A Preference Meta-Model for Logic Programs with Possibilistic Ordered Disjunction

Roberto Confalonieri, Juan Carlos Nieves, and Javier Vázquez-Salceda

Universitat Politècnica de Catalunya
Dept. Llenguatges i Sistemes Informàtics
C/ Jordi Girona Salgado 1-3
E - 08034 Barcelona
{confalonieri,jcnieves,jvazquez}@lsi.upc.edu

Abstract This paper presents an approach for specifying user preferences related to services by means of a preference meta-model, which is mapped to logic programs with possibilistic ordered disjunction following a Model-Driven Methodology (MDM). MDM allows to specify problem domains by means of meta-models which can be converted to instance models or other meta-models through transformation functions. In particular we propose a preference meta-model that defines an abstract preference specification language allowing users to specify preferences in a more friendly way using models. We also present a meta-model for logic programs with possibilistic order disjunction. Then we show how we conceptually map the preference meta-model to logic programs with possibilistic ordered disjunction by means of a mapping function.

1 Introduction

Recently with the adoption of both Service-Oriented Architectures (SOA) and Web services as growing trend for building distributed applications, services can be world-widely advertised and accessed. In this context, service discovery and selection play an important role *w.r.t.* the search and selection of the most suitable services users are looking for. With the rapidly growing number of services that are becoming available, users will be offered in fact with a choice of functionally similar services, which increase the need of enhancing traditional discovery and selection processes with the possibility for the users to express preferences about and relevant to certain services.

On the other hand, expressing and reasoning about user preferences is a complex and challenging task, as preferences cannot be generally explicitly expressed because of the large number of possible alternatives. Nonmonotonic logics have shown to be a potent knowledge representation formalism to reason about preferences [4]. Several extensions of the basic formalism of Answer Set Programming (ASP) have been proposed to model preferences [6], showing how nonmonotonic logics constitute an effective way of resolving indeterminate solutions, reasoning in terms of preferred answer sets of a logic program. Unfortunately, nonmonotonic logics by themselves are not flexible enough and not well designed for modeling orderings on belief sets specified in terms of preferences on their elements [4].

Logic programs with ordered disjunction (or LPODs) offer one way to overcome this problem as they permit to explicitly represent preference information directly into head rules [2]. In this way, the language can capture user qualitative preferences by means of disjunction rules, represent choices among different alternatives and specify a preference order between the answer sets through some comparison criteria. However, in some scenarios the preference information can be subject to uncertainty and preference-aware reasoning methods that can handle uncertainty are needed [5]. For this reason in [5] the authors have proposed an extension of the semantics of logic programs with ordered disjunction in order to cope with the degree of uncertainty in the reasoning process. In particular, they have defined a possibilistic semantics for capturing *logic programs with possibilistic ordered disjunction* (or LPPODs) which is close to the proof theory of possibilistic logic and answer set semantics.

ASP has become quite popular in knowledge representation problems as it is based on solid theoretical foundation, it is expressively rich, and its semantics and computational properties well understood today. Moreover many efficient ASP solvers such as *dlv* [8] and *smodels* [14] are available. As such ASP has called the attention from the industry, and points out to be a promising knowledge representation method in many application areas. Nevertheless, ASP has not been applied to Service-Oriented Architectures yet. One of the main obstacles towards the adoption of nonmonotonic preference representation methods is determined by the lack of an ASP programming environment. Writing correct and efficient ASP programs is in fact a difficult task and users are required to have the sufficient expertise to encode real problems in ASP. For this reason one of the open issues of ASP is the development of ASP programming environments and friendly interfaces [10].

In this paper we propose an approach for specifying user preferences related to services by means of a preference meta-model which we map to LPPODs following a Model-Driven Methodology (MDM). MDM allows to specify problem domains by means of meta-models which can be converted to instance models or other meta-models through transformation functions. We propose a preference meta-model that defines an abstract preference specification language allowing the users to specify preferences about services in a more friendly way using models. We also present a meta-model for logic programs with possibilistic order disjunction showing how we conceptually map the preference meta-model to the model of LPPODs by means of a mapping function.

The paper is organised as follows. In Section 2 the syntax and semantics of logic programs with possibilistic ordered disjunction are presented and the main characteristics of the MDM approach are described. In Section 3 we propose a model-driven framework for capturing preferences in SOA and a preference meta-model is described. In Section 4 we present the meta-model for logic programs with possibilistic ordered disjunction and a conceptual mapping. In Section 5 we describe how MDM can be applied in the modeling of a simple user preference scenario. Finally in Section 6 we discuss our approach, draw some conclusions and outline future work.

2 Background

In this section we introduce the reader with some basic concepts *w.r.t.* the syntax and semantics of logic programs with possibilistic ordered disjunction and Model-Driven Methodology.

2.1 Logic Programs with Possibilistic Ordered Disjunction

Logic programs with possibilistic ordered disjunction (LPPODs) are logic programs with ordered disjunction with possibilistic values added to each rule [5]. The syntax of a logic program with possibilistic ordered disjunction is based on the syntax of ordered disjunction rules [2] and of possibilistic logic [7].

