27

A Federated Cooperation Architecture for Expert Systems Involved in Layout Optimization *

H. Afsarmanesh, M. Wiedijk, L.O. Hertzberger
University of Amsterdam, The Netherlands
email: hamideh@fwi.uva.nl, wiedijk@fwi.uva.nl

F.J. Negreiros Gomes, A. Provedel, R.C. Martins, E.O.T. Salles
Universidade Federal do Espírito Santo, Vitória-ES - Brazil
email: negreiro@inf.ufes.br

Abstract

In shoe and handbag industry, there are many CIM activities that can at best be represented by expert systems. One example of such an activity is the optimization system that supports the cutting of regular/irregular shapes out of uniform/non-uniform base leather material. This paper focuses on and describes a layout optimization environment with five different kinds of agents, where every agent is an intelligent expert system. In order to perform the necessary tasks in the layout optimization activity, the involved agents need to cooperate, and share and exchange their information both with each other and with other agents involved in other CIM activities. In general, an agent representing a CIM subactivity can be represented either by an intelligent system, a simple file system, or an information management system. A natural architecture to represent such agents' cooperation and information sharing is a federated/distributed network of interrelated agents. In this paper we first address and describe the PEER federated architecture and then present a PEER implementation of the agents involved in the layout optimization system and the architecture to support their cooperation.

Keywords

Federated Databases, Integrated CIM Activities, Optimization Expert Systems

1 INTRODUCTION

To support the concurrent engineering approach, different CIM activities must cooperate and exchange information. For instance, the layout optimization in shoe and handbag industry requires to access the information from the design and production planning activities. The optimization expert systems addressed in this paper represent a Decision Support System based on the Visual Interactive Modeling that is further described in (Afsarmanesh et al., 1994c; Angehrn et al., 1990; Bell, 1991; Negreiros et al., 1993). This system aims to optimize the

*The research described here has been partially supported by the CIMIS.net project ECLA 004:76102.

L. Camarinha-Matos et al. (eds.), *Balanced Automation Systems*

layout of a set of moulds to be cut on a piece of raw material. To gain the best (optimized) layout, the system must be integrated with other CIM subsystems such as CAD, CAQC, PPC, and with a numeric control machine that executes the actual cutting. One approach that can support the representation of CIM environment activities and support both their inter-activity and intra-activity cooperations is to apply a federated/distributed information management architecture. Using this approach, every CIM activity is first divided into a number of sub activities and represented by several agents, where an agent is an intelligent system, a simple file system, or an information management system. Then, the sharing of information among the agents is established through integration (Import/Export mechanism). The PEER federated system used in this paper provides such integration architecture.

The intelligent optimization system described in this paper is an approach to cutting of layouts of different shapes out of the base leather material (Afsarmanesh et al., 1994c). The shapes can be either regular (rectangular) or irregular (geometry represented by polygons). Also, the base leather can itself be a uniform rectangular piece or a non-uniform partitioned surface with some defects. Several expert systems however can be defined to support all possible situations involved in cutting of layouts. In this paper, three expert systems, namely OS-IU, OS-IN, and OS-RU are defined to support different possible cases of layout optimizations. Two other kinds of expert systems are also defined to perform other closely related tasks in this application environment. Namely, there is a need for a surface inspection system, called SI, to determine the uniformity of the base leather material, and there is a need for user interface systems, called UI, to support the interactions among the human operators and the optimization system.

In this paper, first the PEER federated information management system is applied to this application to support the definition of these distinct expert systems as PEER agents. Then, a PEER federation of cooperative agents is defined that interrelates OSs, SI, and several UIs agents. Finally, the interrelations among agents are established using the SDDL integration language of PEER. The resulting network will support the access to the optimization system from any UI agent defined in the network, and will automatically support the distributed query processing on optimization data.

The PEER system (Afsarmanesh et al., 1993; Tuijnman and Afsarmanesh, 1993; Afsarmanesh et al., 1994a) described in this paper is an object-oriented federated/distributed database system designed and implemented at the University of Amsterdam. It primarily supports the complex information management requirements set by the industrial automation application environments. The research described in (Afsarmanesh et al., 1994b) describes some concurrent engineering requirements and the specific PEER capabilities to satisfy them. The federated architecture of PEER introduces an *integration facility* to support the cooperation and information sharing of autonomous CIM agents with heterogeneous data representation (Afsarmanesh et al., 1993). To better support users of the integration facility and for high level access to data and meta-data, two powerful and user-friendly interface tools are developed (Afsarmanesh et al., 1994a). The Schema Manipulation Tool and the Database Browsing Tool are both window-oriented and implemented using X-windows on SUN workstations. These interface tools support users with their access, retrieval and modification of both data and meta-data in PEER agents. More details and examples on the two tools are presented in (Wiedijk and Afsarmanesh, 1994; Afsarmanesh et al., 1994a). A prototype implementation of the PEER federated system is developed in the C language that runs on UNIX, on a network of SUN workstations.

