Chapter 7
Structural Models of Knowledge Representation

Constructing so-called *ontologies*[1] is one of the main goals of applying *structural models of knowledge representation*, which have been introduced in Sect. 2.4. In Artificial Intelligence and in computer science, an ontology[2] is defined as a formal specification (conceptualization) of a certain (application) domain which is defined in such a way that it can be used for solving various problems (in the scope of this domain) with the help of general reasoning methods.[3] Such a specification is of the *structural* form. It can be treated as a kind of encyclopedia for the domain which contains descriptions of notions, objects, relations between them, etc.

Automated reasoning with a general technique is possible if we separate the *domain knowledge* from this generic (for a given technique) reasoning scheme. Semantic networks, frames, and scripts are typical structural models for representing domain knowledge. We introduce them in the next three sections.

When we introduce notions concerning structural models of knowledge representation, we refer to the corresponding definitions and notations of *description logics*. These logics were introduced in the 1980s and the 1990s as formal models of ontology representations in Artificial Intelligence. They are used, as well, for constructing efficient generic reasoning schemes, which are mentioned above.[4]

[1]Although there is an analogy between the notion of ontology in computer science and the notion of ontology in philosophy, we should differentiate between the two notions. In philosophy ontology is the study of being, its essential properties and its ultimate reasons.

[2]The system *Cyc*, which is developed by D. Lenat, is one of the biggest AI systems based on an ontology-based approach.

[3]Such standard reasoning methods are analogous to a universal reasoning scheme, which is discussed in a previous chapter.

[4]Description logics are introduced formally in Appendix D.

M. Flasiński, *Introduction to Artificial Intelligence*,
DOI 10.1007/978-3-319-40022-8_7

7.1 Semantic Networks

Semantic networks were introduced by Allan M. Collins and Ross Quillian in 1969 [56] as a result of their research into (natural) language understanding. They assumed that formulating knowledge in the form of a set of *notions* which relate to one another allows us to understand this knowledge better. Therefore, knowledge systems are constructed in just such a way. For example, in mathematics we introduce successive notions referring to the notions defined already. This is shown for geometry in Fig. 7.1a. Let us notice that notions which are successively introduced (from top to bottom in the figure) are particular cases of notions which have been introduced already. In other words, a new notion has all the properties of its predecessor notions and it has also new specific properties. Thus, a notion which is introduced later constitutes a *subclass* of a notion which has been introduced earlier. For example, *Trapezoid* is a subclass of *Quadrilateral*, which in turn is a subclass of *Polygon*, etc. This relation is represented by directed edges, which are labeled by *is subclass* in semantic networks. In description logics we talk about *general concept inclusion* and we denote it as follows:

$$Trapezoid \sqsubseteq Quadrilateral,$$
$$Quadrilateral \sqsubseteq Polygon, \text{etc.}$$

We construct taxonomies in the natural sciences to systematize our knowledge in such a way. For example, a part of such a taxonomy defined for the notion of *Animals* is shown in Fig. 7.1b. Let us notice that concept inclusion is also defined in this case, i.e., classes which are placed at lower levels are subclasses of certain classes placed above.

Sometimes we define an ontology (or its parts) in such a way that new concepts are constructed with the help of a few simple elementary notions. Such simple elementary notions are called *atomic concepts*. For example, for a "color ontology" we can assume the following atomic concepts, which correspond to primary colors[5]: *red* (*R*), *green* (*G*), *blue* (*B*). Then, we can define successive (complex) concepts: *yellow* (*Y*) ≡ *red mixed with green*[6]; *violet* (*V*) ≡ *red mixed with blue*; *white* (*W*) ≡ *red mixed with green mixed with blue*. In a description logic such a definition of complex concepts (here, colors) is expressed in the following way:

$$Y \equiv R \sqcup G,$$
$$V \equiv R \sqcup B,$$
$$W \equiv R \sqcup G \sqcup B.$$

[5]In the RGB (Red-Green-Blue) color model.

[6]We assume that secondary colors are obtained with the help of additive color mixing, i.e., by mixing visible light from various colored light sources.

