Automatic Composition of Form-Based Services in a Context-Aware Personal Information Space

Rania Khéfifi[1], Pascal Poizat[2], and Fatiha Saïs[1]

[1] LRI, CNRS and Paris Sud University,
{rania.khefifi,fatiha.sais}@lri.fr
[2] LIP6, CNRS and Paris Ouest University
pascal.poizat@lip6.fr

Abstract. Personal Information Spaces (PIS) help in structuring, storing, and retrieving personal information. Still, it is the users' duty to sequence the basic steps in different online procedures, and to fill out the corresponding forms with personal information, in order to fulfill some objectives. We propose an extension for PIS that assists users in achieving this duty. We perform a composition of form-based services in order to reach objectives expressed as workflow of capabilities. Further, we take into account that user personal information can be contextual and that the user may have personal information privacy policies. Our solution is based on graph planning and is fully tool-supported.

Keywords: Service Composition, Ontologies, Contextual Data, Personal Information, Privacy, Graph Planning

1 Introduction

Personal Information Spaces (PIS) support users in structuring, storing, and retrieving their personal information. However, with regards to online procedures, *e.g.,* administrative processes, users are left alone to find out which services can be used to achieve parts of the procedures, how to sequence parts of these services, and how to fill the service forms using their personal information. Service composition [8, 2] supports the realization of business processes out of the automatic assembly of online services. Still, a first issue is to deal with data at the *good level of abstraction*. Service composition algorithms that support data do it at the type level. If some service requires a data of type d, any value will be ok. This is not realistic in a PIS where data is contextual. Depending on a context of interest, not all possible values for data type d are equivalent and/or valid to be used as a service input. A second issue is that personal information is *sensitive*. Users should be able to specify access policies to be endorsed while passing data to the composed services.

Contributions. We present in this paper a service composition approach for the *realization of online procedures that are expressed as workflows*. This approach is *context-aware*. It considers the contextual usability of user personal information that are to be transmitted in online service forms. Further, it supports personal

information *access policies* while computing the compositions. Our approach is *automatic* and *tool-supported* through the use of semantic annotations, the encoding of the composition requirements into an AI planning problem, and an extension of a state-of-the-art graph planning algorithm.

Related Work. Several service composition approaches have used context information, *e.g.,* [9, 13, 10]. Most of them consider context with a technical perspective (*e.g.,* device type, battery charge, GPS information) that is used to select functionality (services). We consider contexts in a more general sense, together with possible relations (subsumption, disjointness) between them that enable contextual reasoning. We do not only use contexts to select services but also to select the best (in terms of usability) data to give to these services. Some approaches [5, 1, 4] take into account policies in the composition process. While [5] supports them after composition generation and [1, 4] at execution time, we consider policies directly in the composition process. In a previous work [11] we have developed a service composition approach based on graph planning. The work we present here can be seen as an extension of it that supports contexts and access policies.

Outline. After giving preliminaries on graph planning in Section 2, our models are introduced in Section 3. The description of the way the composition issue can be solved using an extension of graph planning is presented in Section 4. We end with conclusions and perspectives in Section 5. Due to lack of room, it was not possible to put all details in this paper. An extended version that includes a detailed example is available online in the authors' Web pages.

2 Preliminaries

Planning is *"the task of coming up with a sequence of actions that will achieve a goal"* [12]. A (propositional) *planning problem* can be modeled by a tuple $\mathbf{\Pi} = (\mathbf{P}, \mathbf{A}, \mathbf{I}, \mathbf{G})$, where $\mathbf{P}$ is a set of propositions, $\mathbf{A}$ is a set of actions, each action $\mathbf{a} \in \mathbf{A}$ with a set of preconditions $\mathbf{Pre}(\mathbf{a}) \subseteq \mathbf{P}$, a set of negative effects $\mathbf{Eff}^-(\mathbf{a}) \subseteq \mathbf{P}$, and a set of positive effects $\mathbf{Eff}^+(\mathbf{a}) \subseteq \mathbf{P}$ such that $= \mathbf{Eff}^-(\mathbf{a}) \cap \mathbf{Eff}^+(\mathbf{a}) = \emptyset$, also denoted with $\mathbf{a} = (\mathbf{Pre}(\mathbf{a}), \mathbf{Eff}^-(\mathbf{a}), \mathbf{Eff}^+(\mathbf{a}))$, $\mathbf{I} \subseteq \mathbf{P}$ is the input, or initial state, of the problem, and $\mathbf{G} \subseteq \mathbf{P}$ is the goal of the problem.

