3
Preference Metamodel

Chapter 2 presented a user study in which a set of preference specifications in natural language was collected. These specifications allowed us to analyse patterns and common expressions that people typically adopt to express their preferences. In this chapter, we describe a preference metamodel developed based on our previous study. The goal is to provide constructions to model user preferences in a way independent from the target application area and to allow users to use a language as close as possible to natural language. Besides using our user study as a source to build our model, we also took into account existing preference representation models in computer science (which will be discussed in Chapter 4) and other areas, such as psychology and philosophy.

As preferences are expressed in terms of entities of specific application areas, we first introduce an ontology metamodel, before detailing the preference metamodel. The ontology metamodel defines how these entities are structured so that they can be referred to in preference specifications.

All diagrams presented in this chapter are modelled with Unified Modeling Language (UML). Entities of our metamodel are highlighted in **boldface**, and examples of preferences and other entities are in *italics*. UML was adopted to introduce our metamodel, because it is simple, widely known and used. However, as UML lack a formal semantics, we also present a specification of our preference metamodel using the Z notation (Wordsworth 1992) (Appendix B), which is a formal specification language used for describing and modelling software. This formal specification models constraints and other aspects of our metamodel that are described informally in this chapter and cannot be represented in UML.

3.1
Ontology Metamodel

Most of the existing work in the context of preferences and decision making defines a decision problem by considering a set of alternatives (or outcomes), which are structured in terms of features $X_1, ..., X_n$, each of which associated with a particular domain $Dom(X_i) = x_1^i, ..., x_{n_i}^i$. The set of all possible alternatives is therefore $Dom(X_1) \times ... \times Dom(X_n)$. The subset of possible alternatives is called

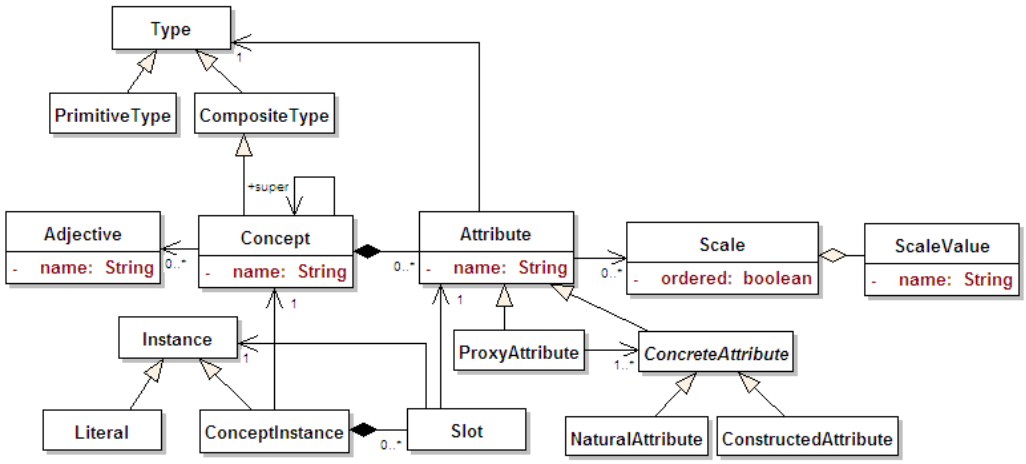


Figure 3.1: Ontology metamodel.

feasible outcomes.

Our experimental study has shown that users adopt a richer vocabulary than attributes and their (single) associated domain to refer to entities of the application area, thus not only making statements about the specific features of entities. Based on the analysis of the specifications collected in our study, we have defined an ontology metamodel, depicted in Figure 3.1, which structures entities of an application area. It is also built on the foundation of decision making research work (Keeney 1944).

Our model defines a main coarse-grained entity, namely concepts. A **concept** represents a class of elements that is composed of attributes, and is denoted by a name. As it is a **type** composed of finer-grained entities, it is a **composite type**. A concept typically represents a class of concrete entities of the world, such as a *laptop*.

Attributes, which are also denoted by a name, have a type, which defines the domain of values that can be assigned to a particular attribute. Based on our study, we identified three kinds of attributes: (i) objective attributes (e.g. *size of the RAM memory*); (ii) attributes that are built subjectively by individuals (e.g. *quality*); and (iii) attributes that represent a collection of other attributes, serving as a proxy to them (e.g. *laptop size*, for which the real attributes are the dimensions of the laptop). These three kinds of attributes are referred to as **natural attributes**, **constructed attributes** and **proxy attributes**, respectively. The first two kinds are collectively referred to as **concrete attributes**. We adopted these terms because they match similar concepts proposed by Keeney (Keeney 1944), used to measure the achievement of objectives.

