Chapter 7: Results and Conclusions

The time has come to draw this thesis to a close. This chapter starts with a summary of the thesis so far. Some experimental results of running the constraint-understanding system on ‘real’ information models are presented in Section 7.2. Section 7.3 then discusses what the research described in this thesis achieves, and what the major contributions of the thesis are. The potential for future work is discussed in Section 7.4.

7.1 Thesis Summary

This thesis has presented a method for understanding the constraints on information models, by extracting knowledge about the constraints which is at a higher level of abstraction than the models themselves. The motivation for this work was to facilitate reasoning about the constraints on information models, in particular about the comparative semantics of the constraints on two models (see Section 1.2.3).

The particular underlying motivation for this research was to understand the constraints on the source and target models for the task of schema-to-schema mapping, which is to generate automatically a program to map instances from one data repository to another (these repositories conform to the source model and the target model respectively). Schema-to-schema mapping was introduced in Section 1.3, and the potential for future work in this area is discussed in Section G.4.1 of Appendix G.

Understanding the constraints on information models is a special case of code understanding, so much of the relevant work for comparison comes from the field of reverse engineering. The topic of code understanding is reviewed in Chapter 2 (along with its inverse process, code generation).

This thesis proposes a case-based reasoning (CBR) approach to understanding constraints (and to understanding code in general). Chapter 3 provides background on CBR. CBR has many attractive advantages, as discussed in Section 3.4. The main advantage of CBR which justifies its application to code understanding is its ability to solve problems in domains which are poorly understood, or where it is impossible to specify completely all the rules or to formalise the domain as a complete general model. However, such domains usually have lots of cases (experiences of what has worked or happened in the past), which are useful for problem-solving in weak-theory domains. Understanding the constraints on information models at higher levels of abstraction than the models themselves is theory-weak in that it is impossible to completely specify all the higher-level knowledge that could be useful to support particular kinds of understanding (such as the comparative understanding of constraints for schema-to-schema mapping). However, an analysis of worked examples (as in Appendix B) provides experiences of the kinds of higher-level knowledge that would be needed to handle those examples. This knowledge can be encoded as cases - each case suggests what higher-level knowledge needs to be extracted from a given fragment of an information model.

A system for understanding the constraints on information models has been developed by the Author. This *constraint-understanding system* was presented in Chapters 4 to 6. The system applies specifically to models written in the EXPRESS language [34], though it could in principle be generalised to other modelling languages. It has a case library (Appendix C), in which each case associates a fragment of EXPRESS code with a higher-level constraint (HLC) that captures the abstract semantics of that fragment. The system works by matching cases to the input model, and extracting the HLCs suggested by those cases that match, as described in terms of the system's external behaviour in Chapter 4, and algorithmically in Appendix D. The extraction of HLCs from a model constitutes an understanding of the constraints on that model at a higher level of abstraction than the model itself.

Understanding the constraints on information models is an unorthodox CBR problem in that an input model constitutes a series of small, local problems rather than one large, global problem. Case selection is not so much a matter of choosing the best of a set of cases which are competing to solve a problem, but rather a matter of applying every applicable case (but not the inapplicable ones). This means that retrieval needs to be more tightly constrained not to miss any relevant cases than the retrieval algorithms commonly used in CBR (which are discussed in Section 3.2.2). However, a system that uses an understanding of constraints to support a task such as transferring constraints for schema-to-schema mapping (Section 1.3.2), rather than simply understanding constraints for the sake of it, would not need to extract every HLC that can be extracted. It could use heuristics to guide which HLCs to extract (see Appendix G, Section G.3.3).

Some constraints on a model are stated explicitly as rules, while others are implied by the structure of the model. An example of the latter would be an entity having a mandatory attribute `age` of type `INTEGER`, while a domain rule such as `{0 <= age < 130}` would be an example of the former. However, no matter how constraints are expressed as EXPRESS code, they can always have implicit semantics. It is the purpose of HLCs to make those semantics explicit.