Two actions $\mathbf{a}$ and $\mathbf{b}$ are *independent* iff $\mathbf{Eff}^-(\mathbf{a}) \cap (\mathbf{Pre}(\mathbf{b}) \cup \mathbf{Eff}^+(\mathbf{b})) = \emptyset$ and $\mathbf{Eff}^-(\mathbf{b}) \cap (\mathbf{Pre}(\mathbf{a}) \cup \mathbf{Eff}^+(\mathbf{a})) = \emptyset$. A set of actions is independent when its actions are pairwise independent.

Among the different techniques to solve planning problems, Graphplan [3] is a technique that yields a compact representation of relations between actions and represent the whole problem world. It has proven to be very efficient and has been applied with success to Web service composition [11] and to composition repair [14]. The Graphplan algorithm is based on the computation of a *planning graph* (Fig. 1), which is a directed acyclic graph composed of interleaved layers called proposition layers $\mathbf{PL}_i \subseteq \mathbf{P}$ and actions layers $\mathbf{AL}_i \subseteq \mathbf{A}$.

The first proposition layer, $\mathbf{PL}_0$, is made up of the propositions in $\mathbf{I}$. The Graphplan algorithm then performs *graph expansion*. Given a proposition layer

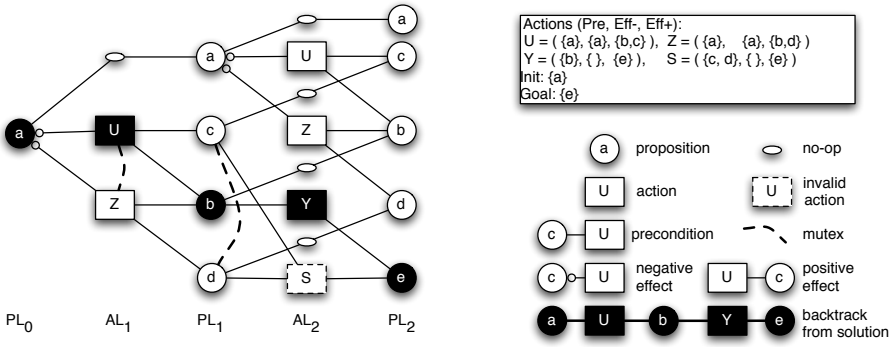

Fig. 1. Graphplan example

$\mathbf{PL}_i$, an action **a** is added in $\mathbf{AL}_i$ if its preconditions and negative effects are in $\mathbf{PL}_i$. If so, all positive effects of **a** are added in $\mathbf{PL}_i$, and **a** is connected with precondition arcs (resp. negative effect arcs) to its preconditions (resp. negative effects) in $\mathbf{PL}_{i-1}$ and with positive effect arcs to its positive effects in $\mathbf{PL}_i$. Specific actions (no-ops) are used to keep propositions from one layer to the next one. Graph planning also introduces the concept of mutual exclusion (mutex) between non independent actions. Mutual exclusion is reported from a layer to the next one while building the graph.

The expansion phase stops either when the objective is reached, *i.e.,* if **G** is included in $\mathbf{PL}_i$, without mutex between elements in **G**, or with a fix-point, *i.e.,* if $\mathbf{PL}_i = \mathbf{PL}_{i-1}$. In second case there is no solution to the planning problem. In the first case, the Graphplan algorithm then performs a *backtrack* from the goal propositions in $\mathbf{PL}_i$ to the initial proposition layer $\mathbf{PL}_0$. Planning graphs whose computations have stopped at level k enable to retrieve all solutions up to this level. Additionally, planning graphs enable to retrieve solutions in a concise form, taking benefit of actions that can be done in parallel (denoted ||).

