# Application Study: Robot Arm Control

## An experience using distributed AIM

**Arnoud Visser**
**Michiel Wiedijk**
**Frank Tuijnman**
**Hamideh Afsarmanesh**

**November 1993**

u

**University of Amsterdam**

## 1. The Robot Arm application at UvA

In the robot arm application we deal with three agents that are heterogeneous in their representation of information. Therefore they cannot directly exchange information. Using the AIM data modeling mechanism we approach to solve this problem through the integration of the AIM schema's of three agents.

Three agents are active: a high level robot controller (HLI)[Meij91], a vision system (SCILIMAGE or SCILAIM)[Kate90] and a CAD system. They are the intelligent systems of the Archon architecture as indicated in figure 1. Sophisticated robot controllers monitor the operations of a robot. If a contingency occurs, the controller needs information about the situation, before plans can be made to recover. A good example of such valuable information is the identity of unexpected objects that are found in the working area. As soon as the identity of those objects is known, decisions how to proceed can be made. To acquire the identity of an object, the HLI has to direct a camera to the proper position, get a description of the object from Scilimage and compare it with the information produced by a solid modeling CAD system. This process is called "sensor operation *Identify Object*"

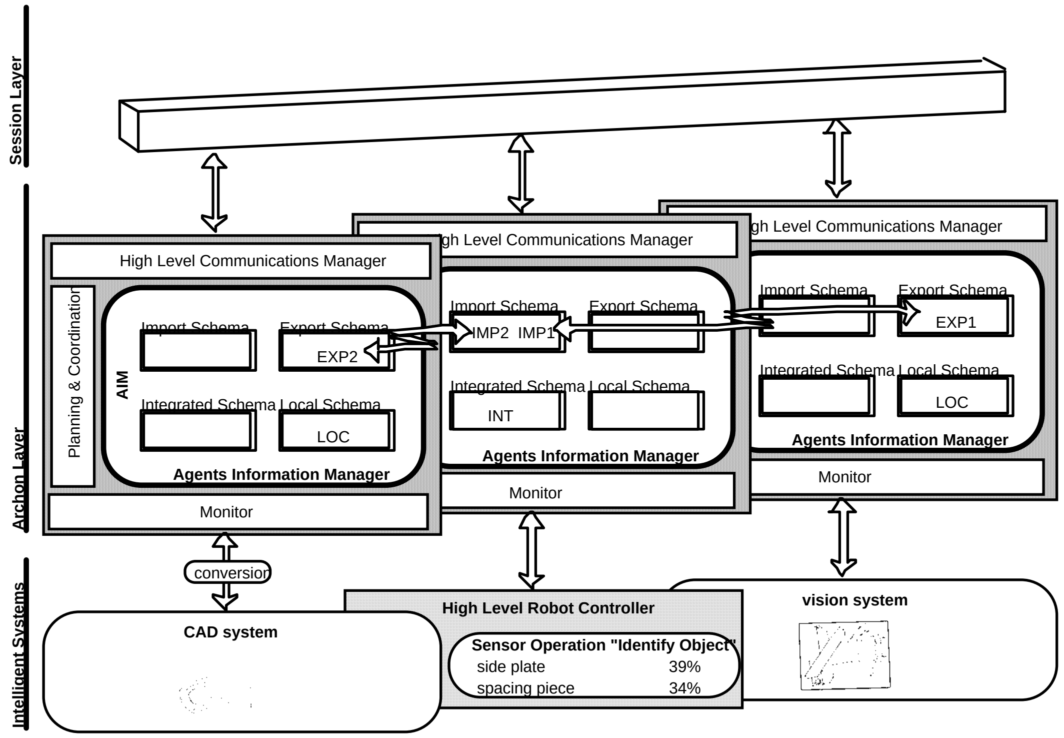

*Figure 1: The Archon architecture for the Robot Arm Application*

The main problem is that the two descriptions of the object provided by the CAD system and the vision system are different, both in semantics and representation of the world. We can generate procedures inside the sensor operation to filter out the information that one needs for the matching algorithm. However, this approach has several disadvantages:

• the algorithm becomes quite complicated because it has to take care of the details of the particular representations.
• the code of the sensor operation has to be changed for every change in the data-structure of the other agents.
• the algorithm is only applicable for those two agents. The use of an other CAD system or vision system will force the user to rewrite her/his code.

In the document UvA/Archon/TN-0013/1-93 an analysis was made on how the ARCHON layer, using the AIM module, could make the entire task easier for the sensor operation. The aim was to convert the data-structures in the import schemas of the HLI agent, which are the same as the export schema's of the other agents, to an integrated schema of the HLI agent.