The remaining of this paper is organized as follows. A brief description of the PEER's federated architecture is provided in Section 2. Section 3 first addresses the description of the realization of the optimization system agents, the user interface agent, and the surface inspection agent of

the layout optimization environment. Then, the PEER implementation of these agents and their integration is presented.

2 PEER FEDERATED COOPERATION ARCHITECTURE

PEER is a federated object-oriented information management system that primarily supports the sharing and exchange of information among cooperating autonomous and heterogeneous nodes (Afsarmanesh et al., 1993; Tuijnman and Afsarmanesh, 1993; Wiedijk and Afsarmanesh, 1994; Afsarmanesh et al., 1994a). The PEER federated architecture consists of a network of tightly/loosely interrelated nodes. Both the *information* and the *control* are distributed within the network. PEER does not define a single global schema on the shared information to support the entire network of database systems, unlike many other distributed object-oriented database systems (Kim et al., 1991). The interdependencies between two nodes' information are established through the schemas defined on their information; thus there is no need to store the data redundantly in different nodes.

2.1 Distributed Schema Management

Every agent is represented by several schemas; a local (LOC) schema, several import (IMPs) schemas and several export (EXPs) schemas and an integrated (INT) schema (Afsarmanesh et al., 1993). 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.

The 'local' schema is the private schema defined on the physical data in an agent. Derived from the local schema are 'export' schemas that each define a particular view on some local objects. Export schemas restructure and represent several related concepts in one schema. For every export schema, the exporter agent manages the information on agents who can access it, by keeping their agent-id, their access rights on the exported information, and the agreed schema modification conditions to notify the agents who use this export schema of its changes. To obtain access to data in another agent's export schema, an agent has to input it as 'import' schema. An import schema in one agent is the same as an export schema in another agent.

Originally, an agent's 'integrated' schema is derived from its local schema and various import schemas. In later stages of integration, instead of the local schema, the previous integrated schema will be used as base. At the level of the integrated schema, the physical distribution of information becomes hidden, and the contributing agents are no longer directly visible to the end user. Different agents can establish different correspondences between their own schema and other agents' schemas, and thus there is no single global schema for the network, unlike other "federated" database systems, such as in ORION2 (Kim et al., 1991), that define one global schema to support the entire network of agents.

The schemas for an agent are defined using the PEER Schema Definition and Derivation Language (SDDL) (Afsarmanesh et al., 1993). SDDL includes facilities for defining schemas, types, and maps, and for specifying the derivation among a derived schema and a number of base schemas using a set of type and map derivation primitives. An integrated type is constructed

from other types by the union, restrict, and subtract primitives. Map integration is accomplished by specifying a map as either a union of other maps, or as a threaded map, or some combination of these two primitives.

2.2 SDDL Schema Integration/Derivation Primitives

The following schema derivation/integration primitives define the relationships between the types and maps in the derived (or integrated) schema and the types in the base schemas (called the base types). For example, to derive the agent's integrated schema for the first time, the base schemas are the local schema and the import schemas. Now first, the integrated types must be defined from their base types to appropriately classify all objects in the integrated schema. The next step is to use map derivation expressions to define necessary mappings for each derived type, using the base mappings originally defined on the base types.

A derived type is defined by a type derivation expression. Below, the semantics of the type derivation primitives are provided, where every 'T_i' stands for a "type-name@schema-name", 'I(T_i)' represents the set of all members of type T_i, 'a' represents a member object, and the 'restriction(a)=TRUE' defined on members of type T_i that checks if 'a' satisfies the restriction.

1. Type Rename: T = T_1
   interprets as: a ∈ I(T) ⇔ a ∈ I(T_1), where T is the derived type and T_1 is a type in a base schema S_1.
2. Type Union: T = **union** ($T_1, \ldots, T_n$)
   interprets as: a ∈ I(T) ⇔ a ∈ I(T_j), where 1 <= j <= n, T is the derived type and T_j is a type in a base schema S_j.
3. Type Subtract: T = **subtract** (T_1, T_2)
   interprets as: a ∈ I(T) ⇔ a ∈ I(T_1) ∧ a ∉ I(T_2), where T is the derived type and T_1 and T_2 are types from base schemas S_1 and S_2. Due to the support of multiple inheritance in PEER, T_2 does not have to be a subtype of T_1.
4. Type Restrict: T = **restrict** (T1, restriction)
   interprets as: a ∈ I(T) ⇔ a ∈ I(T1) ∧ restriction(a)=TRUE, where T is the derived type and T_1 is a type in a base schema S_1.

