter, Gerd Stumme, and Rudolf Wille, editors. *Formal Concept Analysis, Foundations and Applications*, volume 3626 of *Lecture Notes in Computer Science*. Springer, 2005.

[HBC+10] 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. Technical Report CLIP2/2010.0, Technical University of Madrid (UPM), School of Computer Science, March 2010. Under consideration for publication in *Theory and Practice of Logic Programming (TPLP)*.

[HPBG05] 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.

[ICH10] 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.

[Jea07] D. Jordan and et. al. Web Services Business Process Execution Language Version 2.0. Technical report, IBM, Microsoft, et. al, 2007.

[Kha07] Rania Khalaf. Note on Syntactic Details of Split BPEL-D Business Processes. Technical Report 2007/2, Institut für Architektur von Anwendungssystemen, Universität Stuttgart, Universitätsstrasse 38, 70569 Stuttgart,Germany, July 2007.

[KKL07] Oliver Kopp, Rania Khalaf, and Frank Leymann. Reaching Definitions Analysis Respecting Dead Path Elimination Semantics in BPEL Processes. Technical Report 2007/04, Institut für Architektur von Anwendungssystemen, Universitätsstraße 38, 70569 Stuttgart, Germany, November 2007.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming*. Springer, 2nd Ext. Ed., 1987.

[MH91] K. Muthukumar and M. Hermenegildo. Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation. In *International Conference on Logic Programming (ICLP 1991)*, pages 49–63. MIT Press, June 1991.

[MWL08] 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.

[NNH05] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 2005. Second Ed.

[Obj09] Object Management Group. *Business Process Modeling Notation (BPMN), Version 1.2*, January 2009.

[vdAP06] 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.

[vdAtH05] 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.

[vdAtHKB03] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[Wor08] The Workflow Management Coalition. *XML Process Definition Language (XPDL) Version 2.1*, 2008.

[YG07] Ustun Yildiz and Claude Godart. Information Flow Control with Decentralized Service Compositions. In *ICWS*, pages 9–17, 2007.