# 4

# Relationships with other Formalisms

Ulrike Sattler

Diego Calvanese

Ralf Molitor

**Abstract**

In this chapter, we are concerned with the relationship between Description Logics and other formalisms, regardless of whether they were designed for knowledge representation issues or not. We concentrated on those representation formalisms that either (1) had or have a strong influence on Description Logics (e.g., modal logics), (2) are closely related to Description Logics for historical reasons (e.g., semantic networks and structured inheritance networks), or (3) have similar expressive power (e.g., semantic data models). There are far more knowledge representation formalisms than those mentioned in this section. For example, “verb-centered” graphical formalisms like those introduced by Simmons [1973] are not mentioned since we believe that their relationship with Description Logics is too weak.

## 4.1 AI knowledge representation formalisms

In artificial intelligence (AI), various “non-logical” knowledge representation formalisms were developed, motivated by the belief that classical logic is inadequate for knowledge representation in AI applications. This belief was mainly based upon cognitive experiments carried out with human beings and the wish to have representational formalisms that are close to the representations in human brains. In this Section, we will discuss some of these formalisms, namely semantic networks, frame systems, and conceptual graphs. The first two formalisms are mainly presented for historical reasons since they can be regarded as ancestors of Description Logics. In contrast, the third formalism can be regarded as a “sibling” of Description Logics since both have similar ancestors and live in the same time.

### 4.1.1 Semantic networks

Semantic networks originate in Quillian's *semantic memory models* [Quillian, 1967], a graphical formalism designed to represent "word concepts" in a definitorial way, i.e., similar to the one that can be found in an encyclopedia definition. This formalism is based on labelled graphs with different kinds of edges and nodes. Besides others, Quillian's networks allow for *subclass/superclass* edges, for *and*- and *or* edges, and for *subject/object* edges between nodes.

Following Quillian's memory models, a great variety of *semantic network* formalisms were proposed; an overview of their history can be found in [Brachman, 1979]. In general, semantic networks distinguish between *concepts* (denoted by *generic nodes*) and *individuals* (denoted by *individual nodes*), and between *subclass/superclass edges* and *property edges.* Using subclass/superclass links, concepts can be organised in a specialisation hierarchy. Using property edges, properties can be associated to concepts, that is, to the individuals belonging to the concept the properties are associated with. Figure 4.1 contains a hierarchy of animals, birds, fishes, etc. Interestingly, the cognitive adequacy of this approach was proven empirically [Collins and Quillian, 1970].

The two kinds of edges interact with each other: A property is *inherited* along subclass/superclass edges—if not modified in a more specific class. For example, birds are equipped with skin because animals are equipped with skin, and birds

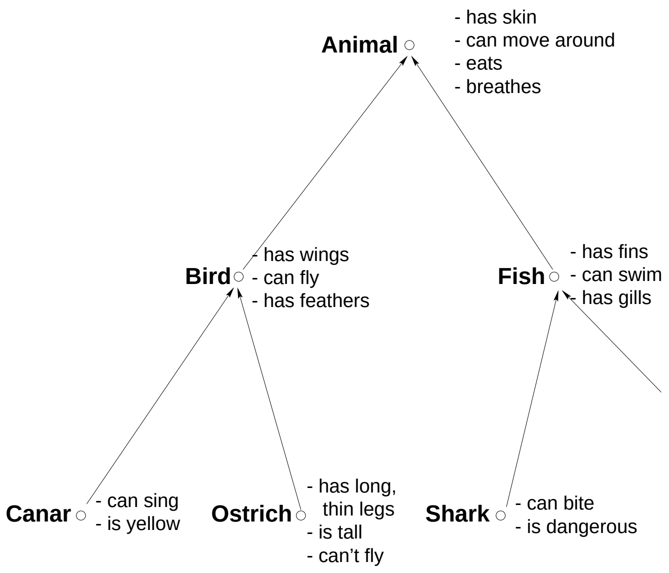

Fig. 4.1. A semantic network describing animals.

inherit this property because of the subclass/superclass edge between birds and animals. In contrast, although ostriches are birds, they do not inherit the property "can fly" from birds because this property is "modified" for ostriches.

Intuitively, it should be possible to translate subclass/superclass edges into concept definitions, for example,[1]

$$\mathsf{Shark} \equiv \mathsf{Fish} \sqcap \mathsf{CanBite} \sqcap \mathsf{IsDangerous}.$$

According to Brachman [1985], the above translation is not always intended. Subclass/superclass edges can also be read as *primitive* concept definitions, that is, they impose only necessary properties but not sufficient ones. Hence the above translation might better be

$$\mathsf{Shark} \sqsubseteq \mathsf{Fish} \sqcap \mathsf{CanBite} \sqcap \mathsf{IsDangerous}.$$

Due to the lack of a precise semantics, there are even more readings of subclass/superclass edges which are discussed in Woods [1975], [1977b; 1985]. A prominent reading is the one of *inheritance by default*, which can be specified in different ways, thus leading to misunderstandings and to the question which of these specifications is the "right" one (see also Chapter 6).

As a consequence of this ambiguity, new formalisms mainly evolved along two lines: (1) To capture inheritance by default, various non-monotonic inheritance systems, respectively various ways of reasoning in non-monotonic inheritance systems, were investigated [Touretzky *et al.*, 1987; 1991; Selman and Levesque, 1993]. (2) To capture the monotonic aspects of semantic networks, a new graphical formalism, *structured inheritance networks*, was introduced and implemented in the system KL-ONE [Brachman, 1979; Brachman and Schmolze, 1985]. It was designed to cover the declarative, monotonic aspects of semantic networks, and hence did not specify the way in which (non-monotonic multiple) inheritance was supposed to function in conflicting situations. Brachman and Schmolze [1985] argue that KL-ONE does not allow for cancellation or inheritance by default because such mechanisms would make taxonomies meaningless. Indeed, all properties of a given concept could be cancelled, so that it would fit everywhere in the taxonomy. Their proposition is to make a strict separation of default assertions and conceptual descriptions.

Brachman and Schmolze [1985], besides pointing out the computation of the taxonomy as a core system service, describe the meaning of various concept constructors that were implemented in KL-ONE, for example conjunction, universal value restrictions, role hierarchies, role-value-maps, etc. Moreover, we find a clear distinction between individuals and concepts, and between a terminological and an assertional formalism.

[1] In the following, we use standard Description Logics as defined in Chapter 2.

Later [Levesque and Brachman, 1987], Kl-One was provided with a well-defined "Tarski-style" semantics which fixed the precise meaning of its graphical constructs and led to the definition of the first *Description Logic* [Levesque and Brachman, 1987], at that time also called terminological languages, concept languages, or Kl-One based languages. Besides giving a precise meaning to semantic networks, this formalisation allowed the investigation of inference algorithms with respect to their soundness, completeness, and computational complexity. For example, it turned out that subsumption in Kl-One is undecidable, mainly due to role-value-maps [Schmidt-Schauß, 1989].

### 4.1.2 Frame systems

Minsky [1981] introduced frame systems as an alternative to logic-oriented approaches to knowledge representation, which he thought were not adequate to "simulate common sense thinking" for various reasons. His system provides record-like data structures to represent prototypical knowledge concerning situations and objects and includes defaults, multiple perspectives, and analogies. Nowadays, semantic networks and frame systems are often viewed as the same family of formalisms. However, in standard semantic networks, properties are restricted to primitive, atomic ones, whereas, in general, properties in frame systems can be complex concepts described by frames.

One goal of the frame approach was to gather all relevant knowledge about a situation (e.g., entering a restaurant) in one object instead of distributing this knowledge across various axioms. Roughly spoken, a situation (or an object) is described in one *frame*. Similar to entries in a record, a frame contains *slots* to represent properties of the situation described by the frame. Reasoning comes in two shapes: (1) Using a "partial matching", more specific frames are embedded into more general ones, thus giving, for example, meaning to a new situation or classifying an object as a kind of, say, bird. (2) Searching for slot *fillers* to collect more information concerning a specific situation. A variety of expert systems [Fikes and Kehler, 1985; Christaller *et al.*, 1992; Gen, 1995; Flex, 1999] are based on a frame-based formalism and are further enhanced with rules, triggers, daemons, etc.

Despite the fact that frame systems were designed as an alternative to logic, the monotonic, declarative part of this formalism could be shown to be captured using first-order predicate logic [Hayes, 1977; 1979]. To our knowledge, no precise semantics could be given for the non-declarative, non-logic, or non-monotonic aspects of frame systems. Hence neither their expressive power nor the quality of the corresponding reasoning algorithms and services can be compared with other formalisms.

In the remainder of this section, we show how the monotonic part of a frame-based knowledge base can be translated into an $\mathcal{ALUN}$ TBox [Calvanese *et al.*, 1994].[1] Since there is no standard syntax for frame systems, we have chosen to use basically the notation adopted by Fikes and Kehler [1985], which is used also in the KEE [2] system.

A *frame definition* is of the form **Frame** : $F$ **in KB** $\mathcal{F}$ $E$, where $F$ is a *frame name* and $E$ is a *frame expression*, i.e., an expression formed according to the following syntax:

$$\begin{array}{rcl} E & \longrightarrow & \textbf{SuperClasses}:\ F_1,\ldots,F_h \\ & & \textbf{MemberSlot}:\ S_1 \\ & & \quad\textbf{ValueClass}:\ H_1 \\ & & \quad\textbf{Cardinality.Min}:\ m_1 \\ & & \quad\textbf{Cardinality.Max}:\ n_1 \\ & & \ldots \\ & & \textbf{MemberSlot}:\ S_k \\ & & \quad\textbf{ValueClass}:\ H_k \\ & & \quad\textbf{Cardinality.Min}:\ m_k \\ & & \quad\textbf{Cardinality.Max}:\ n_k \end{array}$$

$F_i$ denotes a frame name, $S_j$ denotes a slot name, $m_j$ and $n_j$ denote positive integers, and $H_j$ denotes slot constraints. A *slot constraint* can be specified as follows:

$$\begin{array}{rcl} H & \longrightarrow & F \mid \\ & & (\text{INTERSECTION } H_1\ H_2) \mid \\ & & (\text{UNION } H_1\ H_2) \mid \\ & & (\text{NOT } H) \end{array}$$

A *frame knowledge base* $\mathcal{F}$ is a set of frame definitions.

For example, Figure 4.2 shows a simple KEE knowledge base describing courses in a university. Cardinality restrictions are used to impose a minimum and maximum number of students that may be enrolled in a course, and to express that each course is taught by exactly one individual. The frame AdvCourse represents courses which enroll only graduate students, i.e., students who already have a degree. Basic courses, on the other hand, may be taught only by professors.

Hayes [1979] gives a semantics to frame definitions by translating them to first-order formulae in which frame names are translated to unary predicates, and slots are translated to binary predicates.

In order to translate frame knowledge bases to $\mathcal{ALUN}$ knowledge bases, we first define the function $\Psi$ that maps each frame expression into an $\mathcal{ALUN}$ concept expression as follows: Each frame name $F$ is mapped onto an atomic concept $\Psi(F)$,

[1] Not only the translation but also the example are by Calvanese *et al.* [1994].

[2] KEE is a trademark of Intellicorp. Note that a KEE user does not directly specify her knowledge base in this notation, but is allowed to define frames interactively via the graphical system interface.

**Frame**: Course **in KB** University
  **MemberSlot**: enrolls
    **ValueClass**: Student
    **Cardinality.Min**: 2
    **Cardinality.Max**: 30
  **MemberSlot**: taughtby
    **ValueClass**: (UNION GradStudent
                Professor)
    **Cardinality.Min**: 1
    **Cardinality.Max**: 1

**Frame**: AdvCourse **in KB** University
  **SuperClasses**: Course
  **MemberSlot**: enrolls
    **ValueClass**: (INTERSECTION
                GradStudent
                (NOT Undergrad))
    **Cardinality.Max**: 20

**Frame**: BasCourse **in KB** University
  **SuperClasses**: Course
  **MemberSlot**: taughtby
    **ValueClass**: Professor

**Frame**: Professor **in KB** University

**Frame**: Student **in KB** University

**Frame**: GradStudent **in KB** University
  **SuperClasses**: Student
  **MemberSlot**: degree
    **ValueClass**: String
    **Cardinality.Min**: 1
    **Cardinality.Max**: 1

**Frame**: Undergrad **in KB** University
  **SuperClasses**: Student

Fig. 4.2. A Kee knowledge base.

each slot name $S$ onto an atomic role $\Psi(S)$, and each slot constraint $H$ onto the corresponding Boolean combination $\Psi(H)$ of concepts. Then, every frame expression of the form

**SuperClasses** : $F_1, \ldots, F_h$
**MemberSlot** : $S_1$
  **ValueClass** : $H_1$
  **Cardinality.Min** : $m_1$
  **Cardinality.Max** : $n_1$
...
**MemberSlot** : $S_k$
  **ValueClass** : $H_k$
  **Cardinality.Min** : $m_k$
  **Cardinality.Max** : $n_k$

is mapped into the concept

$$\begin{array}{l}
\Psi(F_1) \sqcap \cdots \sqcap \Psi(F_h) \sqcap \\
\forall \Psi(S_1).\Psi(H_1) \sqcap \geqslant m_1\, \Psi(S_1) \sqcap \leqslant n_1\, \Psi(S_1) \sqcap \\
\cdots \\
\forall \Psi(S_k).\Psi(H_k) \sqcap \geqslant m_k\, \Psi(S_k) \sqcap \leqslant n_k\, \Psi(S_k).
\end{array}$$

Making use of the mapping $\Psi$, we obtain the $\mathcal{ALUN}$ knowledge base $\Psi(\mathcal{F})$ corresponding to a frame knowledge base $\mathcal{F}$, by introducing in $\Psi(\mathcal{F})$ an inclusion assertion $\Psi(F) \sqsubseteq \Psi(E)$ for each frame definition **Frame** : $F$ **in KB** $\mathcal{F}$ $E$ in $\mathcal{F}$.

$$\begin{array}{rcl}
\mathsf{Course} & \sqsubseteq & \forall\mathsf{enrolls}.\mathsf{Student} \sqcap \geqslant 2\,\mathsf{enrolls} \sqcap \leqslant 30\,\mathsf{enrolls} \sqcap \\
 & & \forall\mathsf{taughtby}.(\mathsf{Professor} \sqcup \mathsf{GradStudent}) \sqcap = 1\,\mathsf{taughtby} \\
\mathsf{AdvCourse} & \sqsubseteq & \mathsf{Course} \sqcap \forall\mathsf{enrolls}.(\mathsf{GradStudent} \sqcap \neg\mathsf{Undergrad}) \sqcap \leqslant 20\,\mathsf{enrolls} \\
\mathsf{BasCourse} & \sqsubseteq & \mathsf{Course} \sqcap \forall\mathsf{taughtby}.\mathsf{Professor} \\
\mathsf{GradStudent} & \sqsubseteq & \mathsf{Student} \sqcap \forall\mathsf{degree}.\mathsf{String} \sqcap = 1\,\mathsf{degree} \\
\mathsf{Undergrad} & \sqsubseteq & \mathsf{Student}
\end{array}$$

Fig. 4.3. The $\mathcal{ALUN}$ knowledge base corresponding to the KEE knowledge base in Figure 4.2.

