Preference and Similarity-based Behavioral Discovery of Services

Farhad Arbab[1] and Francesco Santini[1,2]

[1] Centrum Wiskunde & Informatica, Amsterdam, Netherlands
[Farhad.Arbab,F.Santini]@cwi.nl
[2] Dipartimento di Matematica e Informatica, Università di Perugia, Italy
francesco.santini@dmi.unipg.it

Abstract. We extend Constraint Automata by replacing boolean constraints with semiring-based soft constraints. The obtained general formal tool can be used to represent preference-based and similarity-based queries, which allow a user more freedom in choosing the behavior of the service to finally use, among all possible choices. A user states his preferences through a "soft" query, and obtains results that satisfy this query with different levels of preference. The soft requirements may involve a parameter data of the service operations, or the (names of the) operations themselves. Moreover, we introduce a first implementation of the search procedure by using declarative (Soft) Constraint Programming.

1 Introduction

Service-orientation is a design paradigm to build computer software in the form of services. The term "service" refers to a set of related software functionalities that can be reused for different purposes. In this sense, the service becomes more important than the software. A *Service-Oriented Architecture* (*SOA*) offers some benefits as return on investment, organisational agility and interoperability as well as a better alignment between business and IT. In such loosely-coupled environments, the automatic discovery process becomes more complex, and a user's decision has to be supported taking into account his (often not crisp) preferences, some semantic information on the related domain knowledge, and the behavior signature of services, describing the sequence of its operations [16,10]. For instance, a user may need to find an on-line purchase service satisfying the following requirements: *i)* charging activity is before shipping activity, *ii)* to purchase a product, the requester first needs to log into the system and finally log out of the system, and iii) filling the electronic basket of the user may consist a succession of "add-to-basket" actions (a similar scenario is proposed in [16]).

In this paper, we define a formal framework that considers both user's preferences and (stateful) service behavior during the search procedure, in order to retrieve multiple results for the same preference-based query; in this way, the end user has the possibility to choose among different results by selecting the service that maximizes his requirements. In the above mentioned purchase scenario, for example, he may prefer to pay with a credit card instead of with a bank transfer. Moreover, using the same framework, we also show how it is possible to

represent similarity-based search, in order to find the services that perform operations "similar" to those requested. These services can be valid alternatives for the user in case the "best" service is not available at the moment, for example, due to failures or high number of requests. In Sec. 2 we report the related work, showing that no general formal framework has been proposed in the literature for such tasks, and this dearth is even more striking if we consider the behavior of the services (i.e. the sequence of the operations).

In Sec. 3 we summarize the background on soft constraints, showing the basic operations of this parametric preference-based framework [5,4].

As a first result of this paper, in Sec. 4 we extend Constraint Automata [2] in order to deal with preferences on data-constraints: instead of (classical) boolean constraints, we adopt semiring-based soft constraints, which can model any preference system as long as it can be cast into a semiring algebraic structure. However, also boolean constraints can be represented with semirings (see Sec. 3), and used in the same framework as well. Services behavior has been already described as Constraint Automata in [13].

In Sec. 5 we show how to model preference-based queries according to the theory presented in Sec. 4. The names of the service operations correspond to the names on the automaton transitions (i.e., the synchronization constraints [2]), and they represent the behavior of the service the user is interested in. At the same time, soft data-constraints model the preference on the data taken as I/O by each operation. For example, the data required for the *charging* operation can involve a bank transfer, a credit card number, or pay-on-delivery with cash. For instance, a user may prefer the second method over the other two. In Sec. 5 we also demonstrate that the formal results of Sec. 4, as the join and hiding operations on automata, and simulation/bisimulation relationships between automata, can be used to reason on preference-based queries.

In Sec. 6 we focus our attention on similarity-based search. In this scenario, a user is interested in retrieving also services with "similar" operations; therefore, soft constraints define a preference for the operation names, and not for their I/O data as in Sec. 5. For example, instead of the *Charging* service, a service named *SendEstimate* could be used by a user to receive a purchase-estimate (and then buy with a phone call). In this case, the names (and the services) *Charging* and *SendEstimate* are "similar". Moreover, we show how to solve this search problem as a *Soft Constraint Satisfaction Problem* (*SCSP*) [5,4].

We suppose the availability of meaning and similarity-scores of names as computed from a proper domain-specific ontology [14] (as proposed by other works in Sec. 2): in this paper we focus on the representation of preference/similarity-based queries, and the formal computational framework we propose solve them. Finally, in Sec. 7 we draw our final conclusions and explain our future work.

2 Related Work

In [16] the authors propose a new behavior model for WSs using automata and logic formalisms. Roughly, the model associates messages with activities and adopts the IOPR model (i.e., Input, Output, Precondition, Result) in *OWL-S* [14] to describe activities. An automaton structure is used to model the service

behavior. A new query language is developed to express temporal and semantic properties on service behaviors. Query evaluation algorithms are developed; an optimization approach using tree structures and heuristics is shown to improve the performance. However, similarity-based search is not mentioned in [16].

The model presented in [17] relies on a simple and extensible keyword-based query language and enables efficient retrieval of approximate results, including approximate service compositions. Since representing all possible compositions and all approximate concept references can result in an exponentially-sized index, the authors investigate clustering methods to provide a scalable mechanism for service indexing. In [9] the problem of behavioral matching is translated to a graph matching problem, and existing algorithms are adapted for this purpose.

In [3], the authors propose a crisp translation from interface description of WSs to classical crisp *Constraint Satisfaction Problems*. Therefore, no service behavior is considered and it is not possible to quantitatively reason on similarity/preference involving different services: it is not possible to widen the results of a query by obtaining similar services. In [19], a semiring-based framework is used to model and compose QoS features of WSs. However, no notion of similarity relationship is given in in [19].

In [7] the authors propose a novel clustering algorithm that groups names of parameters of web-service operations into semantically meaningful concepts. These concepts are then leveraged to determine similarity of inputs (or outputs) of web-service operations. In [15] the authors propose a framework of fuzzy query languages for fuzzy ontologies, and present query answering algorithms for these query languages over fuzzy *DL-Lite* ontologies.

In [10] the authors propose a metric to measure the similarity of semantic services annotated with an *OWL ontology*. Similarity is calculated by defining the intrinsic information value of a service description based on the "inferencibility" of each of *OWL Lite* constructs. The authors of [11] present an approach to hybrid semantic WSs matching that complements logic-based reasoning with approximate matching based on syntactic Information-Retrieval-based similarity computations. Finally, in [18], the authors propose a retrieval method to assess the similarity of available service interfaces with a provided desired-service description, extended to include semantically similar words according to *wordNet*.

The solution in this paper appears to be more general and comprehensive compared to the works in this section, since it can be adapted to any semiring-like metrics, and comes with several formal tools for reasoning over the queries (e.g., join and hiding in Sec. 5). Moreover, most of the proposed works do not consider the service behavior at all, but only their interfaces. In addition, we propose an implementation based on (*Soft*) *Constraint Programming*, which proves to be expressive and efficient at the same time, adopting (off-the-shelf) AI-based solving techniques in its underlying machinery.