LPPODs Syntax: A signature $\mathcal{L}$ is a finite set of elements called atoms. Atoms negated by $\neg$ will be called extended atoms. The concept of atom will be used without paying attention if it is an extended atom or not. A *possibilistic atom* is a pair $p = (a, q) \in \mathcal{A} \times \mathcal{Q}$ where $\mathcal{A}$ is a set of atoms and ($\mathcal{Q}$,$\leq$) a finite lattice[1]. The projection $*$ to any possibilistic atom p is defined as follows: $p^* = a$. Given a set of possibilistic atoms M, the generalization of $*$ over M is defined as: $M^* = \{p^* \mid p \in M\}$. Given a lattice ($\mathcal{Q}$,$\leq$), a possibilistic ordered disjunction rule r is of the form:

$$\alpha : c_1 \times \ldots \times c_n \leftarrow b_1, \ldots, b_m,\ \mathit{not}\ b_{m+1} \ldots,\ \mathit{not}\ b_{m+k}$$

where $\alpha \in \mathcal{Q}$ and $c_i(1 \leq i \leq n)$, $b_j(1 \leq j \leq m+k)$ are atoms. Sometimes a possibilistic ordered disjunction clause is denoted as: $\alpha : c_1 \times \ldots \times c_n \leftarrow \mathcal{B}^+,\ \mathit{not}\ \mathcal{B}^-$ where $\mathcal{B}^+ = \{b_1, \ldots, b_m\}$ and $\mathcal{B}^- = \{b_{m+1}, \ldots, b_{m+k}\}$. The projection $*$ for a possibilistic ordered disjunction rule r, is $r^* = c_1 \times \ldots \times c_n \leftarrow \mathcal{B}^+,\ \mathit{not}\ \mathcal{B}^-$. It can be observed that the ordered disjunction clause r^* is an ordered disjunction clause as was defined in [2]. $n(r) = \alpha$ is a necessity degree representing the certainty level of the information described by r. A possibilistic constraint C is of the form $\mathcal{TOP}_{\mathcal{Q}} :\leftarrow \mathcal{B}^+,\ \mathit{not}\ \mathcal{B}^-$, where $\mathcal{TOP}_{\mathcal{Q}}$ is the top of the lattice ($\mathcal{Q}$,$\leq$) and $\leftarrow \mathcal{B}^+,\ \mathit{not}\ \mathcal{B}^-$ is a constraint as in standard ASP [1]. Please notice that any possibilistic constraint must have the top of the lattice ($\mathcal{Q}$,$\leq$). This restriction is motivated by the fact that, like constraints in standard Answer Set Programming, the purpose of the possibilistic constraint is to eliminate possibilistic models. Hence, it is assumed that there is no uncertainty about the information captured by a possibilistic constraint. As in possibilistic ordered disjunction rules, the projection $*$ for a possibilistic constraint C is $C^* = \leftarrow \mathcal{B}^+,\ \mathit{not}\ \mathcal{B}^-$.

A *logic program with possibilistic ordered disjunction* (LPPOD) is a tuple of the form $P := \langle (Q, \leq), N \rangle$ such that N is a finite set of possibilistic ordered

[1] Only finite lattices are considered.

disjunction rules and possibilistic constraints. The generalization of $*$ over P is defined as follows: $P^* := \{r^* \mid r \in N\}$. Notice that P^* is an ordered disjunction logic program.

LPPODs Semantics: Before defining the possibilistic semantics for capturing LPPODs, basic operations between sets of possibilistic atoms and a relation of order between them are introduced.

Definition 1. *Given $\mathcal{A}$ a finite set of atoms and $(\mathcal{Q},\leq)$ a lattice, $\mathcal{PS} = 2^{\mathcal{A}\times\mathcal{Q}}$ is considered as the finite set of all the possibilistic atom sets induced by $\mathcal{A}$ and $\mathcal{Q}$. Let $A, B \in \mathcal{PS}$, the operators $\sqcap$, $\sqcup$ and $\sqsubseteq$ can be defined as follows:*

$$\begin{aligned}
A \sqcap B &= \{(x, GLB\{q_1, q_2\})|(x, q_1) \in A \ \wedge (x, q_2) \in B\} \\
A \sqcup B &= \{(x, q)|(x, q) \in A \ and \ x \notin B^*\} \cup \{(x, q)|x \notin A^* \ and \ (x, q) \in B\} \cup \\
&\quad \{(x, LUB\{q_1, q_2\})|(x, q_1) \in A \ and \ (x, q_2) \in B\}. \\
A \sqsubseteq B &\Longleftrightarrow A^* \subseteq B^*, \ and \ \forall x, q_1, q_2, (x, q_1) \in A \wedge (x, q_2) \in B \ then \ q_1 \leq q_2.
\end{aligned}$$

The semantics of LPPODs is close to the proof theory of possibilistic logic and answer set semantics. As in answer set semantics definition, the possibilistic semantics is defined based on a syntactic reduction.