The $\mathcal{ALUN}$ knowledge base corresponding to the KEE knowledge base given in Figure 4.2 is shown in Figure 4.3.

The correctness of the translation follows from the correspondence between the set-theoretic semantics of $\mathcal{ALUN}$ and the first-order interpretation of frames [Hayes, 1979; Borgida, 1996; Donini *et al.*, 1996b]. Consequently,

- verifying whether a frame $F$ is satisfiable in a knowledge base and
- identifying which of the frames are more general than a given frame,

are captured by concept satisfiability and concept subsumption in $\mathcal{ALUN}$ knowledge bases. Hence reasoning for the monotonic, declarative part of frame systems can be reduced to concept satisfiability and concept subsumption in $\mathcal{ALUN}$ knowledge bases.

### 4.1.3 Conceptual graphs

Besides Description Logics, conceptual graphs [Sowa, 1984] can be viewed as descendants of frame systems and semantic networks. Conceptual graphs (CGs) are a rather popular (especially in natural language processing) and expressive formalism for representing knowledge about an application domain in a graphical way. They are given a formal semantics, e.g., by translating them into (first-order) formulae.

In the CG formalism, one is, just as for Description Logics, not only interested in *representing* knowledge, but also in *reasoning* about it. Reasoning services for CGs are, for example, deciding whether a given graph is *valid*, i.e., whether the corresponding formula is valid, or whether a graph $g$ is *subsumed by* a graph $h$, i.e., whether the formula corresponding to $g$ implies the formula corresponding to $h$. Since CGs can express all of first-order predicate logic [Sowa, 1984], these reasoning problems are undecidable for general CGs. In the literature [Sowa, 1984; Wermelinger, 1995; Kerdiles and Salvat, 1997] one can find complete calculi for validity of CGs, but implementations of these calculi may not terminate for formulae that are not valid. An approach to overcome this problem, which has also been employed in the area of Description Logics, is to identify decidable fragments of the formalism. The most prominent decidable fragment of CGs is the class of *simple conceptual graphs* (SGs) [Sowa, 1984], which corresponds to the conjunctive, positive, and existential fragment of first-order predicate logic (i.e., existentially quantified conjunctions of atoms). Even for this simple fragment, however, subsumption is still an NP-complete problem [Chein and Mugnier, 1992].[1]

Although Description Logics and CGs are employed in very similar applications, precise comparisons were published, to our knowledge, only recently [Coupey and Faron, 1998; Baader *et al.*, 1999c]. These comparisons are based on translations of CGs and Description Logic concepts into first-oder formulae. It turned out that the two formalisms are quite different for several reasons:

(i) CGs are translated into *closed* first-order formulae, whereas Description Logic concepts are translated into formulae in one free variable;
(ii) since Description Logics use a variable-free syntax, certain identifications of variables expressed by cycles in SGs and by co-reference links in CGs cannot be expressed in Description Logics;
(iii) in contrast to CGs, most Description Logics considered in the literature only allow for unary and binary relations but not for relations of arity greater than 2;
(iv) SGs are interpreted by existential sentences, whereas almost all Description Logics considered in the literature allow for universal quantification.

Possibly as a consequence of these differences, so far no natural fragment of CGs that corresponds to a Description Logic has been identified. In the sequel, we will illustrate the main aspects of the correspondence result presented by Baader *et al.* [1999c], which strictly extends the one proposed by Coupey and Faron [1998].

### *Simple Conceptual Graphs*

*Simple conceptual graphs* (SGs) as introduced by Sowa [1984] are the most prominent decidable fragment of CGs. They are defined with respect to a so-called *support.* Roughly spoken, the support is a partially ordered signature that can be used to fix the a primitive ontology of a given application domain. It introduces a set of *concept types* (unary predicates), a set of *relation types* ($n$-ary predicates), and a set of *individual markers* (constants). As an example, consider the support $\mathcal{S}$ shown in Figure 4.4, where $\top$ is the most general concept type representing the entire domain. The partial ordering on the individual markers is flat, i.e., all individual markers are pairwise incomparable and the so-called *generic marker* $*$ is more general than

[1] Since SGs are equivalent to conjunctive queries (see also Chapter 16), the NP-completeness of subsumption of SGs is also an immediate consequence of NP-completeness of containment of conjunctive queries [Chandra and Merlin, 1977].

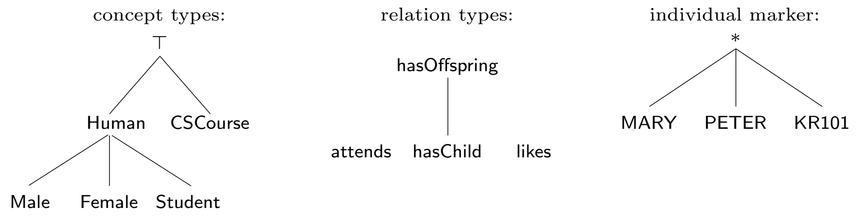

Fig. 4.4. An example of a support.

all individual markers. In this example, all relation types are assumed to have arity 2 and to be pairwise incomparable except for hasOffspring, which is more general than hasChild. The partial orderings on the types yield a fixed specialization hierarchy for these types that must be taken into account when computing subsumption relations between SGs. For binary relation types, this partial ordering resembles a role hierarchy in Description Logics.

An *SG over the support* $\mathcal{S}$ is a labelled bipartite graph of the form $g = (C, R, E, \ell)$, where $C$ is a set of *concept nodes*, $R$ is a set of *relation nodes*, and $E \subseteq C \times R$ is the edge relation.

As an example, consider the SGs depicted in Figure 4.5: the SG $g$ describes a woman Mary having a child who likes its grandfather Peter and who attends the computer science course number KR101; the SG $h$ describes all mothers having a child who likes one of its grandparents.

Each concept node is labelled with a concept type (such as Female) and a *referent*, i.e., an individual marker (such as MARY) or the generic marker $*$. A concept node is called *generic* if its referent is the generic marker; otherwise, it is called *individual concept node.* Each relation node is labelled with a relation type $r$ (such as hasChild), and its outgoing edges are labelled with indices according to the arity of $r$. For example, for the binary relation hasChild, there is one edge labelled with 1 (leading to the parent), and one edge labelled with 2 (leading to the child).

Simple graphs are given a formal semantics in first-order predicate logic (FOL) by the operator $\Phi$ [Sowa, 1984]: each generic concept node is related to a unique variable, and each individual concept node is related to its individual marker. Concept types and relation types are translated into atomic formulae, and the whole SG $g$ is translated into the existentially closed conjunction of all atoms obtained from the nodes in $g$.

In our example, this operator yields

$$\begin{aligned}\Phi(g) \quad = \quad & \exists x_1.(\mathsf{Female}(\mathsf{MARY}) \wedge \mathsf{Human}(\mathsf{PETER}) \wedge \mathsf{Student}(x_1) \wedge \\ & \quad \mathsf{CScourse}(\mathsf{KR101}) \wedge \mathsf{hasChild}(\mathsf{PETER}, \mathsf{MARY}) \wedge \\ & \quad \mathsf{hasChild}(\mathsf{MARY}, x_1) \wedge \mathsf{likes}(x_1, \mathsf{PETER}) \wedge \mathsf{attends}(x_1, \mathsf{KR101})),\end{aligned}$$

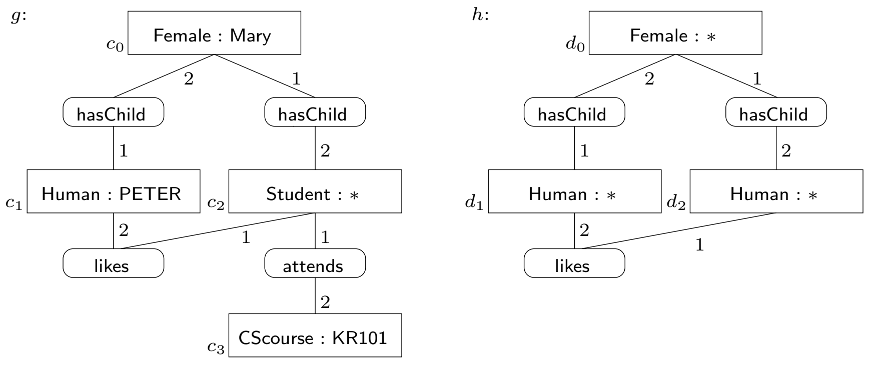

Fig. 4.5. Two simple graphs.

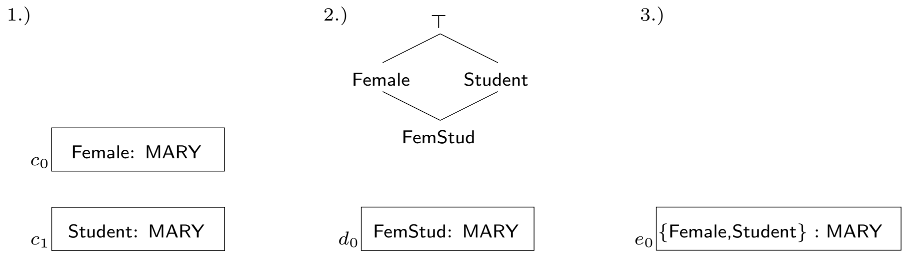

Fig. 4.6. Expressing conjunction of concept types in SGs.

$$\begin{aligned}\Phi(h) \quad = \quad & \exists x_0 x_1 x_2.(\mathsf{Female}(x_0) \wedge \mathsf{Human}(x_1) \wedge \mathsf{Human}(x_2) \wedge \\ & \qquad \mathsf{hasChild}(x_1, x_0) \wedge \mathsf{hasChild}(x_0, x_2) \wedge \mathsf{likes}(x_2, x_1)),\end{aligned}$$

where $x_1$ in $\Phi(g)$ is (resp. $x_0$, $x_1$, and $x_2$ in $\Phi(h)$ are) introduced for the generic concept node $c_2$ (resp. the generic concept nodes $d_0$, $d_1$, and $d_2$).

In general, there are three different ways of expressing conjunction of concept types. For example, suppose we want to express that Mary is both female and a student. This can be expressed by a SG containing one individual concept node for each statement (see Figure 4.6, 1.).[1] A second possibility is to introduce a new concept type in the support for a common specialization of Female(MARY) and Student(MARY) (see Figure 4.6, 2.). Finally, such a conjunction can be represented by labelling the corresponding concept node with a *set* of concept types instead of a single concept type (see Figure 4.6, 3.; for details on how to handle SGs labelled with sets of concept types see [Baader *et al.*, 1999c]).

*Subsumption with respect to a support $\mathcal{S}$* for two SGs $g$, $h$ is defined by a so-called *projection* from $h$ to $g$ [Sowa, 1984; Chein and Mugnier, 1992]: $g$ is *subsumed by* $h$ w.r.t. $\mathcal{S}$ iff there exists a mapping from $h$ to $g$ that (1) maps concept nodes

[1] Note that this solution cannot be applied if the individual marker MARY were substituted by the generic marker ∗, because the two resulting generic concept nodes would be interpreted by different variables.

(resp. relation nodes) in $h$ onto more specific (w.r.t. the partial ordering in $\mathcal{S}$) concept nodes (resp. relation nodes) in $g$ and that (2) preserves adjacency.

In our example (Figure 4.5), it is easy to see that $g$ is subsumed by $h$, since mapping $d_i$ onto $c_i$ for $0 \leq i \leq 2$ yields a projection w.r.t. $\mathcal{S}$ from $h$ to $g$.

Subsumption for SGs is an NP-complete problem [Chein and Mugnier, 1992]. In the restricted case where the subsumer $h$ is a tree, subsumption can be decided in polynomial time [Mugnier and Chein, 1992].

## *Concept Descriptions and Simple Graphs*

In order to determine a Description Logic corresponding to (a fragment of) SGs, one must take into account the differences between Description Logics and CGs mentioned before.

- Most Description Logics only allow for role terms corresponding to binary relations and for concept descriptions describing connected structures. Thus, Baader *et al.* [1999c] and Coupey and Faron [1998] restrict their attention to connected SGs over a support $\mathcal{S}$ containing only unary and binary relation types.
- Due to the different semantics of SGs and concept descriptions (closed formulae vs. formulae in one free variable), Coupey and Faron restrict their attention to SGs that are trees. Baader *et al.* introduce so-called *rooted* SGs, i.e., SGs that have one distinguished node called the *root*. An adaption of the operator $\Phi$ yields a translation of a rooted SG $g$ into a FO formula $\Phi(g)(x_0)$ with one free variable $x_0$.
- Since all Description Logics considered in the literature allow for conjunction of concepts, Baader *et al.* allow for concept nodes labelled with a set of concept types instead of a single concept type in order to express conjunction of atomic concepts in SGs. Coupey and Faron avoid the problem of expressing conjunction of atomic concepts: they just do not allow for (1) conjunctions of atomic concepts in concept descriptions, and (2) for individual concept nodes in SGs.

The Description Logic considered by Baader *et al.*, denoted by $\mathcal{ELIRO}_1$, allows for *existential restrictions* and *intersection of concept descriptions* ($\mathcal{EL}$), *inverse roles* ($\mathcal{I}$), *intersection of roles* ($\mathcal{R}$), and *unary one-of concepts* ($\mathcal{O}_1$). For the constants occurring in the one-of concepts the *unique name assumption* applies, i.e., all constants are interpreted as different objects. Coupey and Faron only consider a fragment of the Description Logic $\mathcal{ELI}$.