The type of an attribute can be either composite, already introduced, or primitive. **Primitive types** are the basic building blocks to create composite types. These types, presented on the left hand side of Figure 3.2, can be a single character

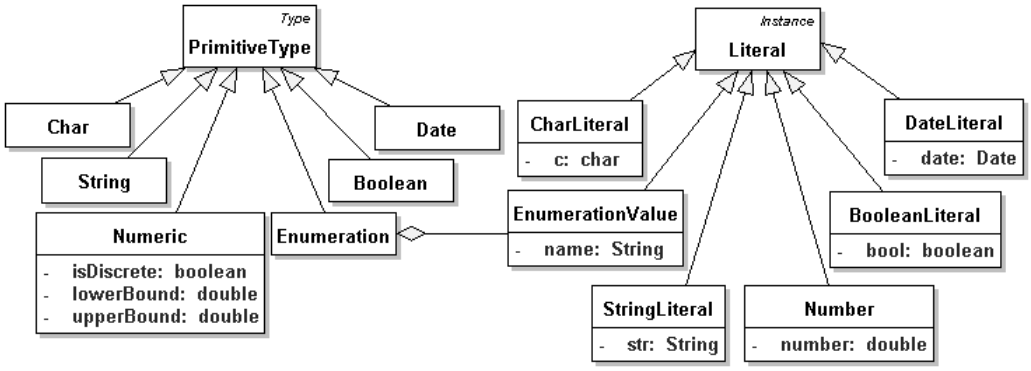


Figure 3.2: Ontology primitive types.

(**char**), a **string** (a sequence of characters), **numeric** (discrete or continuous, optionally with lower and upper bounds), **boolean** (true and false), a **date** and an **enumeration** (a set of elements, possibly having a natural order).

Even though instances of a concept always have a particular value assigned to each of its attributes, users typically use fuzzy domains to express preferences about attributes values, e.g. "*I want a big screen*." Therefore, we associate attributes not only with their type but also with a set of **scales**, which are composed of **scale values**. An example could be a scale *size*, whose values can be *tiny*, *small*, *normal*, *big* and *huge*.

Another common construction adopted by our study participants is adjectives. This is illustrated by the expression "*fast laptop*," which can be translated to a laptop whose processor speed is high and RAM memory is big. This construction is supported by associating a concept with a set of possible **adjectives**, which has a precise meaning defined in a specific ontology. Adjectives are not grouped to form a scale (as above), because adjectives can be translated to a set of statements in terms of attributes, as described in the example. The statements translated from adjectives are, in turn, situated in an ordered domain according to an attribute type, or in a scale.

Finally, types are generic representations for classes of particular **instances**. Instances of primitive types are **literals**, which are specific values assigned for **slots**, which in turn hold a value associated with an attribute. This value is one of the possible values of the primitive type associated with the attribute. For instance, if the attribute is associated with an enumeration, the literal assigned for the slot must be an enumeration value. The different literals are shown on the right hand side of Figure 3.2. **Concept instances**, on the other hand, are a composition of these slots, which are place holders for values (instances) assigned for each of the attributes that are associated with the concept of the instance.

3.2 Propositional Formulae

In different parts of preference specifications, users use propositional-formula-like constructions, for example, to express conditions (*if the laptop is a Mac*) or to restrict possible attribute values (*screen size between 14" and 15"*). So, we have developed a representation for propositional logic formulae, which are used for different purposes in our model, such as building constraints and conditions.

We captured these constructions in the model shown in Figure 3.3, which allows modelling formulae with three logic operators **not** (`NotFormula`), **and** (`AndFormula`), and **or** (`OrFormula`). We define a generic entity, namely `AtomicFormula`, to represent **atomic formulae**, and in particular we define three kinds of them, detailed below.

– **Attribute value specification**, which indicates a restriction over a domain of an attribute. For instance, "*Screen.size = 15 inches*". As an attribute is always part of a concept, we represent an attribute by <*concept*>.<*name*>, where *concept* is the concept that the attribute is part of, and *name* is its name. We represent concept names initiated with uppercase, and attribute names initiated with lowercase.