A derived mapping is defined by a map derivation expression. Semantics of the primitives defined for map integration are provided below, where the union primitive is represented by '{ , }', the treading primitive is represented by '.', every 'm_i' in the following definitions stands for a "mapping-name@schema-name" and 'a.m' stands for the range-object (value) related to object 'a' by mapping 'm'.

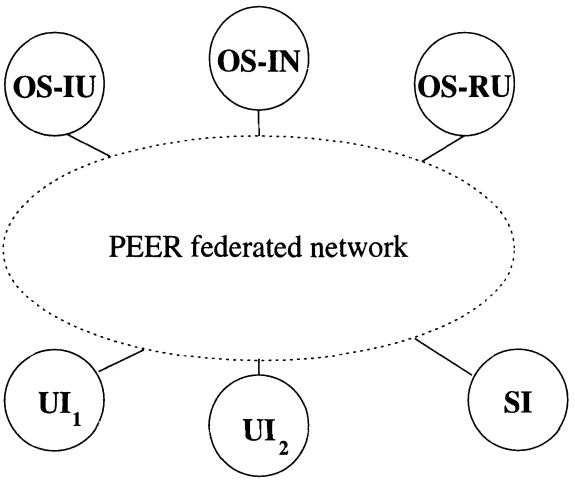
1. Map Rename: m = m_1
   interprets as: a.m = b ⇔ $a.m_1$ = b, where m is a mapping of type T in the derived schema and m_1 is a mapping of type T_1 in a base schema S_1, and where T and T_1 are not disjoint.
2. Map Union: m = { m_1,...,m_n }
   interprets as: a.m = b ⇔ $a.m_i$ = b, where m is a mapping of type T in the derived schema, 1 <= i <= n and m_i is a mapping of type T_i in schema S_i, and where for all i, 1 <= i <= n, T and T_i are not disjoint.
3. Map Threading: m = $m_1.m_2. \ldots .m_n$

interprets as: a.m = k $\Leftrightarrow a.m_1 = b \wedge b.m_2 = c \wedge ... \wedge j.m_n$ = k, where m is a mapping of type T in the derived schema, and m_i is a mapping of type T_i in a base schema S_i, and where T and T_1 may not be disjoint.

In the examples in section 3.2 the need for these primitives are illustrated. For instance, Example 1 in that section needs to apply the type and map rename and the type restrict, and Example 2 needs the type and map union primitives.

3 INTEGRATION OF OPTIMIZATION EXPERT SYSTEMS

This section describes the integration of optimization expert systems by a network of cooperating agents that together optimize the layout of moulds on leather plates. Each agent performs a specific task for which it usually needs to share and exchange information with other agents in the network. Below we describe the five kinds of agents that are involved in the optimization activity. Figure 1 shows the network of agents. A User Interface (UI) agent is the agent that



UI	User Interface Agent
SI	Surface Inspection Agent
OS-IU	Layout optimization of irregular pieces on a uniform rectangular leather plate
OS-IN	Layout optimization of irregular pieces on an irregular plate
OS-RU	Layout optimization of regular pieces on an uniform rectangular plate

Figure 1 The federated network of agents involved in the optimization of layouts

performs the interaction with the human operator and interfaces to the other agents that each perform a part of the optimization. The definition of a UI agent is based on the Visual Interactive Modeling (VIM) paradigm. This subsystem is based on the idea of Modeling by Example (MbE) as a means of enhancing cooperation between user and the system. The UI supports decision

making while taking into consideration four important dimensions of: End-user (decision-maker) modeling, Modeling as a concrete, visual and incremental process, Reactive system behavior, and Supplying active support (act as a consultant) (Afsarmanesh et al., 1994c). The Surface Inspection (SI) agent inspects the leather plates. Its function is the determination of the geometry of a plate and the detection and classification of defects (Salles et al., 1994). There are three specific optimization systems (OS-IU, OS-IN, and OS-RU) that optimize the layout of moulds on a leather plate. An optimization expert system consists of two processes: *problem processing* and *knowledge processing*. The problem processing performs the central control of the solution search process, while receiving the basic algorithms from the knowledge processing and coupling it with the interactive requirements of the user. The OS agents contain both the knowledge to perform the heuristic search, and the production rules to represent the problem (Afsarmanesh et al., 1994c). Each optimization system is specialized in the optimization of a specific combination of a mould category and plate category. The moulds can be classified as regular or irregular. The plates can be classified as uniform rectangular or irregular. Optimizer Agent OS-IU optimizes the irregular cutting on a uniform surface, it determines the allocation of irregular pieces on a uniform rectangular plate. Optimizer Agent OS-IN optimizes the irregular cutting for non-uniform surfaces by allocating irregular pieces on an irregular plate. Optimizer Agent OS-RU optimizes guillotine regular cutting for uniform surfaces by allocating regular pieces on a uniform rectangular plate.