An example is given in Figure 1. The extraction phase gives plans $\mathbf{U};\mathbf{Y}$, $\mathbf{Z};\mathbf{Y}$, and $(\mathbf{U}||\mathbf{Z});\mathbf{S}$. However, **U** and **Z** are in mutual exclusion. Accordingly, since there is no other way to obtain **c** and **d** than with exclusive actions, these two propositions are in exclusion in the next proposition layer, making **S** impossible. The only possible solution plans are therefore $\mathbf{U};\mathbf{Y}$ and $\mathbf{Z};\mathbf{Y}$. Note that other nodes are indeed in mutual exclusion but we have not represented this for clarity.

3 Modeling of the Composition Problem

In this section, we formalize the problem of composing form-based services in context-aware personal information spaces. We will present the different inputs of this composition problem. In Section 4 we will then address how this problem can be solved automatically using an extension of graph planning.

3.1 Ontologies

An ontology can be denoted by a tuple $\mathcal{O} = (\mathcal{C}, \mathcal{P}, \mathcal{R})$ where $\mathcal{C}$ are classes (semantic concepts), $\mathcal{P}$ are properties of concepts (either other concepts or literal values), and $\mathcal{R}$ are relations between concepts. We consider four possible kinds of relations in $\mathcal{R}$: *subsumption* ($\preceq$), *part-whole relationship* ($\sqsubset$), *equivalence*, or synonymy relationship ($\equiv$), and *disjunction* ($\perp$). Disjunction respects subsumption, *i.e.*, $\forall c_1, c_1', c_2, c_2' \in \mathcal{C}, c_2 \preceq c_1, c_2' \preceq c_1', c_1 \perp c_1' \Rightarrow c_2 \perp c_2'$. Further, $\preceq^*$ denotes the transitive closure of $\preceq$. In this work we sometimes rely on simpler ontologies, with only classes and a subsumption relation. In such a case we have $\mathcal{O} = (\mathcal{C}, \preceq)$.

3.2 Semantic Structures

In [6], we have developed the notion of semantic context-aware PIS based on three principles: modeling, contextualization, and instantiation of personal information. Modeling is achieved using an ontology, $\mathcal{O} = (\mathcal{C}, \mathcal{P}, \mathcal{R})$, that describes the personal information types (PIT), their properties, and relations. Contexts are also described with an ontology, $\mathcal{O}_{\text{Cont}} = (\mathcal{C}_{\text{Cont}}, \preceq_{\text{Cont}})$. Given two contexts c_1 and c_2, $c_2 \preceq c_1$ means that all property values that are valid for c_1 are also valid for c_2. To instantiate the user's personal information we have considered that a property value for a property p is defined by a tuple $pv = (v, c, \delta)$ where v is the value, c is a context of the context ontology, and δ is a real number in [0..1] that represents the usability of value v for property p in context c.

To foster automation of the composition process, we further assume that the user structures the PIS with reference to categories of personal information using an ontology $\mathcal{O}_{\text{Info}} = (\mathcal{C}_{\text{Info}}, \preceq_{\text{Info}})$. On the other side, within the context of a research project[3], we are working with an SME partner that develops services for administrations and enterprises. Each service is semantically annotated with semantic information for the forms fields, for the outputs it can produce, and with the set of functionalities, or capabilities, it achieves. The first two correspond to the PIT ontology. For the later we rely on an ontology of capabilities, $\mathcal{O}_{\text{Cap}} = (\mathcal{C}_{\text{Cap}}, \preceq_{\text{Cap}})$. Finally, services can be organized in categories too. For this we assume an ontology of service categories, $\mathcal{O}_{\text{Serv}} = (\mathcal{C}_{\text{Serv}}, \preceq_{\text{Serv}})$

3.3 Policies

The user may express policies on the use of the personal information to be given to services. Given a personal information category x (from $\mathcal{C}_{\text{Info}}$) and a service category y (from $\mathcal{C}_{\text{Serv}}$), a policy authorization (or authorization for short) is a couple (x, y), also denoted by $x \triangleright y$. Its meaning is that any personal information of a category that is x or subsumes it can be given to any service of a category that is y or subsumes it. A policy set, or policy for short, is a set of authorizations.

