Using the Expressive Power of Constraint Programming to Simplify the Task of Specifying DFX Guidelines

Marc van Dongen, Barry O'Sullivan and James Bowen
Department of Computer Science
UCC, Cork, Ireland
{dongen|osullb|J.Bowen}@cs.ucc.ie

Abstract

Increasingly, it is being realized 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 linking the Galileo6 language to an electronics design CAD package called Visula. We overview the basic concepts of Galileo6 and demonstrate its utility by showing how it can be used to encode some Design for Assembly guidelines provided by one of the partners in the ESPRIT project. These guidelines have been successfully applied to commercial PCB[1] designs provided by a company involved in our ESPRIT consortium but, in order to respect commercial confidentiality, in this paper we use a design from an electronics magazine for the purposes of illustration.

[1]Printed Circuit Board

Using the Expressive Power of Constraint Programming to Simplify the Task of Specifying DFX Guidelines[2]

Marc van Dongen, Barry O'Sullivan and James Bowen
Department of Computer Science
UCC, Cork, Ireland
{dongen|osullb|J.Bowen}@cs.ucc.ie

1 Introduction

Increasingly, it is being realized that success in manufacturing requires integration between the various phases of the product life cycle. This integration is sometimes called Concurrent Engineering (CE). One of the key aspects of CE is that, during the design of an artifact, due consideration should be given to facilitating the down-stream phases of the life-cycle. This aspect of CE 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. In Section 3, we overview the basic concepts of Galileo6, the constraint-based language that we are developing. In Section 4, we demonstrate the utility of Galileo6 by showing how it can be used to encode some Design for Assembly guidelines provided by one of the partners in the ESPRIT project. These guidelines have been successfully applied to commercial PCB[3] designs provided by a company involved in our ESPRIT consortium but, in order to respect commercial confidentiality, in this paper we use a design from an electronics magazine for the purposes of illustration. In Section 5, we provide a concluding discussion.

2 Constraints and 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 [5].

Our past research [1, 2, 4] has shown that frame-based constraint networks offer several features which make them attractive as a basis on which to build a language for Concurrent Engineering 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 organized in an inheritance hierarchy.

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

[3]Printed Circuit Board

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 Concurrent Engineering 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[4]:

> *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.*

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[5].

3 An Overview of the Galileo6 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. 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. Line 1 simply states that this is a main compilation module, that is an executable program, rather than a library defining concepts to be exported to other modules. Line 3 defines the concept of a `brick` as something which can be represented by a frame

[4] A similar guideline was used as an example in [2] but we present it again here to facilitate a self-contained paper.

[5] Indeed, this is not the end of the versatility. A user interacting with a Galileo6 program does not have to specify exact values for parameters; the system can also extract information from interval specifications. Suppose, for example, that the circuit designer has not yet decided on an exact crystal frequency although he knows it will be between 15 and 25 Mhz. He can provide this looser information by inputting the specification 15 =<`the_cpu_crystal.freq`=< 25. This statement from the user (which is also treated as a constraint) interacts with the constraint given above to cause the machine to infer that `the_test_machine.maxFreq` >= 25.

```
 1 module main(main).
 2
 3 domain brick =::= (length : real, breadth : real, height : real).
 4
 5 function volume(brick) -> real =::=
 6     {B -> V : V = B.length * B.breadth * B.height}.
 7
 8 relation small(brick) =::= {B : volume(B) < 50}.
 9
10 brick1 : brick.
11 brick2 : brick.
12 brick3 : brick.
13
14 all brick(B) : B.length > 30 implies volume(B) > 1000.
15
16 exists brick(B) : small(B).
17
18 exist 2 brick(B) : volume(B) > 500.
```

Figure 1: Simple Galileo6 program

structure that has three fields (`length`, `breadth` and `height`), each of type real; the word "domain" is Galileo6-speak for data type, the word domain stemming from the constraint processing literature [5]. Lines 5-6 define a function, `volume`, which maps from a brick to its volume. Line 8 defines a relation or predicate, `small`, over the bricks; it is specified that a brick is small if its volume is less than 50. Lines 10-12 declare the existence of three objects, `brick1`, `brick2` and `brick3`, each of type brick.