Although it may seem that there is only one agent of each kind, there may be more than one instance of a kind of agent. For instance, the network in Figure 1 defines two User Interface agents (UI1, UI2) that support different users. Their functionalities and the information representation are in principle the same, but they differ for instance in the layout instantiations that they manage.

Section 3.1 describes the information that is managed by each agent in this network and contains a PEER schema that defines that information. In section 3.2 the integration of these agents and the information that they share and exchange is described using the PEER integration architecture.

3.1 Agent information definitions

This section describes the schema definitions for the agents in the layout optimization environment network. Please notice that in the following definitions // represents comments. The following base entities are defined in the local schema of every agent in the network. For simplicity reasons we define these base entities here once, so that they do not need to be repeated in the schema definition of every agent.

```
type MEASURE_UNIT subtype_of INTEGERS
type DEGREE_UNIT subtype_of REALS
type CLASSIFICATION subtype_of STRINGS
type WASTE subtype of REALS
type POLYGON_COMPONENT
  element: POINT
  next: POLYGON
type POLYGON subtype_of POLYGON_COMPONENT
type POINT
  x: MEASURE_UNIT
  y: MEASURE_UNIT
```

A measure unit represents a length as an integer value in a specified measure. A degree unit represents a rotation as a real value. Waste is an integer value that specifies the percentage of

leather plate material that is not used for one of the pieces. The geometry of plates, mould and defects are represented by polygons. These polygons are modeled as a closed circular linked list in counter clockwise orientation. PEER supports lists through the generic recursion abstraction as described in (Afsarmanesh and McLeod, 1989). Within th recursive definition the element mapping represents a point of the polygon and the next mapping represents the next element in the list. Points are represented by their x and y coordinates.

User Interface agent UI

The User Interface (UI) agent has four main tasks. The first task is to collect the information about leather plates that are used for the cutting of the leather using moulds. The inspection of these plates is done by the Surface Inspection agent (SI). The second task is the interaction with the user that indicates which moulds and how many of each mould have to be cut from a particular leather plate. The third task is to determine which Optimization System (OS-IU, OS-IN, or OS-RU) has to be used to optimize the layout. The decision on the kind of OS is made depending on the kind of plate and kind of moulds that are used for the layout. The fourth task is to display the resulting layouts that are generated by the OSs. The UI locally manages the information about leather plates and moulds. The local schema (LOC) definition is shown below. Section 3.2 describes how the UI agent integrates the inspection information from the SI agent and the resulting layouts from the OSs with its local information.

```
define_schema LOC
  type PLATE
  type UNIFORM_RECTANGULAR_PLATE subtype_of PLATE
    plength: MEASURE_UNIT
    pwidth: MEASURE_UNIT
  type IRREGULAR_PLATE subtype_of PLATE
    partitions: IRREGULAR_PLATE_PARTITION
  type IRREGULAR_PLATE_PARTITION
    partition_geometry: POLYGON
    associated_defects: POLYGON
    partition_classification: CLASSIFICATION
  type MOULD
    mdemand: INTEGERS
  type REGULAR_MOULD subtype_of MOULD
    mlength: MEASURE_UNIT
    mwidth: MEASURE_UNIT
    reference_point: POINT
    permitted_rotation: DEGREE_UNIT
  type IRREGULAR_MOULD subtype_of MOULD
    mgeometry: POLYGON
    permitted_rotations: DEGREE_UNIT
    mclassification: CLASSIFICATION
end_schema LOC
```

The UI agent classifies the leather plates as uniform rectangular and irregular. The uniform rectangular plate is represented by its length and width. Irregular plates are represented by partitions, where each partition is represented by the geometry of the partition (a polygon), the associated defects (a set of polygons) and a classification of the partition. The moulds are classified as regular or irregular. A regular mould is represented by its length and width and the number of times that the piece (mould) must be cut (demand). The irregular mould is represented by its geometry (polygon), demand, permitted rotations, a reference point that is defined by the lowest left corner of the mould (usually the origin), and a classification. For instance a specific irregular mould instance imould23 is represented as imould23(mdemand=3,

mgeometry=polygon45,permittedrotations=30,210, mclassification="good") with polygon instance polygon45 represents the geometry of that mould.