3 Semirings and Soft Constraint Satisfaction Problems

A c-semiring [5] (simply semiring in the sequel) is a tuple $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$, where A is a possibly infinite set with two special elements $\mathbf{0}, \mathbf{1} \in A$ (respectively

the bottom and top elements of A) and with two operations $+$ and $\times$ that satisfy certain properties over A: $+$ is commutative, associative, idempotent, closed and $\mathbf{0}$ is its unit element and $\mathbf{1}$ is its absorbing element; $\times$ is closed, associative, commutative, distributes over $+$, $\mathbf{1}$ is its unit element, and $\mathbf{0}$ is its absorbing element. The $+$ operation defines a partial order $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$; we say that $a \leq_S b$ if b represents a value *better* than a. Moreover, $+$ and $\times$ are monotone on $\leq_S$, $\mathbf{0}$ is its min and $\mathbf{1}$ its max, $\langle A, \leq_S \rangle$ is a complete lattice and $+$ is its *least upper bound* operator (i.e., $a+b = lub(a, b)$). Some practical instantiations of the generic semiring structure are the *boolean* $\langle \{false, true\}, \vee, \wedge, false, true \rangle$,[3] *fuzzy* $\langle [0..1], \max, \min, 0, 1 \rangle$, *probabilistic* $\langle [0..1], \max, \hat{\times}, 0, 1 \rangle$ and *weighted* $\langle \mathbb{R}^+ \cup \{+\infty\}, min, \hat{+}, \infty, 0 \rangle$ (where $\hat{\times}$ and $\hat{+}$ respectively represent the arithmetic multiplication and addition).

Given a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and $a, b \in A$, we define the residuated negation [6] of a as $\neg a = \max\{b : b \times a = \mathbf{0}\}$, where max is according to the ordering defined by $+$ [6]. Note that over the boolean semiring the negation operator exactly corresponds to the logic negation, since $\neg 0 = \max\{b : b \times \mathbf{0} = \mathbf{0}\}$ and $b = 1$, while when $\neg 1 = \max\{b : b \times 1 = \mathbf{0}\}$ then the only possibility is $b = 0$.

A *soft constraint* [5] may be seen as a constraint where each instantiation of its variables has an associated preference. Given $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and an ordered finite set of variables V over a domain D, a soft constraint is a function which, given an assignment $\eta : V \to D$ of the variables, returns a value of the semiring, i.e. $c : (V \to D) \to A$. Let $\mathcal{C} = \{c \mid c : D^{|I \subseteq V|} \to A\}$ be the set of all possible constraints that can be built starting from S, D and V: any function in $\mathcal{C}$ depends on the assignment of only a (possibly empty) finite subset I of V, called the *support*, or *scope*, of the constraint. For instance, a binary constraint $c(x, y) = x + y$ (i.e., $\{x, y\} = I \subseteq V$) is defined on the support $supp(c) = \{x, y\}$. Note that $c\eta[v := d]$ means $c\eta'$ where η' is η modified with the assignment $v := d$. Note also that $c\eta$ is the application of a constraint function $c : V \to D \to A$ to a function $\eta : V \to D$; what we obtain is a semiring value $c\eta = a$.[4]

Given the set $\mathcal{C}$, the combination function $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ is defined as $(c_1 \otimes c_2)\eta = c_1\eta \times c_2\eta$ [5]; $supp(c_1 \otimes c_2) = supp(c_1) \cup supp(c_2)$. Given the set $\mathcal{C}$, the combination function $\oplus : \mathcal{C} \oplus \mathcal{C} \to \mathcal{C}$ is defined as $(c_1 \oplus c_2)\eta = c_1\eta + c_2\eta$ [4]; $supp(c_1 \oplus c_2) = supp(c_1) \cup supp(c_2)$. Informally, the $\otimes/\oplus$ builds a new constraint which associates with each tuple of domain values for such variables a semiring element which is obtained by multiplying/summing the elements associated by the original constraints to the appropriate sub-tuples. Given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the *projection* [5] of c over $V \backslash \{v\}$, written $c \Downarrow_{(V \backslash \{v\})}$ is the constraint c' such that $c'\eta = \sum_{d \in D} c\eta[v := d]$. Informally, projecting means eliminating some variables from the support. Given a soft constraint c, $\neg c$ is a constraint such that if $c\eta = a$ then $\neg c\eta = \neg a$ [6], where $supp(c) = supp(\neg c)$.

The partial order $\leq_S$ over $\mathcal{C}$ can be easily extended among constraints by defining $c_1 \sqsubseteq_S c_2 \iff c_1\eta \leq_S c_2\eta$. In order to define constraint equivalency we have $c_1 \equiv_S c_2 \iff c_1\eta =_S c_2\eta$ and $supp(c_1) = supp(c_2)$.

[3] The *boolean* semiring can be used to represent classical crisp constraints.

[4] the constraint function $\bar{a}$ always returns the value $a \in A$ to all assignments of domain values, e.g. $\bar{\mathbf{0}}$ and $\bar{\mathbf{1}}$ functions always return $\mathbf{0}$ and $\mathbf{1}$ respectively.

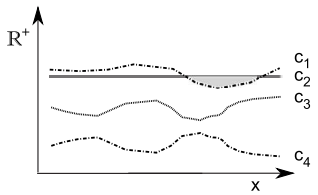

Fig. 1. A graphical representation of four *weighted* constraints, e.g., $c_2 = c_3 \otimes c_4$.

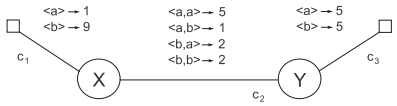

Fig. 2. A soft CSP based on a *weighted* semiring.

In Fig. 1 we show a graphical example of four *weighted* constraints (i.e., defined in the *weighted* semiring), where we have $c_3 \otimes c_4 = c_2$, $c_4 \sqsubseteq c_3$, $c_3 \sqsubseteq c_2$, $c_3 \sqsubseteq c_1$, but $c_2 \not\sqsubseteq c_1$ because of the gray region, where $c_1 \sqsubseteq c_2$ instead; moreover, in Fig. 1 we can see that $supp(c_1) = supp(c_2) = supp(c_3) = supp(c_4) = \{x\}$.

