Interfacing a Constraint Programming Language with an Existing Electronics CAD Package

Marc van Dongen, Barry O'Sullivan, James Bowen, Alex Ferguson
{dongen|osullb|J.Bowen|abf}@cs.ucc.ie,
Computer Science Department, National University of Ireland, Cork,
College Road, Cork, Ireland

September 1997

Abstract

In this note, we report on an ESPRIT-funded project in which we are integrating the Galileo6 constraint language with an electronics design CAD package called Visula. We describe some aspects of the integration and show the functionality of the system, and conclude by discussing our findings.

1 Introduction

Increasingly, it is being realised that success in manufacturing requires integration between the various phases of the product life cycle [Ulrich and Eppinger, 1995]. One of the key aspects of this integration is that, during the design of an artifact, due consideration should be given to facilitating the down-stream phases of the life-cycle. This is frequently known as "Design for X" (or DFX), where the X ranges over such issues as manufacturability, servicability and so on.

Vendors of leading CAD software have started to incorporate DFX features into their packages. In addition, vendors have discovered from customer feedback that engineering companies require flexibility as well as functionality: companies which use CAD packages want to specify their own DFX guidelines as well as, or even instead of, relying on standard DFX guidelines supplied by the CAD vendors.

If engineers from user companies are to augment CAD packages with company-specific guidelines, it is important that a language be provided which simplifies this task as much as possible. We are developing a language for this purpose, based on the computational paradigm of constraints. The language, called Galileo6, is intended to be generic in two senses: it can be used to encode guidelines from any product domain and guidelines expressed in it can be applied to designs encoded in a variety of CAD formats.

We report on some aspects of an ESPRIT-funded project in which we are linking the Galileo6 language to an electronics design CAD package called Visula. In Section 2 we briefly describe the Galileo6 constraint language. In Section 3 we describe the integration of Galileo6 and Visula. In Section 4, we illustrate how the system can be used to specify company specific guidelines and critique designs with them. In Section 5, we provide a concluding discussion.

2 An Overview of the Galileo6 Constraint Language

Galileo6 is the latest in a series of constraint programming languages for Concurrent Engineering that one of the authors (Bowen) has been developing since the mid 1980s [Bowen and Bahler, 1992; Bahler *et al.*, 1994]. The main difference between Galileo6 and its predecessors is that Galileo6 comes with a programmable mechanism for accessing CAD databases.

Writing a program in Galileo corresponds to formulating a theory in first order logic [Bowen and Bahler, 1991; 1993]. Defining application-specific concepts in the form of domain, constant, function or relation definitions corresponds to defining a language in first order logic. Constraints in Galileo correspond to sentences in logic. Unlike the CLP languages, which only admit universal quantification, Galileo allows arbitrary nesting of existential and universal quantifers in constraints.

The language supports structured domains and what are called *list comprehensions* in the functional programming community. Finally, the language provides a number of pseudo-quantifiers which can be used to state that there should be exactly/at least/at most n things in a design satisfying some constraint. Expressions involving such quantifiers can be transformed to constraints in terms of lengths of list comprehensions.

3 Integrating Galileo6 and Visula

In this section we discuss some aspects of integrating Galileo6 and an existing electronics CAD design package called Visula.