In both papers, the correspondence result is based on translating concept descriptions into syntax trees. For example, consider the $\mathcal{ELIRO}_1$-concept

$$\begin{aligned} C \;=\; & \mathsf{Female} \sqcap \exists \mathsf{hasChild}^{-}.(\mathsf{Human} \sqcap \{\mathsf{PETER}\}) \sqcap \\ & \exists(\mathsf{hasChild} \sqcap \mathsf{likes}).(\mathsf{Male} \sqcap \mathsf{Student} \sqcap \exists \mathsf{attends}.\mathsf{CScourse}) \end{aligned}$$

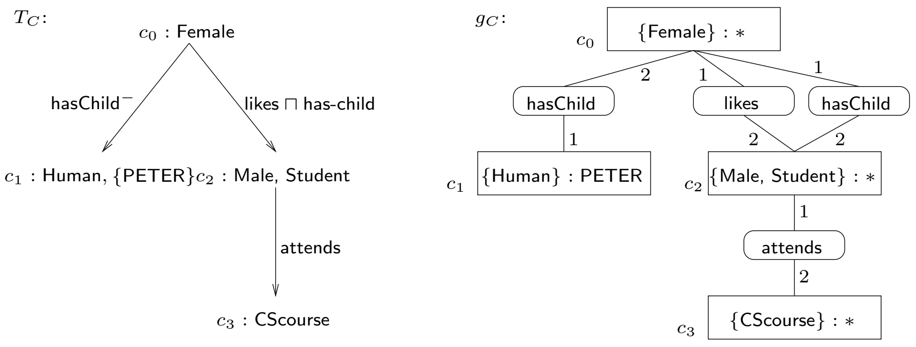

Fig. 4.7. Translating concept descriptions into simple graphs.

describing all daughters of Peter who have a dear child that is a student attending a computer science course. The syntax tree corresponding to $C$ is depicted on the left hand side of Figure 4.7.

One can show [Baader *et al.*, 1999c] that, if concept descriptions $C$ are restricted to *contain at most one unary one-of concept in each conjunction*, the corresponding syntax tree $T_C$ can be easily translated into an equivalent rooted SG $g_C$ that is a tree[1] (see Figure 4.7). Conversely, every rooted SG $g$ that is a tree and that contains only binary relation types can be translated into an equivalent $\mathcal{ELIRO}_1$-concept description $C_g$. There are, however, rooted SGs that can be translated into equivalent $\mathcal{ELIRO}_1$-concept descriptions though they are not trees. For example, the rooted SG $g$ depicted in Figure 4.5 is equivalent to the concept description

$$\begin{aligned} C_g \quad = \quad & \{\mathsf{MARY}\} \sqcap \mathsf{Female} \sqcap \exists \mathsf{hasChild}^-.(\mathsf{Human} \sqcap \{\mathsf{PETER}\}) \sqcap \\ & \exists \mathsf{hasChild}.(\mathsf{Student} \sqcap \exists \mathsf{attends}.(\{\mathsf{KR101}\} \sqcap \mathsf{CScourse}) \sqcap \exists \mathsf{likes}.\{\mathsf{PETER}\}) \end{aligned}$$

In general, the above correspondence result can be strengthened as follows [Baader *et al.*, 1999c]: Every rooted SG $g$ containing only binary relation types can be transformed into an equivalent rooted SG that is a tree if each cycle in $g$ with more than 2 concept nodes contains at least one individual concept node. Hence, each such rooted SG can be translated into an equivalent $\mathcal{ELIRO}_1$-concept description.

Note that the SG $h$ with root $d_0$ in Figure 4.5 cannot be translated into an equivalent $\mathcal{ELIRO}_1$-concept description $C_h$ because, in $\mathcal{ELIRO}_1$, one cannot express that the grandparent (represented by the concept node $d_1$) and the human liked by the child (represented by the concept node $d_2$) must be the same person.

The correspondence result between $\mathcal{ELIRO}_1$ and rooted SGs allows for transferring the tractability result for subsumption between SGs that are trees to $\mathcal{ELIRO}_1$. Furthermore, the characterization of subsumption based on projections between graphs was adapted to $\mathcal{ELIRO}_1$ and other Description Logics, e.g., $\mathcal{ALE}$, and is

[1] In this context, a tree may contain more than one relation between two adjacent concept nodes.

used in the context of inference problems like matching and computing least common subsumers [Baader and Küsters, 1999; Baader *et al.*, 1999b]. Conversely, the correspondence result can be used as a basis for determining more expressive fragments of conceptual graphs, for which validity and subsumption is decidable. Based on an appropriate characterization of a fragment of conceptual graphs corresponding to a more expressive Description Logic (like $\mathcal{ALC}$), one could use algorithms for these Description Logics to decide validity or subsumption of graphs in this fragment.

## 4.2 Logical formalisms

In this section, we will investigate the relationship between Description Logics and other logical formalisms.

Traditionally, the semantics of Description Logics is given in a Tarski-style model-theoretic way. Alternatively, it can be given by a translation into predicate logic, where it depends on the Description Logic whether this translation yields first order formulae or whether it goes beyond first order, as it is the case for Description Logics that allow, e.g., for the transitive closure of roles or fixpoints. Due to the variable-free syntax of Description Logics and the fact that concepts denote sets of individuals, the translation of concepts yields formulae in one free variable. Following the definition by Borgida [1996], a concept $C$ and its translation $\pi(C)(x)$ are said to be *equivalent* if and only if, for all interpretations[1] $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and all $a \in \Delta^{\mathcal{I}}$, we have

$$a \in C^{\mathcal{I}} \text{ iff } \mathcal{I} \models \pi(C)(a).$$

A Description Logic $\mathcal{DL}$ is said to be *less expressive* than a logic $\mathcal{L}$ if there is a translation that translates all $\mathcal{DL}$-concepts into equivalent $\mathcal{L}$ formulae. Such a translation is called *preserving*.

Please note that there are various other ways in which equivalence of formulae and logics being "less expressive than" others could have been defined [Baader, 1996a; Kurtonina and de Rijke, 1997; Areces and de Rijke, 1998]. For example, a less strict definition is the one that only asks the translation to be satisfiability preserving.

To start with, we give a translation $\pi$ that translates $\mathcal{ALC}$-concepts into predicate logic and which will be useful in the remainder of this section. For those familiar with modal logics, please note that this translation parallels the one from propositional modal logic [van Benthem, 1983; 1984]; the close relationship between modal logic and Description Logic will be discussed in Section 4.2.2. For $\mathcal{ALC}$, the translation of concepts into predicate logic formulae can be defined in such a way that the resulting formulae involve only two variables, say $x, y$, and only unary and binary

[1] In the following, we view interpretations both as Description Logic and predicate logic interpretations.

predicates. In the following, $\mathcal{L}^k$ denotes the first order predicate logic over unary and binary predicates with $k$ variables.

The translation is given by two mappings $\pi_x$ and $\pi_y$ from $\mathcal{ALC}$-concepts into $\mathcal{L}^2$ formulae in one free variable. Each concept name $A$ is also viewed as a unary predicate symbol, and each role name $R$ is viewed as a binary predicate symbol. For $\mathcal{ALC}$-concepts, the translation is inductively defined as follows:

$$\begin{array}{rclrcl}
\pi_x(A) & = & A(x), & \pi_y(A) & = & A(y), \\
\pi_x(C \sqcap D) & = & \pi_x(C) \wedge \pi_x(D), & \pi_y(C \sqcap D) & = & \pi_y(C) \wedge \pi_y(D), \\
\pi_x(C \sqcup D) & = & \pi_x(C) \vee \pi_x(D), & \pi_y(C \sqcup D) & = & \pi_y(C) \vee \pi_y(D), \\
\pi_x(\exists R.C) & = & \exists y.R(x,y) \wedge \pi_y(C), & \pi_y(\exists R.C) & = & \exists x.R(y,x) \wedge \pi_x(C), \\
\pi_x(\forall R.C) & = & \forall y.R(x,y) \supset \pi_y(C), & \pi_y(\forall R.C) & = & \forall x.R(y,x) \supset \pi_x(C).
\end{array}$$

Other concept and role constructors that can easily be translated into first order predicate logic without involving more than two variables are inverse roles, conjunction, disjunction, and negation on roles, and one-of[1].

If a Description Logic allows for number restrictions $\geqslant n\, R$, $\leqslant n\, R$, the translation either involves *counting quantifiers* $\exists^{\geq n}$, $\exists^{\leq n}$ (and still involves only two variables) or equality (and involves an unbounded number of variables):

$$\begin{array}{rclcl}
\pi_x(\geqslant n\, R) & = & \exists^{\geq n} y.R(x,y) & = & \exists y_1,\ldots,y_n.\, \bigwedge_{i \neq j} y_i \neq y_j \wedge \bigwedge_i R(x,y_i) \\
\pi_x(\leqslant n\, R) & = & \exists^{\leq n} y.R(x,y) & = & \forall y_1,\ldots,y_{n+1}.\, \bigwedge_{i \neq j} y_i \neq y_j \supset \bigvee_i \neg R(x,y_i)
\end{array}$$

For qualified number restrictions, the translations can easily be modified with the same consequence on the number of variables involved.

So far, all Description Logics were less expressive than first order predicate logic (possibly with equality or counting quantifiers). In contrast, the expressive power of a Description Logic including the transitive closure of roles goes beyond first order logic: First, it is easy to see that expressing transitivity $(\rho^+(x,y) \wedge \rho^+(y,z)) \supset \rho^+(x,z)$ involves at least three variables. To express that a relation $\rho^+$ is the transitive closure of $\rho$, we first need to enforce that $\rho^+$ is a transitive relation including $\rho$—which can easily be axiomatized in first order predicate logic. Secondly, we must enforce that $\rho^+$ is *the smallest* transitive relation including $\rho$—which, as a consequence of the Compactness Theorem, cannot be expressed in first order logic.

**Internalisation of Knowledge Bases:** So far, we were concerned with preserving translations of concepts into logical formulae, and thus could reduce satisfiability of concepts to satisfiability of formulae in the target logic. In Description Logics, however, we are also concerned with concept consistency and logical implication w.r.t. a TBox, and with ABox consistency w.r.t. a TBox.

Furthermore, TBoxes differ in whether they are restricted to be acyclic, allow for

[1] In this case, the translation is to $\mathcal{L}^2$ with constants.

cyclic definitions, or allow for general concept inclusion axioms (see Chapter 2 for details). In first order logic, the equivalent to a TBox assertion is simply a universally quantified formula, and thus it is not necessary to make the above mentioned distinction between, for example, pure concept satisfiability and satisfiability with respect to a TBox—provided that cyclic TBoxes are read with descriptive semantics [Baader, 1990a; Nebel, 1991] (cyclic TBoxes read with least or greatest fixpoint semantics go beyond the expressive power of first order predicate logic). In the following, we consider only the most expressive form of TBoxes, namely those allowing for general concept inclusion axioms. Given a preserving translation $\pi$ from Description Logic concepts into first order formulae and a TBox $\mathcal{T} = \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n\}$, we define

$$\pi(\mathcal{T}) = \forall x. \bigwedge_{i=1}^{n} (\pi_x(C_i) \supset \pi_x(D_i)).$$

Then it is easy to show that

- a concept $C$ is satisfiable with respect to $\mathcal{T}$ iff the formula $\pi_x(C) \wedge \pi(\mathcal{T})$ is satisfiable.
- a concept $C$ is subsumed by a concept $D$ with respect to $\mathcal{T}$ iff the formula $\pi_x(C) \wedge \neg\pi_x(D) \wedge \pi(\mathcal{T})$ is unsatisfiable.
- given two index sets $I$, $J$, an ABox $\{R_k(a_i, a_j) \mid \langle i,j,k\rangle \in I\} \cup \{C_j(a_i) \mid \langle i,j\rangle \in J\}$ is consistent with $\mathcal{T}$ iff the formula

  $$\bigwedge_{\langle i,j,k\rangle \in I} R_k(a_i, a_j) \wedge \bigwedge_{\langle i,j\rangle \in J} \pi_x(C_j)(a_i) \wedge \pi(\mathcal{T})$$

  is satisfiable, where the $a_i$-s in the formula are constants corresponding to the individuals in the ABox.

Observe that, if all concepts in a TBox $\mathcal{T}$ can be translated to $\mathcal{L}^2$ (resp. $\mathcal{C}^2$), then the translation $\pi(\mathcal{T})$ of $\mathcal{T}$ is also a formula of $\mathcal{L}^2$ (resp. $\mathcal{C}^2$).

Hence in first order logic, reasoning with respect to a knowledge base (consisting of a TBox and possibly an ABox) is not more complex than reasoning about concept expressions alone—in contrast to the complexity of reasoning for most Description Logics, where considering even acyclic TBoxes can make a considerable difference (for example, see [Calvanese, 1996b; Lutz, 1999a]). This gap is not surprising since first order predicate logic is far more complex than most Description Logics, namely undecidable.

In the following, we investigate logics that are more closely related to Description Logics, namely restricted variable fragments, modal logics, and the guarded fragment.

### 4.2.1 Restricted variable fragments

One possibility to define decidable fragments of first-order logic is to restrict the set of variables which are allowed inside formulae and the arity of relation symbols. As mentioned in the previous section, we use $\mathcal{L}^k$ to denote first order predicate logic over unary and binary predicates with at most $k$ variables. Analogously, $\mathcal{C}^k$ denotes first order predicate logic over unary and binary predicates with at most $k$ variables and counting quantifiers $\exists^{\geq n}$, $\exists^{\leq n}$.

With the exception of the Description Logics introduced by Calvanese *et al.* [1998a] and Lutz *et al.* [1999], the translation of Description Logic concepts into predicate logic formulae involves predicates of arity at most 2.

From the translations in the previous section, it follows immediately that

- $\mathcal{ALCR}$ is less expressive than $\mathcal{L}^2$ and that
- $\mathcal{ALCNR}$ is less expressive than $\mathcal{C}^2$.

As we have shown above, general TBox assertions can be translated into $\mathcal{L}^2$ formulae. These facts together with the linearity of the translation yields upper bounds for the complexity of $\mathcal{ALCR}$ and $\mathcal{ALCNR}$ (even though these bounds are far from being tight): $\mathcal{L}^2$ and $\mathcal{C}^2$ are known to be NExpTime-complete [Grädel *et al.*, 1997a; Pacholski *et al.*, 2000] (for $\mathcal{C}^2$, this is true only if numbers in counting quantifiers are assumed to be coded in unary, an assumption often made in Description Logics), hence satisfiability and subsumption with respect to a (possibly cyclic) TBox are in NExpTime for $\mathcal{ALCR}$ and $\mathcal{ALCNR}$.

