AGILE

Automatic Generation of Instructions in Languages of Eastern Europe

---

Title ***Final model of the CAD/CAM domain***

Authors Richard Power

Deliverable *MODL2*
Status *Draft*
Availability *Public*
Date *July 1999*

INCO COPERNICUS PL961104

Abstract:

The preliminary domain model described in Agile deliverable MODL1 has been revised and expanded. It now covers five new target texts containing more advanced material. While MODL1 focused on the programs for constructing and interrogating the T-box, in particular the Application Programmer's Interface, the present report focuses on the ontology itself. With reference to the new target texts we describe the systems of concepts employed for modelling planning schemas, actions, and domain objects.

More information on AGILE is available on the project web page and from the project coordinators:

URL:	http://www.itri.brighton.ac.uk/projects/agile
email:	agile-coord@itri.bton.ac.uk
telephone:	+44-1273-642900
fax:	+44-1273-642908

1. Introduction

In the deliverable MODL1 (Power, 1998) we explained the role of domain modelling in Natural Language Generation and the distinction between T-box and A-box; we also described an Application Programmer's Interface (API), implemented in CLOS (Common Lisp Object System), which facilitates the task of defining a T-box and editing an A-box. In the present deliverable MODL2, the emphasis shifts from form to content. Using the API and other utilities we have enlarged the T-box so that it can cover the meanings of some fairly complex procedures for using CAD/CAM applications. Five target texts indicating these procedures are given in Appendix 1.

As explained in MODL1, the CAD/CAM domain model is rooted in an 'Upper Model' (Bateman et al., 1990). The purpose of the Upper Model is to classify domain entities under abstract semantic concepts and relations that are typically expressed through syntax. The CAD/CAM domain model has a different aim, that of defining those semantic configurations that make sense in the domain. To this end, it classifies objects by the roles they can play in actions: some objects can be drawn, some can be selected, some can be clicked, and so forth.

The main part of this report describes the classification scheme for procedures, events (especially actions) and objects. Procedures determine the large-scale structure of the texts; as constituents they contain actions, typically expressed by clauses; actions in their turn refer to objects, typically expressed by nominals. A concluding section provides an assessment of the Agile domain model.

2. Procedures

A procedure is defined in the T-box as follows.

```
(define-concept PROCEDURE (INSTRUCTION-SCHEME)
    ((GOAL :type USER-ACTION)
     (METHODS :type METHOD-LIST :optional T)
     (SIDE-EFFECT :type USER-EVENT :optional T)))
```

The only obligatory attribute of a procedure is its goal. In principle an instruction could specify a goal only ('Save the drawing'), but for non-trivial texts a method for achieving the goal will also be specified: in other words, the instructions tell you not only what to do, but how to do it. Since there might be more than one way of achieving the goal, we provide a 'methods' attribute whose value is a list of methods; in most cases this list will have just one member. Finally, instructions often contain a side-effect, i.e. an incidental result of achieving the goal which is worth mentioning because it reassures users that they are on the right track. For instance, when you save a drawing, a message might appear confirming that the drawing has been saved under a given filename.

A list of methods is defined as follows.

```
(define-concept METHOD-LIST (INSTRUCTION-SCHEME)
    ((FIRST :type METHOD*)
     (REST :type METHOD-LIST :optional T)))
```

As usual, a list is defined through 'first' and 'rest' attributes so that it can be extended to any desired length. Each element of the list must be of type 'method'; in the code, an asterisk is added to this concept name since in CLOS 'method' is a protected symbol.

```
(define-concept METHOD* (INSTRUCTION-SCHEME)
    ((CONSTRAINT :type OPERATING-SYSTEM :optional T)
     (PRECONDITION :type PROCEDURE :optional T)
     (SUBSTEPS :type PROCEDURE-LIST)))
```

The 'constraint' attribute is of marginal importance: it is convenient for software instructions because different platforms often require slightly different methods. The 'precondition' attribute, also used in the Drafter project (Paris et al, 1995), specifies a sub-procedure that must be performed before the main body of the method (i.e. the substeps). Often a precondition is necessary in order to make the actions in the substeps available. For instance, when saving a drawing, you cannot enter its name unless a suitable dialog box has already been opened. The substeps comprise a sequence of sub-procedures modelled by the concept 'procedure-list'.

```
(define-concept PROCEDURE-LIST (INSTRUCTION-SCHEME)
    ((FIRST :type PROCEDURE)
     (REST :type PROCEDURE-LIST :optional T)))
```

Since the elements of a procedure list are procedures, which may have methods of their own, the T-box provides for goal-subgoal structures nested to any required depth. This is why the precondition and the steps of a method are defined as procedures rather than actions.