Surface Inspection agent SI

The Surface Inspection (SI) agent inspects the leather plates. Within the SI agent however, the leather plates are referred to as surface entities. The task of the SI agent is to determine of the surface geometry, and the detection and classification of defects. The local schema LOC of the SI agent is defined as follows.

```
define_schema LOC
  type INSPECTED_SURFACE
    surface_geometry: POLYGON
    associated_defects: DEFECT
  type DEFECT
    defect_geometry: POLYGON
    defect_classification: STRINGS
end_schema LOC
```

The inspected surfaces are represented by their geometry and the associated defects, where the defects themselves are represented by their geometry and classification.

The Optimization Systems OSs

The Optimization Systems generate an optimized layout for locating the pieces (moulds) on a plate. Following are the integrated schema definitions of the information used by each OS agent. Section 3.2 describes the integration among the UI agent and the OS agents to support the sharing and exchange of information in detail. Here we give a brief description of the integration. Each OS agent imports the plate and mould information from the UI agent. After the layout optimization is completed by the OS agent, it stores the resulting layouts with the allocation of the moulds on the plate locally and exports them to the UI agent. In the remaining of this section the three optimization systems are defined by their integrated schemas. For simplicity reasons, the local schemas of the OS agents are not presented since they all only include the layout and mould-allocation types similar to what is included in their respective INT schema.

Optimization System OS-IU: Irregular Cutting for Uniform Surface

The Optimization System OS-IU generates an optimized layout for a uniform rectangular plate and a set of irregular pieces. Following is the integrated schema of the OS-IU agent.

```
derive_schema INT from LOC, IMP-from-UI
  type PLATE
    plate_length: MEASURE_UNIT
    plate_width: MEASURE_UNIT
  type MOULD
    mould_demand: INTEGERS
    mould_geometry: POLYGON
    permitted_rotations: DEGREE_UNIT
  type LAYOUT    // irregular pieces, rectangular plate
    allocation: MOULD_ALLOCATION
    total_waste: WASTE
  type MOULD_ALLOCATION
    mould_id: MOULD
    dx: MEASURE_UNIT
    dy: MEASURE_UNIT
    theta: DEGREE_UNIT
end_schema INT
```

A layout on a rectangular plate is represented by the allocation of the moulds on the plate and the total waste of material. A mould allocation is represented by a X-axis shift (dx) and Y-axis shift (dy) of the mould (identified by mould_id) and the rotation (theta).

Optimization System OS-IN: Irregular Cutting for Non Uniform Surface

The Optimization System OS-IN generates an optimized layout for an irregular plate and a set of irregular pieces. Following is the integrated schema of the OS-IN agent.

```
derive_schema INT from LOC, IMP-from-UI
  type PLATE
    partitions: PLATE_PARTITION
  type PLATE_PARTITION
    partition_geometry: POLYGON
    associated_defects: POLYGON
    partition_classification: CLASSIFICATION
  type MOULD
    mould_demand: INTEGERS
    mould_geometry: POLYGON
    permitted_rotations: DEGREE_UNIT
    mould_classification: CLASSIFICATION
  type LAYOUT    // irregular pieces, irregular plate
    partition_allocation: PARTITION_ALLOCATION
    total_waste: WASTE
  type PARTITION_ALLOCATION
    partition_id: PLATE_PARTITION
    allocation_on_partition: MOULD_ALLOCATION
    partition_waste: WASTE
  type MOULD_ALLOCATION
    mould_id: MOULD
    dx: MEASURE_UNIT
    dy: MEASURE_UNIT
    theta: DEGREE_UNIT
end_schema INT
```

A layout on an irregular plate is represented by partition allocations and the total waste of material. A partition allocation is similar to the mould allocation on a rectangular plate, except that the waste is represented for the partition only.