Appendix B gives a classification of HLCs, focusing on some of those that could support a comparative understanding of the constraints on two models. A particular kind of HLC (such as an entity being constrained to have exactly one of a set of properties) could be realised in various ways in concrete modelling languages such as EXPRESS (e.g. as an enumeration type, a select type, a `ONEOF` supertype expression, etc.) As mentioned in Section 6.4, the (manual) process of identifying HLCs involves analysing models with similar abstract constraint semantics, which gives rise to HLCs that capture those abstract semantics, and cases that associate them with concrete EXPRESS code fragments. Appendix B is by no means a complete classification of the HLCs that could arise from EXPRESS models, but is intended to give a taste of the sorts of HLCs that would be useful for a comparative understanding of constraints. In any case, there is a potentially infinite number of kinds of HLCs that could be invented, and the way to set about doing so is to analyse what HLCs are needed to understand sample models in terms of the abstract similarities of the constraints that they imply. The process of identifying kinds of HLCs is experiential in nature, and gives rise to a set of cases with which to seed the case library.

An experimental implementation of the constraint-understanding system has been written in Java[1] (Grand [30]), as discussed in Appendix E.

The constraint-understanding system is built on top of a frame-based system, which is described in Chapter 5. EXPRESS models, HLCs and cases are all represented as frames, as described in Chapter 6. As discussed in Section 5.2, EXPRESS itself is used as the language for frame classes (of which the frames themselves are instances). An EXPRESS model defines the frame classes as entities, and their slots as attributes of those entities. The frame class model for the constraint-understanding system (Appendix A) includes an EXPRESS metamodel for frames which represent EXPRESS models (Section A.1). This is primarily intended for the EXPRESS models that are input to the constraint-understanding system for the extraction of HLCs. However, since EXPRESS itself is also used as the frame class language, a free advantage of the EXPRESS metamodel is that frame classes themselves are represented as frames, which could be useful for introspective reasoning in the future.

A textual language is required for humans to write frames in, as when seeding the case library. A frame instance language (FIL) was invented for this purpose, in which instances of entities in EXPRESS models (since EXPRESS is used as the frame class language) can be written. FIL is described in Section 5.3. As well as being a language for humans to write seed cases in, FIL is also a convenient output format for the frames in the computer's memory. FIL code can be augmented with HTML (Musciano and Kennedy [49]) tags, such as hypertext links for frame references, as discussed in Appendix E, Section E.5. This allows humans to peruse frame networks using readily available HTML browsers.

1. "Java" is a trademark of Sun Microsystems, Inc.

Although at first sight the frame system looks like a replication of the record structures typically found in programming languages, it has the major advantage over these of being data-driven. Frame classes only have to be declared to the system at runtime (when it parses the frame class model); they do not have to be hardwired into the program code. Making the knowledge of frame classes explicit in this way is more understandable to humans and easily extensible, since new frame classes (in particular, new kinds of HLCs) can simply be declared in the frame class model without having to modify and recompile the program. It also brings greater flexibility, since the case matching algorithm can be (and has been) written generically, without having to hardwire into the algorithm all the frame classes it could match.

The frame system certainly increases the running time of the constraint-understanding system. However, performance could, in principle, be enhanced by automatically compiling the frame classes to Java classes.[2]

This section has summarised the thesis. The next section presents the results of running the constraint-understanding system on some sample models, and Section 7.3 will draw conclusions about the major contributions of this thesis.

7.2 Experimental Results

The constraint-understanding system that was implemented and presented in this thesis has been tested on a variety of EXPRESS models. Many of these are ‘toy’ models: minimal fragments of EXPRESS, written to test cases individually (as presented throughout the thesis, particularly Appendix B).