  An attribute is associated with a concept, and this concept can be the type of different attributes, which are part of different concepts. Consequently, preferences for an attribute value (of a concept) can be different, when this concept is related to different concepts, i.e. different contexts, e.g. preferences for colour depend on which concept colour is associated to: *I prefer white*

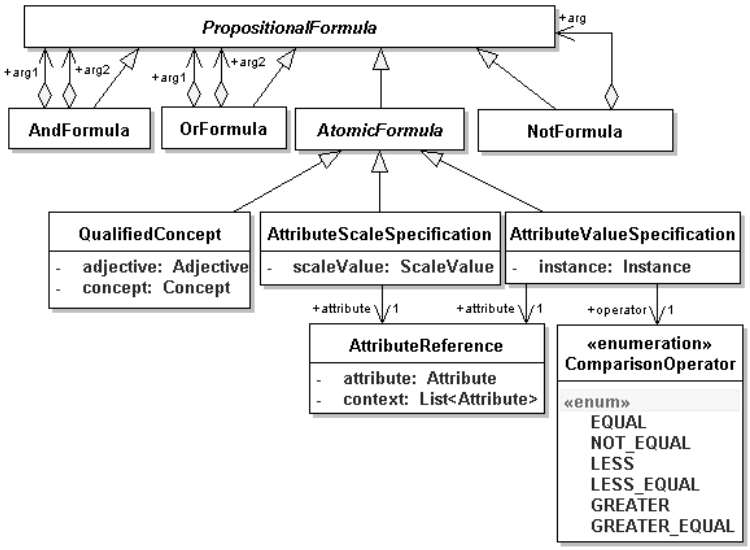


Figure 3.3: Propositional formula model.

houses and *I prefer red cars*. Therefore, this type of formula is represented as shown below:

<context><attribute><comparison operator><instance>.

An example is:

context: Trip.returnFlight, Flight.company
attribute: Company.group
comparison operator: = (equal)
instance: OneWorld

In this example, the context is composed of two attributes for defining the context of the attribute *Company.group*. Note that the type of a previous attribute in the context list must match the concept to which the following attribute is part of (the type of *returnFlight* is *Flight*). The same occurs with the last attribute of the context and the attribute. In Figure 3.3, context and attribute together are represented as an attribute reference.

– **Attribute scale specification**, which is represented as

<context><attribute><scale value>.

It also establishes a restriction over values of a particular attribute, but instead of specifying specific values, a scale value associated with the attribute is specified. Example:

context: Laptop.ramMemory
attribute: RAMMemory.size
scale value: Big

– **Qualified concept**, which qualifies a concept with an adjective, e.g. "*fast laptop*." In theory, the notion of context could also be adopted to associate the concept being qualified with a particular context, but this was not observed in our study of how humans express preferences, so we did not include it in our metamodel.

3.3 Preference Metamodel

In this section, we describe the preference metamodel, divided into multiple parts. We first present, in Section 3.3.1, an overview of all preference types and priorities of our metamodel, and how we model conditions and contexts associated with those preferences. Next, we describe preference constructions to express goals and constraints (Section 3.3.2). Constraints, in turn, are used to construct more sophisticated preferences, namely preference statements, which are presented in Section 3.3.3. Then, we detail how to express preferences over preferences, i.e. priorities (Section 3.3.4). Finally, Section 3.3.5 shows a set of examples of the different kinds of preferences presented.

3.3.1 Overview

Our metamodel is composed of different kinds of preferences and how to express priority among them. All of these are shown Figure 3.4, which overviews our metamodel, showing its main entities and how they are connected by generalisation or association. This figure also shows the two entities that are part of our preference metamodel, namely conditions and decision context, and indicate when preferences and priorities are applicable. These entities are represented in the same way, i.e. by a propositional logic formula, but they have two different meanings, as detailed next.

(i) **Condition.** Values assigned to certain attributes might impact preferences over other attributes, concepts, and so on. A condition specifies a set of values for attributes, to which a preference is subjected. Example: "*if Laptop.brand = Mac, <preference>*." Conditions can also be used for specifying priorities.

