8. Conclusions

8.1 Statement of problem

The foremost concern of designers in electronics in picking a design environment has always been to combine state-of-the-art design tools. Today design tools support designers during all phases of the design, starting with the capturing of a high-level, behavioural design description, through the various transformation and validation stages, finally yielding a low-level description of what has to be physically fabricated. Before the availability of standard and powerful design description languages for all domains and all levels of detail, achieving data integration between tools was a major difficulty. Since their standardization in 1987, the languages VHDL and EDIF have become the Esperanto of electronic design. Today all major EDA tool vendors support one or both languages to capture and exchange design data from the behavioural, structural, and physical domains.

With ever-growing sizes of designs and design teams, design management has moved into focus. EDA frameworks have been introduced to provide data, control, and presentation integration services to help manage large designs. While the first EDA frameworks have focused on data integration, it was realized that for all practical purposes, data exchange through design files solves designers' needs. Since then, the attention of framework vendors has moved to design management. Versatile graphical browsers have been built on top of the basic data management services provided by the frameworks that allow designers to query and manipulate the design state at the granularity of design objects. Design processes can be automated through net-based flow editors. Framework tools communicate with each other through flexible message links.

Today, two facts prevent the effective incorporation of file-based design tools into EDA frameworks:

1. Framework vendors, in an attempt to stay domain independent, have focused on enabling technology for design data management, while neglecting powerful design information management. Especially the VHDL language offers a flexible configuration scheme that has to be mapped to a framework's conceptual schema for design information management.
2. File-based design data exchange has always been considered a bad thing, something that must eventually be overcome. Hence, framework-oriented projects like the German DASSY project and standardization bodies like CFI

have long focused on developing procedural interfaces. Procedural interfaces where thought to rather sooner than later replace design files because of their obvious advantages. Unfortunately, design description languages flourish and are well accepted by designers as well as tool vendors. Languages like VHDL and EDIF have become the prime interface for off-the-shelf tools. Procedural interfaces today are almost always clumsy add-on's, used as technology demonstrators but hardly usable to exchange real designs.

8.2 Solution approach

To overcome these problems, we contribute three major assets to the design automation community:

1. Up to today, conceptual schemas for EDA frameworks have been developed separately from schemas that capture the semantics of design descriptions. In fact, there are two nearly disjoint communities working on overlapping topics. We have developed a conceptual schema that explicitly takes into account both, object-based design information management that supports team-oriented design in an integrated environment and requirements that stem from the design description language VHDL. The resulting conceptual schema could be mapped naturally onto the schema of the Nelsis CAD framework. This effort shows that little is in fact needed to conceptually unify framework-based design information management and the use of high-level design languages.
2. Language processing is traditionally regarded as being in the realm of tool vendors who can rely on standard compiler technology to construct design file input and output processors. We advocate the use of language processors not only for detailed design file analysis but also to extract features with the purpose of design information management. Design files need not be analysed down to every single detail to access useful information to be managed by a framework. Instead, we have defined and implemented a specification language and associated toolkit that facilitates the specification and extraction of useful information from standard design files. The toolkit utilizes our specification language to specify both analysis and reconstruction of design files. Our approach is novel both from the compiler construction and design encapsulation viewpoints.

3. While commercial frameworks offer extension languages (e.g. Cadence's *Skill*, Mentor's *Ample*, JCF's *Elk*), design tool *encapsulation* relies on standard UNIX shell wrappers around black-box integrated tools. We have defined the architecture of a new framework service that is based around the publicly available extension language *Tcl* that combines familiar shell-like syntax with easy extendability and close integration with the language processing toolkit. This new service fits nicely into CFI's proposed framework architecture reference model and into the existing Nelsis framework implementation.

The ignorance of the success of design description languages has lead to the strange situation that on the one hand there is mature framework technology, offering powerful design data management and versatile and user-friendly graphical browsers whereas, on the other hand, there are state-of-the-art file-based design tools that do not match. The unfortunate result is that *open* frameworks have started to become one of the unfulfilled promises of the past, similar to the general problem solvers in artificial intelligence research.

8.3 Technology developed

In this thesis, we present technology that tries to bring together the opponents. Incorporating the extended conceptual schema and new framework service for design tool encapsulation certainly does not provide the optimal solution to design tool integration, but it provides a missing link that provides wider acceptance of open EDA frameworks amongst designers and tool vendors. In Chapter 2, we have shown that there are more aspects to tool integration than simply being black- or white-box. The conclusion was that fine-grained design information management is not necessary and in fact not desirable for file-based design tools. On the other hand, file-based storage of design data prevents effective design information management. Hence, we propose to store design data at the granularity of design objects, but to import and export them in design files. In Chapter 4 we have described a new framework service that encompasses services for scripting tool wrappers using a language more suitable than standard Unix shells. The most important contribution of this new service is a language independent facility to process design files. While such a facility maintains the domain independence of the framework, it allows to extract design information from design files on import and to recombine text chunks associated to design objects managed by the framework on export. Thus, a framework is free to store design data at the granularity of design objects. At this granularity a web of relationships between design objects may be established and maintained automatically by the framework. Graphical browsers can be used by designers to help understand their design.

This web of objects and relationships is based on the conceptual schema developed in Chapter 5. It has to be emphasized that this schema is based on a formally defined modelling technique. We have chosen to use the Xplain data model over its competitor EXPRESS because of its clear graphical notation and built-in semantic constraints. The conceptual schema is based on the schema of the Nelsis CAD framework and adds just enough types and relationships to support dynamic binding of components to design objects and configurations as offered by VHDL. With this schema it is possible to directly map VHDL design descriptions to a web of design objects.