However, the constraint-understanding system has also been tested on larger, more ‘realistic’ information models, to see how it handles those. This section briefly presents

2. An example of code generation (c.f. Section 2.1).

some results of running the system on the following three models:

- Course World: a small model (138 lines of EXPRESS code), devised by the Author to test the range of HLCs and cases in the current system (see Appendix F);
- AP212 [35]: a large model for electrotechnical design (6044 lines of EXPRESS code);
- The EDIF 4 0 0 (Electronic Design Interchange Format, Version 4 0 0) Level 0 Information Model (Lau [40]): a very large model (28279 lines of EXPRESS code).

Figure 7-1 tabulates the results of running the above three models through the constraint-understanding system. The table has a row for each HLC currently implemented (as listed in Section B.6 of Appendix B), a column for each model, and each entry in the table is the number of instances of a particular HLC that were extracted from a particular model.

	Course World	AP212	EDIF 4 0 0 Level 0
`conjunction`	8	411	1164
`bound`	2	0	0
`range`	1	0	0
`exactly_one`	1	35	249
`at_most_one`	1	10	76
`non_null`	13	415	911
`zero_or_more`	3	3	926
`equation`	2	241	274
`adder`	1	3	8
`multiplier`	1	3	4

Figure 7-1. Table to show the numbers of higher-level constraints of each kind extracted from three sample models.

So the constraint-understanding system is capable of processing some real models, and extracting HLCs from them according to the cases in its case library. These results are fairly encouraging, although it is difficult to judge how meaningful the HLCs extracted from a model are until an attempt is made to use them in support of a task which uses the constraint-understanding system as a substratum (such as transferring constraints for schema-to-schema mapping, as explained in Section 1.3.2).

In the case of the Course World model, the system extracted just those HLCs that it should have extracted (given the case library in Appendix C), and no others. Appendix F substantiates this claim by including all the frames that were generated in processing this model (as FIL files, output from the system). The Author makes no such claim for the AP212 and EDIF models, because they are simply too large for human checking to be practical.

HTML versions (c.f. Appendix E, Section E.5) of the FIL files output by the constraint-understanding system after processing the Course World and AP212 models, which hold all the frames generated while processing these models, will be available for browsing on the Author's web site (Broad [8]) for the foreseeable future.

It is disappointing that the constraint-understanding system failed to extract any `bound` and `range` HLCs at all from the two 'real' models, AP212 and EDIF 4 0 0 Level 0. The reason for this is that the cases for extracting bounds are too inflexible, in that they only work for relational operations in entities' domain rules such that one operand is an explicit attribute and the other a numeric literal. Real models have plenty of bound and range constraints, but these tend to be attached to defined types rather than entities (like in Figure B-8 of Appendix B). The 'variable' for a bound extracted from a defined type's domain rule is `SELF`, rather than an explicit attribute. The current constraint-understanding system could handle this by being seeded with many more cases which cover the combinations of possibilities, but it will be more economical to extend the system with more sophisticated adaptation techniques than the simple reinstantiation currently implemented. More sophisticated adaptation techniques will avoid the combinatorial explosion which arises when we try to store all variants of a case explicitly (see Appendix G, Section G.2.2).

This section has presented the experimental results of running the constraint-understanding system on some sample EXPRESS models. The next section draws conclusions about the major contributions of this thesis, and how well the aims and objectives of the thesis (see Section 1.5) have been met.

7.3 Conclusions

The major contribution of this thesis is the method for understanding constraints on information models. The following are considered to be the important ideas that arise from this thesis:

- The application of CBR to the task of understanding constraints, and the idea that it could be applied to code understanding in general. There appears to have been very little previous work in applying CBR to code understanding, although some research in this area clearly has some affinity with CBR, especially cliché recognition (see Section 2.2.2).
- There have been no previous attempts to automate the understanding of constraints on information models, to the best of the Author's knowledge - certainly not to extract higher-level knowledge about the abstract semantics of constraints with a view to the comparative understanding of constraints on two models. The approach taken in this thesis of extracting HLCs from models has much in common with cliché recognition, which has been applied to program code, but no one seems to have applied code understanding techniques to information models.
- Transferring constraints from the target model to the source model for schema-to-schema mapping, so that they can be checked on source instances *before* they are mapped to the target repository (as explained in Section 1.3.2), has some useful advantages:
  - Checking these constraints at all ensures that the source instances will not violate the integrity of the target model;