A SCSP [5] is defined as a quadruple $P = \langle S, V, D, C \rangle$, where $C \subseteq \mathcal{C}$ is the constraint set of the problem P. The *best level of consistency* notion defined as $blevel(P) = Sol(P) \Downarrow_{\emptyset}$, where $Sol(P) = \bigotimes C$ [5]. A problem P is α-consistent if $blevel(P) = \alpha$ [5]; P is instead simply "consistent" iff there exists $\alpha >_S \mathbf{0}$ such that P is α-consistent [5]. P is inconsistent if it is not consistent. Figure 9 shows a SCSP as a graph: S corresponds to the *weighted* semiring, i.e. $\langle \mathbb{R}^+ \cup \{+\infty\}, \min, \hat{+}, \infty, 0 \rangle$. Variables ($V = \{x, y\}$) and constraints ($C = \{c_1, c_2, c_3\}$) are represented respectively by nodes and arcs (unary for c_1 and c_3, and binary for c_2), and semiring values are written to the right of each variable assignment of the constraint, where $D = \{a, b\}$. The solution of P in Fig. 9 associates a preference to every domain value of x and y by combining all the constraints, i.e. $Sol(P) = \bigotimes C$. For instance, for the assignment $\langle a, a \rangle$ (that is, $x = y = a$), we compute the sum of 1 (which is the value assigned to $x = a$ in constraint c_1), 5 (which is the value assigned to $\langle x = a, y = a \rangle$ in c_2) and 5 (which is the value for $y = a$ in c_3). Hence, the resulting preference value for this assignment is 11. For the other assignments, $\langle a, b \rangle \to 7$, $\langle b, a \rangle \to 16$ and $\langle b, b \rangle \to 16$. The *blevel* for the example in Fig. 9 is 7, related to the assignment $x = a$, $y = b$.

4 Soft Constraint Automata

Constraint Automata were introduced in [2] as a formalism to describe the behavior and possible data flow in coordination models (e.g., the `Reo` language [2]); they can be considered as acceptors of *Timed Data Streams* (*TDS*) [1]. In this section we extend some of the definitions given in [2] in order to obtain *Soft Constraint Automata* (*SCA*). We now recall the definition of *TDS* from [1], while extending it using the softness notions provided in Sec. 3: we name this result as *Timed Weighted Data Streams* (*TWDS*). For convenience, we consider only infinite behavior and infinite streams that correspond to infinite "runs" of our soft automata, omitting final states including deadlocks.

Definition 1 (Timed Weighted Data Streams). *Let Data be any set and* $Data^{\omega} = \{\lambda \mid \lambda : \{0, 1, 2, \dots\} \to Data\}$, *where* $Data^{\omega}$ *is the set of infinite*

sequences over Data, and given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$*, a Timed Weighted Data Stream (*TWDS*) is defined as:*

$$TWDS = \{\langle \lambda, l, a\rangle \in Data^{\omega} \times \mathbb{R}^{\omega}_{+} \times A^{\omega} : \forall k \geq 0 : l(k) < l(k+1) \text{ and } \lim_{k \to +\infty} l(k)\}$$

Thus, a *TWDS* triplet $\langle \lambda, l, a\rangle$ corresponds to a data stream $\lambda \in Data^{w}$, a time string $l \in \mathbb{R}^{\omega}_{+}$ and a preference stream $a \in A^{\omega}$. The time stream l indicates, for each data item $\lambda(k)$, the moment $l(k)$ at which it is being input or output, while the preference stream provides a preference score for each $\lambda(k)$.

Constraint Automata [2] use a finite set $\mathcal{N}$ of names, e.g., $\mathcal{N} = n_1, \ldots, n_n$, where n_i stands for the i-th input/output port. The transitions of *SCA* are labeled with pairs consisting of a non-empty subset $N \subseteq \mathcal{N}$ and a soft (instead of crisp as in [2]) data-constraint c. Soft data-constraints can be viewed as an association of data assignments with a preference for that assignment. Formally,

Definition 2 (Soft Data-Constraints). *Soft data-constraints are functions* $c : (d_{N \subseteq \mathcal{N}} \to Data) \to A$ *defined over a semiring* $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$*, where* $d_{\mathcal{N}} = \{d_n \mid n \in \mathcal{N}\}$ *represents the data items associated with every port* n *in* $\mathcal{N}$*, and* Data *is the domain of values that pass through ports in* $\mathcal{N}$*. Soft data-constraints are given by the following grammar:*

$$c_{d_{N \subseteq \mathcal{N}}} ::= \bar{\mathbf{0}} \mid \bar{\mathbf{1}} \mid c_1 \oplus c_2 \mid c_1 \otimes c_2 \mid \neg c$$

where $d_{N \subseteq \mathcal{N}}$ *is the support of the constraint, i.e. the set of data names that determine its preference (see Sec. 3).*

Informally, a soft data-constraint is a function that returns a preference value $a \in A$ given an assignment for the names d_N in its support. In the sequel, we write $SDC(N, Data)$, for a non-empty subset N of $\mathcal{N}$, to denote the set of soft data-constraints. We will use SDC as an abbreviation for $SDC(N, Data)$. Note that in Def. 2 we assume a global data domain $Data$ for all names, but, alternatively, we can assign a data domain $Data_{n_i}$ for every name $n_i \in \mathcal{N}$.

We can say that an assignment η for the names d_N satisfies c with a preference of $a \in A$, if $c\eta = a$. Equivalence and logical implication for soft data-constraints are defined as in Sec. 3: we respectively write $c_1 \equiv c_2$, and $c_1 \sqsubseteq c_2$.

Note that by using the *boolean* semiring (see Sec. 3), thus within the same semiring-based framework, we can exactly model the "crisp" data-constraints presented in the original definition of Constraint Automata [2]. Therefore, Constraint Automata are contained in Def. 3. Note also that weighted automata, with weights taken from a proper semiring, have already been defined in the literature [8]; in *SCA*, weights are determined by a constraint function instead.

Definition 3 (Soft Constraint Automata). *A Soft Constraint Automaton is a tuple* $\mathcal{T}_S = (\mathcal{Q}, \mathcal{N}, \longrightarrow, \mathcal{Q}_0, S)$ *where* i) S *is a semiring* $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, ii) $\mathcal{Q}$ *is a finite set of states,* iii) $\mathcal{N}$ *is a finite set of names,* iv) $\longrightarrow$ *is a finite subset of* $\mathcal{Q} \times 2^{\mathcal{N}} \times SDC \times \mathcal{Q}$*, called the transition relation of* $\mathcal{T}_S$*, and* v) $\mathcal{Q}_0 \subseteq \mathcal{Q}$ *is the set of initial states. We write* $q \xrightarrow{N,c} p$ *instead of* $(q, N, c, p) \in \longrightarrow$*. We call* N *the name-set and* c *the guard of the transition. For every transition* $q \xrightarrow{N,c} p$

we require that i) $N \neq \emptyset$, and ii) $c \in SDC$ (see Def. 2). $\mathcal{T}_S$ is called finite iff $\mathcal{Q}, \longrightarrow$ and the underlying data-domain Data are finite.