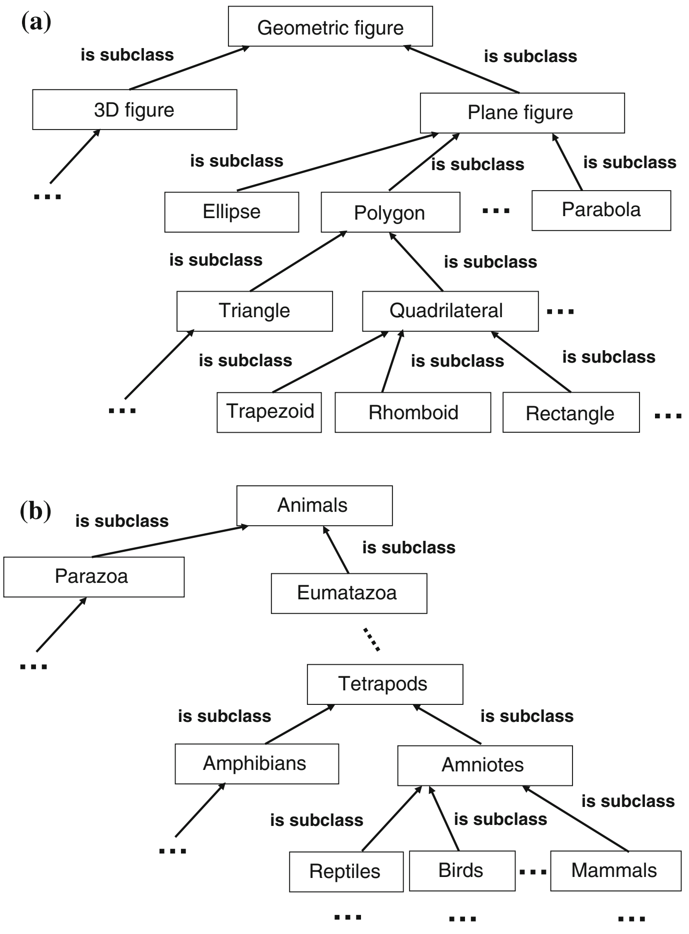

Fig. 7.1 Examples of simple semantic networks (ontologies): **a** in geometry, **b** in biology

Objects are the second generic element of semantic networks. Objects represent individuals of a certain domain. We say that objects are *instances* (examples) of a certain class (concept). For example, an object *John Smith* (a specific person having this name and this surname, who is identified in a unique way by a Social Security number) can be an instance of a class *American*. The existence of such an instance (in this case, a person) is represented by *American(John Smith)* in descriptive logics. Let us notice that a class (concept) can be treated as a set of objects. We introduce a relation *is* (*is a*) in semantic networks for denoting the fact that an object belongs to a class.

For example, a part of a semantic network which contains two objects *John Smith* and *Ava Smith*, and their characteristics is shown in Fig. 7.2a. As we can infer from this representation, the object *John Smith* is a *male* and a *colonel*. The class *colonel*