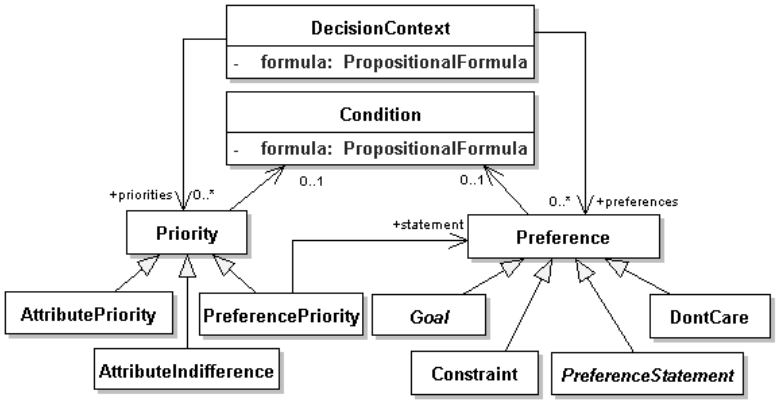
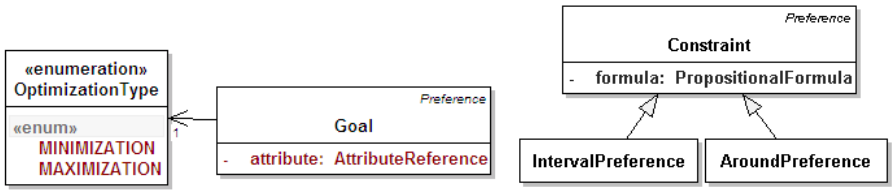


Figure 3.4: Overview of the preference metamodel.



3.5(a): Goal. 3.5(b): Constraint.

Figure 3.5: Goals and Constraints.

(ii) **Decision context.** Decision contexts model a state of world in which a particular set of preferences and priorities is relevant. Example: "*if the purpose of buying the laptop is for business work, <preference>*." Moreover, adjectives of the ontology as well as scales have a specific meaning for a particular decision context, i.e. "light laptop," for instance, can have different interpretations in personal or business contexts.

3.3.2 Simple Preferences

When users know the application area in which they are expressing preferences, they are aware of the possible values that concepts and attributes can have and they have preferences over these values, which impose restrictions on what these values should be. Moreover, users might not be interested in values or certain attributes. This kind of preferences are presented in this section, which we call **simple preferences**, as opposed to preferences presented in the next section, which involve expressing not only attributes and their values, but also a more sophisticated vocabulary, involving expressive speech acts and rates. Three kinds of preferences are introduced next: goals, constraints and don't care.

Goals (Figure 3.5(a)) represent an overall rule that indicates when an attribute value is preferred to another. Users state that they prefer to minimise or maximise (**optimisation type**) a particular attribute (**attribute goal**). An examples is: "*Minimise Laptop.price*."

A **constraint** (Figure 3.5(b)) represents a restriction over the values that can be selected for a particular attribute or set of attributes — it depends on which attributes are referred to in the propositional formula associated with the constraint. Users can specify any formula, but two predefined formulae are provided, as they are commonly expressed by users, as described next.

(i) **Interval preference**: it indicates that the value of an attribute should fall within a range of two provided values. Example: "*Laptop.hardDrive.size between 500GB and 750GB*." The propositional formula associated with this preference is an *and* formula composed of two attribute value specifications,

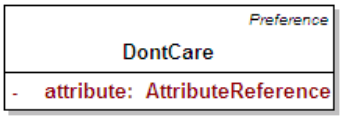


Figure 3.6: Don't care preference.

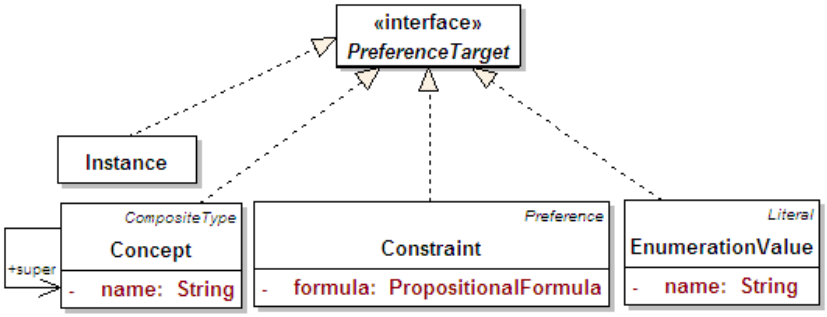


Figure 3.7: Preference targets.

one with the comparison operator *less than* (or *less than or equal to*) and the other with the comparison operator *greater than* (or *greater than or equal to*). Both attribute value specifications must refer to the same attribute.

(ii) **Around preference**: it indicates that an attribute should have a value that is close to a provided reference value, i.e. the closer the attribute value is to the reference value, the better. Example: "*Laptop.hardDrive.size around 500GB*." The propositional formula associated with this preference is an attribute value specification, whose comparison operator is *equal to*.