The intuitive meaning of an *SCA* $\mathcal{T}_S$ as an operational model for service queries is similar to the interpretation of labeled transition systems as formal models for reactive systems. The states represent the configurations of a service. The transitions represent the possible one-step behavior, where the meaning of $q \xrightarrow{N,c} p$ is that, in configuration q, the ports in $N \in \mathcal{N}$ have the possibility of performing I/O operations that meet the soft guard c and that leads from configuration q to p, while the other ports in $\mathcal{N} \backslash N$ do not perform any I/O operation. d_N represent the names of the data associated with ports in N, i.e. the data transformed by the I/O operations through ports in N.

In Fig. 3 we show an example of a (deterministic) *SCA*. In Fig. 4 we define the *weighted* constraints c_1 and c_2 that describe the preference (e.g., a monetary cost) for the two transitions in Fig. 3, e.g. $c_1\eta[d_L := 2] = 5$.

Definition 4. *For a Soft Constraint Automaton $\mathcal{T}_S = (\mathcal{Q}, \mathcal{N}, \longrightarrow, \mathcal{Q}_0, \mathcal{S})$, a state $q \in \mathcal{Q}$, $N \subseteq \mathcal{N}$ and $P \subseteq \mathcal{Q}$, we define*

$$sdc_{\mathcal{T}_S}(q, N, P) = \bigoplus \{c : q \xrightarrow{N,c} p \text{ for some } p \in P\}$$

where $\bigoplus$ corresponds to the application of the $\oplus$ operator presented in Sec. 3 to all the constraints of the set ($\oplus$ is commutative and associative).

Intuitively, $sdc_{\mathcal{T}_S}(q, N, P)$ is the weakest (i.e., with the best preference) soft data-constraint that ensures the existence of an N-transition from state q to P. Note that $sdc_{\mathcal{T}_S}(q, N, P) = \bar{\mathbf{0}}$ if there is no N-transition from q to a P-state.

We define the language accepted by $\mathcal{T}_S$ as

$$\mathcal{L}_{TWDS} = \bigcup_{q \in Q_0} \mathcal{L}_{TWDS}(\mathcal{T}_S, q)$$

where $\mathcal{L}_{TWDS}$ denotes the language accepted by the state q (viewed as the starting state) of $\mathcal{T}_S$. The accepted languages on $\mathcal{N} = \{L, M\}$ are defined as the set of all TWDS pairs $\langle\langle \lambda, l, a_i\rangle, \langle \mu, m, a_j\rangle\rangle$ that have an infinite run in $\mathcal{T}_S$ starting in state q. The data streams λ and μ correspond to the data elements that flow through, respectively, L and M; l and m contain the time instants at which these operations take place. $\mathcal{L}_{TWDS}(\mathcal{T}_S, q)$ consists of all pairs $\langle\langle \lambda, l, a_i\rangle, \langle \mu, m, a_j\rangle\rangle$ such there exists a transition $q \xrightarrow{N,c} \bar{q}$ that satisfies the following conditions:

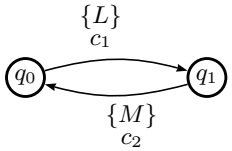

Fig. 3. A Soft Constraint Automaton.

$c_1 : (\{d_L\} \to \mathbb{N}) \to \mathbb{R}^+$ s.t. $c_1(d_L) = d_L + 3$

$c_2 : (\{d_M\} \to \mathbb{N}) \to \mathbb{R}^+$ s.t. $c_2(d_M) = d_M + 5$

Fig. 4. c_1 and c_2 in Fig 3.

$$l(0) < m(0) \wedge N = \{L\} \wedge c\eta[d_L := \lambda(0)] = a_i \wedge \langle\langle \lambda', l', a_i' \rangle, \langle \mu, m, a_j \rangle\rangle \in \mathcal{L}_{TWDS}(\mathcal{T}_S, \bar{q})$$

$$m(0) < l(0) \wedge N = \{M\} \wedge c\eta[d_M := \mu(0)] = a_j \wedge \langle\langle \lambda, l, a_i \rangle, \langle \mu', m', a_j' \rangle\rangle \in \mathcal{L}_{TWDS}(\mathcal{T}_S, \bar{q})$$

$$l(0) = m(0) \wedge N = \{L, M\} \wedge c\eta[d_L := \lambda(0), d_M := \mu(0)] = a_k \wedge \langle\langle \lambda', l', a_i' \rangle, \langle \mu', m', a_j' \rangle\rangle$$

where $\langle\langle \lambda', l', a_i' \rangle, \langle \mu', m', a_j' \rangle\rangle \in \mathcal{L}_{TWDS}$ and $a_i, a_j, a_k >_S \mathbf{0}$. Although the above definition is circular in case $q = \bar{q}$, a proper monotone operator can be formally defined [2]. As an example, the language accepted by the automaton in Fig. 3 equals the set $\{\langle\langle \lambda, l, a_i \rangle, \langle \mu, m, a_j \rangle\rangle \in TWDS \times TWDS \mid l < m < l'\}$.

We now define the soft join-operator of two *SCA*, performing the (natural) join of two $\mathcal{L}_{TWDS}$. We can use this operator to merge two queries (see Sec. 5).

Definition 5 (Soft Product-Automaton (soft join)). *The soft product-automaton of two* SCA $\mathcal{T}_S^1 = (\mathcal{Q}_1, \mathcal{N}_1, \longrightarrow_1, \mathcal{Q}_{0,1}, \mathcal{S})$ *and* $\mathcal{T}_S^2 = (\mathcal{Q}_2, \mathcal{N}_2, \longrightarrow_2, \mathcal{Q}_{0,2}, \mathcal{S})$ *on the same semiring* S *is defined as* $\mathcal{T}_S^1 \bowtie \mathcal{T}_S^2 = (\mathcal{Q}_1 \times \mathcal{Q}_2, \mathcal{N}_1 \cup \mathcal{N}_2, \longrightarrow, \mathcal{Q}_{0,1} \times \mathcal{Q}_{0,2})$*, where* $\longrightarrow$ *is given by the following two rules:*

$$\frac{q_1 \xrightarrow{N_1, c_1} p_1, q_2 \xrightarrow{N_2, c_2} p_2, N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, c_1 \otimes c_2} \langle p_1, p_2 \rangle} \quad (1) \qquad \frac{q_1 \xrightarrow{N, c} p_1, N \cap \mathcal{N}_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N, c} \langle p_1, p_2 \rangle} \quad (2)$$

The first rule is applied when there are two transitions in the automata which can be fired together. This happens only if there is no shared name in the two automata that is present on one of the transitions, but not present on the other one. In this case, the transition in the resulting automaton is labelled with the union of the name sets on both transitions, and the data-constraint is the conjunction of the data-constraints of the two transitions. The second rule is applied when a transition in one automaton can be fired independently of the other automaton, which happens when the names on the transition are not included in the other automaton. The proof for correctness of the soft join directly derives from the correctness of the crisp join [2].

