Chapter VI

Ontologies for Model-Driven Service Engineering

Introduction

In Chapter II we discussed the fundamental properties and concepts of a service. Concepts like interface, contract, service provider and service consumer are universal (i.e., they apply to all types of services). However, in as much as they are intuitive and universal, service concepts such as the aforementioned lack widely agreed upon semantics. The term *semantics* is used by disciplines such as philosophy, mathematics, and computer science to refer to “the meaning of things.” Meaning is usually attributed to a concept via its association with other concepts. In everyday speech, defining, for example, a “car” to be a kind of a “vehicle” is an attempt to attribute meaning to “car” by associating it with another, more abstract concept called “vehicle.” If the recipient of this definition already understands the concept of a vehicle, then he/she can also understand the concept of car via its association with the more abstract/generic concept vehicle.
Ultimately, attributing semantics to concepts requires two things: the use of linguistic constructs such as “is a kind of” or “is similar to” to describe their associations to other concepts, and a dictionary that contains concepts of various degrees of abstraction/specificity. Such dictionary must contain some primitive concepts that are not defined in terms of any other concepts in the vocabulary. If, for example, we know that our audience does not know the meaning of *vehicle*, we can define a vehicle to be “a kind of machinery used for transportation” and define terms such as *machinery* and *transportation* using other concepts, if necessary. Naturally, this cannot continue ad infinitum, as our vocabulary must contain a finite number of such concepts.

Semantics are therefore used to achieve common understanding and consensus on the meaning of various abstract or concrete concepts. In turn, common understanding is needed when information and knowledge regarding such concepts need to be communicated. When a car salesperson needs to communicate information about cars to his or her potential customers, he or she will use written or verbal words to convey information about the cars. Standard terms used in the course of everyday life are assumed to be shared by the car salesman and his or her clients. More technical terms, however (e.g., those describing technical details about the car’s engine) might have to be defined. Assuming that they are relatively free from too much technical jargon, car brochures manage to get information across to the intended buyers because they contain information described in common language terms. The semantics of such terms are built in the human language used to describe them, and the majority of people share a common vocabulary of that language.

So far we have assumed that the recipients of such communication of semantics are humans. Thanks to the inherent flexibility of natural (“human”) languages such as English, French, and so on, semantics can be captured and transmitted fairly easily, by using linguistic constructs. This is how humans acquire knowledge about new concepts. When, however, one or more of the parties involved in such communication are software programs, human languages become unsuitable for conveying semantics. One of the main problems is that one cannot always assume that the receiving and transmitting software programs will share the same vocabulary of concepts. To be able to teach a software program the concept of a car by associating it with a vehicle, we need to know whether the program already understands (i.e., has some kind of internal representation) of the concept of a vehicle. We also need to ensure that the program understands the meaning of the semantic association “is a kind of” in order to infer the properties of the new concept car via its association to the more generic concept vehicle. Unfortunately, this is not always the case. Different programs usually make different assumptions about how to internally represent and use knowledge. If, for example, one such program models internally a car as an “automobile”, any attempt to communicate with such program knowledge about cars would be futile.