Definition 2 (Reduction $r^M_\times$). *Let $r = \alpha : c_1 \times \ldots \times c_n \leftarrow \mathcal{B}^+, \ not \ \mathcal{B}^-$ be a possibilistic ordered disjunction clause and M be a set of atoms. The $\times$-possibilistic reduct $r^M_\times$ is defined as follows:*

$$r^M_\times := \{\alpha : c_i \leftarrow \mathcal{B}^+ | c_i \in M \ and \ M \cap (\{c_1, \ldots, c_{i-1}\} \cup \mathcal{B}^-) = \emptyset\}$$

Definition 3 (Reduction $P^M_\times$). *Let $P = \langle (Q, \leq), N\rangle$ be a LPPOD and M be a set of atoms. The $\times$-possibilistic reduct $P^M_\times$ is defined as follows:*

$$P^M_\times = \bigcup_{r \in N} r^M_\times$$

Observe that the program $P^M_\times$ is a possibilistic positive extended logic program.[2] Once a LPPOD P has been reduced by a set of possibilistic atoms M, it is possible to test whether M is a possibilistic answer set of the program P by considering the following definition.[3]

Definition 4 (Possibilistic answer set). *Let $P = \langle (Q, \leq), N\rangle$ be a LPPOD and M be a set of possibilistic atoms such that M^* is an answer set of $(P^{M^*}_\times)^*$. M is a possibilistic answer set of P if and only if $P^{M^*}_\times \vdash_{PL} M$ and $\nexists M' \in \mathcal{PS}$ such that $M' \neq M$, $P^{(M')^*}_\times \vdash_{PL} M'$ and $M \sqsubseteq M'$.*

By the original (no possibilistic) ordered disjunction rule definition, it is possible to represent preferences among possibilistic answer sets by considering degrees of satisfaction denoted as $deg_M(r)$ and defined by the following definition.

[2] A positive program is a program without negation as failure atoms.

[3] $\vdash_{PL}$ denotes the inference under possibilistic logic.

Definition 5 (Rule Satisfaction Degree). *Let M be a possibilistic answer set of a LPPOD P. Then M satisfies the rule r*

$$\alpha : c_1 \times \ldots \times c_n \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1} \ldots,\ not\ b_{m+k}$$

- *to degree 1 if $b_j \notin M^*$ for some j $(1 \leq j \leq m)$, or $b_i \in M^*$ for some i $(m+1 \leq i \leq m+k)$,*
- *to degree j $(1 \leq j \leq n)$ if all $b_l \in M^*$ $(1 \leq l \leq m)$, $b_i \notin M^*$ $(m+1 \leq i \leq m+k)$, and $j = min\{r \mid c_r \in M^*, 1 \leq r \leq n\}$.*

To distinguish between preferred possibilistic answer sets, the satisfaction degree of a possibilistic answer set M *w.r.t.* a rule, denoted by $deg_M(r)$, provides a ranking of the possibilistic answer sets of a LPPOD, and a preference order on the possibilistic answer sets can be obtained by means of a comparison criteria. In [5] the authors have proposed three criteria for comparing possibilistic answer sets, respectively *possibilistic cardinality*, *possibilistic inclusion* and *possibilistic Pareto*, which are the possibilistic version of the original criteria of [2].
The set of possibilistic atoms M satisfying a degree i is defined as follows:

Definition 6. *Let M be a set of possibilistic atoms and P be a LPPOD. Then $M^{i,\alpha}(P) = \{r \in P \mid deg_M(r) = i \text{ and } n(r) \geq \alpha\}$.*

Given a set of possibilistic atoms M, $n(M)$ is defined as $min\{\alpha \mid (a, \alpha) \in M\}$. Three preference relations can be defined. The possibilistic version of cardinality-based preference can be defined as follows:

Definition 7. *Let M_1 and M_2 be possibilistic answer sets of a LPPOD P. M_1 is possibilistic cardinality-preferred to M_2, $(M_1 >_{pc} M_2)$ iff $\exists$ i such that $\mid M_1^{i,\alpha}(P) \mid > \mid M_2^{i,\alpha}(P) \mid$ and $\forall j < i$, $\mid M_1^{j,\alpha}(P) \mid = \mid M_2^{j,\alpha}(P) \mid$, where $\alpha = min\{n(M_1), n(M_2)\}$.*

The inclusion-based preference is defined as:

Definition 8. *Let M_1 and M_2 be possibilistic answer sets of a LPPOD P. M_1 is possibilistic inclusion-preferred to M_2, $(M_1 >_{pi} M_2)$ iff $\exists$ k such that $M_2^{k,\alpha}(P) \subset M_1^{k,\alpha}(P)$ and $\forall$ $j < k$, $M_1^{j,\alpha}(P) = M_2^{j,\alpha}(P)$, where $\alpha = min\{n(M_1), n(M_2)\}$.*

Lastly, the possibilistic Pareto-based preference is:

Definition 9. *Let M_1 and M_2 be possibilistic answer sets of a LPPOD P. M_1 is possibilistic pareto-preferred to M_2, $(M_1 >_{pp} M_2)$ iff $\exists$ $r \in P$ such that $deg_{M_1}(r) < deg_{M_2}(r)$, and $\nexists r' \in P$ such that $deg_{M_1}(r') > deg_{M_2}(r')$, and $n(r) \geq min\{n(M_1), n(M_2)\}$.*

One interesting characteristic of LPPODs is that they provide a mean to represent preferences among problem solutions and allow to represent preferences which can depend on incomplete knowledge. As LPPODs are based on extended nonmonotonic logic, incomplete information can be expressed by means of default negation.

