Using Constraint Programming to Simplify the Task of Specifying DFX Guidelines*

Marc van Dongen†, Barry O'Sullivan†, James Bowen†,
Alex Ferguson†, and Mike Baggaley‡
{dongen|osullb|J.Bowen|abf}@cs.ucc.ie, M.Baggaley@x400.icl.co.uk
† Computer Science Department, National University of Ireland, Cork,
College Road, Cork, Ireland
‡ D2D, West Avenue, Kidsgrove, Stoke-on-Trent, ST7 1TL, UK

Abstract

Increasingly, it is being realised that success in manufacturing requires integration between the various phases of the product life cycle. 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.

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

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

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.

In this paper, 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 introduce the notion of constraints; we indicate how they can be used to express multi-directional influences between an artifact and its life-cycle environment and discuss how this capacity for multi-directional inference can be used to support Concurrent Engineering (CE). In Section 3, we overview the basic concepts of Galileo6, the constraint-based language that we are developing. In Section 4 we describe the integration of Galileo6 and Visula. In Section 5, we use some Design For Assembly (DFA) rules to show how the resulting system can be used by designers to specify their own design criteria, and how this can facilitate their task of critiquing their designs. In Section 6, we report on the experience of one of the user companies in the ESPRIT consortium. In Section 7, we provide a concluding discussion.

2 Constraint-Based Approaches to Concurrent Engineering

Although the word "constraint" has many meanings in colloquial language, it has a specific meaning in the Computer Science literature. In this context, a *constraint* is a declarative statement which specifies some relationship that must be true about a group of entities; in other words, a constraint restricts the values that may be assumed by a group of one or more *parameters*. A *constraint network* is a collection of such constraints [Mackworth, 1977].

Our past research [Bowen and Bahler, 1991; 1992; Bahler *et al.*, 1994] has shown that frame-based constraint networks offer several features which make them attractive as a basis on which to build a language for CE applications. A frame-based constraint network is one in which parameters are not required to be scalars; instead, they can be either scalars or frames – complex data structures which can be organised in an inheritance hierarchy.

In a frame-based constraint network, frames can be used to represent: the artifact being designed; the components from which the artifact is configured or the materials from which it is made; and the life-cycle environment in which the artifact will be manufactured, tested and deployed. Constraints can be used to express in an explicit way the mutual restrictions exerted on each other by artifact functionality, component/material properties, and life-cycle processes.

A major attraction of constraint networks for CE is that constraints support multi-directional inference: information can flow in any direction through a network. Thus, for example, the impact of a design decision on the options available to a test engineer can be determined by propagating the design decision and its consequences throughout the network. Equally, if testing decisions are made early on, the impact of these decisions can be reflected in restrictions placed on the designer's freedom. By supporting multi-directional inference, one constraint network can support both of these forms of interaction with equal ease.

Consider, for example, the following simple Design for Testability guideline[1]:

> *the CPU crystal on a board should oscillate at a rate which does not exceed the fastest frequency that can be accommodated by the available bed-of-nails tester.*

Suppose we represent the CPU crystal as a frame called `the_cpu_crystal` which contains a slot called `freq` that represents the frequency at which the crystal vibrates and suppose we represent the bed of nails tester as a frame called `the_test_machine` which contains a slot called `maxFreq` that represents the maximum frequency that can be handled by the tester. Then we could represent the above guideline by the following statement in a constraint-based language like Galileo6:

```
the_cpu_crystal.freq =< the_test_machine.maxFreq.
```

Depending on the order in which the circuit designer and test engineer make their decisions, this constraint can propagate restrictive influence between them. If the frequency of the CPU crystal is specified by the circuit designer before the test machine has been selected by the test engineer, this constraint imposes a lower limit on the required frequency handling capability of the test machine and, therefore, it limits the test engineer's choice of equipment. Alternatively, the flow of restrictive influence can go in the opposite direction; if, for example, only one test machine is available in the factory and if a strategic decision has been made that no new equipment can be bought, then this constraint limits the circuit designer's freedom of choice by limiting the oscillation frequency that he can choose for his CPU crystal. Thus, this simple constraint is very versatile.

[1] A similar guideline was used as an example in [Bowen and Bahler, 1991] but we present it again here to facilitate a self-contained paper.