3. Events

Two kinds of events are distinguished: actions and experiences. Actions (e.g. the user drawing an arc) serve as the goals of procedures; experiences (e.g. the user seeing a message on the screen) serve as side-effects. Events are modelled by a concept called USER-EVENT, which subsumes all actions and experiences. All actions also descend from the Upper Model concept DISPOSITIVE-MATERIAL-ACTION, and all experiences from the Upper Model concept PERCEPTION. In fact, the only experiences currently modelled are ones of either seeing or not seeing an object on the screen. Often these will be expressed without explicitly mentioning the person undergoing the experience ('the Save dialog box appears').

All actions within procedures are subsumed by a concept called USER-ACTION, defined as follows:

```
(define-concept USER-ACTION (DISPOSITIVE-MATERIAL-ACTION USER-EVENT)
    ((ACTOR :type USER-ACTOR)))
```

Since it descends from DISPOSITIVE-MATERIAL-ACTION, USER-ACTION inherits the fundamental roles ACTOR (the person performing the action) and ACTEE (the object

that is acted upon). In an analogous way, the experiences SEE and NOT-SEE inherit the roles SENSER and PHENOMENON from the Upper Model concept PERCEPTION; in this case the SENSER is the person undergoing the experience, while the PHENOMENON is the content of the experience (e.g. the dialog box that appears).

Actions are subclassified by two criteria: the nature of the ACTEE, and any other roles that they may have in addition to ACTOR and ACTEE. The most important categories are as follows.

1. DATA-ACTION: an action for which the ACTEE is a complex data structure that might be saved, loaded, opened, or edited. Typical data structures are files, documents, and drawings.
2. LOCATED-ACTION: an action that optional includes a case role specifying some kind of location. The significance of the location can vary from one action to another. For instance, the location for CLICK is the place where you click ('Click the left mouse button *on* the icon'); instead the location for ENTER is a place where you enter text ('Enter the name *in* the Filename field').
3. SIMPLE-ACTION: a broad class of actions that only have the roles ACTOR and ACTEE; the nature of the ACTEE varies so much that it is defined individually on each action.

Apart from ACTOR and ACTEE, LOCATION is by far the commonest case role, but two other roles are occasionally used. The first is OPTIONS, which refers to the list of alternatives in a selection action (e.g. 'Choose Save from the File menu'). The other is RECIPIENT, used for the action of adding an element to some part of a drawing (e.g. to a polyline); here the ACTEE is the element that is added, and the RECIPIENT is the part of the drawing that 'receives' this element.

Although for convenience we have described the ACTOR as a person, or the ACTEE as some kind of object (e.g. a drawing or a dialog box), the actual implementation is more complex. In the latest version of the Upper Model, these attributes are filled by abstract objects representing case relations. This detail is invisible to the person editing the knowledge, who interacts with a diagram that for example shows that the ACTEE of an action is a drawing; behind the scenes, however, the knowledge editor constructs a more complex model in which the ACTEE slot is filled not by an object of type DIALOG-BOX, but by a relation between this object and the whole action. Although less intuitive, this representation is helpful during tactical generation: it means that when selecting a nominal expression referring to an object, the generator has immediate access to its case role.

Appendix 3 gives details of the A-box representations of all the events that occur in text 3. It includes full specifications of the objects filling the case roles, and thus serves to illustrate the modelling of objects as well as events.

4. Objects

All objects descend from CADCAM-OBJECT, which is subsumed by the concept NONDECOMPOSABLE-OBJECT in the Upper Model.