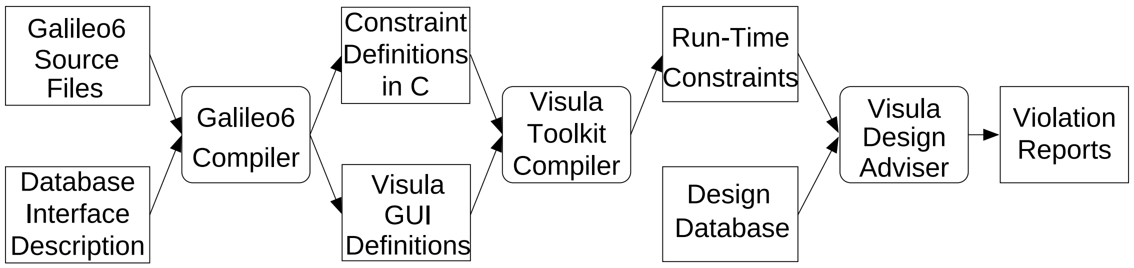

Figure 1: Integration of Galileo6 and Visula

Figure 1 depicts how Galileo6 and Visula are integrated. The *Galileo6 Source Files* at the top left side of the figure implement a number of guidelines as constraints. The *Database Interface Description* at the bottom lefthand side describes the mapping from objects in the Visula database to entities in the Galileo6 world. Both the source files and the interface description are compiled with the Galileo6 compiler, producing (a) *Constraint Definitions in C* implementing the constraints, and (b) *Visula GUI Definitions* describing the mapping between buttons in the GUI and constraint definitions in C. The constraint and GUI definitions are compiled with a standard Visula programmers toolkit into a number of *Run-Time Constraints*. From within the GUI of the Visula Design Adviser, a user can select guidelines (and thereby run-time constraints). After making his selection, a user can critique his design, which will cause the run-time constraints to be applied to his current design. The run-time constraints will read the current contents of the design database, check this for compliance with the guidelines, and write their findings in the form of Natural Language (NL) explanations to *Report Files* which can be read be the user. A formatting mechanism is used to provide automatically generated NL explanations from the Galileo6 program.

4 Critiquing The Design

To illustrate the results of critiquing an example design with Galileo6, we will show what happened when the the design shown in Figure 2 was critiqued with the Galileo6 program from Figure 3.

The program in Figure 3 encodes the following three guidelines as constraints:

- **Corner Fiducials** (lines 10–12): There should be exactly three corners of the bord with a fiducial marks[1] in it. The code generator translates this into the equivalent constraint

[1]Unconnected copper pads on the board, used by assembly machines to ensure accurate placement of components.

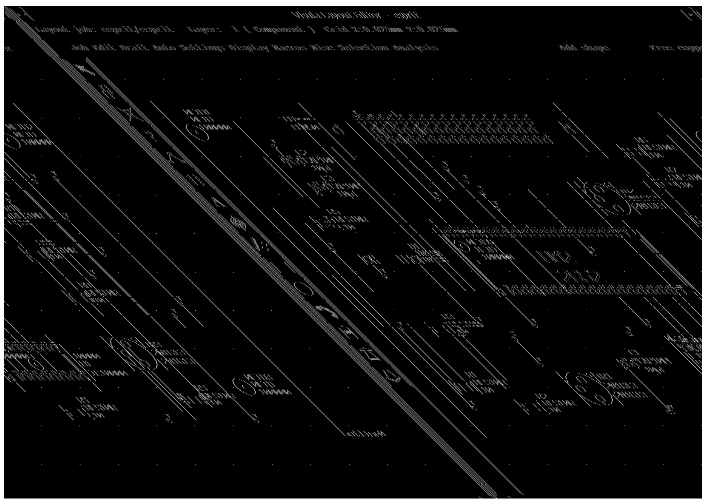

Figure 2: Toy Design

```
3 = length( [ C | exists C in theBoard.corners:  exists fid( F ): in_corner( F, C ) ] ).
```

- **Min Lead Pitch** (lines 14–21): Every component which is an Integrated Circuit with a minimum lead pitch[2] should have two fiducial marks in diagonally opposite corners;
- **IC Existence** (lines 23–24): There should be at least on Integrated Circuit in the design.

There is one constraint which is violated by the design. This is the "Min Lead Pitch" constraint defined in lines 14–21. The constraint is violated because the Integrated Circuit called UM2 in the centre of the design has a lead pitch of less than 0.025", but has only one corner with a fiducial in it, whereas it should have fiducials in two diagonally opposite corners.

```
 1 module main( main ).
 2
 3 import std.
 4 import pcb_concepts( distance/2, diagonal/3, nearest_to/3, in_corner/2 ).
 5
 6 function min_lead_pitch( comp ) -> real =::=
 7   { Ic -> L : L = min( [ distance( P1, P2) | exists P1 in Ic.pins, P2 in Ic.pins: P1 <> P2] ) }
 8   with format( 'the minimum lead pitch of ', #1 ).
 9
10 exist exactly 3 C in theBoard.corners:
11   exists fid( F ): in_corner( F, C )
12 with name = 'Corner Fiducials'.
13
14 all comp( I ):
15   ic( I ) and (min_lead_pitch( I ) < 0.025 [inch]) implies
16     exists C1 in I.corners, C2 in I.corners, fid( F1 ), fid( F2 ):
17       (F1 <> F2) and (C1 <> C2) and
18       diagonal( C1, C2, I.corners ) and
19       nearest_to( F1, C1, I.corners ) and in_corner( F1, C1 ) and
20       nearest_to( F2, C2, I.corners ) and in_corner( F2, C2 )
21 with name = 'Min Lead Pitch'.
22
23 exists comp( C ): ic( C )
24 with name = 'IC Existence'.
```