3 An Overview of the Galileo6 Language

Galileo6 is the latest in a series of constraint programming languages for CE that one of the authors (Bowen) has been developing since the mid 1980s. The main difference between Galileo6 and its predecessors is that Galileo6 comes with a programmable interface mechanism to existing CAD databases. The interface machanism allows for the description of mappings from objects in the CAD database to entities in the Galileo6 world. This allows different companies with different databases to share Galileo6 programs provided their interfaces describe the mappings from their own CAD database format to the Galileo6 concepts referenced in the programs. Also, it allows companies to define mappings from their own design databases to their own company-specific Galileo6 concepts. We will not, however, describe the interface mechanism in this paper as our intent is to focus on the expressiveness offered by constraints. The reader is referred to [Ferguson *et al.*, 1997] for a description of the interface mechanism.

The essential part of a Galileo6 program is a collection of constraint statements. Indeed, if the constraints use only standard pre-defined concepts such as the equality predicate or the addition function, a Galileo6 program will comprise nothing more than a set of constraint statements and, because the program contains a set of constraints, they may be written in any order. Typically, however, constraints for a real-world application can only be expressed if the pre-defined concepts are augmented with application-specific concepts. These additional concepts can be defined locally within the program or they can be defined in a separate compilation module and imported into the program.

Consider, for example the simple Galileo6 program in Figure 1. The line numbers in this figure are not part of the program; they are used here simply for easy reference. We describe the program in a top-down fashion, starting with the constraints and moving downward to application-specific concepts referenced by the constraints.

```
 1 module main( main ).
 2
 3 import std.
 4
 5 function distance( comp, comp ) -> real
 6   =::= { (C1,C2) -> D: D = sqrt( (C1.position.x - C2.position.x)^2 +
 7                                  (C1.position.y - C2.position.y)^2 ) }
 8 with format ('the distance from ', #1, ' to ', #2 ).
 9
10 all comp( C1 ), comp( C2 ):
11   C1 <> C2 and ic( C1 ) implies distance( C1, C2 ) >= 10 [mm].
12
13 exists comp( C ): ic( C ).
```

Figure 1: Simple Galileo6 program

There are two constraints in the program, each implementing a design requirement. The first constraint in lines 10–11 implements the requirement that integrated circuits should be at least 10 millimetres apart from other components. The universal quantifier `all` from Predicate Calculus (PC) is used to implement this requirement stating that, for every two components, `C1` and `C2` say, the following must be true: if they are distinct and `C1` is an integrated circuit, then their distance apart must be at least 10 millimetres.

The predicate symbols `<>` and `>=` referenced in the constraint declaration have a predefined meaning in Galileo6. Other concepts like `comp` (for component) and `mm` (for millimetre) are imported from a module called `std` in line 3. This module also takes care of the interface to the external CAD database.

The function `distance` referenced by the constraint definition is defined in lines 5–8. The function maps two components to their distance apart in reals. The definition for `distance` consists of two sections. The first, in lines 5–7, is the semantic section which describes the *meaning* of the function. The second section, in line 8, is an optional formatting section which is used in the generation of Natural Language (NL) explanations for possible causes of guideline violations. At the end of this section it is shown how good use of formatting sections can improve explanations.

In line 13 a second requirement is implemented by a constraint which uses the PC existential quantifier `exists` to state that there should be at least one component which is an integrated circuit.

The constraints in a Galileo6 program can be used to test designs for compliance with the guidelines imposed by those constraints. Suppose we critique a design which has an integrated circuit in it which is less than 10 millimetres apart from another component, thus violating the guideline implemented in lines 10–11. Because of the violation of the guideline, Galileo6 will provide a Natural Language paraphrase of the guideline as depicted in Figure 2. This text is generated automatically from the constraint definition in lines 10–11, the formatting section in the definition for `distance`, and formatting sections for `ic` and `comp` defined in the library `std`. Had no formatting information been used at all, the explanation given would have been the less natural one depicted in Figure 3. Our experience is that carefully choosing wordings in formatting sections results in good NL explanations [van Dongen *et al.*, 1996].

```
The following constraint was violated:
It must be true that:
for any components, C1 say, and
                    C2 say, the following is true:
     C1 and C2 are equal
  or C1 is not an integrated circuit
  or the distance from C1 to C2 >= 10 mm.
```

Figure 2: Explanation with Formatting

```
The following constraint was violated:
It must be true that:
for any comps, C1 say, and
               C2 say, the following is true:
     C1 = C2
  or not ic( C1 )
  or distance( C1, C2 ) < 10 mm.
```

Figure 3: Explanation without Formatting

4 Integrating Galileo6 and Visula

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

Figure 4 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 and organise them in a number of rulebooks (groups of guidelines). The *Database Interface Description* at the bottom lefthand side describes the mapping from objects in the database to entities in the Galileo6 world as discussed in Section 3. 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 Rulebook Definitions* describing the mapping from a guideline in a rulebook to the constraint definition in C corresponding to that particular guideline. The constraint and rulebook definitions are compiled with a standard Visula programmers toolkit into a number of *Run-Time Constraints*. From within the Visula Design Adviser, a user can select rulebooks with the aid of a Graphical User Interface. Within rulebooks, guidelines (and thereby run-time constraints) can be selected. 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 NL explanations to *Report Files* which can be read be the user.