The objects in the CAD/CAM domain can be classified in two relevant ways: first, by their defining properties; secondly by their roles in actions. These classifications are largely independent. For example, the concept LABELLED-OBJECT covers entities that are defined through their labels; these include MENU, DIALOG-BOX, OPTION, COMMAND, as in the following phrases:

the File menu

the Multiline Styles dialog box

the Multiline Style option

the PLINE command

Although these objects are similar in that they all have labels, they can fulfill diverse roles in actions. By contrast, the concept DATA-OBJECT covers entities that can be subjected to actions like SAVE, LOAD, and EDIT; these include FILE, DOCUMENT, DRAWING and LINE. There is no reason why these concepts need to resemble one another as regards their defining properties: for example, some might have labels, some might not.

Focussing first on the classification by defining properties, there are three major classes in the CAD/CAM domain:

1. UNIQUE-OBJECT A unique object requires no properties except for its type, because usually only one instance is mentioned in any given context. Examples are MULTILINE and ARC (‘the multiline’, ‘the arc’).
2. LABELLED-OBJECT This has a ‘label’ attribute, which may be any short string of characters. Examples are MENU and DIALOG-BOX (‘the File menu’, ‘the Multiline Styles dialog box’).
3. COMPONENT-OBJECT This has an ‘owner’ attribute, the object to which they belong. Examples are END-POINT and COLOR (‘the end point of the multiline’, ‘the element's color’).
4. RELATIONAL-OBJECT A relational object is described through a relation with another object. Examples are DISTANCE-FROM and ANGLE-FROM (‘the distance of the line in relation to the end point’, ‘the angle of the line in relation to the end point’).
5. PLURAL-OBJECT A set of objects of a particular type, and may have a NUMBER attribute. This rudimentary treatment of plurals has been adopted because they occur rarely in the texts we want to generate. Examples are ELEMENTS and POINTS (‘the elements’, ‘three points’).

In the classification by role, we distinguish HARDWARE-TOOL, SOFTWARE-TOOL, and DISPLAYED-OBJECT; the latter class covers anything that can be displayed on the screen, and so subsumes most objects in the CAD/CAM domain. Among displayed objects, the major distinction is between GUI-OBJECT and CONFIGURED-OBJECT. A GUI-OBJECT is any standard component of a graphical user interface: examples are MENU, TOOLBAR and WINDOW. A CONFIGURED-OBJECT is a text or diagram that can be edited by the user: examples are DOCUMENT, STYLE-SPECIFICATION, and DRAWING.

A-box representations of all objects mentioned in text 3 are given in Appendix 3.

5. Conclusions

Although the domain model is still not large, it covers the patterns most often found in CAD/CAM instructional procedures. To a considerable extend it could be expanded further simply by subordinating new concepts directly to existing ones. The aim has been to extract maximum utility from a simple classification scheme, while complying with Occam’s razor

by avoiding unnecessary distinctions. We have accordingly not tried to capture semantic intuitions which, although valid in themselves, play no role in the generation task.

Perhaps the main inadequacy of the current CAD/CAM T-box is the poverty of its connections to the Upper Model. At present the two models are related through only three Upper Model concepts: DISPOSITIVE-MATERIAL-ACTION, PERCEPTION, and DECOMPOSABLE-MATERIAL-OBJECT (apart from the case-role concepts, which are semantically redundant and included only as a convenience in implementing the tactical generator). Since the Upper Model serves to link the domain model to the linguistic resources, we would have hoped for a far richer set of interconnections. There are two possible explanations for this flaw: first, that the Upper Model is failing to make relevant distinctions; secondly, that potential connections have not yet been specified. The second explanation is more plausible, and we will try to address this point as the generators are expanded to cover the full range of linguistic material in the target texts.

References

Bateman, J, Kasper, R., Moore, J. and Whitney, R.. (1990) A general organization of knowledge for natural language processing: the Penman Upper Model. Technical Report, USC/ISI

Keene, S. (1989) Object-oriented programming in Common Lisp: A programmer's guide to CLOS. Adison-Wesley, Reading: Massachusetts.

MacGregor, R. and Bates, R. (1987) The LOOM knowledge representation language. Proceedings of the Knowledge-Based Systems Workshop, St Louis.