2.2 Model-Driven Methodology

Model-Driven Engineering (MDE) refers to the systematic use of models as primary artefacts throughout the Software Engineering (SE) development process. The defining characteristic of MDE is the use of models to represent the important artefacts in a system [13]. Each of the models in a MDE system is constructed from a language specified in a meta-model, which captures the concepts and relationships of the language in a structured and regular form. In relation to these meta-models, the models can then be stored, manipulated and transformed to other models, and to implementation artefacts.

A Model-Driven Methodology (MDM) to development is generally based on the Model Driven Architecture (MDA) [19], an initiative by the Object Management Group (OMG)[4] which specifies a framework of open standards and related technologies. The framework is built upon the metamodel foundation in order to enable a standard specification and interoperability mechanism for tools. So systems and applications are formalized with metamodel descriptions and are visualized by models as metamodel instantiations. Actual code implementations are created automatically by applying predefined transformations from source models to target models and implementation languages.

In the context of this paper, MDD specifies the user preference meta-model (Section 3.3) upon which a *user preference editor* can be created, allowing the modeling and instantiation of corresponding models. By the specification of a meta-model for logic programs with possibilistic ordered disjunction (Section 4.1) and a transformations function it is possible to (semi-)automatically translate the user preference abstract representation to the formalism of logic programs with possibilistic ordered disjunction (Section 4.2).

3 Preferences in Service-Oriented Architectures

The use of nonmonotonic reasoning about preferences in Service-Oriented Architectures is a new and unexplored field. We believe that the applicability of preference and reasoning methods to the services' domain can enhance the service discovery and selection processes and assist the user in services' searches. In order to reuse existing works about service oriented technologies and nonmonotonic preference handling methods for preference representation and reasoning we propose a model-driven framework that attempts to glue these approaches together.

3.1 Model-Driven Framework

Figure 1 shows the model-driven framework we are proposing. The diagram depicts the main components of the framework, and shows the relations between user preferences and services. The meta-model provides a domain independent and technology independent representation of user preferences about services.

[4] `http://www.omg.org/`

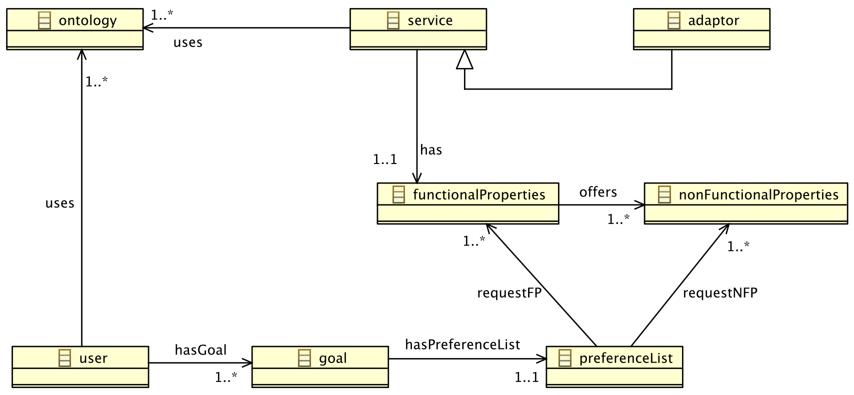

Figure 1. Model-Driven Framework for User Preferences in SOA

The framework is inspired by the Web Service Modeling Framework [9] which has been recently adopted in WSMO [15], although we introduce new concepts covering the relation between user preferences and services. The main component of the framework are: *goals user preferences*, *services*, *ontologies*, and *adaptors*:

– **Services** offer specific functionalities to users, and are described by *functional* and *non-functional properties*. Functional descriptions of services consist in service *input* and *output* and *pre* and *post-conditions*. Non-functional properties are usually related to service usage or domain dependent properties. Services are semantically described through an *ontology*.
– **Goals and User Preferences**: a *goal* is the user-centered view of a service usage. Normally users have specific tasks they want to accomplish and goals are specifications of a desired state of outcomes. Goals are accompanied by a *preference list* which represents the user preferences for the service. Such preferences *request functional* and *non-functional properties*. For instance let us imagine a user is interested in getting a map of restaurants, and she has a set of preferences about the type of restaurants, the map and the cost. Her goal could be to get a service that takes as input an object of type Restaurant and as output type Map, while preferences could be *"I prefer a high resolution map than a low resolution"* or *"I prefer to spend 1 euro than 2 euros, if the service response is fast"*. A goal usually consists of a functional description of the objectives users want to achieve using a Web service. In this sense a user goal is a *hard-constraint w.r.t.* a service functionality. User preferences are requirements the user wants over achieving a goal. They may include Quality of Services (QoS) metrics or domain dependent properties. From this point of view preferences are *soft-constraints w.r.t.* the properties ot the service that fulfills a certain goal.