With architecture and conceptual schema in place, the toolkit presented in Chapter 6 greatly simplifies the construction of language processors for design tool encapsulation. Based on standard compiler construction technology, by exploiting the virtual memory management facilities of modern operating systems, the parsers generated by this toolkit can skip large regions of input text and nevertheless associate this text with design information extracted from a design file. Modular grammars are introduced as a handy means to choose different granularities of analysis for different regions of the input text, further reducing the necessary amount of syntax detail to be specified. The generated parsers automatically create parse trees that can be traversed from Tcl scripts to gather design information necessary to create design objects in the design information management component of the framework. We have renounced the introduction of complicated tree pruning operators because due to the little detail extracted from the input text, parse trees tend to be small. Rather, the two dedicated extension language methods on parse tree nodes *all* and *one* serve as filters during parse tree traversal to help look only at important information.

The reverse direction, exporting design objects to valid design description files, is supported by two mechanisms. Lexical contexts are stored and managed in the framework and are used as file trunks in which text chunks associated with design objects are inserted. In addition, object types in the design information management component can be associated with tree constructors automatically generated from the same syntax specification as the parsers. Such a constructor knows which attribute values to convert into syntax when traversing an object graph to export a design description file. We used this toolkit to generate a parser and tree constructors for VHDL. The syntax specification for VHDL has well below 150 lines, needing a few hundred lines of Tcl code to link the parse trees to the conceptual schema. This amount of coding is well in the range of work accepted by designers as necessary to create a seamless design environment in which it is possible to conduct productive design work.

Chapter 7 describes the implementation of design environments based on our conceptual schema, the encapsulation service with VHDL language processors generated by the toolkit, some commercial design tools from Synopsys, and alternatively, the object-oriented database management system ObjectStore or the Nelsis CAD framework. The implementation based on Nelsis proved most useful because it allows to use the powerful Nelsis browsers and design tools already integrated with Nelsis together with VHDL-based, commercial synthesis and simulation tools. A first prototype of this implementation was demonstrated at the '94 JCF Framework Workshop in Karlsruhe and was well received [Schettler 94a]. The concepts developed in this thesis were also presented at the International Conference on CAD [Schettler 94b].

8.4 Advantages

The technology presented in this thesis provides better data integration between encapsulated design tools and a design information management component of an EDA framework. Design tool encapsulation based on design files may become obsolete one day, when programming interfaces are mature and flexible enough to cover all aspects of an electronic design description. With the current euphoria about the integrative nature of standard design description languages like VHDL (cf. for example [Hüwel 92], [Synopsys 92], [Meersman 94]) this may still take some years. Meanwhile, the conceptual schema we have developed can serve as a starting point for framework developers and standardization bodies like CFI to extend their framework schemas with concepts that are especially important when trying to support VHDL. Amongst these are separation of interface and contents, distinction of domains and levels of detail and the support for the configuration of whole occurrence trees. On the other hand, language standardization may well take into account aspects that are relevant for integrated design environments. While VHDL has gone a long way in this direction by separating interfaces and implementations and the availability of hierarchical configurations, finer control would be desirable over the selection of implementations based on status or version information.

8.5 Disadvantages

Our approach currently lacks an effective means to establish equivalence relationships between input and output data of a design tool. The wrappers could try to deduce some of these relationships by matching names in the result files with objects already in the database [Blackburn 85]. This is a rather clumsy approach because it tries to deduce information from the design data that the design tools already have. Here in fact some help from the encapsulated tool would be called for. A comparatively non-intrusive approach would be to have the tool register its object accesses by a programming interface similar to the one used by Cassotto in his VOV system [Casotto 90]. But this would deviate from pure design tool *encapsulation*. A different approach was suggested in Section 2.8, namely to let the framework emulate a network file system on which the tool would operate.

As discussed in Section 7.5, our approach contributes little to improving control integration in general. Control integration has long been hampered by the lack of sufficiently high-level standards for inter-process communication. As such, tool vendors had to develop proprietary means to enable inter-tool communication (ITC). Unfortunately, tool vendors also seem to consider a good but proprietary ITC mechanism as a major selling point for their tool suites. The interesting work of Silva et al. ([Silva 93]) suggests that opening up these interfaces may be a good selling point for tool suites because users then are able to come up with completely new applications of tools that the vendors have not originally thought of. This unfavourable situation may change with the introduction of vendor technology like ToolTalk [ToolTalk] or industry standards like CORBA [OMG 90], provided that tool vendors accept the need for open, programmable interfaces to their tools.

8.6 Outlook

We have reached our goal of bridging the gap between file-based design data interfacing of design tools and object-based design data management in EDA frameworks. With the approach presented, even file-based design tools can benefit from the sophisticated browsers offered by modern framework technology. Still, the system we have presented in this thesis can by no means represent the final word in tool encapsulation. While a suitable next step on the framework side would be the provision of a network file system on top of a design information management system, further progress largely depends on the improvement of observability and controllability of design tools. The Synopsys tools suite we have selected as base for our design system leaves much room for improvement in this respect. The incorporation of a commercial tool suite certainly is a good selling point for our approach. However, the availability of tool source code would have enabled us to cover control integration more deeply than we have been able to do now. Hopefully, users' pressure will lead tool vendors to open up some of their interfaces to the design automation community. Both designers and vendors will profit from this.