Lines 14 through 18 declare three constraints. Essentially, Galileo6 allows the full syntax of Predicate Calculus (PC) to be used for specifying constraints. Line 14 uses the standard universal quantifier from PC to specify that, for every brick, the following must be true: if its length exceeds 30 its volume must exceed 1000. Line 16 uses the standard existential quantifier from PC to specify that there must be at least one brick which is small. As well as the standard quantifiers, Galileo6 also provides some additional quantifiers; one of these is used in line 18 to specify that there must exist exactly two bricks whose volumes exceed 500.

4 An Example Application

Our purpose in this paper is to illustrate, *using real world examples*, how easy it is to express DFX guidelines in Galileo6. In section 4.1, we present some Design For Assembly (DFA) guidelines that were provided by one of the user companies in our ESPRIT CEDAS consortium. In section 4.2, we show how easily these guidelines are encoded as constraints in Galileo6. In section 4.3, we illustrate these constraints in action by applying them to a design represented in the Visula CAD package developed by the CAD vendor partner in the CEDAS project. Although the guidelines have been applied successfully to real world designs, a design taken from an electronics magazine is used for reasons of simplicity and commercial confidentiality.

4.1 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 a set of guidelines provided to us by one of our partners in the CEDAS project; here they are, in exactly the words that were used by our engineering partner:

- **Background**: *Fiducial marks are unconnected copper pads on the board that are used by assembly machines to ensure accurate placement of the components. There are two types of fiducial mark: Global* *marks are used to register the position of the board; local marks are used with fine pitch components which need greater placement accuracy.*

- **Rule 1**: *There must be three global fiducial marks positioned in three of the four corners of the board. The centre of each mark should be at least 10mm from any edge of the board.*
- **Rule 2**: *Each component that has a lead pitch*[6] *smaller than 0.025" must have two local fiducial marks positioned in diagonally opposite corners of the component. Local fiducial marks should also be at least 10mm from the edge of the board.*
- **Rule 3**: *The preferred fiducial mark is 1mm in diameter, with 4mm clearance*[7]*. for global and 3mm clearance for local marks. The clearance area should be dark and consistent in colour.*

We have not rephrased these guidelines in any way and, with the exception of the stipulation that *the clearance area should be dark and consistent in colour*, we have been able to apply all these guidelines successfully to real designs. The only reason why the "colour" requirement could not be applied is that the designs we have been asked to interface to did not store colour information. If such information had been available, then the "colour" requirement also could have been implemented.

4.2 Encoding the Guidelines in Galileo6

In the process of applying a set of guidelines to a design, encoding the guidelines correctly in the form of a program is crucial. We claim that using Galileo6 can significantly simplify the encoding of guidelines. We hope to convince the reader of this by showing how the natural language guidelines from the previous section have been encoded in Galileo6, by showing the close relationship between the constraints and the original guidelines, and by pointing out the expressive power of the language.

The guidelines in Section 4.1 have been encoded in the Galileo6 program shown in Figure 2. We will examine this program in detail below but, for now, it is useful to point out its overall structure. Line 1 simply specifies that this is a main compilation module. Lines 26 through 43 contain the constraint statements which implement the Natural Language (NL) guidelines given above. These constraint statements use some predefined symbols (such as the `<` comparison operator and the `*` multiplication operator) but they also use some application-specific concepts such as `global, distance, clearance, min_lead_pitch`. In line 3, some of these application-specific concepts are imported from a separate compilation module; others are defined in lines 5 through 24.