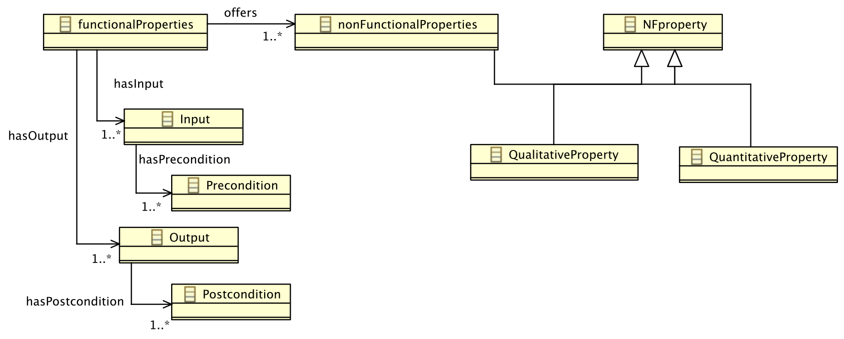

Figure 2. Service Properties Meta-model

- **Service Adaptors** address the handling of heterogeneities occurring between elements that shall interoperate by resolving mismatches between different used terminologies (data level), on communicative behavior between services (protocol level), and on the business process level.
- **Ontologies** provide the formal semantics for the terminology used within all other framework components. We expect to have a framework ontology which will consist in a *service ontology* and *domain ontologies* specific for domain applications.

In order to express preferences about and relevant to services, we need to consider the properties *requested* by the user and the properties *offered* by the service.

3.2 Service Properties Meta-Model

From a provider perspective a service can expose different offered properties associated with the same functionality to address different business requirements (*e.g.* speedy and slow service at different price). We assume that functional descriptions are provided in terms of *input* and *output*. *Pre-* and *post-condition* are constraints about input and output respectively. Offered properties consist in a set of *non-functional properties* which can be *qualitative* and *quantitative* whose values are defined in a *domain ontology* (Figure 2). Non-functional properties can be seen as a kind of service configuration which the provider *offers* for the service usage.

3.3 User Preference Meta-Model

Figure 3 shows the user preference meta-model. From a user perspective, required properties can be considered as a set of constraints on the requested services. To

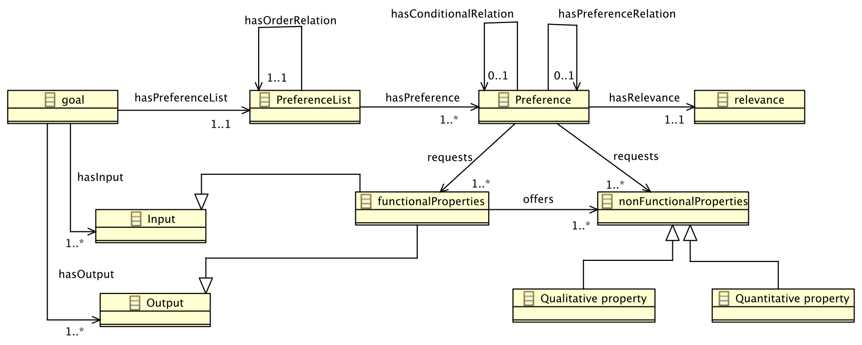

Figure 3. User Preference Meta-model

collect sets of preferences we consider a *preference list* to associate a goal with a set of *preferences*. A *goal* is described by *input* and *output* which specify strong constraints *w.r.t.* a service that fullfills that goal. A preference list is an ordered list of preferences specified by the user, where such order represents a *preference order* between sets of desired properties. Desired properties can be viewed along two dimensions: *functional* and *non-functional*. Functional properties are preferences about input and output of the goal, while non-functional preferences are related to service usage.

To manage qualitative and quantitative properties (*e.g.* Restaurant and Cost), two classes of properties are introduced, *quantitative* and *qualitative properties* respectively. All the properties' values are concepts of the *domain ontology* (*e.g.* Restaurant, Map) or data types (String, Integer *etc*). Preferences may be associated with a degree of *relevance w.r.t.* the preference rules or non-functional properties of a service. A *Preference* has a *preference relation* to be able to specify a preference order (*e.g. "I prefer a map with higher resolution to a lower one"*), and a conditional preference relation to be able to capture *conditional preferences* between properties *e.g.* (*"if the cost of the service is not high I prefer a high map resolution"*).

4 Model Transformations

As we do not want to stick to a particular formalism, language or technology during the solution specification, we have defined a meta-model describing how user preferences and service properties should be specified in a general way. The advantage to have a meta-model is to have a specification general enough which can be then translated to other meta-models and models (representing target languages or formalism) through transformations [18].