The former report showed that such a strategy does not work and included the following reason: the CAD system knows only about undirected edges, while the matching algorithm requires directed edges. For every undirected edge their exist two directed edges. At present, the schema derivation operations of AIM do not support the creation of virtual objects. Consequently, it is not possible to define a transformation that doubles the number of 'edge objects' and creates for each undirected edge (in the CAD database) two directed edges (in the HLI database).

The suggestion in UvA/Archon/TN-0013/1-93 was to find a common representation that is suited for the matching process, but does not double the number of edges with respect to the other databases. In this report we have taken an slightly different approach, namely we will define a schema for the CAD-agent with directed edges. This approach works better for the implementation. Here only the CAD-database must be modified, while in the previously suggested approach the Scilimage-database and the algorithm had to be altered. This process is indicated as the module "conversion", between the Archon layer and the CAD-system in figure 1. As soon as the AIM system supports the creation of 'virtual' objects, we can return to the original CAD-database.

In chapter two a description is given of the objects that are available in the databases of the CAD and the vision system. The third chapter will give a short overview of the matching algorithm, to indicate which information is relevant for the sensor operation "Identify Object". In chapter four a description will be given of the data inside the CAD and vision databases, inclusive the needed modification for the data-structure of the CAD system. Furthermore those schemas are related to the data structure of the database of the robot controller. With those relations specified, we can facilitate the communication between the HCI and the other agents via the Archon Layer.

## 2. The objects

The Robot Arm Applications consists of a High Level robot controller, communicating with two other agents about the identity of certain objects. The sort of objects that one can expect are completely different in distinct environments. In this case we have assumed that the HLI is controlling an assembly robot, that is given the task to produce a product. As prototype of such an industrial product we have selected a benchmark that the Cranfield Institute of Technology has developed. This benchmark is specifically designed to verify the abilities of assembly robots. It contains several mechanical parts, in a large range of size and weight, which require a set of robot motions and accuracy's characteristic for the assembly of small mechanical assemblies. Seventeen parts have to be put on each other in a certain order, to yield a sort of pendulum. These parts are:

- two side plates
- four spacer pegs
- a large spacing piece
- a shaft
- a lever
- eight locking pins

The parts are initially presented on an assembly pallet which also contains an assembly support structure. Figure 2.1a shows the parts in their initial position on the base plate. The assembly is started with putting one of the side plates on the assembly support. Then any of the parts in between the side plates can be assembled, followed by the second side plate. Some constraints on the assembly sequence are that the spacing piece has to be located over two spacing pegs that need to be available first. Also the lever must be assembled after the shaft has been put in place. The final assembly is depicted in figure 2.1b.

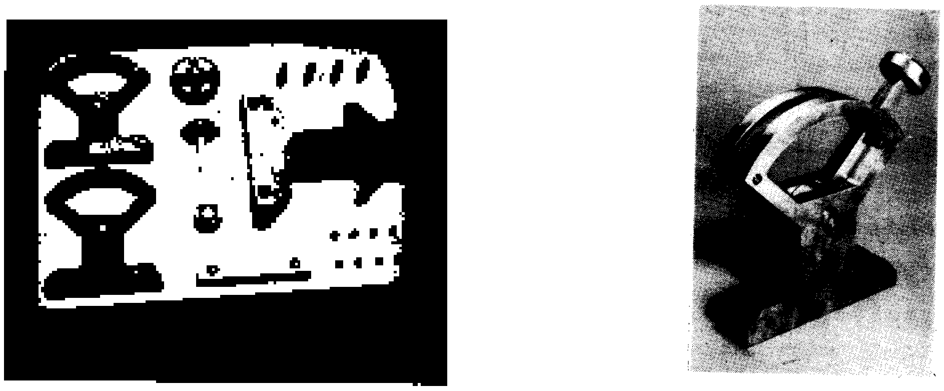

*Figure 2.1a and b: The Cranfield benchmark parts before and after the assembly.*

If we assume that halfway the assembly for some reason the spacing piece is moved from its original place, for instance by a shock. From the perspective of the camera the situation looks as in figure 2.2a. A sophisticated robot controller is able to detect that there is an obstacle on the way. Further assembly is not possible as long as the obstacle is not removed. At this time it is important to know what sort of object is in the way. Is it a piece of dirt, which may be omitted? Or is it a valuable part of the pendulum, which is needed anyway. This is the moment that the diagnosis system of the HLI activates the sensor operation "Identify Object".

To fulfil such sensor operation certain abilities are necessary. A vision system is needed that can process the image (fig. 2.2a) to a set of nodes and edges, as indicated in figure 2.2b. As you can see one of the side plate and the spacing piece are visible in the graph, one on top of the each other. Furthermore, lots of errors and inaccuracies are present. We will see that are matching routine will have difficulties with this complicated figure.