In considering the program in detail, we will do so in a top-down fashion, starting with the constraints and proceeding conceptually downwards to examine the definition of the application-specific concepts that are referenced in these constraints.

There are four constraint statements in this program, even though the NL statement of the guidelines appears to contain only three "rules". Normally, a Galileo6 program contains exactly one constraint statement for each NL guideline. The discrepency in this case is apparent rather than real; it is due to the fact that the first two NL "rules" each contain two sentences and each of these sentences specifies a separate guideline. This might suggest that there should be five constraints but, in fact, the second sentence of "rule" 2 specifies the same requirement as the second sentence of "rule" 1 — this is reflected in the fact that the second sentence in rule 2 contains the word "also".

The guideline expressed in the first sentence of "rule" 1,

> *There must be three global fiducial marks positioned in three of the four corners of the board.*

is implemented by the constraint

```
exist 3 C in theBoard.corners:
  exists fid( F ): global( F ) and distance( F, C) < 20 * mm.
```

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

[7] twice the distance from the centre of the fiducial to the nearest component, where the distance from fiducial F to component C equals the distance from the centre of F to the nearest pin of C

```
1  module main( main ).
2
3  import pcb_concepts( distance/2, diagonal/3, inside/2, local/1, global/1 ).
4
5  constant mm   =::=  100000.0.
6  constant inch =::= 2540000.0.
7
8  relation nearest_to( fid, position, positions ) =::=
9    { (F,Ci,Corners) : distance( F, Ci ) =< min( [ distance( F, C ) | exists C in Corners ] ) }
10   with formats ( positive 'of all ', #3, ', ', #2, ' is the nearest one to ', #1
11                  negative 'in ', #3, ', there is at least one closer to ', #1, ' than ', #2 ).
12
13 function clearance( fid ) -> number =::=
14   { F -> C : C = 2 * min( min( min( [ distance( F, Ic  ) | exists  ic( Ic  ) ] ),
15                                min( [ distance( F, Res ) | exists res( Res ) ] ) ),
16                           min( min( min( [ distance( F, Cap ) | exists cap(  Cap ) ] ),
17                                     min( [ distance( F, Con ) | exists con(  Con ) ] ) ),
18                                min( min( [ distance( F, Trn ) | exists tran( Trn ) ] ),
19                                     min( [ distance( F, Dio ) | exists dio(  Dio ) ] ) ) ) ) }
20   with format ( 'the clearance area of ', #1 ).
21
22 function min_lead_pitch( ic ) -> number =::=
23   { Ic -> L : L = min( [ distance( P1, P2) | existsdif P1 in Ic.pins, P2 in Ic.pins] ) }
24   with format( 'the minimum lead pitch of ', #1 ).
25
26 exist 3 C in theBoard.corners:
27   exists fid( F ): global( F ) and distance( F, C) < 20 * mm.
28
29 all fid( F ), E in theBoard.edges: distance( F, E ) >= 10 * mm.
30
31 all fid( F ):
32   inside( F, theBoard.edges ) and abs( F.radius - 0.5 * mm ) =< 0.1 * mm and
33   ( ( global( F ) and clearance( F ) >=  4 * mm ) or
34     ( local(  F ) and clearance( F ) >=  3 * mm ) ).
35
36 all ic( I ):
37   min_lead_pitch( I ) < 0.025 * inch implies
38     existsdif fid( F1 ), fid( F2 ):
39       local( F1 ) and local( F2 ) and
40       (existsdif C1 in I.corners, C2 in I.corners:
41          diagonal( C1, C2, I.corners ) and
42          nearest_to( F1, C1, I.corners) and nearest_to( F2, C2, I.corners ) and
43          distance( F1, C1 ) =< 0.25 * inch  and distance( F2, C2 ) =< 0.25 * inch).
```