These two kinds of preferences do not state *how much* users prefer the restriction captured by this preference or make any kind of comparison — they are interpreted as "*I want to* ⟨*goal or constraint*⟩."

In addition, users might indicate that a certain attribute or value is totally irrelevant for them, by stating for instance "*I don't care about price.*" This means that the user does not impose any restriction to the values of this attribute, and is indifferent to all possible values that can be assigned to it. This kind of preference is referred to as **don't care**, and refers to an attribute as shown in Figure 3.6.

3.3.3 Preference Statements

Preference statements are sentences provided by users that state preferences over different entities of an application area, such as concepts and instances, and also the relevance of constraints. These entities and constraints are called **preference targets**, as they are targets of the provided preference statements. Figure 3.7 indicates the possible preference targets by showing which previously

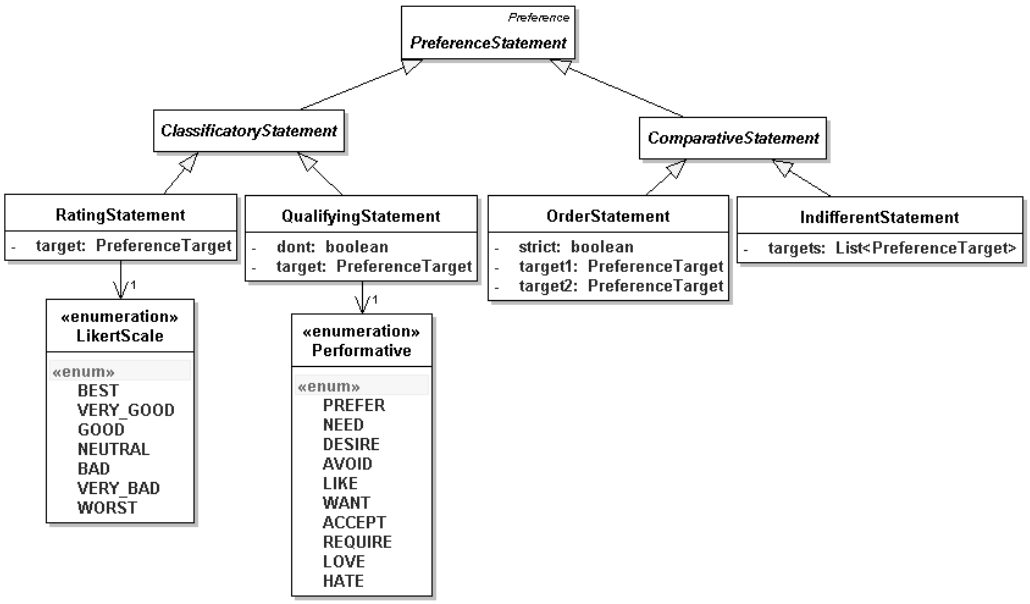


Figure 3.8: Preference statements model.

entities are a type of preference target. As preference statements are constructed by referring generically to targets, they can be any of the elements presented in Figure 3.7.

According to Hansson (Hansson 2001), preferences can be classified in two groups: (i) *monadic* preferences, which evaluate a single target, e.g. "*I like skiing*," and use terms such as "good," "very bad," and "worst;" and (ii) *dyadic* preferences, which indicate a relation between two targets, e.g. "*I prefer skying to surfing*," and use terms such as "better," "worse," and "equal in value to." These two main kinds of statements were indeed identified in our study, which are modelled as **classificatory statement** and **comparative statement**, respectively, as illustrated in Figure 3.8.

Our study showed that users not only classify preference targets as "good" and "bad" (**rating statement**) as suggested by Hansson (Hansson 2001), but also make extensive use of different expressive speech acts in their statement. This is captured by **qualifying statements**, which are speech acts (Searle 1969, Austin 1975) that classify targets with **expressive speech acts**, such as "*I need a light laptop*," "*I avoid Laptop.brand = Acer*." In addition, these expressive speech acts can also be used in the negative form, e.g. "*I don't need a light laptop*," represented by the attribute **don't** in qualifying statements. These different expressive speech acts are natural language expressions that indicate how hard (or soft) a preference is.

The other way of expressing preference statements is by stating an explicit comparison between two or more preference targets. The comparison can state that a target is preferred to another, strictly or not (**order statement**) or that a set of targets are equally preferred (**indifferent statement**).

