A METHODOLOGY FOR DEVELOPING AGENT BASED SYSTEMS FOR ENTERPRISE INTEGRATION

Elizabeth A. Kendall, Margaret T. Malkoun and Chong Jiang
Computer Systems Engineering
Royal Melbourne Institute of Technology
GPO Box 2476V
Melbourne, VIC 3001
AUSTRALIA
email address: kendall@rmit.edu.au

ABSTRACT

Potentially one of the most significant opportunities for enterprise integration is the recent development and advancement of agent based systems. However, before agents can be used as generic building blocks, a methodology must be established for the development of these systems. This methodology must encompass modelling, design, and implementation of the systems. This paper outlines a methodology for the software engineering of agent based systems. The methodology is based upon the IDEF (ICAM Definition) approach for workflow modelling and analysis, the CIMOSA enterprise modelling framework, and the use case driven approach to object oriented software engineering. The methodology is illustrated via a case study in the area of discrete parts manufacturing.

1.0 INTRODUCTION

Potentially one of the most significant opportunities for enterprise integration is the recent advancement of agent based systems. Numerous examples can be found in the literature of applications of agent based systems to enterprise integration (Pan and Tenenbaum, 1991); concurrent engineering (Cutkosky and Engelmore, 1992), (Tenenbaum, Weber, and Gruber, 1992), and (Olsen, Cutkosky, Tenenbaum, and Gruber, 1994); and manufacturing (Kwok and Norrie, 1993). These systems exhibit significant advances in distributed problem solving.

Before agents can be used widely used for enterprise engineering, a methodology must be established for the development of agent based systems. The need for software engineering methodologies for agent based systems has been described elsewhere (Huntbach, Jennings, and Ringwood, 1995). Software engineering is *the establishment and use of engineering principles to obtain economical software that is reliable and works efficiently on real machines* (Pressman, 1992).

Work by (Huntbach, Jennings, and Ringwood, 1995) indicated that an agent could be identified as a process or a data store in a data flow diagram with real time extensions and by a decision table (Cohen and Ringwood, 1995). Agent behavior is actually quite different from a data flow process or a data store. The need to consider *work flow and not just data flow* (Klein, M., 1995) for agents has been discussed, as agents cooperate much as individuals do in organizations or enterprises. (Klein, M., 1995) provides the following table for a comparison of dataflow and workflow representations, and it is evident that workflow representations are more suitable.

Dataflow	**Workflow**
Communication between computer systems	Business processes with human participants
Pass information	Pass tasks
Pre-defined and rigid	Can have exceptions
First in First Out queues	Task queues, reasoning about ordering/ merging
Systems are interchangeable	Agents have unique skills and positions
Requires input data	Requires task execution environment

In this paper, existing methodologies for workflow and object oriented analysis and design are extended to encompass agent oriented systems. The research proposes that as agents must be implemented in software, a workflow representation alone can not be used. An object oriented software engineering methodology is employed, but many aspects of the object oriented model have an

equivalent workflow representation that is discussed. In particular, the IDEF methodology for integrated computer aided manufacturing (ICAM), the CIMOSA modelling framework for enterprise engineering, and the use case driven object oriented software engineering (OOSE) approach are utilized. These tools have been employed because they capture the work flow and active nature of an agent system. They are also mainstream approaches that are widely used for analysis, design, and implementation.

2.0 BACKGROUND

2.1 Agents

2.1.1 Definitions

There are two views of agents: *weak* and *strong* (Wooldridge and Jennings, 1995). A weak definition of agency is items i) through iv) while a strong definition adds one of v) through vii) below:

i) *autonomous* - agents operate without direct intervention ii) *social* - agents interact with other agents iii) *reactive* - agents perceive their environment and respond to changes that occur iv*) pro-active* - agents affect their environment rather than passively allowing their environment to affect them v) *mentalistic notions* - an agent has knowledge, and desires, intentions, commitments, and choices (pro- active behavior) vi) *rationality* - an agent performs actions which further its goals vii) *veracity* and viii) *adaptability or learning*.

Strong agents are also termed reflective in that they reflect on their behavior rather than merely reacting to stimuli or changes. Strong agents are emphasized in this research, although only items i) through vi) above are considered; veracity and learning are not addressed. Strong agents reason about their beliefs to select a plan that could achieve their stated goals. Beliefs in reflective agent oriented systems are primarily represented in extensions to first order predicate calculus. Two examples are dMARS (Distributed Multiagent Reasoning System) (Georgeff, Ingrand, et. al) (Brenton, 1995) and KIF (Knowledge Interchange Format) (Genesereth, Fikes, et. al, 1992).