Figure 2: Source Code for DFA program in Galileo6

which appears in lines 26-27 of the program. As will be shown later, 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 first sentence in rule 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 a global fiducial and
    the distance from F to C < 20 * mm.
```

It is hoped that the reader will agree the paraphrase is very similar to the original guideline sentence. This similarity can be achieved because the very high level nature of the Galileo6 language enables a programmer who is encoding DFX guidelines to craft his constraints so that they are conceptually similar to the original NL guidelines; the similarity of generated paraphrases then follows automatically.

However, there is a slight difference between the above paraphrase and the original guideline, which illustrates a common problem in the elicitation of expert knowledge, namely that experts rarely say what they really mean. Note that the Galileo6 constraint states that there should be three corners of the board each of which should have a global 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: not only was the meaning of "positioned in a corner" left vague but the guideline could even have been interpreted as requiring nine global fiducials, three each in each of three corners of the board.

Before we go on to consider the other constraints in this program, we will discuss some of the concepts referenced in the first constraint. The constraint refers to a field `corners` in an object called `theBoard`; it also refers to a domain `fid`. It also uses a unary predicate, `global`, and a binary function, `distance`, as well as referring to a measurement unit called `mm`.

The predicate `global` and the function `distance` are among the concepts imported from a library module called `pcb_concepts`, as can be seen in line 3. The concept `mm` is defined in line 5 as the number 100000.0 because all dimensions in Visula are specified in terms of a unit called a "design unit" which is 0.00001 of a millimetre.

Although the provenance of `global`, `distance` and `mm` is clear from the program, the two concepts `theBoard` and `fid` seem to spring from thin air. In fact, as we shall now explain, they are automatically defined by the interface to the Visula CAD package. The object `theBoard` is not defined in the program, by contrast with the objects `brick1`, `brick2` and `brick3` which were declared explicitly in the simple program shown in Figure 1. This is because the implementation of Galileo6 we are describing here has been modified (as part of the CEDAS project) to interact directly with the Visula CAD package. Therefore, since the Visula design descriptions generated by our partners always contain boards, by default in this implementation of Galileo the object `theBoard` is assumed to exist and is assumed to denote the board described in a Visula design description file. In essence, a Galileo6 program interacting with a Visula design description is treated as if it contained the following statements:

```
domain position =::= (x:integer, y:integer).
domain edge =::= (start : position, end:position).
domain board =::= (corners : sequence of position,
                     edges : sequence of edge).
theBoard : board.
```

Thus, the object `theBoard` is assumed to be a structure which contains two fields, one a list (sequence) of positions and one a list of edges; an edge is a structure which contains two fields, each of type position; a position is a structure which contains two fields, each of type integer. When a Galileo6 program containing an object called `theBoard` is applied to a Visula design description, the board attributes in the Visula description are automatically copied into the structure called `theBoard` before any of the constraints are executed.

The other concept referenced in this constraint which does not appear to be defined or imported in the program is the domain `fid`. Again, this is a concept that is automatically defined because of the Visula interface. In essence, a Galileo6 program interacting with a Visula design description is treated as if it contained the following statement:

```
domain fid =::= ( position: position,
                  radius:   integer,
                  type:     integer,
                  name:     string ).
```

The second constraint in this program, in line 29, encodes the parts of "rules" 1 and 2 which state that fiducials should be at least 10 millimeters from the edges of the board:

```
all fid( F ), E in theBoard.edges: distance( F, E ) >= 10 * mm.
```

The similarity between the NL paraphrase that would be generated from this constraint