However, both $\mathcal{L}^2$ and $\mathcal{C}^2$ are far more expressive than $\mathcal{ALCR}$ and $\mathcal{ALCNR}$, respectively. For example, both logics allow for the negation of binary predicates, i.e, subformulae of the form $\neg R(x, y)$. In Description Logics, this corresponds to negation of roles, an operator that is rarely considered in Description Logics, except in the weakened form of difference[1] [De Giacomo, 1995; Calvanese *et al.*, 1998a] (Exceptions are the work by Mameide and Montero [1993] and Lutz and Sattler [2000b], which deal with genuine negation of roles). Moreover, $\mathcal{L}^2$ and $\mathcal{C}^2$ allow for "global" quantification, i.e., for formulae of the form $\exists x.\Phi(x)$ or $\forall x.\Psi(x)$ that talk about the whole interpretation domain. In contrast, quantification in Description Logics is, in general, "local", e.g., concepts of the form $\forall R.C$ only constrain all $R$-successors of an individual.

Borgida [1996] presents a variety of results stating that a certain Description Logic is less than or as expressive as a certain fragment of first order logic. We mention only the most important ones:

- $\mathcal{ALC}$ extended with

[1] Difference of roles is easier to deal with than genuine negation, since it does not destroy "locality" of quantification.

**(role constructors)** full Boolean operators on roles, inverse roles, cross-product of two concepts, an identity role *id*, and

**(concept constructors)** individuals ("one-of"),

is as expressive as $\mathcal{L}^2$ (and therefore decidable and, more precisely, NExpTime-complete).

- A further extension of this logic with all sorts of role-value-maps is as expressive as $\mathcal{L}^3$ (and therefore undecidable).

Since both extensions include full Boolean operators on roles, they can simulate a universal role using the complex role $R \sqcup \neg R$, and thus general TBox assertions can be internalised (see Chapter 5). Thus, for these two extensions, reasoning with respect to (possibly cyclic) TBoxes can be reduced to pure concept reasoning—i.e., the TBox can be internalized—and the above complexity results include both sorts of reasoning problems.

Later, a second Description Logic was presented that is as expressive as $\mathcal{L}^2$ [Lutz *et al.*, 2001a]. In contrast to the logic in [Borgida, 1996], this logic does not allow to build a role as the cross-product of two concepts, and it does not provide individuals. However, using the identity role *id* (with $id^{\mathcal{I}} = \{(x,x) \mid x \in \Delta^{\mathcal{I}}\}$ for all interpretations $\mathcal{I}$), we can guarantee that (the atomic concept) $N$ is interpreted as an individual, i.e., a singleton set, using the following TBox axiom:

$$\top \sqsubseteq \exists (R \sqcup \neg R).(N \sqcap \forall \neg id.\neg N)$$

### 4.2.2 Modal logics

Modal logics and Description Logics have a very close relationship, which was first described in [Schild, 1991]. In a nutshell, [Schild, 1991] points out that $\mathcal{ALC}$ can be seen as a notational variant of the multi modal logic $\mathbf{K_m}$. Later, a similar relationship was observed between more expressive modal logics and Description Logics [De Giacomo and Lenzerini, 1994a; Schild, 1994], namely between (extensions of) Propositional Dynamic Logic PDL and (extensions of) $\mathcal{ALC}_{reg}$, i.e., $\mathcal{ALC}$ extended with regular roles. Following and exploiting these observations, various (complexity) results for Description Logics were found by translating results from modal or propositional dynamic logics and the $\mu$-calculus to Description Logics [De Giacomo and Lenzerini, 1994a; 1994b; Schild, 1994; De Giacomo, 1995]. Moreover, upper bounds for the complexity of satisfiability problems were tightened considerably, mostly in parallel with the development of decision procedures suitable for implementations and optimisation techniques for these procedures [De Giacomo and Lenzerini, 1995; De Giacomo, 1995; Horrocks *et al.*, 1999]. In the following, we will describe the relation between modal logics and Description Logics in more detail.

We start by introducing the basic modal logic **K**; for a nice introduction and overview see [Halpern and Moses, 1992; Blackburn *et al.*, 2001]. Given a set of *propositional letters* $p_1, p_2, \ldots$, the set of formulae of the modal logic **K** is the smallest set that

- contains $p_1, p_2, \ldots$,
- is closed under Boolean connectives $\wedge$, $\vee$, and $\neg$, and
- if it contains $\phi$, then it also contains $\Box\phi$ and $\Diamond\phi$.

The semantics of modal formulae is given by so-called *Kripke structures* $M = \langle S, \pi, \mathcal{K} \rangle$, where $S$ is a set of so-called *states* or *worlds* (which correspond to individuals in Description Logics), $\pi$ is a mapping from the set of propositional letters into sets of states (i.e., $\pi(p_i)$ is the set of states in which $p_i$ holds), and $\mathcal{K}$ is a binary relation on the states $S$, the so-called *accessibility relation* (which can be seen as the interpretation of a single role). The semantics is then given as follows, where, for a modal formula $\phi$ and a state $s \in S$, the expression $M, s \models \phi$ is read as "$\phi$ holds in $M$ in state $s$".

$$\begin{array}{lll}
M, s \models p_i & \text{iff} & s \in \pi(p_i) \\
M, s \models \phi_1 \wedge \phi_2 & \text{iff} & M, s \models \phi_1 \text{ and } M, s \models \phi_2 \\
M, s \models \phi_1 \vee \phi_2 & \text{iff} & M, s \models \phi_1 \text{ or } M, s \models \phi_2 \\
M, s \models \neg\phi & \text{iff} & M, s \not\models \phi \\
M, s \models \Diamond\phi & \text{iff} & \text{there exists } s' \in S \text{ with } (s, s') \in \mathcal{K} \text{ and } M, s' \models \phi \\
M, s \models \Box\phi & \text{iff} & \text{for all } s' \in S, \text{ if } (s, s') \in \mathcal{K}, \text{ then } M, s' \models \phi
\end{array}$$

In contrast to many other modal logics, **K** does not impose any restrictions on the Kripke structures. For example, the modal logic **S4** is obtained from **K** by restricting the Kripke structures to those where the accessibility relation $\mathcal{K}$ is reflexive and transitive. Other modal logics restrict $\mathcal{K}$ to be symmetric, well-founded, an equivalence relation, etc. Moreover, the number of accessibility relations may be different from one. Then we are talking about *multi modal logics*, where each accessibility relation $\mathcal{K}_i$ can be thought to correspond to one *agent*, and is quantified using the multi modal operators $\Box_i$ and $\Diamond_i$ (or, alternatively $[i]$ and $\langle i \rangle$). For example, $\mathbf{K_m}$ stands for the multi modal logic **K** with $m$ agents.

To establish the correspondence between the modal logic $\mathbf{K_m}$ and the Description Logic $\mathcal{ALC}$, Schild [1991] gave the following translation $f$ from $\mathcal{ALC}$-concepts using role names $R_1, \ldots, R_m$ to $\mathbf{K_m}$:

$$\begin{array}{rcl}
f(A) & = & A, \\
f(C \sqcap D) & = & f(C) \wedge f(D), \\
f(C \sqcup D) & = & f(C) \vee f(D), \\
f(\neg(C)) & = & \neg(f(C)),
\end{array}$$

$$\begin{aligned} f(\forall R_i.C) &= \Box_i(f(C)), \\ f(\exists R_i.C) &= \Diamond_i(f(C)). \end{aligned}$$

Now, Kripke structures can easily be viewed as Description Logic interpretations and vice versa. Then, from the semantics of $\mathbf{K_m}$ and $\mathcal{ALC}$, it follows immediately that $a$ is an instance of an $\mathcal{ALC}$-concept $C$ in an interpretation $\mathcal{I}$ iff its translation $f(C)$ holds in $a$ in the Kripke structure corresponding to $\mathcal{I}$. Obviously, we can define an analogous translation from $\mathbf{K_m}$ formulae into $\mathcal{ALC}$.

There exists a large variety of modal logics for a variety of applications. In the following, we will sketch some of them together with their relation to Description Logics.

**Propositional Dynamic Logics** are designed for reasoning about the behaviour of programs. *Propositional Dynamic Logic* (PDL) was introduced by Fischer and Ladner [1979], and proven to have an ExpTime-complete satisfiability problem by Fischer and Ladner [1979] and Pratt [1979]; for an overview, see [Harel *et al.*, 2000]. PDL was designed to describe the (dynamic) behaviour of programs: complex programs can be built starting from atomic programs by using non-deterministic choice ($\cup$), composition (;), and iteration ($\cdot^*$). PDL formulae can be used to describe the properties that should hold in a state after the execution of a complex program. For example, the following PDL formula holds in a state if the following condition is satisfied: whenever program $\alpha$ or $\beta$ is executed, a state is reached where $p$ holds, and there is a sequence of alternating executions of $\alpha$ and $\beta$ such that a state is reached where $\neg p \wedge q$ holds:

$$[\alpha \cup \beta]p \wedge \langle(\alpha;\beta)^*\rangle(\neg p \wedge q)$$

Its Description Logic counterpart, $\mathcal{ALC}_{reg}$, was introduced independently by Baader [1991]. $\mathcal{ALC}_{reg}$ is the extension of $\mathcal{ALC}$ with regular expressions over roles[1] and can be seen as a notational variant of Propositional Dynamic Logic. For this correspondence, see the work by Schild [1991] and De Giacomo and Lenzerini [1994a], and Chapter 5.. There exist a variety of extensions of PDL (or $\mathcal{ALC}_{reg}$), for example with inverse roles, counting, or difference of roles, most of which still have an ExpTime satisfiability problem; see, e.g., [Kozen and Tiuryn, 1990; De Giacomo, 1995; De Giacomo and Lenzerini, 1996] and Chapter 5.

**The $\mu$-Calculus** can be viewed as a generalisation of dynamic logic, with similar applications, and was introduced by Pratt [1981] and Kozen [1983]. It is obtained from multi modal $\mathbf{K_m}$ by allowing for (least and greated) fixpoint operators to be

[1] Regular expressions over roles are built using union ($\sqcup$), composition ($\circ$), and the Kleene operator ($\cdot^*$) on roles and can be used in $\mathcal{ALC}_{reg}$-concepts in the place of atomic roles (see Chapter 5).

used on propositional letters. For example, for $\mu$ the least fixpoint operator and $X$ a variable for propositional letters, the formula $\mu X.p \vee \langle\alpha\rangle X$ describes the states with a (possibly empty) chain of $\alpha$ edges into a state in which $p$ holds. In PDL, this formula is written $\langle\alpha^*\rangle p$, and its $\mathcal{ALC}_{reg}$ counterpart is $\exists R_\alpha^*.p$. However, the $\mu$-calculus is strictly more expressive than PDL or $\mathcal{ALC}_{reg}$: for example, the $\mu$-calculus can express *well-foundedness* of a program (binary relation), i.e., there is a $\mu$-calculus formula that has only models in which $\alpha$ is interpreted as a well-founded relation (that is, a relation without any infinite chains). In [De Giacomo and Lenzerini, 1994b; 1997; Calvanese *et al.*, 1999c], this additional expressive power is shown to be useful in a variety of Description Logics applications. The Description Logic counterpart of the $\mu$-calculus extended with number restrictions [De Giacomo and Lenzerini, 1994b; 1997] and additionally with inverse roles [Calvanese *et al.*, 1999c] is proven to have an ExpTime-complete satisfiability problem.

There are two other classes of Description Logics with other forms of fixpoints: in Description Logics, fixpoints first came in through (1) the transitive closure operator [Baader, 1991], which is naturally defined using a least fixpoint, and (2) through terminological cycles [Baader, 1990a], which have a different meaning according to whether a greatest, least, or arbitrary fixpoint semantics is employed [Nebel, 1991; Baader, 1996b; Küsters, 1998].

**Temporal Logics** are designed for reasoning about time-dependent information. They have applications in databases, automated verification of programs, hardware, and distributed systems, natural language processing, planning, etc. and come in various different shapes; for a survey of temporal logics, see, e.g., [Gabbay *et al.*, 1994]. Firstly, they can differ in whether the basic temporal entities are time *points* or time *intervals*. Secondly, they differ in whether they are based on a linear or on a branching temporal structure. In the latter structures, the flow of time might "branch" into various succeeding future times. Finally, they differ in the underlying logic (e.g., Boolean logic or first order predicate logic) and in the operators provided to speak about the past and the future (e.g., operators that refer to the next time point, to all future time points, to a future time point and all its respective future time points, etc.).

In contrast to some other modal logics, temporal logics do not have very close Description Logic relatives. However, they are mentioned here because they are used to "temporalise" Description Logics; for a survey on temporal Description Logics, see [Artale and Franconi, 2001] and Chapter 6. When speaking of "the temporalisation" of a logic, e.g., $\mathcal{ALC}$, one usually refers to a logic with two-dimensional interpretations. One dimension refers to the flow of time, and each state in this flow of time comprises an interpretation of the underlying logic, e.g., an $\mathcal{ALC}$ interpretation. Obviously, the logic obtained depends on the temporal logic chosen for the temporal dimension and on the underlying (description) logic. Moreover, one has the choice to require that the interpretation domain of each time point is the same for all states ("constant domain assumption") or that it is a subset of the domains of the interpretations underlying future states. Examples of temporalised Description Logics can be found in [Wolter and Zakharyaschev, 1999d; Sturm and Wolter, 2002; Artale *et al.*, 2001; Schild, 1993; Lutz *et al.*, 2001b]. An alternative to this temporalisation is to extend a Description Logic with a temporal concrete domain [Baader and Hanschke, 1991a]. This yields a "two-sorted" interpretation domain, consisting of abstract individuals on the one hand and time points or intervals on the other hand. Abstract individuals are then related to the temporal structure using features (functional roles) and the standard concrete domain constructs. An example of such a logic is described by Lutz [2001a].