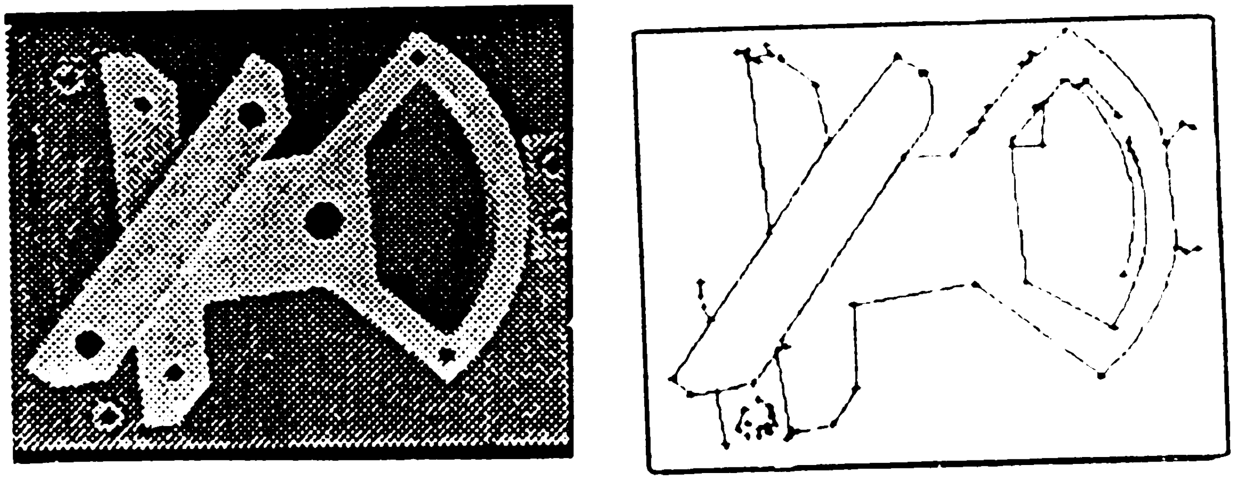

*figure 2.2a and b: picture of the assembly support structure before and after image processing.*

A matching routine will have to compare the set of edges and nodes in the image graph with the models that are available in the CAD-system. Examples of those 3D-objects are given in figure 2.3a and b.

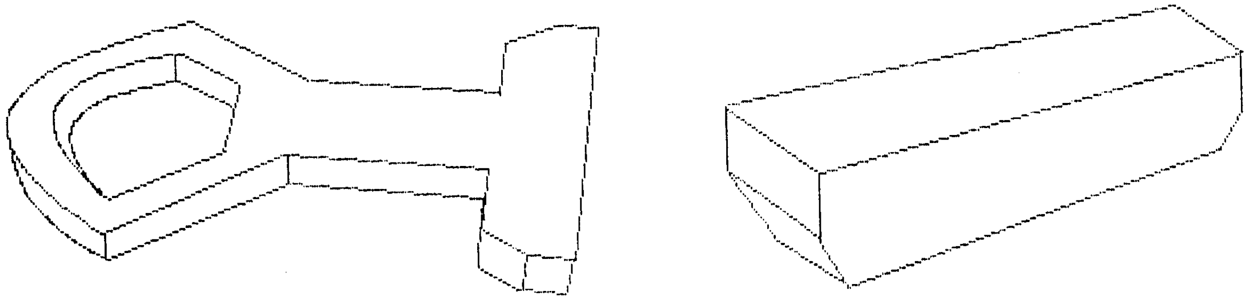

*figure 2.3a and: 3D-model of a side plate and a spacing piece.*

## 3. The matching algorithm

The basic idea behind the algorithm is quite simple. First, it puts up a hypothesis about a one-to-one-mapping between the directed edges of the object graph and the image graph. Then the algorithm compares adjacent edges in the graph to check to which amount that hypothesis is acceptable; the result is a number that expresses the probability of the correctness of the mapping. Then other hypotheses are tested. The mapping with the highest probability is returned as output.

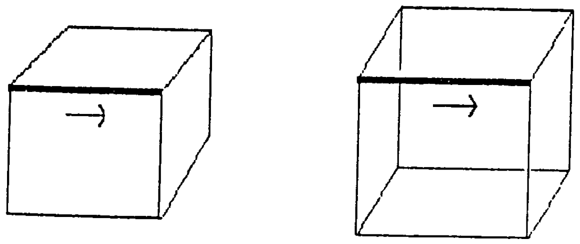

*figure 3a: an example of a (correct) hypothesis, the bold edges represent the two respective edges in the image (left) and the CAD-object (right) which will be checked in equivalence in topology and geometry.*

First, the geometrical data of the original edges is compared. Then the topological equivalence is checked by calculating the geometrical equivalence a certain set of adjacent edges. The way that the adjacent edges are scanned is the core of the algorithm. This scanning occurs in two steps. The first step is that the two faces of which the two original directed edges are a border are cycled simultaneously. These faces are called the *primary faces*, surrounded by the *primary chains*. (fig 3b).