The hiding operator [2] abstracts the details of the internal communication in the *SCA*, and shows the observable external behaviour of a query. In *SCA*, the hiding operator $\exists O[\mathcal{T}_S]$ (see Def. 6) removes all information about names $O \subseteq \mathcal{N}$, and removes the influence of the names in O from the *SDC* of $\mathcal{T}_S$: this operator removes d_O from the support of all soft constraints in $\mathcal{T}_S$.

Definition 6 (Hiding in Soft Constraint Automata). *Let* $\mathcal{T}_S = (\mathcal{Q}, \mathcal{N}, \longrightarrow, \mathcal{Q}_0, S)$ *be an* SCA *and* $N, O \subseteq \mathcal{N}$*. The* SCA $\exists O[T] = (\mathcal{Q}, \mathcal{N} \backslash O, \longrightarrow_O, \mathcal{Q}_{0,O}, S)$ *is defined as follows. Let* $\rightsquigarrow^*$ *be the transition relation such that* $q \rightsquigarrow^* p$ *iff there exists a finite path* $q \xrightarrow{O, c_1} q_1 \xrightarrow{O, c_2} q_2 \xrightarrow{O, c_3} \ldots \xrightarrow{O, c_n} q_n$*, where* $q_n = p$ *and* $c_1, \ldots, c_n$ *are satisfiable (i.e.,* $c_i \not\equiv \bar{\mathbf{0}}$*) and* $\forall c_i.supp(c_i) = d_O$*. The set* $Q_{0,O}$ *of initial states is* $Q_0 \cup \{p \in Q : q_0 \rightsquigarrow^* p \text{ for some } q_0 \in Q_0\}$*. The transition relation* $\longrightarrow_O$*, where* $\Downarrow$ *is the soft constraint projection described in Sec. 3, is given by:*

$$\frac{q \rightsquigarrow^* p, p \xrightarrow{N, c} r, N' = N \backslash O \neq \emptyset, c' = c \Downarrow_{d_{N \backslash O}}}{q \xrightarrow{N', c'}_O r}$$

5 Expressing Preference-based Queries

In this section we use *SCA* (see Sec. 4) to model the queries we adopt to describe *i)* the behavior of the services a user is interested in, and *ii)* the preferences of the user with respect to data exchanged through I/O by the operations. The behavioral signature is described via the automaton states: the operations are described by the names on the transitions of the automata, as described in Sec. 4; the ordering of the operations is enforced by the ordering of the reached states. In addition, we use *SDC* (see Def. 2) to model user's preferences for the data used by the service operations. Assigning these names to the actual names of the services in the database leads to a global preference for that service.

Our model assumes, ignoring details, the existence of a standard vocabulary (i.e., a domain-specific ontology) for messages and activities (e.g., OWL-S [14]). Therefore, we suppose that all the names in the following examples are properly obtained from an ontology on services. Ontologies have already been used in the literature to help preference and similarity-based searches (see Sec. 2), and serve as a common dictionary for queries and services.

In Fig. 5 we show two first examples of soft queries: with q_0 the user looks for a bibliography-search service that is able to search for conference papers by *Author*, while in case q_1 the search is by *Title*. The preferences of the user are summarized by the two soft constraints c_1 and c_2, which are represented in Fig. 6. These examples can be modeled with the *fuzzy* semiring $\langle [0..1], \max, \min, 0, 1 \rangle$: c_1 states that the user prefers to have a search by first name (with a fuzzy score of 1), rather than to have it by full name (i.e. 0.8) or by last name (i.e., 0.2), while c_2 states that the user prefers to have a search using the conference title instead of the paper title. The preference is equal to the bottom value of the semiring where not expressed (here, $\mathbf{0} = 0$). The scenario for these queries corresponds to the fact that the user remembers the first name of the author, or the conference where he met the author, but he has a vague memory of the author's last name, and of the title of the paper the author presented at that conference.

Suppose now that our database contains the four services represented in Fig. 7. According to the preferences expressed by c_1 and c_2 in Fig. 6, queries q_0 and q_1 in Fig. 5 return different preferences for each service, depending on the instantiation of variables d_{Author} and d_{Title}. Considering q_0, services a, b, and d have a respective preference of 0.2, 1, and 0.8. If query q_1 is used instead, the possible results are services c and d, with respective preferences of 1 and

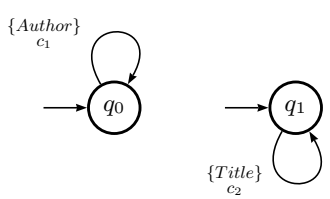

Fig. 5. Two soft Constraint Automata representing two different queries.

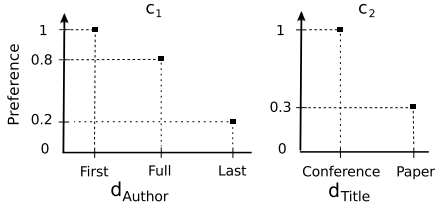

Fig. 6. The definition of c_1 and c_2 in Fig. 5.

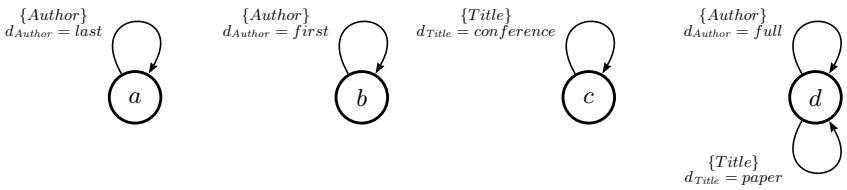

Fig. 7. A database of services for the query in Fig. 5; d perfoms both kinds of search.

0.3. When more than one service is returned as the result of the search, the end user has the freedom to choose the best one according to his preferences: for the first query q_0, he can opt for service b, which corresponds to a preference of 1 (i.e., the top preference), while for query q_1 he can opt for c (top preference as well). A possible programming-like declaration for service a in Fig. 7 is "`void Author(last)`". Note that a fifth possible service in the database may implement a search by name initials, but according to c_1 in Fig. 6, the user's preference for this service would be 0, i.e. $c\eta[d_{Author} := Initials] = 0$.

Note also that we can define n-ary soft constraints for more than one input data at the same time, in order to relate the preference for the values of two or more I/O data. For example, if we want to search by author and title at the same time, we can add a binary constraint c on $d_{Author,Title}$, such that $c\eta[d_{Author} := First, d_{Title} := Conference] = a_1$, $c\eta[d_{Author} := First, d_{Title} := Paper] = a_2$, and $a_1 >_S a_2$. This means that, when we know the first name of the author, we prefer to search using the title of the conference, instead of the paper title.

In Fig. 8 we provide a more complex example of a soft query, where we show the classical on-line purchase scenario cited in Sec. 1. In this case, the requirements of the user are *i)* a login/logout service, *ii)* an electronic basket that can be filled with the user's orders (at least one item has to be added before proceeding further), *iii)* a decision on the shipping method and, finally, *iv)* a payment service. Therefore, this query is expressed with the help of four different states modeling its behavior. The *SDC* expressing the user's preferences are represented in Fig. 8, where we suppose that the user expresses no preference for the data concerning the *Logout* service, i.e., $c_2 = \bar{\mathbf{1}}$. Note that, after the payment, the user can purchase a successive order without logging out.