Optimization System OS-RU: Guillotine Regular Cutting for Uniform Surface

The Optimization System OS-RU generates an optimized layout for a uniform rectangular plate and a set of regular pieces. Following is the integrated schema of the OS-RU agent.

```
derive_schema INT from LOC, IMP-from-UI
  type PLATE
    plate_length: MEASURE_UNIT
    plate_width: MEASURE_UNIT
  type MOULD
    mould_demand: INTEGERS
    mould_length: MEASURE_UNIT
    mould_width: MEASURE_UNIT
    permitted_rotation: DEGREE_UNIT
  type LAYOUT    // regular pieces, rectangular plate
    allocation: MOULD_ALLOCATION
    total_waste: WASTE
  type MOULD_ALLOCATION
    mould_id: MOULD
    dx: MEASURE_UNIT
    dy: MEASURE_UNIT
```

```
    theta: DEGREE_UNIT
end_schema INT
```

The layout representation here is the same as described for the OS-IU agent

3.2 Agent Integration

This section describes two examples of the integration among the UI agent and the OS agents to support the sharing and exchange of information among these agents.

Example 1: OS-IU agent integration with UI information

The first example describes the integration of the mould and plate information of the UI agent within the OS-IU agent. The OS-IU agent needs to import the irregular mould and uniform regular plate information from the UI agent. Thus, the UI agent defines the EXP-for-OS-IU export schema for this purpose as follows:

```
derive_schema EXP-for-OS-IU from_schema LOC
  type UNIFORM_RECTANGULAR_PLATE
    plength: MEASURE_UNIT
    pwidth: MEASURE_UNIT
  type MOULD
    mdemand: INTEGERS
  type IRREGULAR_MOULD subtype_of MOULD
    mgeometry: POLYGON
    permitted_rotations: DEGREE_UNIT
    mclassification: CLASSIFICATION
derivation_specification
  UNIFORM_RECTANGULAR_PLATE = UNIFORM_RECTANGULAR_PLATE@LOC
    plength = plength@LOC
    pwidth = pwidth@LOC
  IRREGULAR_MOULD = IRREGULAR_MOULD
    mdemand = mdemand@LOC
    mgeometry = mgeometry@LOC
    permitted_rotations = permitted_rotations@LOC
    mclassification = mclassification@LOC
end_schema EXP-for-OS-IU
```

Export schemas can only specify a subset of the local information. Here, the only difference between this export schema and the local schema LOC of the UI agent is omitting the generalization hierarchy for the irregular mould. The reason for this omission is that there is only one specific kind of mould defined here. The derivation specification simply specifies that the instances of uniform rectangular plate and the irregular mould are the same as in the local schema.

The OS-IU agent imports this export schema EXP-for-OS-IU as its own import schema IMP-from-UI. In its integrated schema the OS-IU agent defines the following derivation specifications. The UI agent classifies the plate and mould information as described in the derivation specification above. The OS-IU agent only knows about one kind of plate, namely the uniform rectangular, so it renames it to "plate" and similarly renames the irregular mould into "mould". The OS-IU agent defines the length and width of plates with prefix plate_ instead of prefix p, and similarly with prefix mould instead of prefix m_. The derivation renames these mappings accordingly. The value of the mclassification mapping, that is defined in the import schema IMP-from-UI, is used in the integrated schema of OS-IU as a restriction to define the MOULD

type. Namely, only the irregular-moulds for which the mclassification is "good" is of interest to this agent. Furthermore, mclassification is of no importance to the OS-IU agent as a mapping, and so it is simply discarded. Since it is not defined as a mapping in the INT schema of the OS-IU, no derivation specification is necessary to define it here. Following is a part of the derivation specification for the integrated schema (INT) of the OS-IU agent:

```
PLATE = UNIFORM_RECTANGULAR_PLATE@IMP-from-UI
  plate_length = plength@IMP-from-UI
  plate_width = pwidth@IMP-from-UI
MOULD = restrict(IRREGULAR_MOULD@IMP-from-UI,[mclassification@IMP-from-UI="good"])
  mould_demand = mdemand@IMP-from-UI
  mould_geometry = mgeometry@IMP-from-UI
  permitted_rotations = permitted_rotations@IMP-from-UI
LAYOUT = LAYOUT@LOC
  ...
MOULD_ALLOCATION = MOULD_ALLOCATION@LOC
  ...
```

Example 2: UI agent integration with OSs information

The UI agent needs to import the optimized layouts information from the OS agents. Every OS agent defines an export schema for its layout information. These schemas that are imported by the UI agent are defined as follows. Please notice that for simplicity reasons the following schemas are not fully defined. Also, the mould allocation entity is not fully represented here with its details, since it is exactly the same in each OS agent and its definition can be found in the OS agent definitions.