**Hybrid Logics** extend standard modal logics with the the possibility to refer to single states (individuals in the interpretation domain) using so-called *nominals* (see, e.g., [Blackburn and Seligman, 1995; Areces *et al.*, 2000; Areces, 2000] for hybrid logics related to Description Logics). Nominals are simply special propositional variables which hold in exactly one state. Hybrid logics enjoy a variety of "nice" properties whose description goes beyond the scope of this article; for a summary, see [Areces, 2000]. In Description Logics, there are three standard ways to refer to individuals: (1) we can use ABox individuals in ABoxes, (2) we can use the "one-of" concept constructor $\{o_1, \ldots, o_k\}$ which can be applied to individual names $o_i$ and which is present in only a few Description Logics (e.g., in the Description Logic described in [Bresciani *et al.*, 1995]), and (3) we can use nominals in a similar way as in hybrid logics (e.g., [De Giacomo, 1995; Tobies, 2000; Horrocks and Sattler, 2001]), namely as special atomic concepts that are interpreted as singleton sets. For most Description Logics, there is a direct mapping between nominals and the "one-of" constructor and back: let $o_i$ stand for individual names and, at the same time, nominals. Then we can extend the translation $f$ mentioned above to the "one-of" constructor as follows—provided that we make the *unique name assumption* (cf. Chapter 2) either for both the individual names and the nominals or for none of them:

$$f(\{o_1, \ldots, o_k\}) = f(\{o_1\} \sqcup \ldots \sqcup \{o_k\}) = o_1 \vee \ldots \vee o_k$$

ABox individuals can be viewed as a restricted form of nominals, and each ABox in a Description Logic $\mathcal{L}$ can be translated into a single concept of (the extension of) $\mathcal{L}$ with conjunction, existential restriction, and "one-of": first, translate each assertion of the form

$$\begin{array}{ll} C(a) & \text{into } \{a\} \sqcap C \text{ and} \\ R(a,b) & \text{into } \{a\} \sqcap \exists R.\{b\} \end{array}$$

Next, for $C_1, \ldots, C_m$ the resulting concepts of this translation and $U$ a role name not occurring in any $C_i$, define $C = \sqcap_{1 \leq i \leq m} \exists U.C_i$. Then each model of $C$ is a model of the original ABox—provided, again, that the unique name assumption holds either for both individual names and nominals or for none. Vice versa, each model of the original ABox can easily be extended to a model of $C$.

So far, we only mentioned the weakest way in which nominals occur in hybrid logics. The next stronger form are formulae of the form $\varphi @ o_i$ which describes, intuitively, that $\varphi$ holds in the state $o_i$. For $U$ a universal role and $C_\varphi$ the translation of $\varphi$, this formula corresponds to the concept $\exists U.(o_i \sqcap C_\varphi)$. Finally, we only point out that there are even more expressive ways of talking about nominals in hybrid logics using, for example, variables for nominals and quantification over them.

So far for the relation between certain modal logics and certain Description Logics. In the remainder of this section, the relationship between standard Description Logics constructors and their counterpart in modal logics are discussed.

**Number Restrictions:** In modal logics, the equivalent to qualified number restrictions $\geqslant n\, R.C$ and $\leqslant n\, R.C$ [Hollunder and Baader, 1991b] is known as *graded modalities* [Fine, 1972; Van der Hoek and de Rijke, 1995], whereas no equivalent to the standard, weaker form of number restrictions, $\geqslant n\, R$ and $\leqslant n\, R$, has been considered explicitly.

Number restrictions can be said to play a central role in Description Logics: they are present in almost all knowledge representation systems based on Description Logics, several variants have been investigated with respect to their computational complexity (e.g., see [Tobies, 1999c] for qualified number restrictions, [Baader and Sattler, 1999] for symbolic number restrictions and number restrictions on complex roles), and it was proved by De Giacomo and Lenzerini [1994a] that reasoning with respect to (possibly cyclic) TBoxes for the Description Logic equivalent to *converse*-PDL extended with qualified number restrictions (on atomic and inverse atomic roles) is EXPTIME-complete.

In contrast, they play a minor role in modal and dynamic logics. A more prominent role in dynamic logics is played by *deterministic* programs, i.e., programs that are to be interpreted as *functional* relations (cf. Chapter 2). Ben-Ari *et al.* [1982] and Parikh [1981] show that validity (and hence satisfiability) of DPDL (i.e., the logic that is obtained from PDL by restricting programs to be deterministic) is EXPTIME-complete. Moreover, Parikh [1981] has shown that PDL formulae can be linearly translated into DPDL formulae, and this translation was used by De Giacomo and Lenzerini [1994a] to code qualified number restrictions into DPDL formulae. As a consequence, we have that satisfiability and subsumption with respect to (possibly cyclic) TBoxes in $\mathcal{ALC}$ extended with regular expressions over roles and qualified number restrictions is in ExpTime.

**Transitivity:** In modal logics and Description Logics, transitivity comes in (at least) two different shapes, as transitive roles (or frames whose accessibility relation is transitive, like in $\mathbf{K4_m}$) and as the transitive closure operator on roles (or the Kleene star operator on programs in PDL). Interestingly, these two sorts of transitivity differ in their complexity.

Fischer and Ladner [1979] prove that satisfiability in PDL is ExpTime-complete. However, the only operator on programs (or roles) used in the hardness proof is the transitive closure operator. Translated to Description Logics, this yields ExpTime-completeness of satisfiability in $\mathcal{ALC}$ extended with the transitive closure operator on roles.

In contrast, $\mathbf{K4_m}$ is known to be of the same complexity as $\mathbf{K_m}$ (or $\mathcal{ALC}$), namely PSpace-complete [Halpern and Moses, 1992], while providing transitivity: $\mathbf{K4_m}$ is obtained from $\mathbf{K_m}$ by restricting Kripke structures to those where the accessibility relations are transitive. Translated into Description Logics, this means that concept satisfiability in $\mathcal{ALC}$ extended with transitive roles (i.e., the possibility to say that certain roles are interpreted as transitive relations) is in PSpace [Sattler, 1996]. An extension of this Description Logic with role hierarchies was implemented in the Description Logic system FaCT [Horrocks, 1998a]. Although pure concept satisfiability of this extension is ExpTime-hard, its highly optimised implementation behaves quite well [Horrocks, 1998b].

**Inverse Roles:** Without the converse operator on programs/time (or the inverse operator on roles), binary relations are restricted to be used asymmetrically: For example, one is restricted to either model “into the future” or “into the past”, or one must decide whether to use a role “has-child” or “is-child-of”, but may not use both and relate them in the proper way. Hence in both modal and Description Logics, the converse/inverse operator plays an important role since it overcomes this asymmetry, and a variety of logics allowing for this operator were investigated [Streett, 1982; Vardi, 1985; De Giacomo and Massacci, 1996; Calvanese, 1996a; De Giacomo, 1996; Horrocks *et al.*, 1999].

### 4.2.3 Guarded fragments

Andréka *et al.* [1996] introduce guarded fragments as natural generalisations of modal logics to relations of arbitrary arity. Their definition and investigation was motivated by the question why modal logics have such “nice” properties, e.g., finite axiomatisability, Craig interpolation, and decidability. Guarded fragments are obtained from first order logic by allowing the use of quantified variables only if these variables are *guarded* by appropriate atoms[1] before they are used in the body of a formula. More precisely, quantifiers are restricted to appear only in the form

$$\begin{array}{llll} \exists \mathbf{y}(P(\mathbf{x},\mathbf{y}) \wedge \Phi(\mathbf{y})) & \text{or} & \forall \mathbf{y}(P(\mathbf{x},\mathbf{y}) \supset \Phi(\mathbf{y})) & \text{(First Guarded Fragment)} \\ \exists \mathbf{y}(P(\mathbf{x},\mathbf{y}) \wedge \Phi(\mathbf{x},\mathbf{y})) & \text{or} & \forall \mathbf{y}(P(\mathbf{x},\mathbf{y}) \supset \Phi(\mathbf{x},\mathbf{y})) & \text{(Guarded Fragment)} \end{array}$$

for atoms $P$, vectors of variables $\mathbf{x}$ and $\mathbf{y}$, and (first) guarded fragment formulae $\Phi$ with free variables in $\mathbf{y}$ and $\mathbf{x}$ (resp. in $\mathbf{y}$). The *loosely* guarded fragment further allows for a restricted form of conjunction as guards.

Obviously, the translation $(\exists y.R(x,y) \wedge \varphi(y))(x)$ of the $\mathbf{K}$ formula $\Diamond\varphi$ (or of the $\mathcal{ALC}$ concept $\exists R.C_\varphi$) is a formula in the first guarded fragment since the quantified variable $y$ is "guarded" by $R$. A more complex guarded fragment formula is

$$\exists z_1, z_2.(\mathsf{parents}(x, z_1, z_2) \wedge (\mathsf{married}(z_1, z_2) \wedge (\forall y.\mathsf{parents}(y, z_1, z_2) \supset \mathsf{rich}(z_1))))$$

in one free variable $x$, a guard atom $\mathsf{parents}$, and describing all those persons that have married parents and whose siblings (including herself) are rich.

All guarded fragments were shown to be decidable [Andréka *et al.*, 1996]. Grädel [1999] proves that satisfiability of the guarded fragment is in ExpTime—provided that the arity of the predicates is bounded—and 2ExpTime-complete for unbounded signatures. Interestingly, the guarded fragment was shown to remain 2ExpTime when extended with fixpoints [Grädel and Walukiewicz, 1999]. These "nice" properties together with their close relationship to modal/description logics suggest that they are a good starting point for the development of a Description Logic with $n$-ary predicates [Grädel, 1998]: in [Lutz *et al.*, 1999], a restriction of the guarded fragment was proven to be PSpace-complete, where the restriction concerns the way in which variables are used in guard atoms. Roughly spoken, each predicate $A$ comes with a two-fold arity $(i, j)$ and, when $A$ is used as a guard, either all first $i$ variables are quantified and none of the last $j$ are or, symmetrically, all last $j$ variables are quantified and none of the first $i$ are. Hence one might think of the predicates as having two-fold "groupings". A similar logic, the so-called action-guarded fragment AGF is proposed in [Gonçalvès and Grädel, 2000]: it comes with a similar grouping of variables in predicates (which is, when extended with "inverse actions", the same as the grouping in [Lutz *et al.*, 1999]) and, additionally, it divides predicates into those allowed as guards and those allowed in the body of formulae. From a Description Logic perspective, this should not be too severe a restriction since it parallels the distinction between role and concept names. Interestingly, the extension of AGF with counting quantifiers (the first order counterpart of number restrictions), inverse actions, and fixpoints yields an ExpTime logic—provided that

[1] Atoms are formulae $P(x_1, \ldots, x_k)$ where $P$ is a $k$-ary predicate symbol and $x_i$ are variables.

the arity of the predicates is bounded and that numbers in counting quantifiers are coded unarily [Gonçalvès and Grädel, 2000]. This result is even more interesting when noting that the guarded fragment, when extended with number restrictions, functional restrictions, *or* transitivity (i.e., statements saying that certain binary relations are to be interpreted as transitive relations) becomes undecidable [Grädel, 1999].

To the best of our knowledge, the only other $n$-ary Description Logics with sound and complete inference algorithms are $\mathcal{DLR}$ [Calvanese *et al.*, 1998a] and $\mathcal{DLR}_\mu$ [Calvanese *et al.*, 1999c], which seem to be orthogonal to the guarded fragment. An exact description of the relationship between $\mathcal{DLR}$ (resp. $\mathcal{DLR}_\mu$) and the guarded fragment (resp. its extension with fixpoints) is missing so far.

## 4.3 Database models

In this section we will describe the relationship between Description Logics and data models used in databases. We will consider both traditional data models used in the conceptual modeling of an application domain, such as semantic and object-oriented data models, and more recently introduced formalisms for representing semistructured data and data on the web. We will concentrate on the relationship between the formalisms and refer to Chapter 16 for a more detailed discussion on the use of Description Logics in data management [Borgida, 1995].

### 4.3.1 Semantic data models

Semantic data models were introduced primarily as formalisms for database schema design [Abrial, 1974; Chen, 1976], and are currently adopted in most of the database and information system design methodologies and Computer Aided Software Engineering (CASE) tools [Hull and King, 1987; Batini *et al.*, 1992]. In semantic data models, classes provide an explicit representation of objects with their attributes and the relationships to other objects, and subtype/supertype relationships are used to specify the inheritance of properties. Here, we concentrate on the *Entity-Relationship* (ER) model [Chen, 1976; Teorey, 1989; Batini *et al.*, 1992; Thalheim, 1993], which is one of the most widespread semantic data models. However, the considerations we make hold also for other formalisms for conceptual modeling, such as UML class diagrams [Rumbaugh *et al.*, 1998; Jacobson *et al.*, 1998]

#### *4.3.1.1 Formalization*

The basic elements of the ER model are entities, relationships, and attributes, which are used to model the domain of interest by means of an *ER schema*.

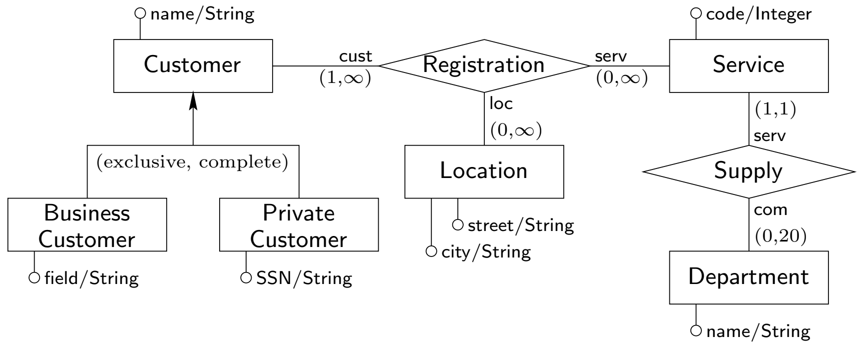

Fig. 4.8. An Entity-Relationship schema.

Figure 4.8 shows a simple ER schema representing the registration of customers for (telephone) services provided by departments (e.g., of a telephone company). The schema is drawn using the standard graphical ER notation, in which entities are represented as boxes, and relationships as diamonds. An attribute is shown as a circle attached to the entity for which it is defined. An *entity type* (or simply *entity*) denotes a set of objects, called its *instances*, with common properties. Elementary properties are modeled through *attributes*, whose values belong to one of several predefined *domains*, such as Integer, String, Boolean, etc. Relationships between instances of different entities are modeled through *relationship types* (or simply *relationships*). A relationship denotes a set of tuples, each one representing an association among a combination of instances of the entities that participate in the relationship. The participation of an entity in a relationship is called an *ER-role* and has a unique name. It is depicted by connecting the relationship to the participating entity. The number of ER-roles for a relationship is called its *arity*.