5 An Example Application

In this section we show how the integration described Section 4 can be used (a) to express DFX guidelines; (b) to critique designs with these guidelines; (c) to use automatically generated explanations to locate entities in the design which caused these guidelines to fail. Two guidelines are used to illustrate this functionality.

5.1 Natural Language Guidelines

Part of the reason why it is so easy to implement guidelines in Galileo6 is that the language enables them to be expressed in such a very high-level fashion that the Galileo6 statement of the guidelines is very similar to the Natural Language expression of them. To illustrate this, we use two guidelines which were taken from a larger set of Design for Assembly (DFA) guidelines provided by one of the partner companies in the CEDAS project.[2] The guidelines

[2]The reader is referred to [van Dongen *et al.*, 1996] to see how the complete set of guidelines has been encoded in Galileo6.

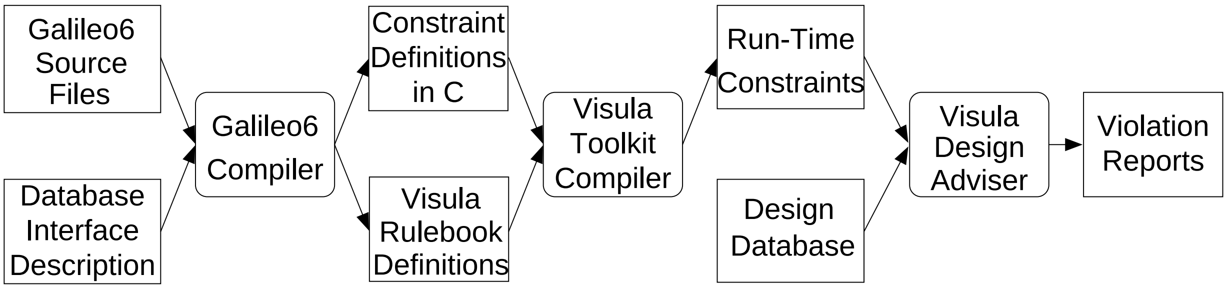

Figure 4: Integration of Galileo6 and Visula

are concerned with the proper placement of *fiducial marks*, which are unconnected copper pads on the board, used by assembly machines to ensure accurate placement of components. The guidelines we have chosen are as follows:

- **Guideline 1**: *There must be three fiducial marks positioned in three of the four corners of the board.*

- **Guideline 2**: *Each integrated circuit that has a lead pitch*[3] *smaller than 0.025" must have two fiducial marks positioned in diagonally opposite corners of the component.*

5.2 Encoding the Guidelines in Galileo6

One of the major bottle necks for user companies in using existing CAD systems is the difficulty of implementing guidelines correctly in the form of a program. We claim that using Galileo6 can significantly simplify the encoding of guidelines. We demonstrate this by showing how the natural language guidelines from the previous section have been encoded in Galileo6, showing the close relationship between the constraints and the original guidelines, and by pointing out the expressive power of the language.

```
module main( main ).

import std.
import pcb_concepts( distance/2, diagonal/3, nearest_to/3, in_corner/2 ).

function min_lead_pitch( comp ) -> real =::=
  { Ic -> L : L = min( [ distance( P1, P2) | exists P1 in Ic.pins, P2 in Ic.pins: P1 <> P2] ) }
  with format( 'the minimum lead pitch of ', #1 ).

rulebook 'ICE volume I':

exist exactly 3 C in theBoard.corners:
  exists fid( F ): in_corner( F, C )
with name = 'Corner Fiducials'.

rulebook 'ICE volume II':

all comp( I ):
  ic( I ) and (min_lead_pitch( I ) < 0.025 [inch]) implies
    exists C1 in I.corners, C2 in I.corners, fid( F1 ), fid( F2 ):
      (F1 <> F2) and (C1 <> C2) and
      diagonal( C1, C2, I.corners ) and
      nearest_to( F1, C1, I.corners ) and in_corner( F1, C1 ) and
      nearest_to( F2, C2, I.corners ) and in_corner( F2, C2 )
with name = 'Min Lead Pitch'.

exists comp( C ): ic( C )
with name = 'IC Existence'.
```

Figure 5: Source Code for DFA Program in Galileo6

The DFA guidelines from Section 5.1 have been encoded in the Galileo6 program shown in Figure 5. In considering the program in detail, we will do so in a top-down fashion, starting with the rulebooks and proceeding conceptually downwards to examine the constraint

[3] the least distance between any pair of adjacent pins

declarations and the definition of the application-specific concepts that are referenced in these constraints.