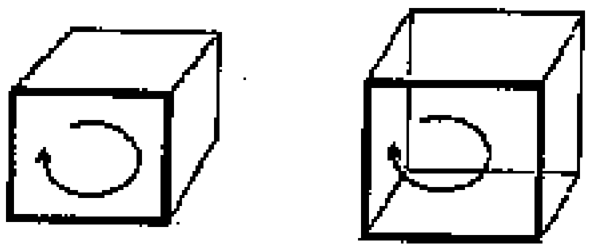

*figure 3b: The primary faces have been matched*

Thereafter, all faces adjacent directly to the primary faces are cycled. Those faces are called the *secondary faces*. (fig 3c and d).

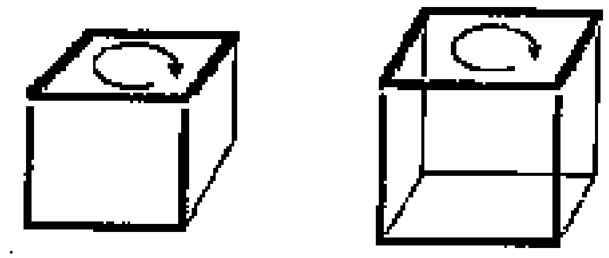

*figure 3c: The first pair of secondary faces have been matched*

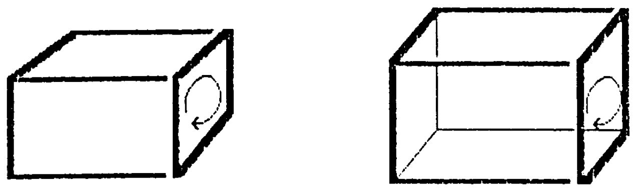

*figure 3d: The second pair of secondary faces have been matched*

For the CAD-object there are still unmatched secondary faces left, but those faces are not visible in the image. This means that no other secondary chains can be matched, and that the algorithm has concluded the calculation of the probability of this hypothesis. It can continue with the testing of another hypothesis.

To increase the speed of the algorithm, against a small price in reliability, there is a possibility to reduce the number of hypotheses that are checked. Not all hypotheses are equally probable. A hypothesis with the longest image edge and the shortest object edge is much less likely to be successful than one between edges of equal relative length. The same holds for other geometrical attributes like curvature and angles. Statistics can be used to sort out the edges on basis of a geometrical cost-function. This ordered set can be used to select a subset of all possible hypothesis, which is likely to contain good candidate hypothesis.

We can summarise the algorithm as follows:

```
• Create the start sets.
• For every pair of directed edges in these sets,
  calculate the probability that they match.
  • Cycle the primary chains, updating the score, and
    assigning matches between the edges until one of
    the chains is either exhausted or rounded.
  • For every matched pair of the primary chains,
    calculate the score of the adjacent secondary
    chains
    • Cycle the secondary chains, updating the score,
      and assigning matches between the edges until one
      of the chains is either exhausted or rounded.
• Output the score and id's of the pair that had the
  highest probability.
```

In this way the topology of the graph's are used to *guide* the matching process. After matching edges X and Y, the algorithm continues with 'some' neighbour of X and 'some' neighbour of Y. Geometry is only used to *qualify* the probability of the mapping, although the topology also contributes to that number. This approach can be justified with the fact that the measurements of geometric features on images are not very reliable. Under projection, angles and length can be reduced or blow up to a very high extent. Guidance on such an unreliable information makes an algorithm unstable. But still the geometry is necessary in the qualification. Without geometrical data the algorithm couldn't see the difference between a cube and a rectangular box, because the topological description of both objects is the same.

The scoring is calculated in the following way:

- The probability of a hypothesis is the mean of the scores of the primary and secondary chains:

$$Hypothesis\_probability = \left( \sum_{i=1}^{N\_chains} chain\_score \Big/ N\_chains \right)$$

  where N_chains is the number of visible and matched chains of the abstract graphs.

  - The score of a chain is the mean of scores of the edges-pairs, multiplied with a penalty (set on 0.5) if the chains are of different length:

$$chain\_score = \left( \sum_{i=1}^{N\_pairs} pair\_score \Big/ N\_pairs \right) \cdot penalty$$

    where N_pairs is the number of edges of the shortest chain.