- Checking these constraints before rather than after carrying out a mapping gains efficiency, as discussed in Section 1.3.2;
- Automatically generating code to check constraints before a mapping (as opposed to writing such code manually) saves human programming effort. This is a difficult problem which requires understanding the comparative semantics of the constraints on the target and source models, so that constraints on the target model can be transferred to the source model if necessary.

However, transferring constraints for schema-to-schema mapping is not a contribution of this thesis, in that it has not been implemented yet.

The state of practical achievement, after a year of working towards the goal of transferring constraints for schema-to-schema mapping, is that the constraint-understanding system has been implemented. This was identified as an important prerequisite to constraint transference. Ideas for realising the goal of transferring constraints are in place, and could be further developed as discussed in Appendix G, Section G.4.1.

The objective that the system be data-driven has been achieved to a large extent. Much of this is inherent in the CBR approach. It is more flexible to represent the associations of EXPRESS code fragments with HLCs as cases, and to have generic algorithms to match those cases against the input model and extract the corresponding HLCs, than to hardwire code into the program to recognise such fragments and extract the HLCs. It makes the system more easily extensible, because new cases can be added by writing them in FIL, without having to modify the program.

The frame system makes the system data-driven to an even greater degree, because it makes it possible to declare new frame classes just by writing them in EXPRESS, again without having to change the program, as would be the case if the frame classes were realised as Java classes instead. However, there are still some special frame classes that the program needs to know about, and to make assumptions about what slots they have:

- The `case_frame` frame class: this is the frame class of the top-level frame of each case, which the FIL parser has to look out for so that it inserts frames of this frame class into the case library as cases. A case frame is assumed to have the distinguished slots `problem`, which the matcher recognises (see Appendix D, Section D.2), and `solution`, which the extractor uses (see Appendix D, Section D.3).
- The EXPRESS frame classes: the frames representing the input EXPRESS model are the starting point for extracting HLCs, so they have to come from somewhere! As discussed in Appendix E, Section E.3, the program creates frames of these frame classes by converting the objects produced by the ExJava parser.
- It was necessary to compromise the data-driven theology for the `conjunction` HLC frame class, because it needs to be treated specially, as explained in Sections 6.2 and 6.3.

This section has concluded the thesis by outlining its major contributions and discussing the achievement of its aims and objectives. The potential for future work is discussed in the next section.

7.4 Future Work

The potential for future work ensuing from this thesis is considerable, and is discussed at length in Appendix G. This section gives a brief summary.

There is room for improvement in the constraint-understanding system itself, as discussed in Section G.1 of Appendix G. In short, more HLCs will be needed, and the system also needs to be extended to cover more of EXPRESS. The FIL language could be extended with some new, powerful features to support more sophisticated case representation.

It was one of the aims to take a CBR approach to understanding constraints. However, the current constraint-understanding system is limited in the CBR that it uses. The following are the main areas for improvement and further development (and are further discussed in

Appendix G, Section G.2):

- The organisation of the case library (and hence the case retrieval algorithm) is based on a simple shallow indexing scheme, as explained in Appendix D, Section D.1.3. This is more efficient than mere linear organisation and retrieval because it prunes out some irrelevant cases from being considered by the matching algorithm, but it is not optimal because it only discriminates on one feature (the frame class of the top-level frame in the case's 'problem' part).
- The system does no partial matching and adaptation, except that it allows variables in cases that are unified with the corresponding values in the retrieval prompt, which is tantamount to reinstantiation (see Sections 3.3.2 and 6.3). The fact that the matching algorithm requires all slots in the 'problem' part of a case to match the prompt makes the system rather inflexible, sometimes requiring an excessive number of variants of each case in the case library.
- The system does not retain any of its own constraint-understanding experiences as cases in the case library - neither cases derived by adaptation, nor cases which chunk together the application of several simpler cases.