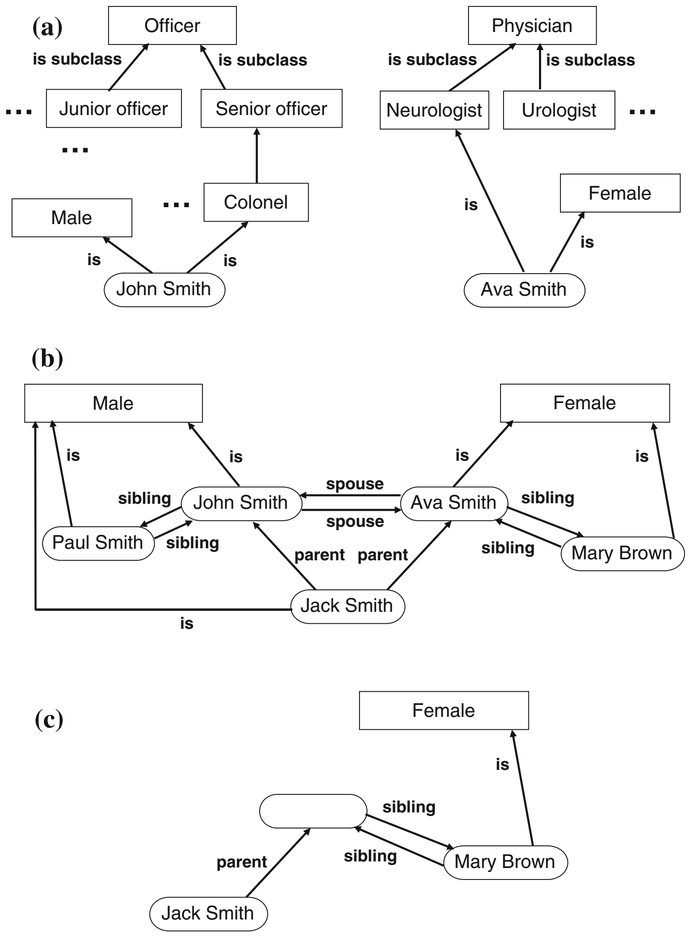

Fig. 7.2 Examples of semantic networks: **a** containing objects, **b** defining roles, and **c** a representation of a query in a system which is based on a semantic network

is a subclass of the class *senior officer* and the class *senior officer* is a subclass of the class *officer*. The object *Ava Smith* is a *female* and a *physician*, strictly speaking a *neurologist*.

Roles are the third generic element of semantic networks. Roles are used for describing relations between objects (sometimes also between classes). For example, we can introduce roles *spouse*, *parent*, and *sibling* in order to represent a genealogical knowledge. A semantic network which represents a part of a genealogy ontology is shown in Fig. 7.2b. We see that, for example, *John Smith* and *Ava Smith* are the parents of *Jack Smith*. In descriptive logics some roles represented by the semantic network shown in Fig. 7.2b can be defined in the following way:

$$\begin{aligned}&parent(Jack\ Smith,\ John\ Smith),\\&parent(Jack\ Smith,\ Ava\ Smith),\\&spouse(John\ Smith,\ Ava\ Smith),\ \text{etc.}\end{aligned}$$

Similarly to the case of concepts, we can define complex roles on the basis of simpler *atomic roles* (and concepts). For example, the role *grandfather* can be defined as a *parent* of a *parent* and a *male*. The role *aunt* can be defined as a *sibling* of a *parent* and a *female*.

A variety of reasoning methods have been developed in order to extract knowledge from semantic networks. One of the simplest is the *method of structural matching*. For example, if we would like to verify the validity of the following proposition:

Mary Brown is an aunt of *Jack Smith*,

then we should define a general structural pattern which represents such a proposition and then we should check whether the pattern can be matched to some part of our semantic network. A structural pattern for our proposition is shown in Fig. 7.2c. Let us notice that some elements are fixed and some elements are not fixed. The object that is a parent for *Jack Smith* is not fixed. If we denote it by *X*, then we have:

$$parent(Jack\ Smith,\ X),$$

since when we look for an aunt of *Jack Smith*, it is unimportant whether she is a sibling of his father or his mother. Of course, one can easily see that such a pattern defined for the verification of our proposition can be matched to a part of the semantic network shown in Fig. 7.2b.

The efficiency of pattern matching methods is a crucial problem of reasoning in semantic networks. A semantic network is a graph from a formal point of view. As we know from computational complexity theory, this problem, i.e., graph pattern matching, is of the non-polynomial complexity. Therefore, at the end of the twentieth century intensive research was carried out for the purpose of constructing methods of efficient graph processing. We discuss such methods in Chap. 8, in which graph grammars are introduced.

7.2 Frames

Frames were introduced by Marvin Minsky in 1975 [203]. As we have mentioned in Sect. 2.4, a frame system can be treated as an extension of a semantic network. The extension consists of replacing the nodes of a network by complex structures called *frames*, which allow us to characterize objects and classes in a detailed way. In the case of objects we talk about *object frames* and in the case of classes we talk about *class frames*.