Paris, C., VanderLinden, K., Fischer, M., Hartley, A., Pemberton, L., Power, R. and Scott, D. (1995) A support tool for writing multilingual instructions. Proceedings of the International Joint Conference on Artificial Intelligence, Montreal: Canada.

Power, R. (1998) Provisional model of the CAD/ CAM domain. Deliverable MODL1, Agile project.

Appendix 1

The extensions to the CAD/CAM domain model were guided by five texts, adapted from the AutoCAD user manual.

Text 1 (pp 47-48)

To create a multiline style:

First open the Multiline Styles dialog box using one of these methods:

Windows: From the Object Properties toolbar or the Data menu, choose Multiline Style.

DOS and UNIX: From the Data menu, choose Multiline style.

1. Choose Element Properties to add elements to the style
2. In the Element Properties dialog box, enter the offset of the multiline element.
3. Select Add to add the element.
4. Choose Color. Then select the element's color from the Select Color dialog box. *Choose Color. The Select Color dialog box appears. Select the element's color.*
5. Choose Linetype. Then select the element's linetype from the Select Linetype dialog box. *Choose Linetype. The Select Linetype dialog box appears. Select the element's linetype.*
6. Repeat these steps to define another element.
7. Choose OK to save the style of the multiline element and to exit the Element Properties dialog box.

Text 2 (p 46)

To draw a line and arc combination polyline:

First draw the line segment.

1. Start the PLINE command using one of these methods. Windows: From the Polyline flyout on the Draw toolbar, choose Polyline. DOS and UNIX: From the Draw menu, choose Polyline.
2. Specify the start point of the line segment.
3. Specify the end point of the line segment.
4. Enter **a** to switch to Arc mode. Then select OK in the Arc mode confirmation dialog box. *Enter **a** to switch to Arc mode. The Arc mode confirmation dialog box appears. Select OK.*
5. Specify the endpoint of the arc.
6. Enter **l** to return to Line mode. Then select OK in the Line mode confirmation dialog box. *Enter **l** to return to Line mode. The Line mode confirmation dialog box appears. Select OK.*
7. (a) Enter the distance of the line in relation to the endpoint of the arc. Enter the angle of the line in relation to the endpoint of the arc. (b) Enter the distance and angle of the line in relation to the endpoint of the arc.

8. Press Return to end the polyline.

Text 3 (p 58)

To draw an arc by specifying three points.

Start the ARC command using one of these methods:

Windows: From the Arc flyout on the Draw toolbar, choose 3 points.

DOS and UNIX: From the Draw menu choose Arc, Then choose 3 points.

1. Specify the start point by entering **endp** and selecting the line so the arc snaps to the endpoint of the line.
2. Specify the second point by entering **poi** and selecting a point to snap to.
3. Specify the endpoint.

Text 4 (p 48-9)

To specify the properties of a multiline and save the style.

From the Data menu, choose Multiline Style.

From the Data menu, choose Multiline Style. The Multiline Style dialog box appears.

First specify the properties of the multiline.

1. In the Multiline Styles dialog box, choose Multiline Properties. *Choose Multiline Properties. The Multiline Properties dialog box appears.*
2. In the Multiline Properties dialog box, select Display Joints to display a line at the vertices of the multiline. *Select Display Joints to display a line at the vertices of the multiline.*
3. Under Caps, select a line or an arc for the start point of the multiline. Then select a line or an arc for the endpoint of the multiline. Lastly, enter an angle.
4. Under Fill, select On to display a background color.
5. Choose Color. Then select the background fill color from the Select Color dialog box.
6. Choose OK to return to the Mutliline Styles dialog box. *Choose OK to return to the Multiline Styles dialog box. The Multiline Properties dialog box disappears.*

Now save the style.

1. Under Name, enter the name of the style..
2. Under Description, enter a description.
3. Select Add to add the style to the drawing.
4. Select Save to save the style to a file.
5. Choose OK and close the dialog box.