Generally speaking, a model transformation takes as input a model conforming to a given meta-model and produces as output another model conforming to

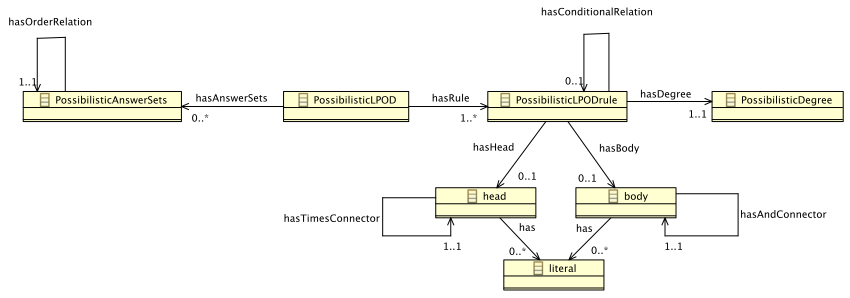

Figure 4. Meta-Model for Logic Programs with Possibilistic Ordered Disjunction

a given meta-model. The process of translating a model to another is specified by a mapping algorithm. The main advantage of this approach is that, from the same meta-model specification, several mappings to different models can be done through different mapping functions [12].

We define a transformation t of a preference model M_1 according to the preference meta-model *P_MM* to a LPPOD model M_2 according to the LPPODs meta-model *LPPOD_MM* as

$$t : M_{1_{P\_MM}} \rightarrow M_{2_{LPPOD\_MM}}$$

To be able to (informally) define a the transformation function t we first need a meta-model which describes the formalism of LPPODs.

4.1 Meta-Model for LPPODs

A proposal of a meta-model for this class of logic programs is shown in Figure 4. A *possibilistic LPOD program* consists of a finite set of *possibilistic LPOD rules* associated with a *possibilistic degree*. Each rule *has* a *head* and a *body*, where in the body a preference relation is specified by means of the logical connector $\times$ (times). Outcomes of LPPOD are *possibilistic answer sets* and a *relation order* between them can be obtained applying the comparison criteria defined in Section 2.1.

4.2 A Mapping to LPPODs

According to MDD the preference meta-model *P_MM* and LPPOD meta-model *LPPOD_MM* dependencies are formulated with model driven mappings (relations). The mappings are specified with a transformation language, among the corresponding elements of the *P_MM* and *LPPOD_MM* meta-models shown in Figure 3 and Figure 4 respectively. As illustrative example the mappings `preferenceListToLPPOD` and `preferenceToPossibilisticLPODRule` have been specified.

The transformation process can be initiated from the `preferenceListToLPPOD` mapping that converts user preferences in a LPPOD. The rule in turn applies a mapping between preference and preference rules of a LPPOD.

```
mapping preferenceListToLPPOD(in pl:P_MM::PreferenceList,
  inout lppod:LPPOD_MM::PossibilisticLPOD) {
 var preferenceRules := pl.preference -> collect(p|map
  preferenceToPossibilisticLPODRule(p,pl,preferenceRules));
 lppod.possibilisticLPODrule := preferenceRules;
}
mapping preferenceToPossibilisticLPODRule(in p:P_MM::Preference,
  in: pl:P_MM::PreferenceList,
  inout: pr:LPPOD_MM::PossibilisticLPODrule) {
 pr.possibilisticDegree := p.relevance;
 pr.head := p.functionalProperties -> collect(fp|map
  functionalPropertiesToHead(fp,p.preferenceOrder));
 pr.body := p.NFproperties -> collect(nfp|map
  nfPropertiesToBody(nfp));
}
mapping functionalPropertiesToHead(in fp:P_MM::functionalProperty,
  in po:P_MM:PreferenceRelation) {
 ...
}
mapping nfPropertiesToBody(in nfp:P_MM::NFPropertyPreference) {
 ....
}
```

5 Applying Model-Driven Methodology to User Preference Modeling

The preference meta-model presented in Section 3.3 can be instantiated to different domain problems and used for expressing user preferences about services in an abstract way. In particular it can be generated by means of a *preference editor*. The meta-model in fact can be specified by the Eclipse Ecore specification. Once the meta-model is available, a preference editor can be implemented using the EMF tools of the Eclipse Platform.[5] The model generated by the editor can be mapped to an instantiated model of a LPPOD by means of the transformation function t (Figure 5).

For example let us consider a user looking for a recommendation service, that takes as input *restaurants* and returns a *map*. She can have a goal where Restaurant is the Input and Map is the Output. She can have preferred values about the input and output of the goal, be undecided about the type of restaurants, and be looking for specific non-functional properties such as the cost of the service