In the following we show that the join and hiding operators presented in Sec. 4 can be used to operate on queries. Query composition is useful in order

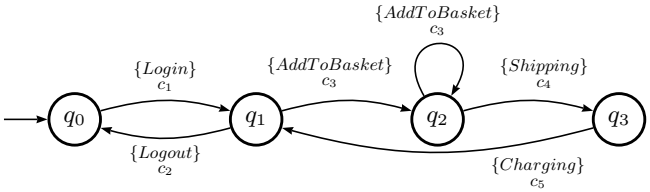

Fig. 8. A more complex soft query for the on-line purchase scenario presented in Sec. 1.

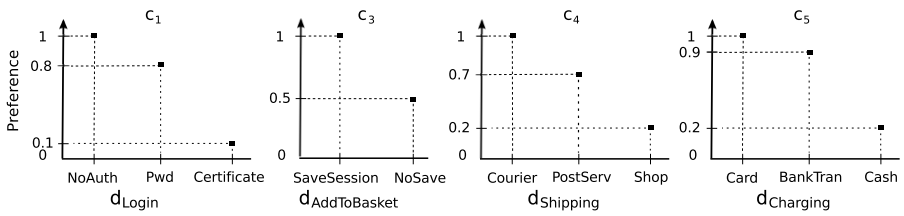

Fig. 9. The definition of c_1, c_3, c_4, c_5 for the query in Fig. 8 (we suppose $c_2 = \bar{\mathbf{1}}$).

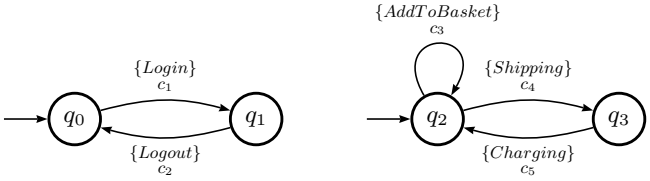

Fig. 10. Two queries that can be composed (i.e., with join) to obtain Fig. 8.

to reuse part of a query into another one, or to split the query into different knowledge domains. For example, in Fig. 4 the first query can be decided by the internal IT department of the company that needs to use the service, according to the internal security regulations, whereas the second query can be decided by the purchasing department of the company.

The hiding operator presented in Sec. 4 can be used to hide some over constraining information from the query, if , for instance, the previous search has led to a "no result" answer for the user. If we suppose to ask query q_0 in Fig. 11 having a database as the one represented in Fig. 7, no result is returned because no service implements a search by *Author* and *Title* at the same time. Then the user may relax the query by hiding *Title*, and ask again obtaining as possible response services a, b and d in Fig. 7.

5.1 Formally Reasoning on the Similarity of Queries

In the sequel we redefine the notions of bisimulation and simulation [2] by considering soft constraints, instead of crisp ones.

Definition 7 (Soft Bisimulation). *Let $\mathcal{T}_S = (\mathcal{Q}, \mathcal{N}, \longrightarrow, \mathcal{Q}_0, S)$ be an* SCA *and let $\mathcal{R}$ be an equivalence relation on $\mathcal{Q}$. $\mathcal{R}$ is called a soft bisimulation for $\mathcal{T}_S$ if, for all pairs $(q_1, q_2) \in \mathcal{R}$, all $\mathcal{R}$-equivalence classes $P \in \mathcal{Q}\backslash\mathcal{R}$, and every $N \subseteq \mathcal{N}$: $sdc_{\mathcal{T}_S}(q_1, N, P) \equiv sdc_{\mathcal{T}_S}(q_2, N, P)$.*

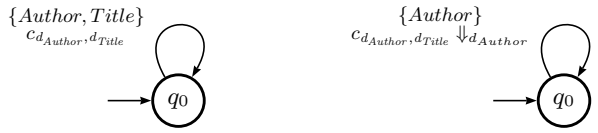

Fig. 11. Hiding information in a soft query.

As a reminder, $c_1 \equiv c_2$ iff $c_1\eta = c_2\eta = a$ for each assignment η (see Sec. 3). States q_1 and q_2 are called bisimulation-equivalent ($q_1 \sim q_2$) iff there exists a bisimulation $\mathcal{R}$ with $(q_1, q_2) \in \mathcal{R}$. Two automata $\mathcal{T}_S^1$ and $\mathcal{T}_S^2$ are bisimulation-equivalent ($\mathcal{T}_S^1 \sim \mathcal{T}_S^2$) iff their initial states are bisimulation-equivalent [2].

Definition 8 (Soft Simulation). *Let $\mathcal{T}_S = (\mathcal{Q}, \mathcal{N}, \longrightarrow, \mathcal{Q}_0, S)$ be an* SCA *and let $\mathcal{R}$ be a binary relation on $\mathcal{Q}$. $\mathcal{R}$ is called a soft simulation for $\mathcal{T}_S$ if, for all pairs $(q_1, q_2) \in \mathcal{R}$, all $\mathcal{R}$-upward closed sets $P \subseteq \mathcal{Q}$, and every $N \subseteq \mathcal{N}$: $sdc_{\mathcal{T}_S}(q_1, N, P) \sqsubseteq sdc_{\mathcal{T}_S}(q_2, N, P)$.*

An automaton $\mathcal{T}_S^2$ simulates another automaton $\mathcal{T}_S^1$ iff every initial state of $\mathcal{T}_S^1$ is simulated by an initial state of $\mathcal{T}_S^2$; this relationship is denoted as $\mathcal{T}_S^1 \preceq \mathcal{T}_S^2$.

Soft bisimulation can be seen as a method to check the equivalence of two $\mathcal{L}_{TWDS}$ languages, while soft simulation concerns language inclusion, as explained in [2] for the crisp version. Moreover, since our timed streams are weighted as explained in Def. 1 we can prove the following proposition:

Proposition 1. *Given two automata $\mathcal{T}_S^1$ and $\mathcal{T}_S^2$ able to parse the languages $\mathcal{L}_{TWDS_1}$ and $\mathcal{L}_{TWDS_2}$ respectively, and $\langle \lambda, l, a_i \rangle \in TWDS_1$ and $\langle \mu, m, a_j \rangle \in TWDS_2$, then:*

- *If $\mathcal{T}_S^1 \sim \mathcal{T}_S^2$ and $\lambda(k) = \mu(k)$, then $a_i =_S a_j$.*
- *If $\mathcal{T}_S^1 \preceq \mathcal{T}_S^2$ and $\lambda(k) = \mu(k)$, then $a_i \geq_S a_j$.*

The proof derives from the fact that, if $\mathcal{T}_S^1 \sim \mathcal{T}_S^2$, then $sdc_{\mathcal{T}_S^1}(q_1, N, P) \equiv sdc_{\mathcal{T}_S^2}(q_2, N, P)$, and, if $\mathcal{T}_S^1 \preceq \mathcal{T}_S^2$, $sdc_{\mathcal{T}_S^1}(q_1, N, P) \sqsubseteq sdc_{\mathcal{T}_S^2}(q_2, N, P)$.