3.3.4 Preference Priority

Users express different preferences about an application area, but these preferences can conflict with each other. In order to resolve these conflicts users indicate preferences over previously stated preferences, showing which are more relevant for them. This notion of preferences over preferences is represented as **priorities** in our metamodel.

The most common way that participants of our study indicated priority was numbering preference statements or restrictions, which is modelled with **preference priority** in our metamodel — see Figure 3.9. The other way was by stating relative importance over attributes, e.g. "*Laptop.quality is more important than Laptop.price*" (**attribute priority**) or "*Laptop.quality and Laptop.price are equally important to me*" (**attribute indifference**).

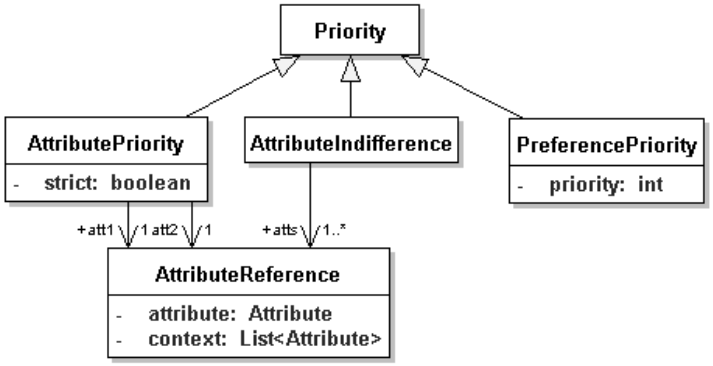


Figure 3.9: Preference priority model.

3.3.5 Interaction among Preferences and Targets

In previous sections, we presented different kinds of preferences: goals, constraints, don't care and four types of statements. Statements can refer to different entities of the ontology metamodel as well as to constraints, which are generically referred to as preference targets. In this section, we show how different statements interact with each target, by showing examples of their each possible combination. These examples are presented in Table 3.1(a). In addition, in order to make the remaining preferences and priorities clear for reader, we also provide examples of goals, don't care preferences, and the three types of priorities in Table 3.1(b). Therefore, Table 3.1 summarises most of the preference expressions that can be represented with our metamodel.

Table 3.1: Examples of preferences and priorities.

Type	Rating	Qualifying	Order	Indifferent
Ontology elements				
Concept	Laptop is good.	I need a laptop.	I prefer laptop to desktop.	I am indifferent to laptops and desktops.
Enumeration value	Color.pink is very good.	I accept Color.blue.	I prefer Color.blue to Color.green.	I am indifferent to Color.blue and Color.green.
Instance	HP is good.	I need a Dell.	I prefer HP to Dell.	I am indifferent to HP and Dell.
Constraints				
Constraint	Laptop.screen = small is very bad.	I like Laptop.screen = big.	I prefer Laptop.screen = big to Laptop.screen = small.	I am indifferent to Laptop.screen = big and Laptop.screen = small.
Interval	Laptop.screen.size between 14” and 15” is good.	I like Laptop.screen.size between 14” and 15”.	I prefer Laptop.screen.size between 10” and 13” to between 14” to 15”.	I am indifferent to Laptop.screen.size between 10” and 13”, and between 14” to 15”.
Around	Laptop.screen.size around 15” is good.	I like Laptop.screen.size around 15”.	I prefer Laptop.screen.size around 15” to around 10”.	I am indifferent to Laptop.screen.size around 15” and around 10”.

(a) Preference statements and targets interaction.

Goal	Minimise Laptop.weight.
Don’t Care	I don’t care about Laptop.bluetooth.
Priorities	
Attribute priority	Laptop.size is more important than Laptop.weight.
Attribute indifference	Laptop.size is as important as Laptop.weight.
Preference priority	1. Minimise price. 2. I need Screen.size ≥ 15”.

(b) Goals, don’t care preferences and priorities.

3.4 Final Remarks

In this chapter, we described a preference metamodel that contemplates patterns and expressions that are used by people and were identified in our study of how humans express preferences. The metamodel includes an ontology metamodel, whose instances describe an application area, such as laptops and their attributes. Propositional formulae are also represented, as many preferences refer to them to indicate preferred attribute values. Our metamodel consists of different types of preferences, such as goals and constraints, or more complex preferences that are associated with natural-language-like expressions, e.g. expressive speech acts. Many of these preferences are not part of existing preference metamodels, and this limitation of existing approaches is discussed in next chapter.