A frame consists of *slots*, which are used for describing features and properties of an object/concept precisely. Each slot consists of *facets*. For example, if we construct a frame of an object which describes some device, then it can be characterized by certain properties such as voltage, temperature, pressure (of gas within the device), etc. Each property is represented by a slot. However, for each property various "aspects" can be defined. For example, for voltage we can define a current value, a unit (mV, V, kV, MV, etc.), an accuracy of measurement, a measuring range, etc. In a slot of a frame facets are used for storing such aspects.

Some default types of *facets* are used in frame systems. The most important ones include the following types.

- *VALUE*—the current value of the slot is stored in this facet.
- *RANGE*—this contains a measuring range or a list of values of the slot which are allowed.
- *DEFAULT*—this contains the default value of the slot. We can assume that this value is valid, if e.g., the facet of a type *VALUE* is not known at the moment.
- *belongs to a class* (for an object frame)—this contains a pointer to a class to which this object belongs.
- *is a subclass of* (for a class frame)—this contains a pointer to a class for which this class is a subclass.

Inheritance is a basic reasoning mechanism in frame systems. It is based on a fundamental property of ontologies, which is that subclasses *inherit* all features of superclasses (in the sense of *general concept inclusion*). Due to this property, if knowledge which concerns a certain class is updated/modified then it can be propagated to subclasses of this class (and to objects which belong to this class). This mechanism is enhanced, additionally, by the fact that an object can belong to more than one class. For example, the object *John Smith* belongs to two classes, *Male* and *Colonel*, in Fig. 7.2a. An analogous property concerns a class which can be a subclass of many classes. In such case, we talk about *multiple inheritance*.

Demon procedures, also called *demons*, are the second reasoning mechanism in frame systems. A *facet* of a slot can be not only of a static form (data, pointer to other frame, etc.), but also of the dynamic form of a demon. This peculiar name for these procedures comes from their idea, which is described by some authors as lying in wait to be invoked. If a demon is invoked by "jostling" its frame, e.g., by demanding some information, then it is awoken and it begins to operate. Similarly to the case of static facets, there are many types of demons. The most popular types include the following cases.

- *if-needed*—activated if a value of a facet is not known and we want to acquire it. Then, a demon tries to acquire/compute it from other frames.
- *if-added*—triggered if a new value has been added to a facet.
- *if-updated*—activated if a value in a facet has been updated.
- *if-removed*—triggered if a value has been removed from a facet.
- *if-read*—activated if a value has been read from a facet.
- *if-new*—triggered if a new frame is generated.