OS-IU

```
define_schema IMP-from-OS-IU same_as_schema EXP-for-UI from_agent OS-IU
  type LAYOUT    // irregular pieces, rectangular plate
    allocation: MOULD_ALLOCATION
    total_waste: WASTE
  type MOULD_ALLOCATION
    ...
end_schema IMP-from-OS-IU
```

OS-IN

```
define_schema IMP-from-OS-IN same_as_schema EXP-for-UI from_agent OS-IN
  type LAYOUT    // irregular pieces, irregular plate
    partition_allocation: PARTITION_ALLOCATION
    total_waste: WASTE
  type PARTITION_ALLOCATION
    partition_id: PLATE_PARTITION
    allocation_on_partition: MOULD_ALLOCATION
    partition_waste: WASTE
  type MOULD_ALLOCATION
    ...
end_schema IMP-from-OS-IN
```

OS-RU

```
define_schema IMP-from-OS-RU same_as_schema EXP-for-UI from_agent OS-RU
  type LAYOUT    // regular pieces, rectangular plate
    allocation: MOULD_ALLOCATION
    total_waste: WASTE
  type MOULD_ALLOCATION
```

```
    ...
end_schema IMP-from-OS-RU
```

The following (extended) integrated schema of the UI agent shows how the layout information imported from the OS agents is integrated by the UI agent. Every OS agent only knows about one kind of layout, that it is the one that it generates. So, it is simply named as LAYOUT. The UI agent however, has to manage three kinds of layouts, one for each of the OS agents. Therefore, it defines a generalization hierarchy to manage these layouts. The different layout entities are extended with RM or IM for regular-moulds or irregular-moulds, and with RP or IP for rectangular-plate or irregular-plate. In the derivation specification below the layouts are specified according to their origin. For instance, the LAYOUT_IM_IP specifies the instances of the LAYOUT entity of agent OS-IN, while the layouts on the rectangular plates are specified as the union of the layouts of the OS-IU and OS-RU agents.

```
derive_schema INT from_schema LOC, IMP-from-OS-IU, IMP-from-OS-IN, IMP-from-OS-RU
  // local type definitions of moulds, plates, etc. will appear here ...

  type LAYOUT
    total_waste: WASTE
  type LAYOUT_RP subtype_of LAYOUT
    allocation: MOULD_ALLOCATION
  type LAYOUT_IM_RP subtype_of LAYOUT_RP  // irregular pieces, rectangular plate
  type LAYOUT_RM_RP subtype_of LAYOUT_RP  // regular pieces, rectangular plate
  type LAYOUT_IM_IP subtype_of LAYOUT     // irregular pieces, irregular plate
    partition_allocation: PARTITION_ALLOCATION
  type PARTITION_ALLOCATION
    ...
  type MOULD_ALLOCATION
    ...
derivation_specification
  LAYOUT = union(LAYOUT@IMP-from-OS-IU, LAYOUT@IMP-from-OS-IN,
                 LAYOUT@IMP-from-OS-RU)
    total_waste = {total_waste@IMP-from-OS-IU, total_waste@IMP-from-OS-IN,
                   total_waste@IMP-from-OS-RU}
  LAYOUT_RP = union(LAYOUT@IMP-from-OS-IU, LAYOUT@IMP-from-OS-RU)
    allocation = {allocation@IMP-from-OS-IU, allocation@IMP-from-OS-RU}
  LAYOUT_IM_RP = LAYOUT@IMP-from-OS-IU
    partition_allocation = partition_allocation@IMP-from-OS-IN
  LAYOUT_RM_RP = LAYOUT@IMP-from-OS-RU
  LAYOUT_IM_IP = LAYOUT@IMP-from-OS-IN
  PARTITION_ALLOCATION = PARTITION_ALLOCATION@IMP-from-OS-IN
    ...
  MOULD_ALLOCATION = union(MOULD_ALLOCATION@IMP-from-OS-IU, MOULD_ALLOCATION@IMP-from-OS-IN,
                           MOULD_ALLOCATION@IMP-from-OS-RU)
    ...
```

The inspected surface information can be integrated by the UI similar to the layout information described and represented above, except that it is a matter of 'renaming' the import types instead of taking the union of types from several import schemas

3.3 Example application

Once the agents involved in this CIM activity are defined and their integration relations are established, agents can cooperate and simply share and exchange their information. This section describes the typical use of the resulting environment. First, we briefly summarize the needs for

information sharing and integration among the involved agents. The UI agent needs to import the surface information (which is exported) by the SI agent. The OS agents needs to import the plate and mould information which is managed locally and exported by the UI agent. The UI agent also needs to import the optimized layouts which are managed and exported by the OS agents. The UI agent then integrates the imported information from SI and OS with its local information and creates its integrated schema. Through the UI's integrated schema, the human operator can simply (and transparently) access all the information that he/she needs and thus optimize the layouts. First, the operator chooses a surface that can be used as the base for the allocation of moulds, using the techniques described earlier in this paper. When the layout is defined, the operator selects an optimization agent that can optimize the designed layout. Later, when the optimization is accomplished successfully the operator can access the resulting layout.