Text 5 (p 75)

To define a boundary set in a complex drawing:

1. Open the Boundary hatch dialog box using one of these methods. Windows: From the Hatch flyout on the Draw toolbar, choose Hatch. DOS and UNIX: From the Draw menu, choose Hatch.
2. Under Boundary, choose Advanced. *Under Boundary, choose Advanced. The Advanced Options dialog box appears. Choose Make New Boundary Set.*
3. In the Advanced Options dialog box, choose Make New Boundary Set.
4. At the Select Objects prompt, specify the corner points for the boundary set and press Return.
5. In the Advanced Options dialog box, choose OK.
6. In the Boundary Hatch dialog box, choose Pick Points.
7. Specify the internal point and press Return.
8. In the Boundary Hatch dialog box, choose Apply to apply.

Appendix 2

To illustrate how procedures are represented, the following print-out gives the full A-box for text 3. To focus on representation of procedures rather than actions or objects, we omit the detailed composition of actions, giving only a summary description within angle brackets. The labels a1, a2 etc. are entities, preceded by their types (e.g. a1 is of type PROCEDURE). Attributes of an entity are listed beneath it: the attribute label is followed by a colon, and the value of the attribute is given on the next line at a further level of indenting, unless it is simple enough to present on the same line.

```
PROCEDURE a1
GOAL: <draw an arc>
METHODS:
  METHOD-LIST a7
  FIRST:
    METHOD* a8
    PRECONDITION:
      PROCEDURE a9
      GOAL: <start ARC command>
      METHODS:
        METHOD-LIST a14
        FIRST:
          METHOD* a15
          CONSTRAINT: <Windows operating system>
          SUBSTEPS:
            PROCEDURE-LIST a17
            FIRST:
              PROCEDURE a18
              GOAL: <choose 3 points from Arc flyout>
        REST:
          METHOD-LIST a26
          FIRST:
            METHOD* a27
            CONSTRAINT: <DOS/Unix operating system>
            SUBSTEPS:
              PROCEDURE-LIST a29
              FIRST:
                PROCEDURE a30
                GOAL: <choose Arc from Draw menu>
              REST:
                PROCEDURE-LIST a37
                FIRST:
                  PROCEDURE a38
                  GOAL: <choose 3 points>
    SUBSTEPS:
      PROCEDURE-LIST a44
      FIRST:
        PROCEDURE a45
        GOAL: <specify start point>
        METHODS:
          METHOD-LIST a51
          FIRST:
            METHOD* a52
            SUBSTEPS:
              PROCEDURE-LIST a53
              FIRST:
                PROCEDURE a54
                GOAL: <enter "endp">
```

```
          REST:
            PROCEDURE-LIST a60
            FIRST:
              PROCEDURE a61
              GOAL: <choose line>
    SIDE-EFFECT: <arc snaps to line endpoint>
  REST:
    PROCEDURE-LIST a72
    FIRST:
      PROCEDURE a73
      GOAL: <specify second point>
      METHODS:
        METHOD-LIST a79
        FIRST:
          METHOD* a80
          SUBSTEPS:
            PROCEDURE-LIST a81
            FIRST:
              PROCEDURE a82
              GOAL: <enter "poi">
            REST:
              PROCEDURE-LIST a88
              FIRST:
                PROCEDURE a89
                GOAL: <choose point>
      SIDE-EFFECT: <start point snaps to point>
    REST:
      PROCEDURE-LIST a99
      FIRST:
        PROCEDURE a100
        GOAL: <specify endpoint>
```

Appendix 3

To illustrate how events and objects are represented, the following print-out gives the full A-box configurations for all the events in text 3. The conventions of the print-out are as described in Appendix 2.

"To draw an arc"

```
DRAW a2
ACTOR:
  USER-ACTOR a5
  RANGE:
    USER a6
ACTEE:
  GRAPHICAL-ACTEE a3
  RANGE:
    ARC a4
```

"Start the ARC command"

```
START-TOOL a10
ACTOR:
  USER-ACTOR a13
  RANGE:
    USER a6
ACTEE:
  SOFTWARE-ACTEE a11
  RANGE:
    SOFTWARE-COMMAND a12
    LABEL: "ARC"
```