    - The score of a edge_pair depends on the amount of length equality, corrected for scaling, the amount of angle-with-successor equality and the amount of curvature equality.

$$length\_eq = \left( 1 - \frac{\mathrm{abs}(\mathrm{get\_length}(e1) - \mathrm{get\_length}(e2))}{\max(\mathrm{get\_length}(e1), \mathrm{get\_length}(e2))} \right)$$

$$curvature\_eq = (1 - \mathrm{XOR}(\mathrm{get\_curvature}(e1), \mathrm{get\_curvature}(e2)))$$

$$angle\_eq = \left( 1 - \frac{\mathrm{abs}(\mathrm{get\_angle}(e1) - \mathrm{get\_angle}(e2))}{(\max(\mathrm{get\_angle}(e1), \mathrm{get\_angle}(e2)) + 180^\circ)} \right)$$

      The score is zero if one of the edges is curved and the other not, otherwise it is the mean of the length equality and the angle equality:

$$pair\_score = curvature\_eq \cdot \left( (length\_eq + angle\_eq) \Big/ 2 \right)$$

In the implementation the probabilities will be given as a percentage from 0 to 100%.

The algorithm works good for images of single objects. Although the different parts of the pendulum are sometimes quite alike (the large spacing piece resembles the 'foot' of the side plate for instance), the algorithm selects always the right object. The probability of the occurrence of selected CAD-model in the image is at least 10% higher than the probabilities of the other objects. The correctness of the algorithm reduces significantly if more several connected objects are visible on the image. For the matching algorithm the face of an other object can be interpreted as the side or secondary face of the original object. The geometrical equivalence between the faces of different objects is low, so the probability of occurrence of any object is indicated as very low by the algorithm.

# 4. Data-schemas

In this chapter we will model the stored data in the different agents for distributed Agent Information Manager (AIM)[UvA/TN11/7-92][Afsar93]. Distributed AIM is a federated, object-oriented database management system designed and implemented primarily to support the industrial automation application environments, where there is a network of cooperating agents.

Distributed AIM defines an information access/sharing mechanism that is based on a global data model, the 3DIS model, to represent the information of each agent and a global language, the 3DIS/ISL query/update language, defined on top of the data model to support database interactions among agents. Using the global data model, agents' heterogeneous 'schemas' are made homogeneous. However, the homogeneity of schema representation does not address the semantic interrelationships (loose or tight integration) that may exist among the data and knowledge of different agents. These interrelationships are established systematically and incrementally through a set of derivation/integration operations defined for distributed schema management of AIM. Clearly, the query language and its capabilities play a major role in the specification of schema integration among the agents.

Every agent is represented by several schemas. The local schema is the schema that models the data stored locally. The various import schemas model the information that is accessible from other databases. An export schema models some information that this database wishes to make accessible to other databases. Usually, an agent defines several export schemas.  The integrated schema presents a coherent view on all accessible local and remote information. The integrated schema can be interpreted as one user's global classification of objects that are classified differently by the schemas in other databases.

In this chapter, the schemas that define objects in the CAD-system, Scilimage and HLI are presented. Which information from the vision- and CAD-agents is relevant for the HLI-agent is analysed, and made available for distribution by the definition of import and export schemas. For the CAD-system we will show the original data-structure, and the data-structure after the modification from undirected edges to directed edges.

## 4.1 The CAD data-structure

There are four basic methods to describe objects: wireframes, polygonal schemes, sculptured surfaces and solid modeling [Requ80]. Wireframes and solid modeling are in widespread use, although wireframes are often considered old-fashioned, since they do not fully describe an object. A wireframe representation can be interpreted in various ways, as shown in figure 4.1.1.

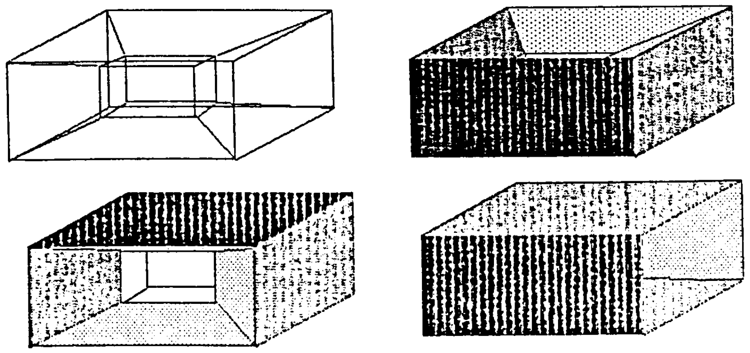
*figure 4.1.1: Ambiguity in the wireframe representation*

Solid modeling has a much more profound theoretical basis. It does not suffer from ambiguity. The two most important ones are constructed solid geometry (CSG) and boundary representation (B-rep).