4 CONCLUSIONS

In this paper we have described the implementation of an integration architecture for expert systems involved in a layout optimization environment using the PEER federated information management system. First, the detailed description of PEER agents for five specific expert systems involved in this application are provided. Then the step by step integration of the shared information among these agents are described. Further, it was shown that a human operator using a User Interface agent can transparently share and exchange information with the Surface Inspection agent and the three distinct Optimization Systems, while the agents could still retain their own local information representations. The complexity of such a federated architecture is an example of a balanced automated system in which the human interaction plays an important role in the leather cutting layout design.

5 REFERENCES

Afsarmanesh, H. and McLeod, D. (1989). The 3DIS: An Extensible Object-Oriented Information Management Environment. *ACM Transaction on Information Systems*, 7:339–377.

Afsarmanesh, H., Tuijnman, F., Wiedijk, M., and Hertzberger, L. (1993). Distributed Schema Management in a Cooperation Network of Autonomous Agents. In *Proceedings of the 4th IEEE International Conference on "Database and Expert Systems Applications DEXA'93"*, Lecture Notes in Computer Science (LNCS) 720, pages 565–576. Springer Verlag.

Afsarmanesh, H., Wiedijk, M., and Hertzberger, L. (1994a). Flexible and Dynamic Integration of Multiple Information Bases. In *Proceedings of the 5th IEEE International Conference on "Database and Expert Systems Applications DEXA'94", Athens, Greece*, Lecture Notes in Computer Science (LNCS) 856, pages 744–753. Springer Verlag.

Afsarmanesh, H., Wiedijk, M., Moreira, N., and Ferreira, A. (1994b). Design of a Distributed Database for a Concurrent Engineering Environment. In *Proceedings of the ECLA.CIM workshop, Florianopolis, Brazil*, pages 35–43. (to appear in the "Journal of the Brazilian Society of Mechanical Sciences", ISSN 0100-7386).

Afsarmanesh, H., Wiedijk, M., Negreiros, F., Lopes, R., and Martins, R. (1994c). Integration of Optimization Expert Systems with a CIM Distributed Database System. In *Proceedings of the ECLA.CIM workshop, Florianopolis, Brazil.* (to appear in the "Journal of the Brazilian Society of Mechanical Sciences", ISSN 0100-7386).

Alvarenga, A., Negreiros, F., Provedel, A., Sastron, F., and Arnalte, S. (1993). Integration of an irregular cutting system into cim – part i; information flows. In *Proceedings of ECLA-CIM'93.*

Angehrn, A., Arnoldi, M., Löthi, H.-J., and Ackermann, D. (1990). *A Context-oriented Approach for Decision Support.* Lecture Notes in Computer Science (LNCS), 439. Springer-Verlag.

Bell, P. (1991). Visual interactive modelling: The past, the present, and the prospects. *European Journal of Operational Research*, 54.

Kim, W., Ballou, N., Garza, J., and Woelk, D. (1991). A Distributed Object-Oriented Database System Supporting Shared and Private Databases. *ACM Transaction on Information Systems*, 9(1):31–51.

Negreiros, F., Alvarenga, A., Lorenzoni, L., Camarinha-Matos, L., and Pinheiro-Pita, H. (1993). Object dynamic behavior for graphical interfaces in cim. In *Proceedings of ECLA-CIM'93.*

Nilsson, N. (1984). *Principles of Artificial Intelligence.* Springer-Verlag.

Pinheiro-Pita, H. and Camarinha-Matos, L. (1993). Comportamento de objetos activos na interface gráfica cim-case. In *4º PPP/AC.*

Salles, E., F.J.Negreiros-Gomes, and G.H.Brasil (1994). Segmentação de Texturas utilizando Operadores de Convoluçãoe Redes Neurais. In *Anais do I Congresso Brasileiro de Redes Neurais, Itajubá-MG Brasil*, pages 189–194.

Tuijnman, F. and Afsarmanesh, H. (1993). Management of shared data in federated cooperative PEER environment. *International Journal of Intelligent and Cooperative Information Systems (IJICIS)*, 2(4):451–473.

Wiedijk, M. and Afsarmanesh, H. (1994). The PEER User Interface Tools Manual. Technical Report CS-94-15, Dept. of Computer Systems, University of Amsterdam.