Chapter 4: Architecture of the System

We now return to the problem of understanding the constraints on information models, which was introduced in Section 1.2. To recap, the underlying motivation for this work is to understand the *comparative* semantics of the constraints on two models (see Section 1.2.3), as would be needed for such tasks as transferring constraints for schema-to-schema mapping (as explained in Section 1.3.2). This research explores a case-based reasoning (CBR) approach to understanding constraints.

Understanding the constraints on a model is realised by extracting *higher-level constraints (HLCs)* from the model. This calls for some explanation:

- To *extract* an HLC from a model means to create an instance of that HLC, which is linked to the model itself.
- Each HLC represents knowledge about the semantics of a constraint on the model. As mentioned in Chapter 1, this thesis considers a *constraint* on a model to be any condition on the model that must not be violated by instances of the model, no matter whether that condition is stated explicitly in a rule, or is implicit in the structure of the model. The knowledge represented by HLCs is at a higher level of abstraction than the model itself, making explicit the implicit semantics of the constraints.

Having covered the relevant background material - code understanding (of which constraint understanding is a special case) in Chapter 2, and CBR in Chapter 3 - this chapter presents the architecture of a system for extracting HLCs from EXPRESS [34] models, laying the foundations for the rest of the thesis. This chapter gives an overview of the external functionality of the system. Here, we are not concerned with how it works internally (that will be covered in Appendix D).

Section 4.1 outlines the essential functionality which the system provides, and gives a top-level architecture for that. Section 4.2 then shows what the architecture looks like with the CBR approach to understanding constraints (i.e. to extracting HLCs).

4.1 Top-Level Architecture

Essentially, the system for understanding constraints takes as input an EXPRESS model, and outputs that EXPRESS model augmented with HLCs that represent explicit knowledge about the semantics of the constraints on that model.

```
SCHEMA people;
     ENTITY living_person;
          name : STRING;
          age : INTEGER;
     WHERE
          age >= 0;
          age < 130;
     END_ENTITY;
END_SCHEMA;
```

Figure 4-1. A model of people.

For example, suppose the model in Figure 4-1 were input to the constraint-understanding system. The following HLCs (written in English for now) might be extracted:

- A discrete, inclusive, lower bound of zero on `age`;
- A discrete, exclusive, upper bound of one hundred and thirty on `age`;
- A discrete numeric range constraint on `age`, whose bounds are the above two HLCs.

The system extracts the two bound constraints from the `WHERE` clauses, and extracts the range constraint from those bound constraints. To generalise, HLCs can be extracted from other HLCs, as well as from the model itself.

Rather than operating directly on the EXPRESS model in its textual form, the system first parses the EXPRESS model into an internal representation. This internal representation will be described in detail in Section 6.1. For now, it can be understood as a collection of data structures (representing entities, attributes, domain rules and the other syntactic units of EXPRESS) which are linked appropriately (e.g. an entity is linked to its attributes and to its domain rules, and so on).

The internal representation includes not just the syntactic constructs of EXPRESS, but also the HLCs that the system extracts from the model - these are linked to the parts of the model from which they are extracted. The (internal representation of the) model, once the system has extracted HLCs from it, shall be referred to as the *augmented model*.

Figure 4-2 depicts diagrammatically the architecture described above: the boxes represent data (in either external or internal form[1]) and the ovals represent processes.

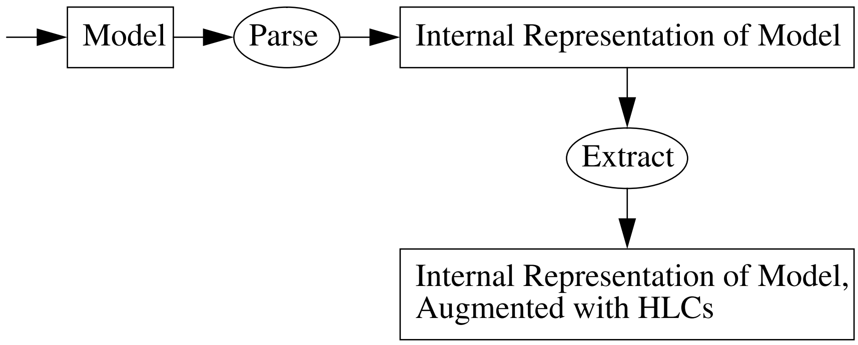

Figure 4-2. Top-level architecture of the constraint-understanding system.

Note that there is no conversion of the augmented model back into textual form, because the whole point of the augmented model is to represent HLCs, linked to the model itself, and HLCs are at a higher level than EXPRESS itself. A system that reasons about HLCs (e.g. to perform transference of constraints for schema-to-schema mapping, as explained in Section 1.3.2) will use the augmented model directly in its internal form.[2]

This section has introduced an architecture for a system to extract HLCs, with no regard to the CBR approach being taken. CBR is used to extract HLCs from the input model, in order

1. *Internal data* are those which are held in the data structures of a running program; *external data* are stored in files.
2. The augmented model can be converted into a suitable form for browsing by humans, however.

to produce the augmented model. Since the CBR is visible externally, this aspect of the architecture is covered in the next section.

4.2 CBR Architecture (for Extraction)

The main impact that the CBR approach to extracting HLCs has on the external architecture of the constraint-understanding system is the presence of a case library, in which the system stores its cases.