[3] Personal Information Management through Internet
`http://genibeans.com/cgi-bin/twiki/view/Pimi/WebHome`

3.4 Workflows and Procedures

Workflows capture the behavioral aspects of online procedures. Given a set of names A, used to label the basic activities, a simple (yet expressive) kind of workflow over A, WF^A, can be modeled following [7] by a tuple $(N, \rightarrow, \lambda)$. N is the set of workflow nodes. It can be further divided into disjoint sets $N = N_A \cup N_{SO} \cup N_{SA} \cup N_{JO} \cup N_{JA}$, where N_A are basic activities of the workflow, N_{SO} are XOR-split nodes, N_{SA} are AND-split nodes, N_{JO} are XOR-join nodes, and N_{JA} are AND-join nodes. XOR-split and XOR-join nodes enable to model exclusive choice, while AND-split and AND-join nodes enable to model parallelism. $\rightarrow \subseteq N \times N$ denotes the control flow, and $\lambda : N_A \rightarrow A$ is a function assigning names to activity nodes. We note $\bullet x = \{y \in N \mid y \rightarrow x\}$ and $x\bullet = \{y \in N \mid x \rightarrow y\}$. We require that workflows are well-structured and without loop. A signicant feature of well-structured workflows is that the XOR-splits and the OR-Joins, and the AND-splits and the AND-splits appear in pairs. Moreover, we require $|\bullet x| \leq 1$ for each x in $N_A \cup N_{SA} \cup N_{SO}$ and $|x \bullet| \leq 1$ for each x in $N_A \cup N_{JA} \cup N_{JO}$.

A procedure is the specification of functionalities that should be achieved to reach some goal. These functionalities correspond to capabilities that will be realized through the use of one or several form-based services. We may then model a procedure by a workflow labelled by capabilities, *i.e.,* defined over $\mathcal{C}_{\text{Cap}}$.

3.5 Services

Services require a set of inputs in order to produce outputs and achieve their capabilities. They are organized in categories. Given the ontologies introduced in 3.2, we model services as a tuple $w = (\text{u}, I, O, K, C)$ where u is the service URI (address of the service form), $I \subseteq \mathcal{P}$ are the service inputs, $O \subseteq \mathcal{P}$ are the service outputs, $K \subseteq \mathcal{C}_{\text{Cap}}$ are the service capabilities, and $C \in \mathcal{C}_{\text{Serv}}$ is the service category. In the sequel we suppose a set of services W being available to the user. Services may not have capabilities. These correspond for example to transformational services (*e.g.,* to retrieve a postal code from a city name).

3.6 Composition Requirement

The requirement of composition is to find out a correct sequence of groups of services, possibly executed in parallel, *i.e.,* a plan in the sense of planning, that altogether are able to achieve some procedure using only the data they produce and the personal information the user agrees to provide them with. Further, one may precise a specific context in which the procedure is to be executed and a minimal usability value for contextualized information (below this threshold information is not relevant). Given the ontologies introduced in 3.2, a composition requirement is a tuple $Req = (Proc, \underline{c}, \epsilon, W, PIS, Pol)$ where $Proc$ is the procedure one wants to achieve, $\underline{c} \in \mathcal{C}_{\text{Cont}}$ is the context in which we apply the procedure ($\top$ for none), $\epsilon \in [0..1]$ is the minimal acceptable usability degree, W are the available services, PIS is the user PIS (*i.e.,* the set of contextualized property values it contains), and Pol is the user policy set.

4 Automatic Encoding and Resolution of the Composition Problem

In Section 3, we have formalized the composition problem that we address. In this section, we present how it can be automatically solved using a planning problem encoding and by extending the Graphplan algorithm. The approach we follow is first to encode the procedure and the services as planning actions.

4.1 Procedure Encoding

We reuse here a transformation from workflows to Petri nets defined in [7] that has been modified to map planning actions in [11]. The behavioral constraints underlying the workflow semantics (*e.g.,* an action being before/after another one) are supported through two kinds to propositions: $\mathbf{r_{x,y}}$ and $\mathbf{c_{x,y}}$. We also have a proposition $\sharp$ for initial states, and a proposition $\surd$ for correct termination states. We may then define actions:

- for each $x \in N_{SA}$, we have an action $\mathbf{a} = \oplus x$, for each $x \in N_{JA}$, we have an action $\mathbf{a} = \overline{\oplus} x$, and for each $x \in N_A$, we have an action $\mathbf{a} = [\lambda(x)]x$. In all three cases, we set:
  $\mathbf{Pre}(\mathbf{a}) = \mathbf{Eff}^-(\mathbf{a}) = \bigcup_{y \in \bullet x} \{\mathbf{r_{x,y}}\}$, and $\mathbf{Eff}^+(\mathbf{a}) = \bigcup_{y \in x\bullet} \{\mathbf{c_{x,y}}\}$.
- for each $x \in N_{SO}$, for each $y \in x\bullet$, we have an action $\mathbf{a} = \otimes x, y$ and we set:
  $\mathbf{Pre}(\mathbf{a}) = \mathbf{Eff}^-(\mathbf{a}) = \bigcup_{z \in \bullet x} \{\mathbf{r_{x,z}}\}$, and $\mathbf{Eff}^+(\mathbf{a}) = \{\mathbf{c_{x,y}}\}$.
- For each $x \in N_{JO}$, for each $y \in \bullet x$, we have action $\mathbf{a} = \overline{\otimes} x, y$ and we set:
  $\mathbf{Pre}(\mathbf{a}) = \mathbf{Eff}^-(\mathbf{a}) = \{\mathbf{r_{x,y}}\}$ and $\mathbf{Eff}^+(\mathbf{a}) = \bigcup_{z \in \bullet x} \{\mathbf{c_{x,z}}\}$
- for each $x \to y$, we have an action $\mathbf{a} = \rightsquigarrow x, y$ and we set:
  $\mathbf{Pre}(\mathbf{a}) = \mathbf{Eff}^-(\mathbf{a}) = \{\mathbf{c_{x,y}}\}$ and $\mathbf{Eff}^+(\mathbf{a}) = \{\mathbf{r_{y,x}}\}$.
- additionally, for any initial action $\mathbf{a}$ we add $\sharp$ in $\mathbf{Pre}(\mathbf{a})$ and $\mathbf{Eff}^-(\mathbf{a})$.
- additionally, for any final action $\mathbf{a}$ we add $\surd$ in $\mathbf{Eff}^+(\mathbf{a})$.

The procedure and the services are inter-related by ordering constraints over the capabilities. Let us suppose a simple procedure with two capabilities in a sequence $k_1 \to k_2$, a service w_1 with capability k_1, and a service w_2 with capability k_2. w_1 should not be put in an action layer before capability k_1 is enabled. This is achieved after putting the action corresponding to step k_1 in the encoding of the procedure in an action layer. In turn, the actions encoding the following steps of the procedure (here only the $\rightsquigarrow$ action, that will enable the action for k_2 later on) should be blocked until capability k_1 has been planned, *i.e.,* here, w_1 has been put in an action layer. For this we propose to rely on two propositions for each capability k in $\mathcal{C}_{\text{Cap}}$: $\mathbf{enabled_k}$ and $\mathbf{done_k}$. In the encoding of the procedure workflow, we then replace any action $a = [k]x$ by two actions a' and $\bar{a'}$ and we set $\mathbf{Pre}(\mathbf{a'}) = \mathbf{Pre}(\mathbf{a})$, $\mathbf{Eff}^-(\mathbf{a'}) = \mathbf{Eff}^-(\mathbf{a})$, $\mathbf{Eff}^+(\bar{\mathbf{a}'}) = \mathbf{Eff}^+(\mathbf{a})$, $\mathbf{Eff}^+(\mathbf{a}) = \{\mathbf{enabled_k}, \mathbf{link}_x\}$, and $\mathbf{Eff}^-(\bar{\mathbf{a}'}) = \{\mathbf{done_k}, \mathbf{link}_x\}$, with $\mathbf{link}_x$ ensuring the correct ordering between a' and $\bar{a'}$.

4.2 Service Encoding

A service can be executed only if all its inputs are available and if its capacities are enabled by the current state of execution of the procedure. The service then generates its outputs and indicates that the capacities have been achieved. Each service $w = (\mathrm{u}, I, O, K, C)$ is therefore encoded as an action $\mathbf{a} = (\mathbf{Pre}(\mathbf{a}), \mathbf{Eff}^-(\mathbf{a}), \mathbf{Eff}^+(\mathbf{a}))$ with $\mathbf{Eff}^-(\mathbf{a}) = \bigcup_{k \in K} \{\mathbf{enabled_k}\}$, $\mathbf{Eff}^+(\mathbf{a}) = O \cup \bigcup_{k \in K} \{\mathbf{done_k}\}$, and $\mathbf{Pre}(\mathbf{a}) = I \cup \mathbf{Eff}^-(\mathbf{a})$.