[5] `http://www.eclipse.org/modeling/emf`

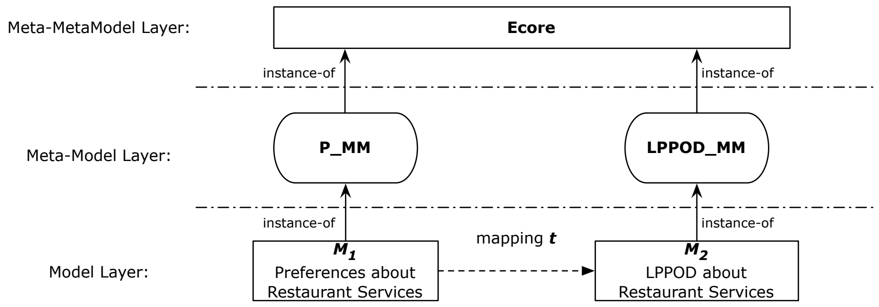

Figure 5. MDD applied to User Preference Modeling

and the map resolution. The request of a personalized service according to her preferences can be expressed as a list of preference such as:

– she prefers Italian to Mexican restaurants
– she prefers a higher map resolution to a lower one if the cost of the service is not $>$ 2 euros (high) and the time of response is not $\geq$ 0.5 sec (slow).
– she prefers a lower map resolution if the service cost is $>$ 2 euros (high) and time of response is $\geq$ 0.5 sec (slow).

Let us imagine we have the model M_1 for the scenario described above. By the transformation function t we can then generate the model of a LPPOD M_2. A further step (not shown here) is represented by a code generation function that converts M_2 to the syntax of LPPODs. The code generation results in the LPPOD code shown in the following example.

Example 1. Let $P = \langle(Q, \leq), N\rangle$ be a LPPOD expressing the user preferences about restaurant maps. First of all we define the lattice ($\mathcal{Q}$,$\leq$) to specify an ordered set of relevance degrees. We consider the lattice $\mathcal{Q} = (\{0, 0.1, 0.2, \ldots, 1\}, \leq)$, and $\leq$ the standard relation between rational numbers. Let N be the set of possibilistic ordered disjunction rules expressing the user preferences about restaurant maps generated by the transformation function t. One possible way to encode the user preferences in the syntax of a LPPOD is:[6]

$r_1 = \mathbf{1} : rest(italian) \times rest(mexican).$
$r_2 = \mathbf{1} : map(high) \times map(low) \leftarrow not\ cost(\_, high), not\ response(\_, slow).$
$r_3 = \mathbf{1} : map(low) \times map(high) \leftarrow cost(\_, high).$
$r_4 = \mathbf{1} : map(low) \times map(high) \leftarrow response(\_, slow).$
$r_5 = \mathbf{1} :\ \leftarrow map(low), map(high).$

Please notice that whenever the α values are equal to 1, the LPPOD behaves like an ordered disjunction program where the possibilistic values of the atoms

[6] The predicates cost(_,high) and response(_,slow) are qualitative predicates for the quantitative value *w.r.t.* the cost and the response time. Such predicates can be built by specific rules in the logic program.

in the answer sets are the top of the lattice. Therefore we can see that there are four possibilistic answer sets satisfying the program,

$$\begin{aligned}M_1 &= \{(rest(italian), 1), (map(high), 1)\}\\ M_2 &= \{(rest(italian), 1), (map(low), 1)\}\\ M_3 &= \{(rest(mexican), 1), (map(high), 1)\}\\ M_4 &= \{(rest(mexican), 1), (map(low), 1)\}\end{aligned}$$

and according to their satisfaction degrees they have the following preference order: $M_1 >_{pp} M_2$, $M_1 >_{pp} M_3$, M_2 and M_3 are equally preferred and $M_3 >_{pp} M_4$. Such a order can be exploited to select the best service that accomplish the user goal. Each of the possibilistic answer set in fact can be used to build a service search where the atoms are used as input parameters for a matchmaker.[7] The matchmaking process returns a list of candidate services *w.r.t.* the possibilistic answer sets used. Non-functional service properties can be converted as a set of possibilistic facts $\mathcal{PF}_i$, where the necessity-value degrees corrispond to normalized non-functional properties values (*e.g.* $0.5 : cost(s_1, high)$) and added to a new LPPOD P_i, such that $P_i = P \cup \mathcal{PF}_i \cup C_i$.[8] The possibilistic answer sets of the new generated LPPODs P_i (one for each original M_i) are candidate service solutions where the possibilistic values drawn the selection of the best service *w.r.t.* the user preferences.

6 Conclusions

In this paper we have presented an approach for assisting the user in expressing preferences about services' searches following the Model-Driven Methodology. We have defined a preference meta-model which allows to represent user preferences about services without being bound to a specific implementation technology. We have conceptually shown how it is possibile to translate preference models to the model of LPPODs. The advantage of having a model for LPPODs is that LPPODs' code can be (semi-)automatically generated by means of a transformation function t.

The proposed preference meta-model provides in fact a flexible way to capture user preferences and represents the basic artefact through which different problem domains can be instantiated. The preference meta-model representation can ease the development of a preference editor which allows users to express preferences relevant to services in a friendly way. This abstract representation can be mapped to LPPODs which, based on ordered disjunction programs, are a flexible way to represent and reason about user preferences.

Concerning related works on user preference representation about services, in most of the existing approaches preferences have been studied in the context of

[7] The matchmaker is a component which is able to perform a syntactic or semantic matching of a user goal against service descriptions.