*Cardinality constraints* can be attached to an ER-role in order to restrict the minimum or maximum number of times an instance of an entity may participate via that ER-role in instances of the relationship [Abrial, 1974; Grant and Minker, 1984; Lenzerini and Nobili, 1990; Ferg, 1991; Ye *et al.*, 1994; Thalheim, 1992; Calvanese and Lenzerini, 1994b]. Minimal and maximal cardinality constraints can be arbitrary non-negative integers. However, typical values for minimal cardinality constraints are 0, denoting no constraint, and 1, denoting mandatory participation of the entity in the relationship; typical values for maximal cardinality constraints are 1, denoting functionality, and $\infty$, denoting no constraint. In Figure 4.8, cardinality constraints are used to impose that each customer must be registered for at least one service. Also, each service is provided by exactly one department, which in turn may not provide more than 20 different services.

To represent inclusions between the sets of instances of two entities or two relationships, so called *IS-A* relations are used. An IS-A relation states the inheritance of properties from a more general entity (resp. relationship) to a more specific one. A *generalization* is a set of IS-A relations which share the more general entity (resp. relationship). Multiple generalizations can be combined in a *generalization hierarchy.* A generalisation can be *mutually exclusive*, meaning that all the specific entities (resp. relationships) are mutually disjoint, or *complete*, meaning that the union of the more specific entities (resp. relationships) completely covers the more general entity (resp. relationship). In Figure 4.8, a mutually exclusive and complete generalisation is used to represent the fact that customers are partitioned into private and business customers.

Additionally, *keys* are used to represent the fact that an instance of an entity is uniquely identified by a certain set of attributes, or that an instance of a relationship is uniquely identified by a set of instances of the entities participating in the relationship.

Although we do not provide a formal definition here, the semantics of an ER schema can be given by specifying which database states are consistent with the information structure represented by the schema; for details see e.g., [Calvanese *et al.*, 1999e].

Traditionally, the ER model has been used in the design phase of commercial applications, and modern CASE tools usually provide sophisticated schema editing facilities and automatic generation of code for the interaction with the database management system. However, these tools do not provide any support for dealing with the complexity of schemata that goes beyond the graphical user interface. In particular, the designer is responsible for checking schemata for important properties such as consistency and redundancy. This may be a complex and time consuming task if performed by hand. By translating an ER schema into a Description Logic knowledge base in such a way that the verification of schema properties corresponds to traditional Description Logic reasoning tasks, the reasoning facilities of a Description Logic system can be profitably exploited to support conceptual database design.

### *4.3.1.2 Correspondence with Description Logics*

Both in Description Logics and in the ER model, the domain of interest is modeled through classes and relationships, and various proposals have been made for establishing a correspondence between the two formalisms. Bergamaschi and Sartori [1992] provide a translation of ER schemas into acyclic $\mathcal{ALN}$ knowledge bases. However, due to the limited expressiveness of the target language, several features of the ER model and desired reasoning tasks could not fully be captured by the proposed translation. Indeed, when relating the ER model to Description Logics, one has to take into account the following aspects:

$$
\begin{aligned}
\mathsf{Registration} &\sqsubseteq \forall \mathsf{custRegistration}.\mathsf{Customer} \sqcap {=1}\,\mathsf{custRegistration} \sqcap \\
&\quad\ \forall \mathsf{locRegistration}.\mathsf{Location} \sqcap {=1}\,\mathsf{locRegistration} \sqcap \\
&\quad\ \forall \mathsf{servRegistration}.\mathsf{Service} \sqcap {=1}\,\mathsf{servRegistration} \\
\mathsf{Supply} &\sqsubseteq \forall \mathsf{servSupply}.\mathsf{Service} \sqcap {=1}\,\mathsf{servSupply} \sqcap \\
&\quad\ \forall \mathsf{comSupply}.\mathsf{Customer} \sqcap {=1}\,\mathsf{comSupply} \\
\mathsf{Customer} &\sqsubseteq \forall \mathsf{custRegistration}^{-}.\mathsf{Registration} \sqcap {\geqslant 1}\,\mathsf{custRegistration}^{-} \\
\mathsf{Location} &\sqsubseteq \forall \mathsf{locRegistration}^{-}.\mathsf{Registration} \\
\mathsf{Service} &\sqsubseteq \forall \mathsf{servRegistration}^{-}.\mathsf{Registration} \sqcap \\
&\quad\ \forall \mathsf{servSupply}^{-}.\mathsf{Supply} \sqcap {=1}\,\mathsf{servSupply}^{-} \\
\mathsf{Department} &\sqsubseteq \forall \mathsf{comSupply}^{-}.\mathsf{Supply} \sqcap {\leqslant 20}\,\mathsf{comSupply}^{-} \\
\mathsf{Customer} &\sqsubseteq \mathsf{BusinessCustomer} \sqcup \mathsf{PrivateCustomer} \\
\mathsf{BusinessCustomer} &\sqsubseteq \mathsf{Customer} \\
\mathsf{PrivateCustomer} &\sqsubseteq \mathsf{Customer} \sqcap \neg\mathsf{BusinessCustomer} \\
\mathsf{Customer} &\sqsubseteq \forall \mathsf{name}.\mathsf{String} \sqcap {=1}\,\mathsf{name}
\end{aligned}
$$

Fig. 4.9. Part of the knowledge base corresponding to the Entity-Relationship schema in Figure 4.8.

(i) The ER model allows for relations of arbitrary arity, while in traditional Description Logics only unary and binary relations are considered.

(ii) The assumption of acyclicity is unrealistic in an ER shema, while it is common in Description Logics knowledge bases.

(iii) Database states are considered to be finite structures, while no assumption on finiteness is usually made on the interpretation domain of a Description Logic knowledge base.

Before discussing these issues in more detail, we show in Figure 4.9 part of the $\mathcal{ALUNI}$ knowledge base corresponding to the ER schema in Figure 4.8, derived according to the translation proposed by Calvanese *et al.* [1994; 1999e]. We have omitted the part corresponding to the translation of most attributes, showing as an example only the translation of the attribute name of the entity Customer.

Due to point (i), when translating ER schemas into knowledge bases of a traditional Description Logic, it becomes necessary to *reify* relationships, i.e., to translate each relationship into a concept whose instances represent the tuples of the relationship. Each entity is translated also into a concept, while each ER-role is translated into a Description Logic role. Then, using functional roles, one can enforce that each instance of the atomic concept $C$ corresponding to a relationship $R$ represents a tuple of $R$, i.e., for each role representing an ER-role of $R$, the instance of $C$ is connected to exactly one instance of the entity associated to the ER-role.

There is, however, one condition, which is implicit in the semantics of the ER model, but which does not necessarily hold once relationships are reified, and which can also not be enforced in Description Logics on the models of a knowledge base: The condition is that the extension of a relationship $R$ does not contain some tuple twice. After reification this corresponds to the fact that there are no two instances of the concept corresponding to $R$ that are connected through all roles of $R$ exactly to the same instances of the entities associated to the roles. However, it can be shown that, when reasoning on a knowledge base corresponding to an ER schema, nothing is lost by ignoring this condition. Indeed, given an arbitrary model of such a knowledge base, one can always find a model in which the condition holds, and thus one that corresponds directly to a legal database state [Calvanese *et al.*, 1994; De Giacomo, 1995; Calvanese *et al.*, 1999e].

Cardinality constraints are translated using number restrictions on the inverse of the roles connecting relationships to entities. To avoid the need for qualified number restrictions, in the translation in Figure 4.9 we have disambiguated the roles by appending to their name the name of the relationship they belong to. An alternative would be to allow the same role to appear in several places, and use qualified number restrictions instead of unqualified ones. While considerably complicating the language, this makes it possible to translate also IS-A relations between relationships, which cannot be captured using the translation proposed by Calvanese *et al.* [1999e]. Also more general forms of cardinality constraints have been proposed for the ER model [Thalheim, 1992], allowing e.g., to limit the number of locations a customer may be registered for, independently of the service. To the best of our knowledge, such types of cardinality constraints cannot be captured in Description Logics in general. Borgida and Weddell [1997] have studied reasoning in Description Logics in the presence of functional dependencies that are more general than unary ones, and which allow one to represent keys of relations. Decidability of reasoning in a very expressive Description Logic augmented with non-unary key constraints has been shown by Calvanese *et al.* [2000b], and Calvanese *et al.* [2001a] have shown that also general functional dependencies can be added without losing ExpTime-completeness.

IS-A relations are simply translated using concept inclusion assertions. Generalisation hierarchies additionally require negation, if they are mutually disjoint, and union, if they are complete.

With respect to point (ii), we observe that the translation of an ER schema containing cycles obviously gives rise to a cyclic Description Logic knowledge base. However, due to the necessity of properly relating a relationship via an ER-role to an entity, even when translating an acyclic ER schema, the resulting knowledge base contains cycles. On the other hand, it is sufficient to use inclusion assertions rather than equivalence, since the former naturally correspond to the semantics of ER schemata.

With respect to point (iii), we observe that one cannot simply ignore it and adopt algorithms that reason with respect to arbitary models. Indeed, the ER model itself does not have the *finite model property* [Cosmadakis *et al.*, 1990; Calvanese and Lenzerini, 1994b], which states that, if a knowledge base (resp. schema) has an arbitrary, possibly infinite model (resp. database state), then it also has a finite one (see also Chapter 5 for more details). A further confirmation comes from the fact that, for correctly capturing ER schemas in Description Logics, possibly cyclic knowledge bases expressed in a Description Logic including functional restrictions and inverse roles are required, and such knowledge bases do not have the finite model property [Calvanese *et al.*, 1994; 1999e]. Therefore one must resort to techniques for finite model reasoning. Calvanese *et al.* [1994] show that reasoning w.r.t. finite models in $\mathcal{ALUNI}$ knowledge bases containing only inclusion assertions is ExpTime-complete, and Calvanese [1996a] presents a 2ExpTime algorithm for reasoning in $\mathcal{ALCQI}$ knowledge bases with general inclusion assertions.

### *4.3.1.3 Applications of the correspondence*

The study of the correspondence between Description Logics and semantic data models has led to significant advantages in both fields. On the one hand, the richness of constructs that is typical of Description Logics makes it possible to add them to semantic data models and take them fully into account when reasoning on a schema [Calvanese *et al.*, 1998g]. Notable examples are:

- the ability to specify not only IS-A and generalisation hierarchies, but also arbitrary Boolean combinations of entities or relationships, which can correspond to forms of negative and incomplete knowledge [Di Battista and Lenzerini, 1993];
- the ability to refine properties along an IS-A hierarchy, such as restricting the numeric range for cardinality constraints, or refining the participation in relationships using universal quantification over roles;
- the ability to define classes by means of equality assertions, and not only to state necessary properties for them.

The correspondence between semantic data models and Description Logics has been recently exploited to add such advanced capabilities to CASE tools. A notable example is the i•com tool [Franconi and Ng, 2000] for conceptual modeling, which combines a user-friendly graphical interface with the ability to automatically infer properties of a schema (e.g., inconsistency of a class, or implicit IS-A relations) by invoking the FaCT Description Logic reasoner [Horrocks, 1998a; 1999].

On the other hand, the basic ideas behind the translation of semantic data models into Description Logics, namely reification and the fact that one can restrict the attention to models in which distinct instances of a reified relation correspond to distinct tuples, have led to the development of Description Logics in which relations of arbitrary arity are first class citizens [De Giacomo and Lenzerini, 1994c; Calvanese *et al.*, 1997; 1998a]. Using such Description Logics, the translation of an ER schema is immediate, since now also relationships of arbitrary arity have their direct counterpart. For example, using $\mathcal{DLR}$ [Calvanese *et al.*, 1998a], the part of the schema in Figure 4.8 relative to the ternary relation Registration can be translated as follows:

$$\begin{aligned}
\mathsf{Registration} &\sqsubseteq (\$1\colon \mathsf{Customer}) \sqcap (\$2\colon \mathsf{Location}) \sqcap (\$3\colon \mathsf{Service}) \\
\mathsf{Customer} &\sqsubseteq \exists[\$1]\mathsf{Registration}
\end{aligned}$$

We refer to Chapter 16, Section 16.2.2 for the details of the translation.

Description Logics could also be considered as expressive variants of semantic data models with incorporated reasoning facilities. This is of particular importance in the context of information integration, where a high expressiveness is required to capture in the best possible way the complex relationships that hold between data in different information sources [Levy *et al.*, 1995; Calvanese *et al.*, 1998d; 1998e].

### 4.3.2 Object-oriented data models

Object-oriented data models have been proposed recently with the goal of devising database formalisms that could be integrated with object-oriented programming systems [Abiteboul and Kanellakis, 1989; Kim, 1990; Cattell and Barry, 1997; Rumbaugh *et al.*, 1998]. Object-oriented data models rely on the notion of *object identifier* at the extensional level (as opposed to traditional data models which are value-oriented) and on the notion of *class* at the intensional level. The structure of the classes is specified by means of *typing* and *inheritance*. Since we aim at discussing the relationship with Description Logics, which are well suited to describe structural rather than dynamic properties, we restrict our attention to the structural component of object-oriented models. Hence we do not consider all those aspects that are related to the specification of the behaviour and evolution of objects, which nevertheless constitute an important part of these data models. Although in our discussion we do not refer to any specific formalism, the model we use is inspired by the one presented by Abiteboul and Kanellakis [1989], and embodies the basic features of the static part of the ODMG standard [Cattell and Barry, 1997]

```
class Customer type-is
  union BusinessCustomer, PrivateCustomer
  end

class PrivateCustomer is-a Customer type-is
  record
    SSN: String
  end

class Service type-is
  record
    code: Integer,
    suppliedBy: Department
  end

class Registration type-is
  record
    cust: Customer,
    regis: set-of record
                    serv: Service
                    loc: Location
                  end
  end
```

Fig. 4.10. An object-oriented schema.