With CSG, every object is represented as a composition of primitive objects like cylinders, cones, and cubes. Representing an object thus means specifying what primitive object is composed of, the dimensions of the primitive objects, their spatial relation, and the way they should be composed: union, difference, or intersection.

In the boundary representation an object is described by specifying its outer shell which is given by the faces it is composed of. The faces can be curved to allow exact modeling of round objects. In turn, every face can be described by its 'border', which is given by the edges, pieces of straight or curved line; every straight edge can be represented by stating start end and vertex.

In our application we have chosen the Modified Winged-Edge representation (MWE), which is of the B-rep type. It is described in [Weil85].

As you can see in figure 4.1.2 the object is specified by an outer shell, which is an set of faces that form the boundary of the object. Every instance of the type SHELL_3D also contains a pointer to the list of edges of those faces: the wireframe. Every frame is limited by a chain of edges. No information is stored about adjacent faces in the FACE_3D, this data is stored inside the EDGE_3D type. Only the pointers to previous and next faces in the list are stored, to support an two-directional list. An undirected edge is the boundary between two faces, a fact stored in the PAIR_OF_FACES membership. The line itself is bound between two nodes. Because the edge is undirected, no indication can be made which is the start or end-node. The only statement that can be made that if one of adjacent faces is cycled in clockwise direction, one speak of an start- and end-node, about an clockwise successor (cwe), and the angle with that successor (cwe_angle), etc. All this information has to be stored for both faces (or for both halfedges). To facilitate an consequent choice of first and the last of an pair, the selection is guided by the flags in the memberships CWEHALF and CCWEHALF.

The approach of this report was to convert all the databases in such a way, that edges were represented in a directed fashion. In this way no virtual objects had to be created. The database schema showed on the right side was the result of this research. The representation is still a boundary representation: the object is defined by the shell of faces that surrounds it. The difference is that the faces are defined by chains of halfedges, instead of undirected edges. Every undirected EDGE_3D has an pointer to its counterpart. Start and end node, clockwise and counter clockwise successor, all this information is now uniquely determined. The price we have to pay from this design is that the length and curvature, features that must be the same for both halfedges by definition, are stored twice. This cost some memory space and requires some truth maintenance routines. The advantage of this approach is the much simpler structure, which allows shorter and faster queries on all information concerning the relation between the directed edge and the other edges of its face.

With the modified schema, we are able to define an export schema (fig. 4.1.5), which the HLI-agent can use to build an integrated schema. This schema is explained in chapter 4.3.

```
define_schema LOC used_schema

  type PAIR_OF_INTS
    first_int: INTEGERS
    last_int: INTEGERS

  type PAIR_OF_REALS
    first_real: REALS
    last_real: REALS

  type NODE_3D
    node_name: STRINGS
    x: REALS
    y: REALS
    z: REALS

  type PAIR_OF_NODES
    first_node: NODE_3D
    last_node: NODE_3D

  type EDGE_3D
    edge_name: STRINGS
    chrono_succ: EDGE_3D
    nodes: PAIR_OF_NODES
    cwe: PAIR_OF_EDGES
    ccwe: PAIR_OF_EDGES
    cwehalf: PAIR_OF_INTS
    ccwehalf: PAIR_OF_INTS
    cw_angle: PAIR_OF_REALS
    ccw_angle: PAIR_OF_REALS
    faces: PAIR_OF_FACES
    curvature: REALS

  type  PAIR_OF_EDGES
    first_edge: EDGE_3D
    last_edge: EDGE_3D

  type FACE_3D
    face_name: STRINGS
    next_face: FACE_3D
    prev_face: FACE_3D
    edge_list: EDGE_3D

  type PAIR_OF_FACES
    first_face: FACE_3D
    last_face: FACE_3D

  type SHELL_3D
    shell_name: STRINGS
    face_list: FACE_3D
    chronologic_edge_list:
                       EDGE_3D

end schema LOC
```

```
define_schema LOC used_schema

  type NODE_3D
    node_name: STRINGS
    x: REALS
    y: REALS
    z: REALS

  type EDGE_3D
    edge_name: STRINGS
    start_node: NODE_3D
    end_node: NODE_3D
    chrono_succ: EDGE_3D
    otherhalf: EDGE_3D
    cwe: EDGE_3D
    ccwe: EDGE_3D
    cw_angle: REALS
    ccw_angle: REALS
    curvature: REALS
    length: REALS
    face: FACE_3D

  type FACE_3D
    face_name: STRINGS
    next_face: FACE_3D
    prev_face: FACE_3D
    edge_list: EDGE_3D

  type SHELL_3D
    shell_name: STRINGS
    face_list: FACE_3D
    chronologic_edge_list:
EDGE_3D

end_schema LOC
```