[8] C_i is a possibilistic constraint that forces the solutions of each P_i to be relevant only *w.r.t.* M_i.

Web services composition [11,17,16]. For instance, in [16] user preferences specified using Conditional Preference Networks (CP-Nets) are used to improve the quality of generated compositions. In [11,17] an augmented version of the logic programming language Golog is used to specify and to integrate user preferences into Web service composition. Although the use of user preferences related to services is not new, our proposed work differs with the cited works on at least two aspects: a) we use a preference-aware and uncertainty-aware preference representation language as part of the user preference selection process; b) we incorporate the use of this language as part of a MDD methodology according to which implementation details will be trasparent to the user. The meta-models we are proposing in fact can improve the time of development of LPPODs and the preference meta-model represents the first step towards a preference editor implementation. Our approach can be generalised to be used in other ASP-based formalisms in order to ease the knowledge modeling and the reuse of existing knowledge. Moreover relations between the original formalism of logic programs with ordered disjunction (LPODs) and other preference handling methods such as CP-Nets have been already explored in [3] showing that a further mapping is feasible.

Interesting issues for future work are the refinement of the preference and LPPODs meta-models and the specification of the transformation function t in a more formal way using the ATL transformation language.[9] Currently we are implementing our approach using the Eclipse Platform and the tools of the Ecore framework. As soon as we have the Ecore meta-models we can start the implementation of the preference editor to model practical domain problems. Although not related to the paper itself, it is worthy to mention that we are also studying the implementation of the solver for LPPODs.

Acknowledgements This work has been funded mainly by the European Commission Framework 7 funded project ALIVE (FP7-215890). Javier Vázquez-Salceda's work has been also partially funded by the Ramón y Cajal program of the Spanish Ministry of Education and Science. The opinions of the authors do not reflect the opinions of the European Commission. We thank anonymous referees for the valuable comments.

References

1. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press, 2003.
2. G. Brewka, I. Niemelä, and T. Syrjänen. Logic Programs with Ordered Disjunction. *Computational Intelligence*, 20(2):333–357, 2004.
3. G. Brewka, I. Niemelä, and M. Truszczyński. Answer Set Optimization. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 867–872. Morgan Kaufmann Publishers, 2003.

[9] `http://www.eclipse.org/m2m/atl/`

4. G. Brewka, I. Niemelä, and M. Truszczyński. Preferences and Nonmonotonic Reasoning. *AI Magazine*, 29(4):69–78, 2008.
5. R. Confalonieri, J. C. Nieves, and J. Vázquez-Salceda. Logic Programs with Possibilistic Ordered Disjunction. Technical Report LSI-09-19-R, Universitat Politècnica de Catalunya - LSI, 2009.
6. J. Delgrande, T. Schaub, H. Tompits, and K. Wang. A classification and Survey of Preference Handling Approaches in Nonmonotonic Reasoning. *Computational Intelligence*, 20(2):308–334, 2004.
7. D. Dubois, J. Lang, and H. Prade. Possibilistic Logic. In D. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*, pages 439–513. Oxford University Press, Oxford, 1994.
8. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR System dlv: Progress Report, Comparisons and Benchmarks. In L. S. A.G. Cohn and S. Shapiro, editors, *Proceedings Sixth International Conference on Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann, 1998.
9. D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.
10. N. Leone. Logic Programming and Nonmonotonic Reasoning: From Theory to Systems and Applications. In *Proceedings of 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, page 1, 2007.
11. S. Mcilraith and T. C. Son. Adapting Golog for Programming the Semantic Web. In *In Fifth International Symposium on Logical Formalizations of Commonsense Reasoning*, pages 195–202, 2001.
12. A. Metzger. *Model-Driven Software Development*, chapter A Systematic Look at Model Transformations, pages 19–33. Computer Science. Springer Berlin Heidelberg, 2005.
13. J. B. Nicolas, N. Farcet, J. M. Jézéquel, B. Langlois, and D. Pollet. Reflective Model Driven Engineering. In *Proceedings of the 6th International Conference on the Unified Modeling Language*, LNCS, pages 175–189. Springer, 2003.
14. I. Niemelä and P. Simons. Smodels - An Implementation of the Stable Model and Well-Founded Semantics for Normal LP. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR97)*, pages 421–430. Springer-Verlag, 1997.
15. D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1), 2005.
16. G. Santhanam, S. Basu, and V. Honavar. On Utilizing Qualitative Preferences in Web Service Composition: A CP-net Based Approach. In *IEEE Congress on Services - Part I*, pages 538–544, July 2008.
17. S. Sohrabi, N. Prokoshyna, and S. A. McIlraith. Web Service Composition Via Generic Procedures and Customizing User Preferences. In *International Semantic Web Conference*, volume 4273 of *LNCS*, pages 597–611. Springer, 2006.
18. A. U. Stephen Mellor, Kendall Scott and D. Weise. *MDA Distilled: Principles of Model-Driven Architecture*. Addison Wesley, 2004.
19. T. Weis, A. Ulbrich, and K. Geihs. Model Metamorphosis. *IEEE Software*, 20(5):46–51, 2003.