The program consists of two rulebooks. The first rulebook called "ICE volume I" is defined in lines 10–14. The second rulebook is called "ICE volume II" and is defined in lines 16–28. Rulebooks, as mentioned earlier, can be used to group guidelines. "ICE volume I" consists of one guideline. "ICE volume II" consists of two guidelines.

The single guideline in "ICE volume I" is implemented as the constraint called "Corner Fiducials" in lines 12–14. The constraint implements Guideline 1 above.

In order to keep this example simple, a number of concepts like `theBoard`, `fid`, and `mm` in the constraint declaration have been imported through the `import` statement in line 3. The concept `distance` has been imported from the library called `pcb_concepts` in line 4.

There is one difference between the above constraint and the original guideline, which illustrates a common problem in the elicitation of expert knowledge, namely that experts rarely say what they really mean. The difference is that the Galileo6 constraint states that there should be three corners of the board each of which should have a fiducial in it; this is what was really meant by the engineer who provided the NL guidelines although it only became clear when we asked him for clarification. The original statement of the guideline was ambiguous: the guideline could have been interpreted as requiring nine fiducials, three each in each of three corners of the board.

As was shown, when a constraint written in Galileo6 is violated, an automatically generated NL paraphrase of the constraint is included in the error report that is produced. The relationship between the first constraint in this program and the sentence in Guideline 1 above is highlighted by showing here the NL paraphrase that would be generated if the constraint were violated:

```
It must be true that:
there exist exactly 3 positions, C say, in the corners of the board,
with the following property:
  there exists a fiducial, F say, such that:
    F is positioned in C.
```

The second rulebook "ICE volume II" consists of two guidelines. The first of these two is implemented as the constraint called "Min Lead Pitch" in lines 18-25. The constraint implements Guideline 2, which specifies that if the minimum lead pitch of an integrated circuit is less than 0.025", then the IC should have two fiducials in two diagonally opposite corners, which is exactly how the requirement has been implemented. The NL paraphrase of this constraint will be shown in Section 5.3.

The function `min_lead_pitch` referred to in line 19 is defined in lines 6–8. It is recalled that the minimum lead pitch of a component is the least distance between any pair of adjacent pins of that component. In the definition for `min_lead_pitch` this is implemented as the minimum distance between any pair of different pins of that component:

```
min( [ distance( P1, P2 ) | exists P1 in Ic.pins, P2 in Ic.pins, P1 <> P2 ] ).
```

The last guideline in rulebook "ICE volume II" is implemented by the constraint named "IC Existence" defined in lines 27–28. Its only purpose here is to show that rulebooks can have multiple guideline-entries. It implements the (fictitious) rule that there should be an integrated circuit in the design.

5.3 Critiquing the Design

To illustrate the results of critiquing an example design with Galileo6, we will show what happened when the Galileo6 program from Figure 5 was used to test the design shown in Figure 6.