4.3 Resolution of the Composition Problem

Once we have encoded the procedure and the services as planning actions, we can apply the 2-step Graphplan algorithm: graph expansion and then backtracking (see Sect. 2). The second step is the same as in the original algorithm [3]. However, due to the contextualization of data and the use of policies, the first step has to be modified as follows.

Contextual Propositions. As far as the encoding of personal information is concerned, we replace basic propositions by tuples $(p, \underline{c}, \delta)$ corresponding to the PIS property values. Such a tuple denotes that some value for property p in known for context $\underline{c}$ (the context used in the composition requirements) with usability degree δ. Other propositions, *e.g.,* corresponding to the encoding of the procedure, are regular with reference to graph planning.

Filtering Out. A preliminary optimizing step is to filter out any proposition corresponding to personal information p for which there is no enabling policy (some $x' \rhd \_ \in Pol$ where x, $x \preceq_{\mathrm{Info}}{}^* x'$, denotes the category of p). We also remove actions corresponding to services that are not allowed to use some of their inputs, *i.e.,* a service w with at least an input $p \in I(w)$ such that there is no $x' \rhd y' \in Pol$, with x, $x \preceq_{\mathrm{Info}}{}^* x'$, being the category of p and y, $y \preceq_{\mathrm{Serv}}{}^* y'$, being the category of w.

Initialization - $\mathbf{PL}_0$. The first proposition layer contains the initial proposition for the procedure, $\sharp$, together with propositions for the contextual property values in the PIS. For the later, for each property value (v, c, δ) for p, we compute a contextual proposition $(p, \underline{c}, \delta')$ where δ' is equal to $\delta \times \Delta_{c,\underline{c}}^{\mathcal{O}_{\mathrm{Cont}}}$ with $\delta \times \Delta_{c,\underline{c}}^{\mathcal{O}_{\mathrm{Cont}}}$ being the semantic similarity measure between two concepts in the ontology $\mathcal{O}_{\mathrm{Cont}}$ (see [6] for the way we compute this for contextual data querying). We put in $\mathbf{PL}_0$ all the $(p, \underline{c}, \delta')$ where δ' is maximal.

Expansion Basic Step. What changes here with reference to [3] is the condition and effect of adding an action $\mathbf{a}$ related to some service w in an action layer. First w should be authorized to use its input data. This is ensured by the filtering step, above. Second, all inputs required for w should be available in the current proposition layer with a degree higher than the threshold, *i.e.,* for each p in $I(w)$, there is some $(p, \underline{c}, \delta)$ in $\mathbf{PL}_{i-1}$ such that $\delta \geq \epsilon$. If so, for each p in $O(w)$,

we add $(p, \underline{c}, \delta')$ in $\mathbf{PL}_i$, where δ' is the minimal value of δ for all the inputs of w in $\mathbf{PL}_{i-1}$. Further, for every k in $K(w)$, we require $\mathbf{enabled_k} \in \mathbf{PL}_{i-1}$ and we add $\mathbf{done_k}$ in $\mathbf{PL}_i$. Since several services may produce the same data, at each step of the expansion we perform a cleaning by keeping in $\mathbf{PL}_i$ only the tuples with the maximal δ, *i.e.,* if we have $(p, \underline{c}, \delta_1)$ and $(p, \underline{c}, \delta_2)$ in $\mathbf{PL}_i$, with $\delta_1 \geq \delta_2$, we keep only the first one.

Nothing changes for actions related to the procedure encoding. Given such an action $\mathbf{a}$, we should have $\mathbf{Pre}(\mathbf{a}) \subseteq \mathbf{PL}_{i-1}$, $\mathbf{Eff}^-(\mathbf{a}) \subseteq \mathbf{PL}_{i-1}$, and then we add $\mathbf{Eff}^+(\mathbf{a})$ in $\mathbf{PL}_{i-1}$. Non independent actions are treated as in [3], using mutex.