Figure 3: Source Code for DFA Program in Galileo6

After applying the run-time constraints to the design, the Visula Design Adviser shows the guidelines that passed by giving them a blue status bar (these appear as dark gray in the window in Figure 4). For each constraint which has been violated, a report is produced with an explanation

[2]The minimum distance between adjacent pins.

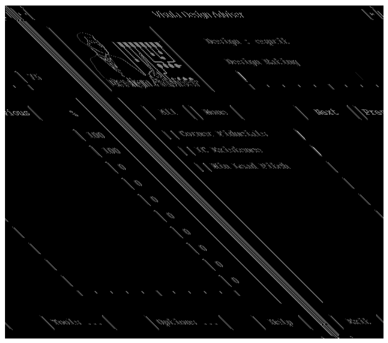

Figure 4: Design Adviser

```
The following constraint was violated:
It must be true that:
for any component, I say, the following is true:
    I isn't an integrated circuit
 or the minimum lead pitch of I >= 0.025 inch
 or there exist  positions, C1 say, in the corners of I, and
                           C2 say, in the corners of I, and
                fiducials, F1 say, and F2 say, such that:
      F1 and F2 are not equal and
      C1 and C2 are not equal and
      C1 and C2 are diagonally opposite corners of the corners of I and
      of all the corners of I, C1 is the nearest one to F1 and
      F1 is in C1 and
      of all the corners of I, C2 is the nearest one to F2 and
      F2 is in C2.

It was violated by each instance in the following:
A component where ( compname = UM2
                    partname = 2652
                    type = ic )
```

Figure 5: Violation Report

of why it failed. These explanations were generated from the Galileo6 constraint-declarations and describe in English why the constraint has failed. In addition the explanation provides a list of all combinations of entities in the design which violated the constraint. Figure 5 shows the violation report for the "Min Lead Pitch" rule. A cross-highlighting facility can be used to locate components which were mentioned in the violation reports: upon selecting the name of a component in a violation report, the Visula software will highlight the component in the design with that particular name.

5 Concluding Discussion

We have reported on the integration of the Galileo6 constraint language with an existing electronics CAD package called Visula. We have shown the functionality of this integration.

End-user companies in our project are using Galileo6 to express their company specific guidelines. Their reaction is that expressing guidelines as constraints in Galileo6 is easy and prevents ambiguity.

6 Acknowledgements

The work reported here was funded by the European Commission under ESPRIT project number 20501, with acronym CEDAS.

References

[Bahler *et al.*, 1994] D. Bahler, C. Dupont, and J. Bowen. An axiomatic approach that supports negotiated resolution of design conflicts in concurrent engineering. In *Proc. AID-94*, 1994.

[Bowen and Bahler, 1991] J. Bowen and D. Bahler. Conditional existence of variables in generalized constraint networks. In *Proc. AAAI-91*, pages 215–220, 1991.

[Bowen and Bahler, 1992] J. Bowen and D. Bahler. Frames, quantification, perspectives and negotiation in constraint networks for life-cycle engineering. *Int. J. of AI in Engineering*, 7:199–226, 1992.

[Bowen and Bahler, 1993] J. Bowen and D. Bahler. Full first-order logic in knowledge representation a constraint-based approach. Technical report, University College Cork, 1993.

[Ulrich and Eppinger, 1995] Karl T. Ulrich and Steven D. Eppinger. *Design for Manufacture*, chapter 9, pages 179–216. McGraw-Hill, New York, 1995.