Bisimulation can be used to check if two queries are equivalent, that is if they search the same services with the same preferences expressed by a user. Simulation between $\mathcal{T}_S^1 \preceq \mathcal{T}_S^2$ can instead be used to check if the query expressed through $\mathcal{T}_S^1$ is entailed by $\mathcal{T}_S^2$, and, consequently, the latter's returned services are a subset of the former's. Note that simulation/bisimulation for weighted automata (i.e., not for *SCA*) has already been defined in the literature [8]

6 Similarity-based Queries

In Sec. 5 we adopted the theory extended in Sec. 4 to represent the queries using crisp synchronization constraints (i.e., "crisp names") and soft data-constraints. This way, it is possible to guide the search according to the preferences of the user concerning the data used by the service operations.

In this section, we focus on similarity-based search, instead of preference-based one: transition names are not crisp anymore, but we allow for different operations considered somehow similar for the purposes of the user's query. Note that a similar service can be used, e.g., when the "preferred" one is down due to a fault, or when it offers bad performances due to the high number of requests. Definition 9 formalizes the notion of soft synchronization-constraint.

Definition 9 (Soft Synch.-constraint). *A soft synchronization-constraint is a function $c : (V \to \mathcal{N}) \to A$ defined over a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$, where V is a finite set of variables for each I/O port, and $\mathcal{N}$ is the set of I/O port names of the* SCA*.*

For example, suppose that a user asks only query q_0 in Fig. 5. The possible results are services a, b and d in Fig. 7, since service c only performs a search based on the *Title* of the paper, and not on the *Author*. However, the two services are very similar, and a user may be satisfied also by retrieving (and then, using) service c. This can be accomplished with the query in Fig. 12, where $c_x\eta[x := Author] = 1$, and $c_x\eta[x := Title] = 0.7$. In Fig. 13 we show a similarity-based query for our on-line purchase scenario: in this case, we have $V = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, and the domain for each of these variables is $\mathcal{N} = \{MutualLogin, Login, Logout, AddToBasket, Shipping, Charging, SendEstimate\}$. A user can also choose for a mutual login in the first step, and for an estimate of the price instead of a direct charging.

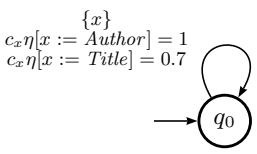

Fig. 12. A similarity-based query for the *Author* / *Title* example.

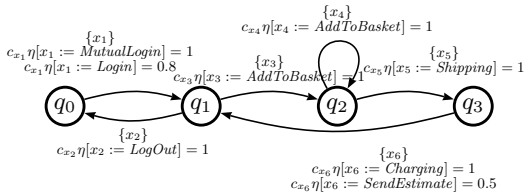

Fig. 13. A similarity-based query for the on-line purchase service.

6.1 A Mapping to Soft Constraint Satisfaction Problems

In this section we map our similarity-based search into a SCSP P (see Sec. 3); by solving P, we find the services in the database closest to the requirements expressed by a user's query. We use this solving method for two fundamental reasons: first, constraint programming is a declarative and very expressive means to quickly define the acceptable results in terms of constraint relationships. Second, SCSPs (and in general, Constraint Programming) come with several AI-based techniques that can be used to cut the search space and improve efficiency. For example, an α-cut on the branch-and-bound search can be used to stop the search as soon as the preference of the partial solution becomes worse than a threshold specified by the user. In this way, we can find only the α-consistent solutions (see Sec. 3), thus sparing computational resources for those solutions with a worse-than-α preference, which would not be selected by the user after all. This is particularly desirable with very large databases of services, containing interfaces with thousands of operations and thousands of behavioral states.

Mapping. We propose a mapping $\mathcal{M}$ such that, given the *SCA* $\mathcal{T}_S = (\mathcal{Q}, \mathcal{N}, \longrightarrow, \mathcal{Q}_0, S)$ (i.e., our query), and a database of services $DB = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_h\}$ represented by h crisp constraint automata [2], we obtain $\mathcal{M}(\mathcal{T}_S, DB) = P$, where $P = \langle S, V, D, C\rangle$ is a SCSP (see Sec. 3). For each transition i in the automaton $\mathcal{T}_S$ modeling the query, we use two variables x_i, y_i representing the source and destination states of the transition. A variable z_i represents the operation

names that we associate with transition i. Therefore, $V = \bigcup\{x_i, y_i, z_i\}$, for $i = 1\,..\,(|\longrightarrow_i| \in \mathcal{T}_S)$. Concerning the domain of the variables in V, $\forall x_i, y_i \in V.D_{x_i,y_i} = \{\mathcal{Q}_1 \cup \mathcal{Q}_2 \cup \cdots \cup \mathcal{Q}_h\} = \mathcal{Q}_{DB}$ (where, e.g., $\mathcal{Q}_1 \in \mathcal{T}_1$), and $\forall z_i \in V.D_{z_i} = \{N \mid N \subseteq \mathcal{N}_{DB}\}$, where $\mathcal{N}_{DB}$ is the set of operation names used by the services in DB. k is the total number of states for the services in the database, i.e., $k = |\ \mathcal{Q}_{DB}\ |$. We identify two different classes of constraints to build $C \in P$:

Automaton-structure constraints. With this set of constraints, we force the solutions (i.e., the crisp automata in DB) to respect the structure of automaton $\mathcal{T}_S$ modeling the query. For each $q_a \xrightarrow{N_l} q_b \in DB$ $(q_a, q_b \in \mathcal{Q}_{DB})$, we have $c_{x_i,y_i}(x_i = q_a, y_i = q_b) = \mathbf{1}$, and $\mathbf{0}$ otherwise. In addition, we also need to enforce the sequence of the states/transitions in the solution, according to the one expressed by the query. For example, if we have $q_{x_i} \longrightarrow q_{y_i}$ and $q_{x_j} \longrightarrow q_{y_j}$, and $y_i = x_j$ in $\mathcal{T}_S$, we need to add $c_{x_j,y_i}(x_j = y_i) = \mathbf{1}$, and $\mathbf{0}$ otherwise.

Name-preference constraints. These constraints model the preference for the name on the transition. For each $q_a \xrightarrow{N_l} q_b \in DB$, $c_{x_i,y_i,z_i}(x_i = q_a, y_i = q_b, z_i = n_l) = a$, where $a \in A$ (the set of preferences in S) represents the preference for name n_l.