Cases suggest which HLCs to extract from models. Each case has two parts:

- The *'problem' part*: a fragment of EXPRESS code (to which fragments of actual models will be matched), or some HLCs (to which HLCs already extracted from actual models will be matched, as in Section 4.1's example of extracting a range constraint from two bound constraints);
- The *'solution' part*: an HLC which characterises the constraint semantics of the 'problem' part at a higher level of abstraction (to extract from actual models when the 'problem' part successfully matches).

The extraction algorithm involves matching fragments of EXPRESS code to the 'problem' parts of cases, and extracting the HLC suggested by the 'solution' part of a case if a code fragment in the model does match the case.

The CBR process involves retrieving cases from the case library to which the input model will be matched. Fragments of the input model that are known to be potential sources of HLCs are used as retrieval prompts. The system navigates the input model, submitting the appropriate fragments as retrieval prompts to the case library, which results in a set of cases being retrieved. Each fragment is matched to the 'problem' part of each of the cases retrieved for that fragment. Each case which matches successfully is used to extract an HLC (the one suggested by the 'solution' part of the case) from the input model. As the HLC from the ‘solution’ part is being instantiated for the input model, it is adapted using a form of reinstantiation which replaces references and variables from the case with references and constants from the input model (as will be further discussed in Section 6.3).

Understanding the constraints on information models is an unusual task for CBR in that instead of having one big problem to solve, it has to navigate the problem situation (i.e. the input model), extracting whatever HLCs it is reminded of by local parts of the model. As a consequence, the CBR method proposed in this thesis has some unorthodox characteristics. In particular, it is not so much a matter of finding the best case to solve a problem as finding all cases that are applicable to the model (assuming we wish to extract all the HLCs that the system knows how to extract from the given model), although it may be necessary to select between cases that are competing to extract the same HLC from a particular fragment of the model (namely cases that are variants of one another).

In the spirit of CBR, each case is supposed to represent an *experience* of extracting an HLC. However, not all cases have to represent the system’s own experience. Indeed, if the system started with no experience, it would not be able to extract any HLCs in the first place to gain experience! Therefore, it is necessary to *seed* the case library with an initial set of cases to cover a range of HLCs and the situations (i.e. the cases’ ‘problem’ parts) in which it is appropriate to extract those HLCs. These *seed cases* represent human experience rather than machine experience, but such experience is very valuable to the system nevertheless.

So, the system could acquire cases in two ways: by being seeded with hand-coded cases provided by a human, and by storing cases that it derived itself using adaptation (i.e. storing an HLC-extracting episode as a case in memory). In fact, the current implementation of the system only uses seed cases - it does not retain its own experiences as cases.

Figure 4-3 shows the architecture implied by the discussion in this section, which is an extension of the architecture depicted in Figure 4-2 to show the CBR components.[3]

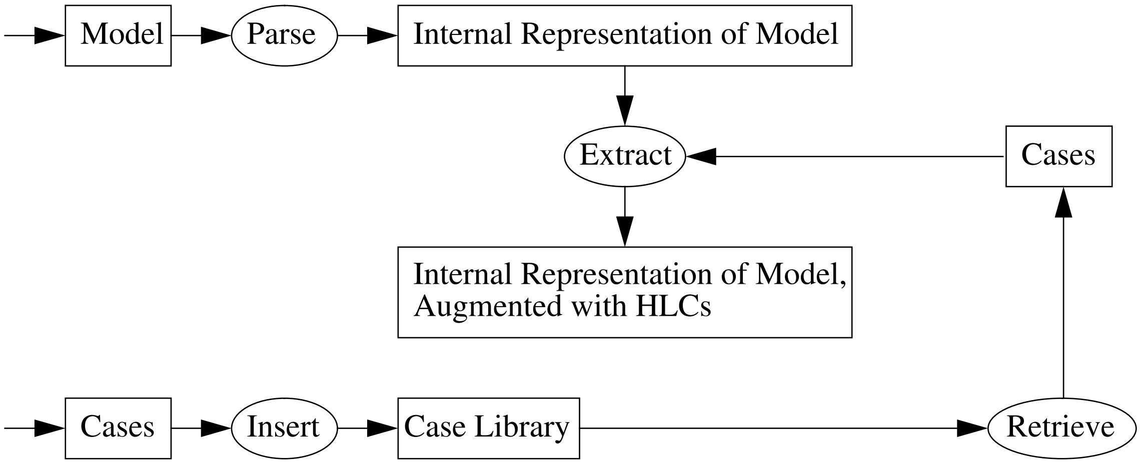

Figure 4-3. Architecture of the constraint-understanding system, with its case library.

As shown in Figure 4-3, the system has two entry points:

- There is the main entry point to the constraint-understanding system, which is used for extracting HLCs from a given input model;
- There is also another entry point, which is used for inserting seed cases into the case library.

This chapter has introduced an architecture for understanding the constraints on EXPRESS models by extracting HLCs. It has described the external functionality of the system with no regard to the representation of the data structures or the implementation of the algorithms. EXPRESS models, HLCs and cases are represented internally using a uniform frame-based representation. Chapter 5 will introduce the frame system (a substratum of the constraint-understanding system), and Chapter 6 will then show how EXPRESS models, HLCs and cases are represented as frames. Chapter 6 will also discuss how cases are used to extract HLCs from EXPRESS models.

3. Note that the arrows in Figure 4-3 do not have clear semantics - the diagram is only meant to provide an informal visualisation of the constraint-understanding system's architecture. The arrows do not necessarily represent the sequence of control flow. In particular, the system does *not* go on to retrieve cases straight after inserting cases into the case library! Cases are retrieved from the case library when the extraction process demands it.