```
It must be true that:
for any fiducial, F say, and
        edge, E say, in the edges of the board, the following is true:
  the distance from F to E >= 10 * mm.
```

and the original sentences

> *The centre of each* (global fiducial) *mark should be at least 10mm from any edge of the board.*
> *Local fiducial marks should also be at least 10mm from the edge of the board.*

speaks for itself.

One point worth noting is the difference between the arguments given to the function `distance` in this second constraint and those given to the function in the first constraint. This illustrates the fact that Galileo6 supports polymorphic functions. In the second constraint, the arguments given to the function comprise a fiducial and an edge (a structure comprising a pair of positions) while in the first constraint the arguments given to the function were a fiducial and a corner (a position).

The third constraint in this program, in lines 31-34 encodes "rule" 3 except for the "colour" stipulation which cannot be implemented because the design databases that we have been asked to interface to did not store colour information. This constraint also includes an additional requirement which was intended by the engineer who gave us the rules but which he never stated explicitly: it states that every fiducial should be inside the edges of the board. We could have made it a separate constraint but, since this constraint is already quantified over all the fiducials, this seemed a natural place to add it. Once again, there is a clear relationship between the Galileo6 statement

```
all fid( F ):
    inside( F, theBoard.edges ) and abs( F.radius - 0.5 * mm ) =< 0.1 * mm and
    ( ( global( F ) and clearance( F ) >=  4 * mm ) or
    ( local(  F ) and clearance( F ) >=  3 * mm ) ).
```

and the requirements that it expresses, as can be seen by comparing these requirements

> *The preferred fiducial mark is 1mm in diameter, with 4mm clearance for global and 3mm clearance for local marks* (and, implicitly, every fiducial should be inside the edges of the board).

with the NL paraphrase of the constraint:

```
It must be true that:
for any fiducial, F say, the following is true:
   F is inside the edges of the board and
   |the radius of F - 0.5 * mm| =< 0.1 * mm and
   F is a global fiducial and
   the clearance area of F >= 4 * mm
or F is inside the edges of the board and
   |the radius of F - 0.5 * mm| =< 0.1 * mm and
   F is a local fiducial and
   the clearance area of F >= 3 * mm.
```

The notion of "preferred" in the given rule was ambiguous and it has been interpreted to mean that the radius of each fiducial should be between 0.4mm and 0.6mm; hence the constraint statement.

One point worth noting about this constraint is the definition of the function `clearance` which it uses. When we sought clarification, we were told that the the clearance of a fiducial is twice the minimum distance of the fiducial to any other component. This is exactly how the function `clearance/1` is defined in lines 13–20. This definition contains several expressions like this:

```
min( [ distance ( F, R ) | exists res( R ) ] ).
```

Although this may appear daunting at first, it is actually standard Mathematical list notation. The domain `res` (resistor) comes with our interface to Visula (just like the domain `fid`, as explained above) so the above expression can be read as follows: *the minimum of the elements in the list of distances from* `F` *to any resistor* `R`. The implementation of Galileo6 is not finished and one feature that has not been implemented yet is domain inheritance hierarchy[8]. This means that, at the moment, for each type of component in the design, an expression of the form `min( [ distance ...]  )` has to be specified in this definition of `clearance`, so there are expressions for resistors, capacitors, connectors and so on. However, the next release of Galileo6 will include an implementation of domain inheritance and at that stage the function `clearance/1` can be defined straightforwardly as follows:

[8]In previous implementations of the Galileo family there always has been the notion of domain hierarchies, but future versions will come with extensions of this notion

```
function clearance( fid ) -> real =::=
  { F -> C: C = 2 * min( [ distance( F, Comp ) | exists component( Comp ) ] ) }
  with format ( 'the clearance area of ', #1 ).
```

The fourth constraint in this program, in lines 36-43, encodes the first sentence in "rule" 2, which specifies that if the minimum lead pitch of a component is less than 0.025", then the component should have two local fiducials in two diagonally opposite corners. The following NL paraphrase would be output if this constraint were violated:

```
It must be true that:
  for any integrated circuit, I say, the following is true:
       the minimum lead pitch of I >= 0.025 * inch
    or there exist different fiducials, F1 say, and F2 say, such that:
         F1 is a local fiducial and
         F2 is a local fiducial and
         there exist different positions, C1 say, in the corners of I, and
                                          C2 say, in the corners of I, such that:
           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
           of all the corners of I, C2 is the nearest one to F2 and
           the distance from F1 to C1 =< 0.25 * inch and
           the distance from F2 to C2 =< 0.25 * inch.
```

It is recalled that, in logic, sentences of the form `A implies B` are tantamount to sentences of the form `not( A ) or B`. This explains why the natural language explanation of this last constraint does not contain the word "implies".

4.3 Critiqueing the Design

In this section the results of critiqueing an example design with Galileo6 are presented. The Galileo6 program from Figure 2 was used to test the design shown in Figure 3.

Two constraints are violated by the design. The first one is that defined in line 29 of Figure 2. It is the constraint which states that each fiducial should be at least 10 millimeters from the edges of the board. The other constraint which is violated by the design is the one defined in lines 36–43. This constraint is violated because component UM2 in the centre of the design has a lead pitch of less than 0.025", but has only one corner with a local fiducial in it, whereas it should have local fiducials in two diagonally opposite corners.

After applying the Galileo6 program to the design, the report shown in Figure 4 is generated. For each constraint which has been violated, the report gives a description of the constraint in English, and a list of each combination of entities in the design which violated it. The descriptions of the constraints have been generated automatically from the definitions of the constraints. Format strings from relation and function definitions have been used to improve the "readability."

5 Concluding Discussion

Our purpose in this paper was to illustrate how the notion of declarative constraint programming, as implemented in the Galileo6 language, simplifies the task of writing programmes which check designs for compliance with DFX criteria. We have done this by using *real world* DFX guidelines that were given to us by a partner company in an ESPRIT consortium. These DFX guidelines were then applied to designs (only one of which was presented in this paper) by interfacing Galileo6 with the commercial CAD package that is used by the company which gave us the guidelines.

Our experience has shown us that, once guidelines have been understood and provided the necessary information is available, encoding the guidelines in Galileo6 can be done in a straightforward manner. We expect that, for an expert, encoding guidelines may be even easier than it was for us because he would not have to seek advice to interpret the rules.

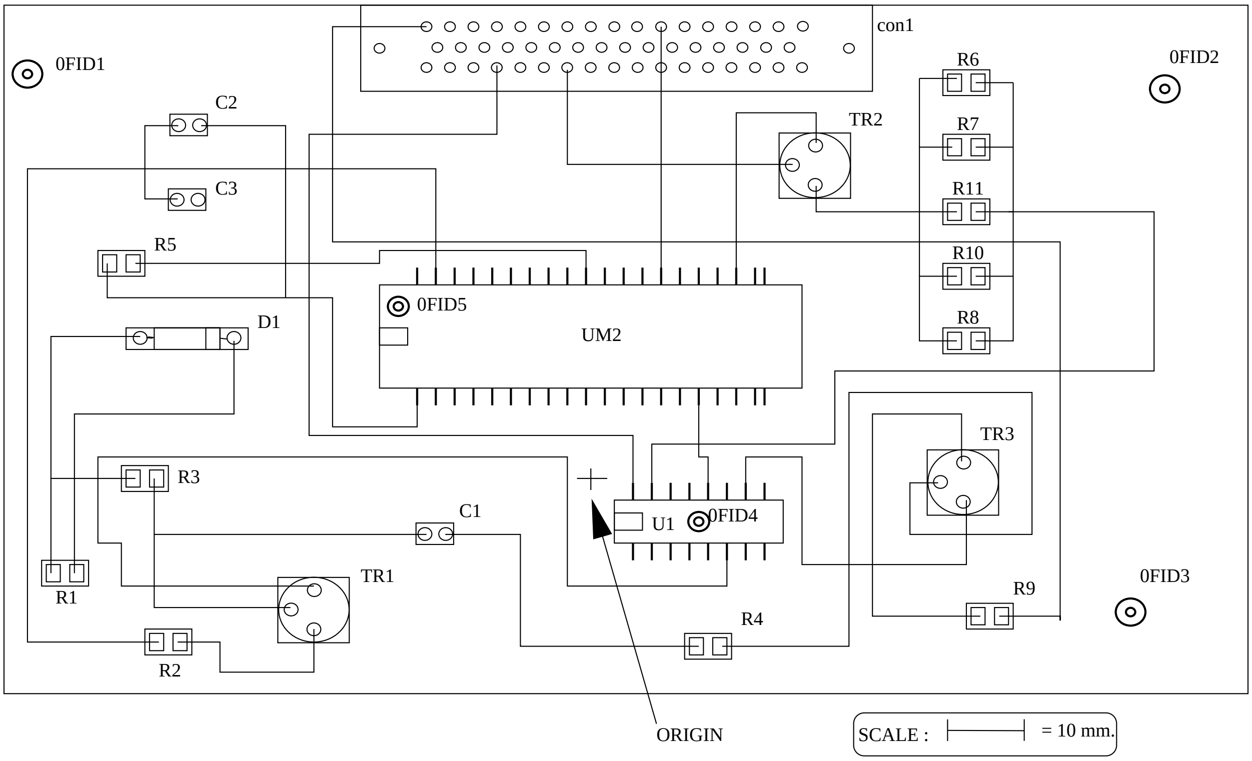

Figure 3: The Test Design