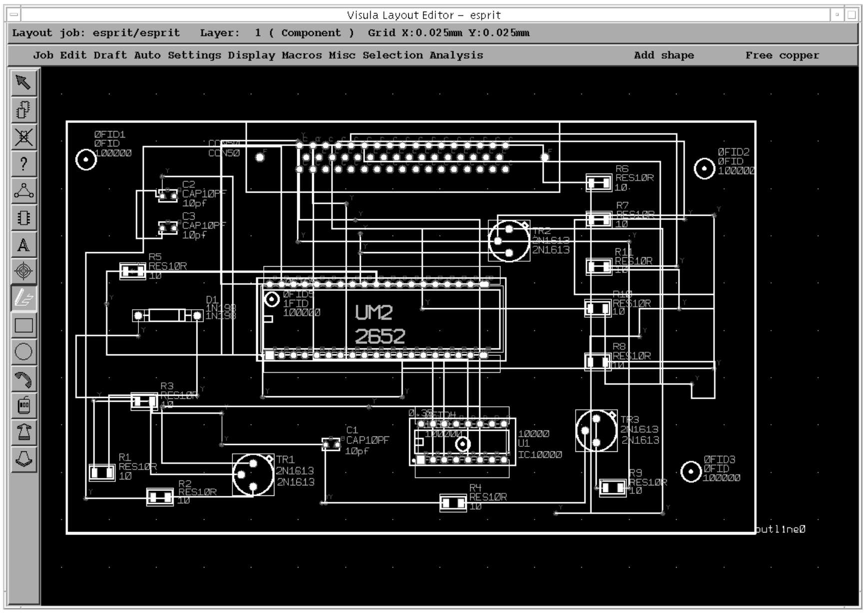

Figure 6: Toy Design

There is one constraint which is violated by this design. This is the "Min Lead Pitch" constraint defined in lines 18–25. The constraint is violated because the IC 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.

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 7). For each constraint which has been violated, a report is produced with an explanation of why it failed. These explanations, as explained in Section 3 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 8 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. This can be used to locate components violating rules.

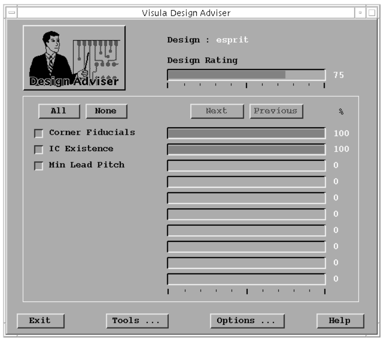

Figure 7: Design Adviser

```
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 comp where : ( compname: UM2
                 partname: 2652
                 type: ic )
```

Figure 8: Violation Report

6 User Experience

In this section we report on the experiences of D2D, one of the end-user companies in the CEDAS consortium.

D2D have been using the Visula system for several years and were provided with an early version of the Galileo6 compilation system during the summer of 1996. They received a one-day training session on Galileo6 and how to use the Galileo6 compilation system. Since then they have used Galileo6 for implementing their company-specific guidelines. Previously, the only mechanism available was to write programs in C or C++ but this was considered to be uneconomic.

After having used it for about half a year, D2D have been able to implement a number of DFA guidelines and critique recent Printed Cirquit Board designs with Galileo6. Prior to this, although they had some company-specific guidelines, these were in the form of paper documents and they had no method of checking whether their designs violated these guidelines. For the first time they have been able to systematically specify their guidelines in the form of programs and critique their designs with them.

D2D found using Galileo6 to be a straightforward and natural way of expressing their guidelines. They also found that the operation of implementing guidelines itself was useful because many of the guidelines contained ambiguities or were not very clearly stated. Obviously Galileo6 requires unambiguous rules, so the process of implementing guidelines also resulted in these guidelines being refined and improved.

As part of the exercise, some existing boards were tested by the newly defined rules and it was discovered that some of the boards did not meet certain guidelines. Whilst some of these violations were unavoidable due to other design requirements, some designs could have been easily altered to fulfil the guidelines.

They also discovered that some of their company specific guidelines were too strict. Closer inspection of some design violations showed that the designs were satisfactory and should not have been rejected. As a result of this, D2D have been able to refine some of their own rules and have been able to improve the quality of their own working rule set.

7 Concluding Discussion

We have shown how the notion of declarative constraint programming, as implemented in Galileo6, simplifies the task of writing programs which check designs for compliance with DFX criteria. We have reported on the integration of the Galileo6 language with an existing electronics CAD package called Visula. We have shown the functionality of this integration and have reported on user company experience with this system.

Our findings are that, once guidelines are understood, encoding the guidelines in Galileo6 can be done in a straightforward manner. Using Galileo6 in a systematic manner facilitates expressing guidelines in the form of programs which can be used to automatically critique designs.

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. Supporting cooperation between multiple perspectives in a constraint-based approach to concurrent engineering. *J. of Design and Manufacturing*, 1:89–105, 1991.

[Bowen and Bahler, 1992] J.A. 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.

[Ferguson *et al.*, 1997] A.B.F. Ferguson, M.R.C. van Dongen, B. O'Sullivan, and J.A. Bowen. The CEDAS Galileo6 system. Technical Report TR-UCC-CS-CPG-97-01, National University of Ireland, Cork, 1997.

[Mackworth, 1977] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

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

[van Dongen *et al.*, 1996] M.R.C. van Dongen, B. O'Sullivan, and J.A. Bowen. Using the expressive power of constraint programming to simplify the task of expressing DFX guidelines. Technical Report TR-UCC-CS-CPG-96-01, University College Cork, 1996.