Example 1. Here we list all the *automaton-structure constraints* with a preference better than $\mathbf{0}$, that model the query in Fig 13: $c_{x_1,y_1}(x_1 = q_a, y_1 = q_b) = 1$, $c_{x_2,y_2}(x_2 = q_c, y_2 = q_d) = 1$, $c_{x_1,y_2}(x_1 = y_2) = 1$, $c_{x_2,y_1}(x_2 = y_1) = 1$, $c_{x_3,y_3}(x_3 = q_a, y_3 = q_b) = 1$, $c_{x_3,y_2}(x_3 = y_2) = 1$, $c_{x_4,y_4}(x_4 = q_a, y_4 = q_b) = 1$, $c_{x_4,y_4}(x_4 = y_4) = 1$, $c_{x_4,y_3}(x_4 = y_3) = 1$, $c_{x_5,y_5}(x_5 = q_a, y_5 = q_b) = 1$, $c_{x_5,y_4}(x_5 = y_4) = 1$, $c_{x_6,y_6}(x_6 = q_a, y_6 = q_b) = 1$, $c_{x_6,y_5}(x_6 = y_5) = 1$, $c_{x_3,y_6}(x_3 = y_6) = 1$, for each $q_a \longrightarrow q_b$ in the service database.
In the following we list also the *name-preference constraints* with a preference better than $\mathbf{0}$: $c_{x_1,y_1,z_1}(x_1 = q_a, y_1 = q_b, z_1 = \{Login\}) = 0.8$, $c_{x_1,y_1,z_1}(x_1 = q_a, y_1 = q_b, z_1 = \{MutualLogin\}) = 1$, $c_{x_2,y_2,z_2}(x_2 = q_a, y_2 = q_b, z_2 = \{Logout\}) = 1$, $c_{x_3,y_3,z_3}(x_3 = q_a, y_3 = q_b, z_3 = \{AddToBasket\}) = 1$, $c_{x_4,y_4,z_4}(x_4 = q_a, y_4 = q_b, z_4 = \{AddToBasket\}) = 1$, $c_{x_5,y_5,z_5}(x_5 = q_a, y_5 = q_b, z_5 = \{Shipping\}) = 1$, $c_{x_6,y_6,z_6}(x_6 = q_a, y_6 = q_b, z_6 = \{Charging\}) = \mathbf{1}$ and $c_{x_6,y_6,z_6}(x_6 = q_a, y_6 = q_b, z_6 = \{SendEstimate\}) = 0.5$, for each $q_a \xrightarrow{n_l} q_b$ in the service database.

7 Conclusion

In this paper, we have proposed a general formal framework to express both preference and similarity-based queries for the SOC paradigm, as WSs. This framework has evolved from Constraint Automata [2] (to model the behavior of services) and semiring-based soft constraint [5,4] (to model preference values): we merged these two ingredients to obtain *SCA*, which comprise the tool we use to model these kinds of queries. The resulting framework can parametrically deal with different systems of semiring-based preferences.

In the future, we will automate the search by implementing the mapping proposed in Sec. 6.1 into a real constraint programming environment, such as *CHOCO* [12], and test the performance results. Moreover, we want to unify both preference and similarity-based queries in a single framework, to be able to express similarity-based queries with preferences on the input data.

References

1. Arbab, F., Rutten, J.J.M.M.: A coinductive calculus of component connectors. In: Recent Trends in Algebraic Development Techniques (WADT) Revised Selected Papers. LNCS, vol. 2755, pp. 34–55. Springer (2002)
2. Baier, C., Sirjani, M., Arbab, F., Rutten, J.J.M.M.: Modeling component connectors in Reo by constraint automata. Sci. Comput. Program. 61(2), 75–113 (2006)
3. Benbernou, S., Canaud, E., Pimont, S.: Semantic web services discovery regarded as a constraint satisfaction problem. In: Flexible Query Answering Systems, 6th International Conference. LNCS, vol. 3055, pp. 282–294. Springer (2004)
4. Bistarelli, S.: Semirings for Soft Constraint Solving and Programming. SpringerVerlag (2004), http://dl.acm.org/citation.cfm?id=993462
5. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. J. ACM 44(2), 201–236 (1997)
6. Bistarelli, S., Santini, F.: A nonmonotonic soft concurrent constraint language to model the negotiation process. Fundam. Inform. 111(3), 257–279 (2011)
7. Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J.: Similarity search for web services. In: Proceedings of Very large data bases. vol. 30, pp. 372–383. VLDB Endowment (2004), http://dl.acm.org/citation.cfm?id=1316689.1316723
8. Droste, M., Kuich, W., Vogler, H.: Handbook of Weighted Automata. Springer Publishing Company, Incorporated, 1st edn. (2009)
9. Grigori, D., Corrales, J.C., Bouzeghoub, M.: Behavioral matchmaking for service retrieval. In: IEEE International Conference on Web Services (ICWS). pp. 145–152. IEEE Computer Society (2006)
10. Hau, J., Lee, W., Darlington, J.: A semantic similarity measure for semantic web services. In: Web Service Semantics Workshop at WWW (2005)
11. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with OWLS-MX. In: Proceedings of Autonomous agents and multiagent systems. pp. 915–922. AAMAS '06, ACM, New York, NY, USA (2006), http://doi.acm.org/10.1145/1160633.1160796
12. Laburthe, F.: CHOCO: implementing a CP kernel. In: Proceedings of TRICS: Techniques for Implementing Constraint programming Systems, a post-conference workshop of CP. pp. 71–85 (2000)
13. Meng, S., Arbab, F.: Web services choreography and orchestration in reo and constraint automata. In: Proceedings of the 2007 ACM Symposium on Applied Computing (SAC). pp. 346–353. ACM (2007)
14. OWL-S: Semantic Markup for Web Services (2004), http://www.w3.org/Submission/OWL-S/
15. Pan, J.Z., Stamou, G., Stoilos, G., Taylor, S., Thomas, E.: Scalable querying services over fuzzy ontologies. In: Proceedings of World Wide Web. pp. 575–584. WWW '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1367497.1367575
16. Shen, Z., Su, J.: Web service discovery based on behavior signatures. In: Proceedings of the 2005 IEEE International Conference on Services Computing - Volume 01. pp. 279–286. SCC '05, IEEE Computer Society, Washington, DC, USA (2005)
17. Toch, E., Gal, A., Reinhartz-Berger, I., Dori, D.: A semantic approach to approximate service retrieval. ACM Trans. Internet Technol. 8(1) (Nov 2007)
18. Wang, Y., Stroulia, E.: Semantic structure matching for assessing web-service similarity. Service-Oriented Computing-ICSOC 2003 pp. 194–207 (2003)
19. Zemni, M.A., Benbernou, S., Carro, M.: A soft constraint-based approach to QoS-aware service selection. In: Service-Oriented Computing - 8th International Conference, ICSOC 2010. LNCS, vol. 6470, pp. 596–602 (2010)