### *4.3.2.1 Formalization*

An *object-oriented schema* is a finite set of class declarations, which impose constraints on the instances of the classes that are used to model the application domain. A *class declaration* for a class $C$ has the form

$$\underline{\text{class}}\ C\ \underline{\text{is-a}}\ C_1, \ldots, C_k\ \underline{\text{type-is}}\ T,$$

where the is-a part, which is optional, specifies inclusions between the sets of instances of the involved classes, while the type-is part specifies through the *type expression* $T$ the structure assigned to the objects that are instances of the class. We consider *union*, *set*, and *record types*, built according to the following syntax, where the letter $A$ is used to denote *attributes*:

$$\begin{array}{rcl} T & \longrightarrow & C \ \mid \\ & & \underline{\text{union}}\ T_1, \ldots, T_k\ \underline{\text{end}} \ \mid \\ & & \underline{\text{set-of}}\ T \ \mid \\ & & \underline{\text{record}}\ A_1{:}\,T_1, \ldots, A_k{:}\,T_k\ \underline{\text{end}}. \end{array}$$

Figure 4.10 shows part of an object-oriented schema modeling the same reality as the Entity-Relationship schema of Figure 4.8. Notice that now registrations are represented as a class and grouped according to the customer, since all registrations related to one customer are collected in the set-valued attribute regis.

The meaning of an object-oriented schema is given by specifying the characteristics of a database state for the schema. The definition of a *database state* makes use of the notions of *object identifier* and *value*. Starting from a finite set $\mathcal{O}^{\mathcal{J}}$ of object identifiers, the set of complex values over $\mathcal{O}^{\mathcal{J}}$ is built inductively by grouping values into finite sets and records. A *database state* $\mathcal{J}$ for a schema is constituted by the set of object identifiers, a mapping $\pi^{\mathcal{J}}$ assigning to each class a subset of $\mathcal{O}^{\mathcal{J}}$, and a mapping $\rho^{\mathcal{J}}$ assigning to each object in $\mathcal{O}^{\mathcal{J}}$ a value over $\mathcal{O}^{\mathcal{J}}$.

Notice that, although the set of values that can be constructed from a set $\mathcal{O}^{\mathcal{J}}$ of object identifiers is infinite, for a database state one only needs to consider the finite subset $\mathcal{V}_{\mathcal{J}}$ of values assigned by $\rho^{\mathcal{J}}$ to the elements of $\mathcal{O}^{\mathcal{J}}$, including the values that are not explicitly associated with object identifiers, but are used to form other values.

The interpretation of type expressions in a database state $\mathcal{J}$ is defined through an *interpretation function* $\cdot^{\mathcal{J}}$ that assigns to each type expression $T$ a set $T^{\mathcal{J}}$ of values in $\mathcal{V}_{\mathcal{J}}$ as follows:

- if $T$ is a class $C$, then $T^{\mathcal{J}} = \pi^{\mathcal{J}}(C)$;
- if $T$ is a union type $\underline{\text{union}}\ T_1, \ldots, T_k\ \underline{\text{end}}$, then $T^{\mathcal{J}} = T_1^{\mathcal{J}} \cup \cdots \cup T_k^{\mathcal{J}}$;
- it $T$ is a record type (resp. set type), then $T^{\mathcal{J}}$ is the set of record values (resp. set values) compatible with the structure of $T$. For records we are using an open semantics, meaning that the records that are instances of a record type may have more components than those explicitly specified in the type [Abiteboul and Kanellakis, 1989].

A database state $\mathcal{J}$ for an object-oriented schema $\mathcal{S}$ is said to be *legal* (with respect to $\mathcal{S}$) if for each declaration

$$\underline{\text{class}}\ C\ \underline{\text{is-a}}\ C_1, \ldots, C_n\ \underline{\text{type-is}}\ T$$

in $\mathcal{S}$, it holds that (1) $C^{\mathcal{J}} \subseteq C_i^{\mathcal{J}}$ for each $i \in \{1, \ldots, n\}$, and (2) $\rho^{\mathcal{J}}(C^{\mathcal{J}}) \subseteq T^{\mathcal{J}}$. Therefore, for a legal database state, the type expressions that are present in the schema determine the (finite) set of values that must be considered. The construction of such values is limited by the depth of type expressions.

#### *4.3.2.2 Correspondence with Description Logics*

When establishing a correspondence between an object-oriented model as the one presented above, and Description Logics, one must take into account that the interpretation domain for a Description Logic knowledge base consists of atomic objects, whereas each object of an object-oriented schema is assigned a possibly structured value. Therefore one needs to explicitly represent in Description Logics the type structure of classes [Calvanese *et al.*, 1994; 1999e; Artale *et al.*, 1996a]. We describe now the translation proposed by Calvanese *et al.* [1994; 1999e], that overcomes this difficulty by introducing in the Description Logic knowledge base concepts and roles with a specific meaning: the concepts AbstractClass, RecType, and SetType are used to denote instances of classes, record values, and set values, respectively. The associations between classes and types induced by the class declarations, as well as the basic characteristics of types, are modeled by means of specific roles: the functional role $\mathsf{value}$ models the association between classes and types, and the role $\mathsf{member}$ is used for specifying the type of the elements of a set. Moreover, the concepts representing types are assumed to be mutually disjoint, and disjoint from the concepts representing classes. These constraints are expressed by the following inclusion assertions, which are always part of the knowledge base that is obtained from an object-oriented schema:

$$\begin{aligned} \mathsf{AbstractClass} &\sqsubseteq\ (= 1\,\mathsf{value}) \\ \mathsf{RecType} &\sqsubseteq\ \forall\mathsf{value}.\bot \\ \mathsf{SetType} &\sqsubseteq\ \forall\mathsf{value}.\bot \sqcap \neg\mathsf{RecType} \end{aligned}$$

The translation from object-oriented schemas to Description Logic knowledge bases is defined through a mapping $\Gamma$, which maps each type expression to a concept expression as follows:

- Each class $C$ is mapped to an atomic concept $\Gamma(C)$.
- Each type expression $\underline{\text{union}}\ T_1, \ldots, T_k\ \underline{\text{end}}$ is mapped to $\Gamma(T_1) \sqcup \cdots \sqcup \Gamma(T_k)$.
- Each type expression $\underline{\text{set-of}}\ T$ is mapped to $\mathsf{SetType} \sqcap \forall\mathsf{member}.\Gamma(T)$.
- Each attribute $A$ is mapped to an atomic role $\Gamma(A)$, and each type expression $\underline{\text{record}}\ A_1 \colon T_1, \ldots, A_k \colon T_k\ \underline{\text{end}}$ is mapped to

$$\begin{aligned} \mathsf{RecType} \sqcap\ & \forall\Gamma(A_1).\Gamma(T_1) \sqcap (= 1\,\Gamma(A_1)) \sqcap \cdots \sqcap \\ & \forall\Gamma(A_k).\Gamma(T_k) \sqcap (= 1\,\Gamma(A_k)). \end{aligned}$$

Then, the knowledge base $\Gamma(\mathcal{S})$ corresponding to an object-oriented schema $\mathcal{S}$ is obtained by taking for each class declaration

$$\underline{\text{class}}\ C\ \underline{\text{is-a}}\ C_1, \ldots, C_n\ \underline{\text{type-is}}\ T$$

an inclusion assertion

$$\Gamma(C)\ \sqsubseteq\ \mathsf{AbstractClass} \sqcap \Gamma(C_1) \sqcap \cdots \sqcap \Gamma(C_n) \sqcap \forall\mathsf{value}.\Gamma(T).$$

We show in Figure 4.11 the knowledge base resulting from the translation of the fragment of object-oriented schema shown in Figure 4.10.

Analogously to the ER model, it is sufficient to use inclusion assertions instead of equivalence assertions to capture the semantics of object-oriented schemas. A translation to an acyclic knowledge base is possible under the assumption that no class in the schema refers to itself, either directly in its type or indirectly via the class declarations[1] [Artale *et al.*, 1996a]. However, since this assumption represents a rather strong limitation in expressiveness, cycles are typically present in object-oriented schemas, and in this case the resulting Description Logic knowledge base

[1] Note that cyclic references cannot appear directly in a type, which is constructed inductively, but only through the class declarations.

$$
\begin{array}{rcl}
\mathsf{Customer} & \sqsubseteq & \mathsf{AbstractClass} \sqcap \forall \mathsf{value}.(\mathsf{BusinessCustomer} \sqcup \mathsf{PrivateCustomer}) \\
\mathsf{PrivateCustomer} & \sqsubseteq & \mathsf{AbstractClass} \sqcap \mathsf{Customer} \sqcap \forall \mathsf{value}.(\mathsf{RecType} \sqcap (= 1\, \mathsf{SSN}) \sqcap \forall \mathsf{SSN}.\mathsf{String}) \\
\mathsf{Service} & \sqsubseteq & \mathsf{AbstractClass} \sqcap \\
& & \forall \mathsf{value}.(\mathsf{RecType} \sqcap (= 1\, \mathsf{code}) \sqcap \forall \mathsf{code}.\mathsf{Integer} \sqcap \\
& & \qquad (= 1\, \mathsf{suppliedBy}) \sqcap \forall \mathsf{suppliedBy}.\mathsf{Department}) \\
\mathsf{Customer} & \sqsubseteq & \mathsf{AbstractClass} \sqcap \\
& & \forall \mathsf{value}.(\mathsf{RecType} \sqcap (= 1\, \mathsf{cust}) \sqcap \forall \mathsf{cust}.\mathsf{Customer} \sqcap \\
& & \qquad (= 1\, \mathsf{regis}) \sqcap \forall \mathsf{regis}.(\mathsf{SetType} \sqcap \\
& & \qquad\qquad \forall \mathsf{member}.(\mathsf{RecType} \sqcap \\
& & \qquad\qquad\qquad (= 1\, \mathsf{serv}) \sqcap \forall \mathsf{serv}.\mathsf{Service} \sqcap \\
& & \qquad\qquad\qquad (= 1\, \mathsf{loc}) \sqcap \forall \mathsf{loc}.\mathsf{Location})))
\end{array}
$$

Fig. 4.11. The specific part of the knowledge base corresponding to the object-oriented schema in Figure 4.10.

will contain cyclic assertions. No inverse roles are needed for the translation, since in object-oriented models the inverse of an attribute is rarely considered. Furthermore, the use of number restrictions is limited to functionality, since all attributes are implicitly functional.

To establish the correctness of the transformation, and thus ensure that the reasoning tasks on an object-oriented schema can be reduced to reasoning tasks on its translation in Description Logics, we would like to establish a one-to-one correspondence between database states legal for the schema and models of the knowledge base resulting from the translation. However, as for the ER model, the knowledge base may have models that do not correspond directly to legal database states. In this case, this is due to the fact that, while values have a treelike structure, the corresponding individuals in a model of the Description Logic knowledge base may be part of cyclic substructures. One way of ruling out such cyclic substructures would be to adopt a specific constructor that allows one to impose well-foundedness [Calvanese *et al.*, 1995], or even exploit general fixed points on concepts [Schild, 1994; De Giacomo and Lenzerini, 1994a; 1997; Calvanese *et al.*, 1999c]. However, it turns out that, in this case, it is not necessary to explicitly enforce such a condition. Indeed, due to the finite depth of nesting of types in a schema, it can be shown that each model of the translation of the schema can be unfolded into one that directly corresponds to a legal database state (more details are provided by Calvanese *et al.* [1999e]).

#### *4.3.2.3 Applications of the correspondence*

Similarly to the ER model, the existence of property-preserving transformations from object-oriented schemas into Description Logic knowledge bases makes it possible to exploit the reasoning capabilities of a Description Logic system for checking relevant schema properties, such as consistency and redundancy [Bergamaschi and Nebel, 1994; Artale *et al.*, 1996a; Calvanese *et al.*, 1998g]. Additionally, several extensions of the object-oriented formalism that are useful for the purpose of conceptual modeling can be considered:

- Not only IS-A, but also disjointness, and, more generally, Boolean combinations of classes can be used.
- Class definitions can be used to specify not only necessary but also necessary and sufficient properties for an object to be an instance of a class [Bergamaschi and Nebel, 1994].
- Cardinality constraints and not only implicit functionality can be imposed on attributes. Having attributes with multiple values could in some cases be a useful alternative to set-valued attributes.
- By admitting also the use of inverse roles in the language, one gains the ability to impose constraints using a relation in both directions, as it is customary in semantic data models. The increase in expressiveness that one obtains this way has indeed been recognized as extremely important by the database community [Albano *et al.*, 1991], and has been included in the recent ODMG standard [Cattell and Barry, 1997].

The basic characteristics of object-oriented data models have also been included in the structural part of the Unified Modeling Language (UML) [Rumbaugh *et al.*, 1998; Jacobson *et al.*, 1998], which is becoming the standard language for the analysis phase of software and information system development. Additionally, UML allows for the definition of generic recursive data structures (both inductive and co-inductive) such as lists and trees, and for their specialisation to specific types. In order to capture also these aspects of UML in Description Logics and take them fully into account when reasoning over a schema, the Description Logic must provide the ability to represent and reason over data structures. In particular, to represent UML schemas, it is necesary to resort to very expressive Description Logics including number restrictions, inverse roles or $n$-ary relations, and fixed point constructs on concepts [Calvanese *et al.*, 1999c]. Also in this case, the reasoning services provided by a Description Logic system can be integrated in CASE tools and profitably exploited to support the designer in the analysis phase [Franconi and Ng, 2000].

### 4.3.3 Semistructured data models and XML

In recent application areas such as data integration, access to data on the web, and digital libraries, the structure of the data is usually not rigid, as in conventional databases, and thus it is difficult to describe it using traditional data models. Therefore, so called *semistructured data models* have been proposed, which are graph-based data models that provide flexible structuring mechanisms, and thus allow one to represent data that is neither raw nor strictly typed [Abiteboul *et al.*, 2000; Abiteboul, 1997; Buneman *et al.*, 1997; Mendelzon *et al.*, 1997]. The *Extensible Markup Language* (XML) [Bray *et al.*, 1998; Abiteboul *et al.*, 2000], which has been introduced as a mechanism for representing structured documents on the web, can in fact also be considered a model for semistructured data. Indeed, XML is by now the way most popular model for data on the Web, and there is a tremendous effort related to XML and the associated standards[1], both in the research community and in industry.