*Figure 4.1.4a and b The original and modified database schema of the CAD-agent.*

```
derive_schema EXP2 from_schema  LOC

  type EDGE_3D
    edge_name: STRINGS
    chrono_succ: EDGE_3D
    otherhalf: EDGE_3D
    cwe: EDGE_3D
    cw_angle: REALS
    curvature: REALS
    length: REALS

  type SHELL_3D
    chronologic_edge_list:
                       EDGE_3D
    shell_name: STRINGS

derivation_specification

  EDGE_3D = EDGE_3D@LOC
    edge_name = edge_name@LOC
    chrono_succ = chrono_succ@LOC
    otherhalf = otherhalf@LOC
    cwe = cwe@LOC
    cw_angle = cw_angle@LOC
    curvature = curvature@LOC
    length = length@LOC

  SHELL_3D = SHELL_3D@LOC
    chronologic_edge_list =
chronologic_edge_list@LOC
end schema EXP2
```

*Figure 4.1.5 The exported database schema of the CAD-agent.*

## 4.2 The Scilimage data-structure

The vision system Scilimage is an interactive image processing environment. In this environment the image is taken by a digital camera. On the image, some edge-detection techniques are applied to get the vertices and segments of the objects. Special care has to be taken on the lighting conditions, because every shadow will result in to additional edges. The segments are stored as directed edges, so one can speak of start and end nodes (fig 4.2.1). The angles are given with the clockwise and counter clockwise neighbours (CW_SUCC_ANGLE, CW_PRED_ANGLE). 'Straight ahead' is defined as 0□, 'straight corner to the left' as -90□. Only a part of this information, relevant for the matching algorithm is exported (fig 4.2.2)

```
define_schema LOC used_schema

  type NODE_2D
    node_2d_name: STRINGS
    x: REALS
    y: REALS

  type EDGE_2D
    edge_2d_name: STRINGS
    start_node: NODE_2D
    end_node: NODE_2D
    chrono_succ: EDGE_2D
    otherhalf: EDGE_2D
    cw_succ: EDGE_2D
    cw_pred: EDGE_2D
    cw_succ_angle: REALS
    cw_pred_angle: REALS
    curvature: REALS
    length: REALS
    at_border: STRINGS
    used: STRINGS

  type GRAPH_2D
    image_name: STRINGS
    chronologic_head: EDGE_2D
    chronologic_tail: EDGE_2D

end_schema LOC
```

*Figure 4.2.1 The  internal database schema of the SCILIMAGE-agent.*

```
derive_schema EXP1 from_schema  LOC

  type EDGE_2D
    edge_2d_name: STRINGS
    chrono_succ: EDGE_2D
    otherhalf: EDGE_2D
    cw_succ: EDGE_2D
    cw_succ_angle: REALS
    curvature: REALS
    length: REALS
    at_border: STRINGS

  type GRAPH_2D
    chronologic_head: EDGE_2D

derivation_specification

  EDGE_2D = EDGE_2D@LOC
    edge_2d_name = edge_2d_name@LOC
    chrono_succ = chrono_succ@LOC
    otherhalf = otherhalf@LOC
    cw_succ = cw_succ@LOC
    cw_succ_angle = cw_succ_angle@LOC
    curvature = curvature@LOC
    length = length@LOC
    at_border = at_border@LOC

  GRAPH_2D = GRAPH_2D@LOC
    chronologic_head = chronologic_head@LOC

end schema EXP1
```

*Figure 4.2.2 The exported database schema of the SCILIMAGE-agent.*

## 4.3 The HLI data-structure

In the abstract graph is constructed by set of points (representing the directed EDGES in the image and the CAD-model), and the relations between those points. There are two types of relations: the relations with the other points of the chain that defines a face, and the relation with the directed EDGE that represents the same boundary, but traversed in opposite direction. The first relation is stored as an pointer to the next POINT of the chain: arc_I. The second relation is stored as an pointer to the POINT representing the otherhalf of the EDGE. This relation is undirected, so care have to taken that the arc_II-relation of both POINT's are set simultaneously, pointing to each other, in an atomic transaction.

The geometrical information is stored in slots curvature, length and cw_succ_angle of the POINT's. In contrast with the schemas of the other agents, were complementary information was stored in the co-ordinates of the nodes, this is only place were geometrical data is found. This feature makes it easy to change the geometry of an object, for instance by scaling. In the abstract graph representation, topology and geometry have been separated totally.