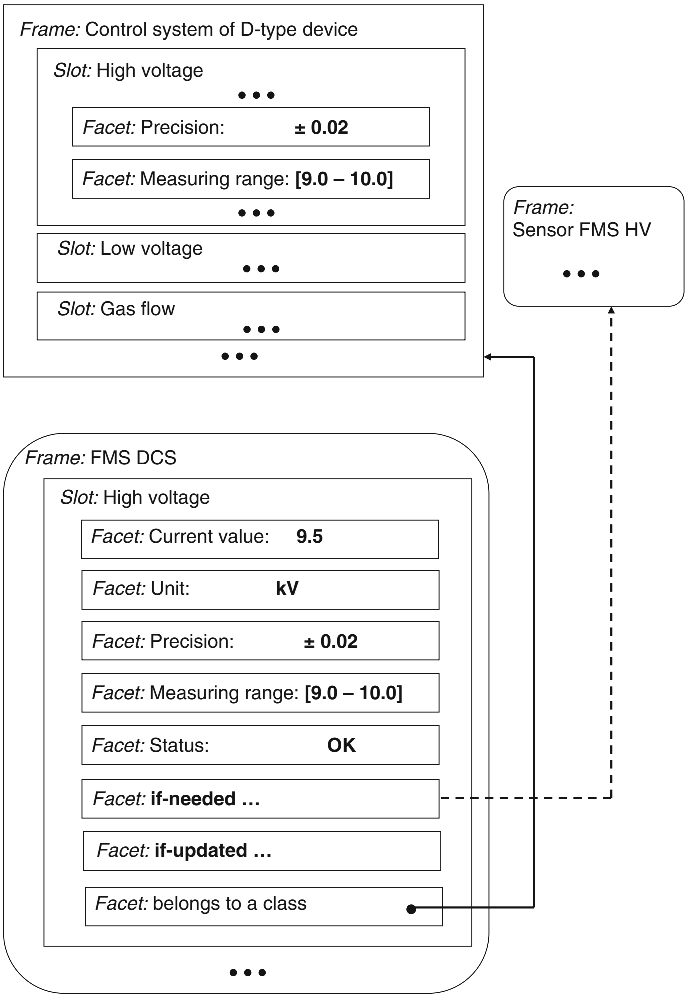

Fig. 7.3 Part of a model defined for a frame-based system for controlling complex industrial-like equipment

Let us analyze a part of a model defined for a frame-based system for controlling complex industrial-like equipment,[7] which is shown in Fig. 7.3. A class frame *Control system of D-type device* contains design characteristics of the device such as *High voltage*, *Low voltage* (a device has two types of electric power supply), *Gas flow*, etc. Each parameter relates to a slot. *Facets* of a slot specify these parameters in a

[7]This example has been defined on the basis of the documentation of the project *Generic Requirements Model for LHC Control Systems*, which was coordinated by the author and Dr. Axel Daneels at *Conseil Européen pour la Recherche Nucléaire* (*CERN*) in Geneva in 1997–1998.

precise way. For example, for high voltage we specify *Precision* (of a measurement), *Measuring range*, etc.

An object frame corresponding to a specific control system called *FMS DCS* belongs to a class defined previously. This means that the system has been constructed according to the design assumptions of this class. Thus, certain design characteristics, e.g., *Precision*, *Measuring range*, are inherited by the *FMS DCS* object frame from this class. As we can see, in the slot *High voltage* the facet *Current value* is filled during monitoring of the device. It is obtained not by inheritance from its class, but by a demon *if-needed*, which gains such information from another object frame called *Sensor FMS HV*. The object frame *Sensor FMS DCS* corresponds to a measuring device which monitors the high voltage of the device *FMS*. (In such a monitoring system, this demon is invoked continuously.) The second demon *if-updated* is included in the slot *High voltage* of the frame *FMS DCS* as well. It is triggered each time a new value is written into the slot *Current value*. If the value is contained within *Measuring range*, then the demon fills the facet *Status* with *OK*. Otherwise, it writes a suitable message specifying the type of the error to the facet *Status*, and it performs certain control actions which concern the device monitored.

Artificial Intelligence systems which are based on frames have influenced the Object-Oriented paradigm in software engineering very strongly. Designing software systems based on objects, classes, and inheritance is nowadays a fundamental technique for implementing information systems.

The efficiency (in the sense of time efficiency) of frame-based AI systems is a basic problem. Since the structure of a frame system does not contain mechanisms that control the reasoning process explicitly, we should possess a programming environment which ensures computational efficiency at an acceptable level. Such programming environments, which allow us to obtain high efficiency for systems containing a lot of frames, have been constructed since the very beginning of frame-based systems.[8]

7.3 Scripts

Scripts were proposed by Roger Schank and Robert P. Abelson in 1977 [264] for Natural Language Processing, NLP. The model is based on the following observation in psychology. If we want to understand a message which concerns a certain event (gossip told by a friend, coverage of a broadcast parliamentary debate, etc.), then we refer to a generalized pattern that relates to the type of this event. This pattern is constructed on the basis of earlier similar events. Then, it is stored in our memory. For example, if a child goes with her/his mom to a local clinic yet again, then

[8]For example, an AI control system containing about 100 class frames and more than 3000 object frames, which has been implemented for the high-energy physics experiment under the supervision of the author and Dr. Ulf Behrens, has processed data in real time (Flasiński M.: Further Development of the ZEUS Expert System: Computer Science Foundations of Design. *DESY Report 94-048*, Hamburg, March 1994, ISSN 0418-9833).

she/he knows from experience that this event consists of the following *sequence of elementary steps*: entering the clinic, going to the reception desk, waiting in a queue, entering the doctor's surgery, being asked "Where does it hurt?" by a doctor, having a checkup, being written a prescription by a doctor and (at last!) exiting the clinic. (When I was a child, then there was always also an obligatory visit to a toy shop.)