Description Logics have traditionally been used to describe and organize data in a more flexible way than what is done in databases, basically using graph-like structures. Hence it seems natural to adopt Description Logics and the associated reasoning services also for representing and reasoning on semistructured data and XML. In the following, we discuss the (rather few) proposals made in the literature. What these proposals have in common is the necessity to resort to fixpoints, either by adopting fixpoint semantics [Nebel, 1991; Baader, 1991], or by using reflexive transitive closure or explicit fixpoint constructs [De Giacomo and Lenzerini, 1997] (cf. also Chapter 5).

For the recent extensive work on the use of Description Logics to provide a semantically richer representation of data on the web we refer to Chapter 14.

#### *4.3.3.1 Relationship between semistructured data and Description Logics*

Michaeli *et al.* [1997] propose to extend a semistructured data model that is an abstraction of the OEM model [Abiteboul *et al.*, 1997] with a layer of classes, representing objects with common properties. Class expressions correspond to Description Logic concepts and the properties for the classes are specified by a set of *classification rules*, which provide sufficient conditions for class membership and are interpreted under a least fixpoint semantics. By a reduction to reasoning in a Description Logic with fixpoint operators [De Giacomo and Lenzerini, 1997; Calvanese *et al.*, 1999c], it is shown that determining class satisfiability and containment under a set of rules is ExpTime-decidable (and in fact ExpTime-complete).

In the following, we discuss in more detail the use of Description Logics to represent and reason on semistructured data, on the example of one typical representative for semistructured data models. In semistructured data models, data is organized in form of a graph, and information on both the values and the schema for the data are attached to the edges of the graph. In the formalism proposed by Buneman *et al.* [1997], the labels of edges in a schema are formulae of a complete first order theory, and the *conformance* of a database to a schema is defined in terms of a special relation, called *simulation*. The notion of simulation is less rigid than the usual notion

[1] `http://www.w3.org/`

of satisfaction, and suitably reflects the need for dealing with less strict structures of data. In order to capture in Description Logics the notion of simulation, it is necessary on the one hand to express the local conditions that a node must satisfy, and on the other hand to deal with the fact that the simulation relation is the greatest relation satisfying the local conditions. Since semistructured data schemas may contain cycles, the local conditions may depend on each other in a cyclic way. Therefore, while the local conditions can be encoded by means of suitable inclusion assertions in $\mathcal{ALU}$, the maximality condition on the simulation relation can only be captured correctly by resorting to a greatest fixed point semantics [Calvanese *et al.*, 1998c; 1998b]. Then, using a Description Logic with fixed point constructs, such as $\mu\mathcal{ALCQ}$ [De Giacomo and Lenzerini, 1994b; 1997] (see also Chapter 5), a so-called *characteristic concept* for a semistructured data schema can be constructed, which captures exactly the properties of the schema. Subsumption between two schemas, which is the task of deciding whether every semistructured database conforming to one schema also conforms to another schema [Buneman *et al.*, 1997], can be decided by checking subsumption between the characteristic concepts of the schemas [Calvanese *et al.*, 1998c].

The correspondence with Description Logics can again be exploited to enrich semistructured data models, without losing the ability to check schema subsumption. Indeed, the requirement already raised by Buneman *et al.* [1997], to extend semistructured data models with several types of constraints, has been addressed by Calvanese *et al.* [1998b], who propose several types of constraints, such as existence and cardinality constraints, which are naturally derived from Description Logic constructs. Reasoning in the presence of constrains is done by encoding also the constraints in the characteristic concept of a schema. Calvanese *et al.* deal also with the presence of incomplete information in the theory describing the properties of edge labels, by proposing the use of a theory expressed in $\mu\mathcal{ALCQ}$, instead of a complete first order theory.

### *4.3.3.2 Relationship between XML and Description Logics*

XML [Bray *et al.*, 1998] is a formalism for representing documents that are structured by means of nested tags. Recently, XML has gained popularity also as a formalism for representing (semistructured) data and exchanging it over the Web. Figure 4.12 shows two example XML documents containing respectively data about customers and their registration to services provided by various departments (e.g., of a telephone company). A part of an XML document consisting of a *start tag* (e.g., `<Customer>`), the matching *end tag* (e.g., `</Customer>`), and everything in between is called an *element*. Elements can be arbitrarily nested, and can have associated *attributes*, specified by means of attribute-value pairs inside the start tag (e.g., `type="business"`). Intuitively, each XML document can be viewed as a finite

```
<?xml version="1.0"?>
<!DOCTYPE Customers SYSTEM "services.dtd">

<Customers>
  <Customer type="business">
    <Name>FIAT</Name>
    <Field>manufacturing</Field>
    <Registered service="522">
       <Location><City>Torino</City>
                 <Address>...</Address>
       </Location>
       <Location>...</Location>
    </Registered>
    <Registered service="612">
       <Location>...</Location>
    </Registered>
  </Customer>

  <Customer type="private">
    <Name>...</Name>
    <SSN>...</SSN>
    <Registered service="214">
       <Location>...</Location>
    </Registered>
  </Customer>
  ...
</Customers>
```

```
<?xml version="1.0"?>
<!DOCTYPE Services SYSTEM "services.dtd">

<Services>
  <Department name="standard-services">
    <Service code="522">
       <Name>call-back when busy</Name>
       <Cost>...</Cost>
       ...
    </Service>
    <Service code="214">
       <Name>three-party call</Name>
    </Service>
  </Department>

  <Department name="business-services">
    <Service code="612">
      <Name>conference call</Name>
    </Service>
    ...
  </Department>
</Services>
```

Fig. 4.12. Two XML documents specifying respectively customers and services.

ordered unranked tree[1], where each element represents a node, and the children of an element are those elements directly contained in it. How XML documents are viewed as trees is defined, together with an API for accessing and manipulating such trees/XML-documents, by the *Document Object Model*[2], which defines, besides element nodes, also other types of nodes, such as attributes, comments, etc.

In XML, it is possible to impose a structure on documents by means of a *Document Type Declaration* (DTD) [Bray *et al.*, 1998]. A DTD consists of a set of declarations: For each *element type* used in the XML document, the DTD must contain a declaration that specifies, by means of a regular expression, how elements can be nested within elements of that type. The keyword `#PCDATA` is used to specify that the *element content* (i.e., the part enclosed by the tags) is free text without nested elements. For each attribute appearing in the XML document, the DTD must contain a declaration specifying the name of the attribute, the type of the elements it is associated to, and additional properties (e.g., the type and whether the attribute is optional or mandatory). Figure 4.13 shows part of the DTD for the XML documents in Figure 4.12. We refer to [Bray *et al.*, 1998] for a precise definition of the syntax and semantics of XML DTDs.

[1] In an *unranked* tree each node can have an arbitrary finite number of child nodes. The tree is *ordered* since the order among children of the same node matters.

[2] `http://www.w3.org/DOM/`

```
<!-- File: services.dtd -->

<!ELEMENT Customers  (Customer)+ >
<!ELEMENT Customer   (Name, (Field|SSN), Registered+) >
<!ELEMENT Registered (Location)+ >
...
<!ELEMENT Services   (Department)+ >
<!ELEMENT Department (Service)* >
<!ELEMENT Service    (Name, Cost?, ...) >
<!ELEMENT Name       #PCDATA >
...

<!ATTLIST Customer   type    (business|private) "private">
<!ATTLIST Registered service IDREF              #REQUIRED>
<!ATTLIST Department name    CDATA              #REQUIRED>
<!ATTLIST Service    code    ID                 #REQUIRED>
...
```

Fig. 4.13. Part of the Document Type Declaration $S$ for the XML documents in Figure 4.12.

We illustrate the method for encoding XML DTDs into Description Logics knowledge bases proposed in [Calvanese *et al.*, 1999d]. For simplicity, we do not consider XML attributes, although they can easily be dealt with by introducing suitable roles. Due to the presence of regular expressions, to encode DTDs in Description Logics, it is necessary to resort to a Description Logic equipped with constructs for building regular expressions over roles (cf. Chapter 5). Notice that the encoding of DTDs into Description Logic knowledge bases must allow for representing unranked trees and at the same time for preserving the order of the children of a node. For example, the DTD in Figure 4.13 enforces that the content of a `Customer` element consists of a `Name` element, followed by (in DTDs, *concatenation* is denoted with "`,`") either a `Field` or an `SSN` element (*alternative* is denoted with "`|`"), followed by an arbitrary number (but at least one) of `Registered` elements (*transitive closure* is denoted with "`+`"). To overcome these difficulties, Calvanese *et al.* [1999d] propose to represent XML documents (i.e., ordered unranked trees) by means of binary trees, and provide an encoding of DTDs in Description Logics that exploits such a representation. Figure 4.14 shows the binary tree corresponding to one of the XML documents in Figure 4.12.

Figure 4.15 shows part of the axioms encoding the DTD in Figure 4.13. The two roles f and r are used to encode binary trees, and such roles are globally functional (axiom (4.1)). Moreover, the *well-founded* construct (cf. Chapter 5) $\mathit{wf}(\mathsf{f} \sqcup \mathsf{r})$ is used to express that there can be no infinite chain of objects, each one connected to the next by means of $\mathsf{f} \sqcup \mathsf{r}$. Such a condition turns out to be necessary to correctly capture the fact that XML documents correspond to trees that are *finite*. For each

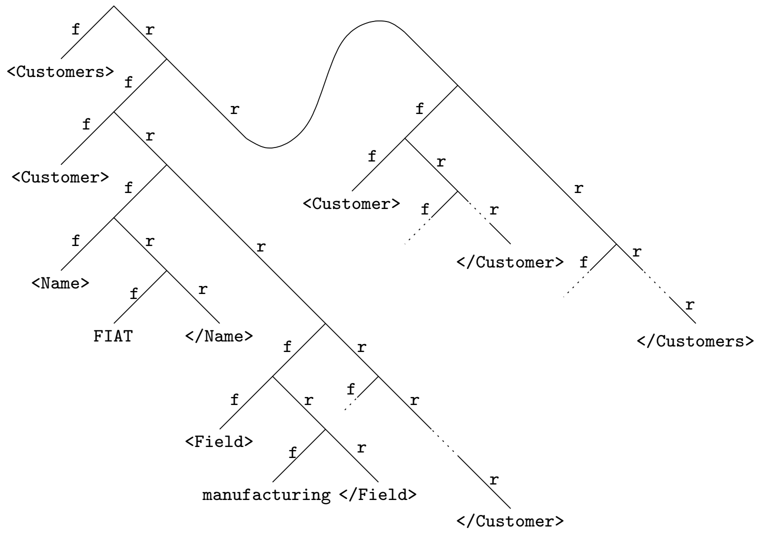

Fig. 4.14. The binary tree corresponding to the XML document on the left hand side of Figure 4.12.

element type $E$, the atomic concepts $\mathsf{Start}E$ and $\mathsf{End}E$ represent respectively the start tags (4.2) and end tags (4.3) for $E$, and such tags are leaves of the tree (4.4). The remaining leaves of the tree are free text, represented by the atomic concept PCDATA (4.5). Using such concepts and roles, one can introduce for each element type $E$ appearing in a DTD $D$ an atomic concept $E_D$, and encode the regular expression specifying the structure of elements of type $E$ in a suitable complex role, exploiting constructs for regular expressions over roles (including the $id(\cdot)$

$$\begin{array}{rcll}
\top & \equiv & \leqslant 1\,\mathsf{f} \sqcap \leqslant 1\,\mathsf{r} \sqcap wf(\mathsf{f} \sqcup \mathsf{r}) & (4.1)\\
\mathsf{Start}E & \sqsubseteq & \mathsf{Tag} \quad \text{for each element type } E & (4.2)\\
\mathsf{End}E & \sqsubseteq & \mathsf{Tag} \quad \text{for each element type } E & (4.3)\\
\mathsf{Tag} & \sqsubseteq & \forall(\mathsf{f} \sqcup \mathsf{r}).\bot & (4.4)\\
\mathsf{PCDATA} & \sqsubseteq & \forall(\mathsf{f} \sqcup \mathsf{r}).\bot \sqcap \neg\mathsf{Tag} & (4.5)\\
\mathsf{Customers}_S & \equiv & \exists\mathsf{f}.\mathsf{StartCustomers} \sqcap \exists(\mathsf{r} \circ (id(\exists\mathsf{f}.\mathsf{Customer}_S) \circ \mathsf{r})^+).\mathsf{EndCustomers} & \\
\mathsf{Customer}_S & \equiv & \exists\mathsf{f}.\mathsf{StartCustomers} \sqcap \exists(\mathsf{r} \circ id(\exists\mathsf{f}.\mathsf{Name}_S) \circ \mathsf{r} & \\
 & & \qquad \circ\, (id(\exists\mathsf{f}.\mathsf{Field}_S) \sqcup id(\exists\mathsf{f}.\mathsf{SSN}_S)) \circ \mathsf{r} & \\
 & & \qquad \circ\, (id(\exists\mathsf{f}.\mathsf{Registered}_S) \circ \mathsf{r})^+).\mathsf{EndCustomer} & \\
\mathsf{Name}_S & \equiv & \exists\mathsf{f}.\mathsf{StartName} \sqcap \exists(\mathsf{r} \circ id(\exists\mathsf{f}.\mathsf{PCDATA}) \circ \mathsf{r}).\mathsf{EndName} & \\
 & \vdots & &
\end{array}$$

Fig. 4.15. Part of the encoding of the DTD $S$ in Figure 4.13 into a Description Logics knowledge base.

construct). This is illustrated in Figure 4.15 for part of the element types of the DTD in Figure 4.13. We refer to [Calvanese *et al.*, 1999d] for the precise definition of the encoding.

The encoding of DTDs into Description Logics can be exploited to verify different kinds of properties on DTDs, namely *inclusion*, *equivalence*, and *disjointness* between the sets of documents conforming respectively to two DTDs. Such reasoning tasks come in different forms. For *strong* inclusion (resp. equivalence, disjointness) both the document structure *and* the actual tag names are of importance when comparing documents, while for *structural* inclusion (resp. equivalence, disjointness) one abstracts away from the actual tag names, and considers only the document structure [Wood, 1995]. *Parametric* inclusion (resp. equivalence, disjointness) generalizes both notions, by considering an equivalence relation between tag names, and comparing documents modulo such an equivalence relation. By exploiting the encoding of DTDs into Description Logics presented above, all forms of inference on DTDs can be carried out in deterministic exponential time [Calvanese *et al.*, 1999d].