"From the Arc flyout on the Draw toolbar, choose 3 points"

```
CHOOSE a19
ACTOR:
  USER-ACTOR a25
  RANGE:
    USER a6
ACTEE:
  DISPLAYED-ACTEE a20
  RANGE:
    OPTION a21
    LABEL: "3 Points"
OPTIONS:
  GUI-ACTEE a22
  RANGE:
    FLYOUT a23
    LOCATION:
      TOOLBAR a24
      LABEL: "Draw"
    LABEL: "Arc"
```

"From the Draw menu choose Arc"

```
CHOOSE a31
ACTOR:
  USER-ACTOR a36
  RANGE:
    USER a6
ACTEE:
  DISPLAYED-ACTEE a32
  RANGE:
    OPTION a33
    LABEL: "Arc"
OPTIONS:
  GUI-ACTEE a34
  RANGE:
    MENU a35
    LABEL: "Draw"
```

"choose 3 points"

```
CHOOSE a39
ACTOR:
  USER-ACTOR a43
  RANGE:
    USER a6
ACTEE:
  DISPLAYED-ACTEE a40
  RANGE:
    OPTION a41
    LABEL: "3 Points"
OPTIONS:
  GUI-ACTEE a42
```

"Specify the start point"

```
SPECIFY-COMPONENT a46
ACTOR:
  USER-ACTOR a50
  RANGE:
    USER a6
ACTEE:
  CONFIGURED-ACTEE a48
  RANGE:
    START-POINT a49
    OWNER:
      ARC a4
LOCATION:
```

DISPLAYED-ACTEE a47

"entering **endp**"

```
ENTER a55
ACTOR:
```

```
  USER-ACTOR a59
  RANGE:
    USER a6
ACTEE:
  CONFIGURED-ACTEE a56
  RANGE:
    COMMAND-LINE a57
    LABEL: "endp"
LOCATION:
  GUI-ACTEE a58
```

‘selecting the line’

```
CHOOSE a62
ACTOR:
  USER-ACTOR a66
  RANGE:
    USER a6
ACTEE:
  DISPLAYED-ACTEE a63
  RANGE:
    LINE a64
OPTIONS:
```

GUI-ACTEE a65

“the arc snaps to the endpoint of the line”

```
SNAP a67
ACTOR:
  USER-ACTOR a71
  RANGE:
    USER a6
LOCATION:
  GRAPHICAL-ACTEE a68
  RANGE:
    ARC a4
ACTEE:
  DISPLAYED-ACTEE a69
  RANGE:
    END-POINT a70
    OWNER:
      LINE a64
```

“Specify the second point”

```
SPECIFY-COMPONENT a74
ACTOR:
  USER-ACTOR a78
  RANGE:
    USER a6
ACTEE:
  CONFIGURED-ACTEE a76
  RANGE:
    POINT a77
```

```
    OWNER:
      LINE a64
    NUMBER: 2
LOCATION:
  DISPLAYED-ACTEE a75
```

"entering **poi**"

```
ENTER a83
ACTOR:
  USER-ACTOR a87
  RANGE:
    USER a6
ACTEE:
  CONFIGURED-ACTEE a84
  RANGE:
    COMMAND-LINE a85
    LABEL: "poi"
LOCATION:
  GUI-ACTEE a86
    METHOD* a8
```

"selecting a point"

```
CHOOSE a90
ACTOR:
  USER-ACTOR a94
  RANGE:
    USER a6
ACTEE:
  DISPLAYED-ACTEE a91
  RANGE:
    POINT a92
OPTIONS:
  GUI-ACTEE a93
```

"to snap to"

```
SNAP a95
ACTOR:
  USER-ACTOR a98
  RANGE:
    USER a6
LOCATION:
  GRAPHICAL-ACTEE a96
  RANGE:
    START-POINT a49
    OWNER:
      ARC a4
ACTEE:
  DISPLAYED-ACTEE a97
  RANGE:
    POINT a92
```

"Specify the endpoint"

```
SPECIFY-COMPONENT a101
ACTOR:
  USER-ACTOR a105
  RANGE:
    USER a6
ACTEE:
  CONFIGURED-ACTEE a103
  RANGE:
    END-POINT a104
    OWNER:
      ARC a4
LOCATION:
  DISPLAYED-ACTEE a102
```