Such a representation defines the typical course of a certain event. Knowledge of such a form can be used in an AI system for predicting the course of events or for reasoning: What should be done in order to achieve a specific goal? In the case of Natural Language Processing problems, if some information is lacking, we can "guess" it with the help of a script. However, if a message is not *structured* in a proper way (e.g., the message is chaotic, the chronology of an event is disturbed), then matching an ambiguous description of a specific event to a pattern event (script) can be difficult.

Summing up, a *script* can be defined as a structural representation which describes an event of a certain type in a generalized/stereotyped way,[9] taking into account a particular context. The definition of a script is formalized with the help of the following elements.

- *Agents* are objects which can impact on other objects and which can be influenced by other objects. In the example of visiting a local clinic, a *child*, a *mother*, a *doctor*, etc. are agents.
- *Props* are things which occur in a script. In our example a *clinical thermometer* and a *prescription* are props.
- *Actions* are elementary events which are used for constructing the whole event. In our example *writing* a prescription and *exiting* the clinic are actions.
- *Preconditions* are propositions which have to be true at the moment of starting an inference with the help of a script, e.g., *a child is ill*, *a local clinic is open*, etc.
- *Results* are propositions which have to be true at the moment of ending an inference with the help of a script, e.g., *a prescription is written out by a doctor*.

In a standard Schank-Abelson model actions are defined in a hierarchical, two-level way. *Scenes* are defined at a higher level (e.g., *doctor is giving a child a checkup*) and are represented with the help of *conceptual dependency graphs*, which have been introduced in Sect. 2.4. *Elementary acts*, which correspond to nodes of CD graphs, are defined at a lower level. Elementary acts are constructed with *conceptual primitives* (introduced in Sect. 2.4) such as *PTRANS*, which denotes "change the physical location of an object", *SPEAK*, which denotes "produce a sound", etc.

At the end of the chapter, let us consider a (simplified) example of constructing a script. Let us assume that Paul has told the following gossip to me.

> Mark was angry with Tom. Therefore, Mark backbit Tom during a party. When Tom found out about it, he became offended at Mark. Paul decided to reconcile Mark with Tom. So, he invited them to a pub. Mark, after drinking a few beers, apologized to Tom for backbiting. As a result Tom mended fences with Mark.

[9] Such a stereotyped sequence of elementary steps which define an event is sometimes called a *stereotyped scenario*.

This story can be represented with the help of the object shown in the lower left-side part of Fig. 7.4.

The next day I read the following article on a dispute between Ruritania and Alvania in my favorite newspaper.