Expansion Termination. Expansion stops with a fix point or with success. The later is reached when there is the $\surd$ proposition in the current proposition layer, which means that we have successfully completed the procedure. In such as case a solution can be obtained using backtrack. Fix point is if the current proposition layer $\mathbf{PL}_i$ add no new proposition with reference to $\mathbf{PL}_{i-1}$. In order to support data contextualization, we consider a tuple $(p, \underline{c}, \delta_i)$, $\delta_i \geq \epsilon$, to be a new proposition either if there is no tuple $(p, \underline{c}, \_)$ in $\mathbf{PL}_{i-1}$ or if there is a tuple $(p, \underline{c}, \delta_{i-1})$ such that $\delta_{i-1} < \epsilon$. This is because having a new contextual value with a degree above the requirement threshold may yield new possibilities for service-related actions.

4.4 Tool Support

We have defined an Eclipse Modeling Framework model for the models presented in this paper, namely composition requirements with their constituents, and planning problems. Using an ATL model-to-model transformation, we transform the former into the later and then dump the planning problem into a text file using an Acceleo model-to-text transformation. We have implemented our modifications to the graph planning expansion structure in a Java implementation of the Graphplan algorithm[4]. This operates on the textual planning problem file to retrieve solution plans. We are currently packaging our tool support to make it freely available.

5 Conclusion

In this paper we have presented a service composition approach that supports the contextualization of personal information and related user policies. This is achieved using an encoding as a planning problem and the extension of a graph planning technique, which provides full automation of the process. As future work, we plan to study the joint use of several contexts and the contextualization of services, *i.e.,* enabling context-oriented constraints for input and output data in online services. The usability degree we use can be seen as a form of non-functional information used in composition. We plan to study the combination of it with user-specific preferences over non-functional service attributes.

[4] `http://sourceforge.net/projects/jplan/`

Acknowledgement. This work is supported by project "Personal Information Management through Internet" (PIMI-ANR-2010-VERS-0014-03) of the French National Agency for Research.

References

1. Anand, P., Vladimir, K., Lalana, K., Anupam, J.: Enforcing policies in pervasive environments. In: Proc. of MobiQuitous (2004)
2. Bartalos, P., Bieliková, M.: Automatic Dynamic Web Service Composition: A Survey and Problem Formalization. Computing and Informatics 30(4), 793–827 (2012)
3. Blum, A., Furst, M.L.: Fast Planning Through Planning Graph Analysis. Artificial Intelligence 90(1-2), 281–300 (1997)
4. Dulay, N., Damianou, N., Lupu, E., Sloman, M.: A Policy Language for the Management of Distributed Agents. In: IJAOSE (2002)
5. Hutter, D., Volkamer, M.: Information Flow Control to Secure Dynamic Web Service Composition. In: Security in Pervasive Computing (2006)
6. Khéfifi, R., Poizat, P., Saïs, F.: Modeling and Querying Context-Aware Personal Information Spaces. In: Proc. of DEXA(2) (2012)
7. Kiepusewski, B.: Expressiveness and suitability of languages for control flow modelling in workflows. Queensland University of Technology, Brisbane (2003)
8. Marconi, A., Pistore, M.: Synthesis and Composition of Web Services. In: Proc. of SFM (2009)
9. Mostéfaoui, S.K., Hirsbrunner, B.: Towards a Context-Based Service Composition Framework. In: Proc. of ICWS (2003)
10. Mrissa, M., Benslimane, D., Maamar, Z., Ghedira, C.: Towards a semantic- and context-based approach for composing web services. IJWGS 1(3/4), 268–286 (2005)
11. Poizat, P., Yan, Y.: Adaptive Composition of Conversational Services through Graph Planning Encoding. In: Proc. of ISoLA(2) (2010)
12. Russell, S.J., Norvig, P., Canny, J.F., Malik, J.M., Edwards, D.D.: Artificial Intelligence: A Modern Approach. Prentice hall Englewood Cliffs (1995)
13. Sheshagiri, M., Sadeh, N., Gandon, F.: Using Semantic Web Services for Context-Aware Mobile Applications. In: Proc. of MobiSys (2004)
14. Yan, Y., Poizat, P., Zhao, L.: Repair vs. Recomposition for Broken Service Compositions. In: Proc. of ICSOC (2010)