The instances of this data structure, have a one-to-one relationship with the objects in the other agents. So instead of acquiring information about an object in the (export-)format of the other agents, followed by a conversion to the abstract graph format by sensor operation, direct access to the attributes of the abstract graph is possible via the definition of an integrated schema (fig 4.3). Conversions to the appropriate type- and map-names is defined with derivation primitives defined in AIM. This makes that the algorithm don't have to care about implementation details of other agents, but directly can query in the abstract-graph format, with the queries defined in UvA/Archon/TN-0013/1-93.

```
define_schema IMP1 same_as_export_schema EXP1 from SCILIMAGE

  type EDGE_2D
    edge_2d_name: STRINGS
    chrono_succ: EDGE_2D
    otherhalf: EDGE_2D
    cw_succ: EDGE_2D
    cw_succ_angle: REALS
    curvature: REALS
    length: REALS
    at_border: STRINGS

  type GRAPH_2
    image_name: STRINGS
    chronologic_edge_head: EDGE_2D

end_schema IMP1

define_schema IMP2 same_as_export_schema EXP2 from_agent CAD

  type EDGE_3D
    edge_name: STRINGS
    chrono_succ: EDGE_3D
    otherhalf: EDGE_3D
    cwe: EDGE_3D
    cw_angle: REALS
    curvature: REALS
    length: REALS

  type SHELL_3D
    chronologic_edge_list: EDGE_3D
    shell_name: STRINGS

end schema IMP2
```

*Figure 4.3.1 The imported database schemas of the HLI-agent.*

```
derive_schema INT from_schema IMP1,IMP2
  type POINT
    point_name: STRINGS
    chrono_succ: POINT
    arc_I: POINT
    arc_II: POINT
    succ_angle: REALS
    curvature: REALS
    length: REALS
    at_border: STRINGS

  derivation_specification
  POINT = union(EDGE_2D@IMP1,EDGE_3D@IMP2)

  point_name  = {edge_2d_name@IMP1,edge_name@IMP2}
  chrono_succ = {chrono_succ@IMP1,chrono_succ@IMP2}
  arc_I       = {cw_succ@IMP1,cwe@IMP2}
  arc_II      = {otherhalf@IMP1,otherhalf@IMP2}
  succ_angle  = {cw_succ_angle@IMP1,cw_angle@IMP2}
  curvature   = {curvature@IMP1,curvature@IMP2}
  length      = {length@IMP1,length@IMP2}
  at_border   = at_border@IMP1

  type ABSTRACT_GRAPH
    chronologic_head: POINT
    graph_name: STRINGS

  derivation_specification
  ABSTRACT_GRAPH = union(GRAPH_@D@IMP1,SHELL_3D@IMP2)

    chronologic_head = {chronologic_edge_head@IMP1,
                           chronologic_edge_list@IMP2)}
    image_name = {image_name@IMP1,shell_name@IMP@}

end schema INT
```

*Figure 4.3.2 The database schema of the HLI-agent.*

## 5 Results

The modification of the existing software to a version that makes use of the advantages of the AIM data modeling mechanism has leaded to a reduction of the needed code by a factor of 2.5. The original software approach, inclusive conversion routines, needed 2049 lines, the new approach only 796.

Further is this code more flexible in respect to the data-structures of the other agents. One can cope with modification of the export-schema of an other agent by a small change in the derivation-specifications of the integrated schema, while in the original set-up this would lead to a change in the conversion routines.

## 6 Conclusion

In this technical note a description is given how in an robot application three autonomous agents can use the AIM module of the Archon layer to facilitate the exchange the complex data structures between the agents. A complete description of the data schemas of a vision system and a CAD-system are given, and the way to relate those schemas to the data schemas of the robot controller agent. The benefits are

- a drastic reduction in the code required for data exchange
- a complete isolation of all data modelling issues in the Archon Layer
- a uniform description of all relevant data schemas used by various agents

## 7 References

| | |
|---|---|
| [Afsar93] | H. Afsarmanesh, F. Tuijnman, M. Wiedijk and L.O. Hertzberger: 'Distributed Schema Management in a Cooperation Network of Autonomous Agents' Proceedings of the 4th IEEE Internal Confererence on 'Database and Expert Systems Applications (DEXA)", Springer Verlag, LNCS 720, Sept. 1993. Also published in: Archon Technical Report TR46, 1993 |
| [Kate90] | T.K. ten Kate, R. van Balen, A.W.M. Smeulders, F.C.A. Groen, G.A. den Boer: 'SCILAIM: A mult-level interactive image processing environment' Pattern Recognition Letters 11 (1990) 429-441 |
| [Meij91] | G.R. Meijer: 'Autonomous shopflour systems, a study into exception handling for robot control' PhD Thesis, Universiteit van Amsterdam, June 1991 |
| [Requ80] | A.A.G. Requicha: 'Representations for rigid solids: theory, methods and systems' Computer Surveys, 12(4), pp. 437-464 |
| [Weil85] | K.Weiler: 'Edge-based data structures for solid modeling in curved-surface environments' IEEE Computer Graphics and Applications, 5(1), pp. 21-40. |