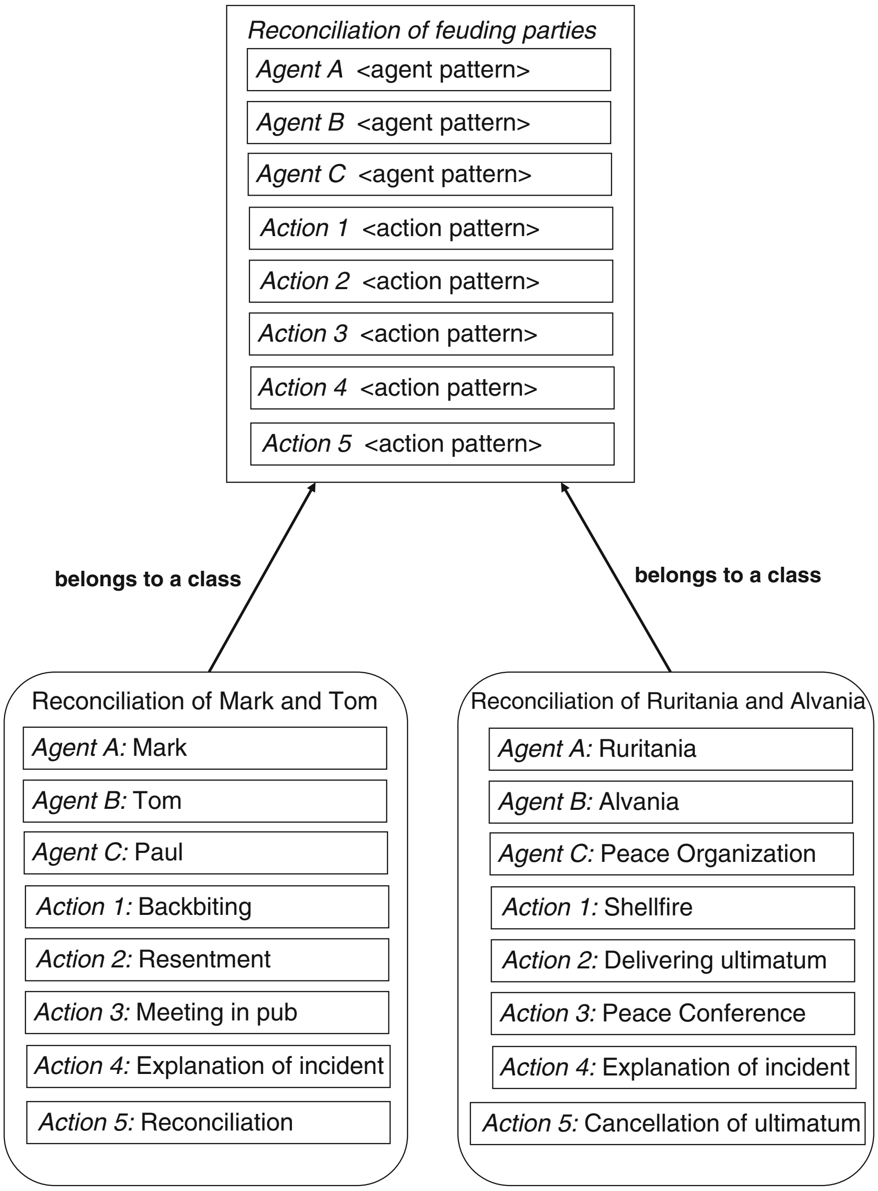
Reconciliation of feuding parties
Agent A <agent pattern>
Agent B <agent pattern>
Agent C <agent pattern>
Action 1 <action pattern>
Action 2 <action pattern>
Action 3 <action pattern>
Action 4 <action pattern>
Action 5 <action pattern>
belongs to a class
belongs to a class
Reconciliation of Mark and Tom
Agent A: Mark
Agent B: Tom
Agent C: Paul
Action 1: Backbiting
Action 2: Resentment
Action 3: Meeting in pub
Action 4: Explanation of incident
Action 5: Reconciliation
Reconciliation of Ruritania and Alvania
Agent A: Ruritania
Agent B: Alvania
Agent C: Peace Organization
Action 1: Shellfire
Action 2: Delivering ultimatum
Action 3: Peace Conference
Action 4: Explanation of incident
Action 5: Cancellation of ultimatum

Fig. 7.4 An example of defining a script

Two weeks ago the Ruritanian troops attacked the territory of Alvania. The next day the ambassador of Alvania delivered an ultimatum to the foreign minister of Ruritania. Then, the Peace Organization decided to organize a peace conference. Representatives of Ruritania apologized to representatives of Alvania and explained it was a misunderstanding. As a result, Alvania canceled its ultimatum.

This article can be represented with the help of the object shown in the lower right-side part of Fig. 7.4. For both objects, we can define a generalized description of a reconciliation of feuding parties, which is, in fact, their class of abstraction. Such a class is, according to a definition introduced above, a *script*. This script contains a stereotyped sequence of elementary acts of the scenario, as shown in Fig. 7.4.

Bibliographical Note

Structural models of knowledge representation are introduced in classic monographs concerning Artificial Intelligence [189, 211, 315]. This area is discussed in detail in [36, 280, 281]. Monographs [130, 266] are good introductions to descriptive logics.