```
The following constraint was violated:
It must be true that:
for any fiducial, F say, and
        edge, E say, in the edges of the board, the following is true:
  the distance from F to E >= 10 * mm.

It was violated by each instance in the following:
 INSTANCE:
    (an edge from [  78mm,  65mm] to [ -70mm,  65mm],
     a fiducial '0FID1' at position [ -66mm,  57mm])
 INSTANCE:
    (an edge from [ -70mm,  65mm] to [ -70mm, -23mm],
     a fiducial '0FID1' at position [ -66mm,  57mm])
********************************************************************************
The following constraint was violated:
It must be true that:
for any integrated circuit, I say, the following is true:
     the minimum lead pitch of I >= 0.025 * inch
  or there exist different fiducials, F1 say, and F2 say, such that:
       F1 is a local fiducial and
       F2 is a local fiducial and
       there exist different positions, C1 say, in the corners of I, and
                                        C2 say, in the corners of I, such that:
         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
         of all the corners of I, C2 is the nearest one to F2 and
         the distance from F1 to C1 =< 0.25 * inch and
         the distance from F2 to C2 =< 0.25 * inch.

It was violated by each instance in the following:
 INSTANCE:
    (an integrated circuit 'UM2' at position [  -2mm,  22mm])
********************************************************************************
```

Figure 4: Violation Report

Although the expressive power of natural language is very high, ambiguities can arise easily. In the previous section we have shown that even *simple* guidelines such as the ones we have used in our example, can be ambiguous. Using Galileo6 as a specification language could prevent this.

References

[1] Bowen J and Bahler D, 1992. "Frames, Quantification, Perspectives and Negotiation in Constraint Networks for Life-Cycle Engineering," *Int. J. of AI in Engineering*, **7**, 199-226.

[2] Bowen J and Bahler D, 1991. "Supporting cooperation between multiple perspectives in a constraint-based approach to concurrent engineering," *J. of Design and Manufacturing*, **1**, 89-105.

[3] Bowen J and Bahler D, 1991. "Conditional Variable Existence in Generalized Constraint Networks," *Proc. 9th Natl. Conf. on Artificial Intelligence (AAAI-91)*, 251-256.

[4] Bahler D, Dupont C and Bowen J, 1994, "An axiomatic approach that supports negotiated resolution of design conflicts in concurrent engineering" *Proc. International Conference on Artificial Intelligence in Design (AID-94).*

[5] Mackworth A, 1977, "Consistency in Networks of Relations," *Artificial Intelligence*, 8, 99-118.