The way forward to achieving the goal of transferring constraints for schema-to-schema mapping (as introduced in Section 1.3.2) will entail research into understanding the comparative semantics of the constraints on two models (Section 1.2.3). Section G.3 of Appendix G discusses how constraints could be compared to achieve such an understanding. In short, it will require:

- Inferring correspondences between the constraints on two models (which will require knowledge of the structural mapping between these two models - how to represent this knowledge will have to be investigated first) (Section G.3.1);
- Assessing the semantic equivalences and differences between corresponding constraints on the two models (Section G.3.2).

In the context of comparative constraint-understanding, the extraction of HLCs will be able to be controlled: rather than extracting indiscriminately all the HLCs that can be extracted from a given model, the constraint-understanding system could avoid extracting HLCs that it does not need for the task which it is being used for. Section G.3.3 discusses heuristics for this extraction control.

Section G.4.1 discusses how transference of constraints for schema-to-schema mapping could be achieved, which will build on the work on comparative understanding of constraints. As mentioned in Section 1.3.2, transferring a constraint from the target model to the source model will entail:

- Deciding whether it is necessary to transfer the constraint (if it is already implied by the source model, it is unnecessary and should be avoided);
- Transforming the constraint from how it is expressed on the target model, if structural differences between the two models mean that it has to be expressed differently on the source model.

In fact, transferring constraints for schema-to-schema mapping is not the only task which could use an understanding of the comparative semantics of the constraints on two models. Section G.4.2 identifies and briefly describes some other possible applications of this sort of understanding.

This chapter has summarised the thesis. It has presented some experimental results of running the constraint-understanding system on 'real' models. It has drawn conclusions regarding the major contributions of this thesis, and how well the aims and objectives (as stated in Section 1.5) have been achieved. It has discussed the potential for future work. This thesis is a stepping stone towards understanding the comparative semantics of the constraints on two models (as needed for such tasks as transferring constraints for schema-to-schema mapping).

Section 7.5 brings the curtain down on this thesis by considering the wider implications that it may have for automatic programming in the distant future.

7.5 Afterword

Understanding the constraints on information models is a special case of code understanding in general, as discussed in Chapter 2. This thesis has discussed a case-based approach to understanding constraints, in which each case associates a fragment of code with an abstract characterisation of the semantics of that code (in the form of an HLC). This idea could surely be generalised to understanding all sorts of code, such as imperative code for example. Indeed, understanding the constraints implied by functions and procedures in EXPRESS would, of necessity, link research on understanding constraints to research on understanding imperative code (such as that referred to in Chapter 2, which was mainly concerned with imperative code).

It is argued in Appendix G, Section G.4.1, that a bidirectional approach to CBR could exploit the existing cases for understanding constraints to use them for transferring constraints from a target model to a source model (and hence planting code in a schema-to-schema mapping program to check those constraints), by matching the 'solution' parts of cases to HLCs and applying the 'problem' parts of those cases that match to realise an HLC as concrete code. In principle, the notion of bidirectional CBR should also apply to code generation in general, using cases for code understanding to suggest how to realise an abstract specification as a concrete implementation.

Ultimately, it may be possible to achieve automatic programming by taking a case-based approach, writing new programs by retrieving and adapting old programs. Reuse has already been identified as an issue of paramount importance in the field of software engineering (as discussed in Section 2.4), and case-based programming is clearly latent in that approach. The work described in this thesis, taken with the short-term future work discussed in Appendix G, represents a small step towards this dream.