Agent pro-active behavior is represented in terms of a plan library. An agent selects a plan from the plan library on the basis of the goals that are to be accomplished; goals represent states of affairs that an agent wants to achieve, test for, or maintain (Georgeff and Rao, 1995). A plan is instantiated when a triggering event occurs that satisfies its invocation and context conditions, such as the posting of a new goal. An instantiated plan is an intention. The body of a plan is a set of tasks that can be subgoals, actions, assertions to and retractions from the belief database, and queries and messages to other agents. When an intention is formed, these tasks are executed by the agent in an effort to reach the stated goal.

2.1.2 Agent Cooperation

Agents operate in a distributed environment; therefore, agents require mechanisms and behavior that allow them to communicate with other agents. KQML (Knowledge Query and Manipulation Lanugage) (Finin, Weber, et al, 1993) is a language for communication among agent based programs. With KQML, agents can query for and offer information, along with other options.

Agents must cooperate and negotiate with each other. When agents negotiate, they engage in conversations that can be represented in speech acts and state machines. Languages have been developed to support this. COOL is an extension to KQML (Barbuceanu and Fox, 1995), and with it agents can make proposals and counterproposals, accept and reject goals, and notify other agents of goal cancellation, satisfaction, or failure. Another language, AgenTalk (Kuwabara, Ishida, and Osato, 1995), has an inheritance or specialization mechanism for introducing new protocols as extensions of predefined ones.

Related work by (Dunskus et al., 1995) has resulted in a preliminary categorization of the basic types of agents. Selectors, Estimators, Evaluators, Critics, Praisers, and Suggestors have been considered as narrow scope, or single function agents. This approach is valuable in that it addresses given applications at a more abstract level and allows new problems to be viewed as extensions or specializations of problems that have been seen before. Basic types of negotiation between these agents, in particular conflict resolution (Klein, 1991), have also been identified.

2.2 Object Oriented Software Engineering

2.2.1 Definition

In a complete view, the notion of an object encompasses all of the following features (Booch, 1994):

1. *Compound objects* - an object can have properties or attributes that are in fact objects themselves.
2. *Inheritance* - an object type can have subtypes (classes can have subclasses).
3. *Encapsulation* - an object can hide or protect data and behavior
4. *Identity* - a system can distinguish between two objects that "look" the same.
5. *State* - an object's behavior is dependent on its history
6. *Behavior* - an object can act and react through receiving and sending messages

2.2.2 Object Oriented Analysis

During object oriented analysis, an object oriented model is developed for a given application. The Object Modelling Technique (OMT) (Rumbaugh et al., 1991), consists of three complementary models: the object, dynamic and functional models. The object model indicates objects and relationships between objects. The dynamic model includes scenarios and event traces of object interactions. The functional model resembles a data flow diagram, with passive objects as data stores. State transition diagrams are used to provide an overall view of an object's dynamic behavior and to gain insight into inheritance. Subclasses inherit the state diagram of a superclass, adding states and transitions. Therefore, the protocols of AgenTalk (section 2.1.2) are in fact objects that can be specialized using inheritance, as stated in (Kuwabara, Ishida, and Osato, 1995).

In the use case approach, also called OOSE (Object Oriented Software Engineering) (Jacobson, 1992), a use case is a scenario or a description of how users interact with the system in a certain mode of operation. A user in a particular role is known as an actor, and each actor *uses* one or more use cases. Objects relevant to a use case are identified, and their dynamic behavior is analysed. One use case can *extend* another one, meaning that it is inserted into the first use case, adding to its definition. Extend relationships can be conditional, occurring under only certain circumstances. Use cases can also be described at varying levels of abstraction, and inheritance can be utilized to provide specialization. A use case may be purely abstract, or it may be concrete and therefore able to be instantiated.

2.3 Enterprise Modelling Methodologies

Methodologies have been proposed for enterprise modelling. The IDEF (ICAM Definition) method (Bravoco and Yadav, 1985a and 1985b) is a standard for requirements definition in manufacturing and other organizations. IDEF comprises three modeling methodologies: the function model ($IDEF_0$), information model ($IDEF_1$), and dynamic model ($IDEF_2$). In the function model, the manufacturing or other organization is described in terms of a hierarchy of decisions, activities, or actions. Each function is given an ICOM (Input, Control, Output, and Resource or Mechanism) representation.

The CIMOSA modelling framework (Computer Integrated Manufacturing Open System Architecture) (Kosanke, 1992) centers on types of major constructs. In CIMOSA, an enterprise activity is analogous to an IDEF function, except it can have control input and output, and resource or mechanism input and output. Both the CIMOSA and $IDEF_0$ models can be taken to a high degree of detail, as shown in Figure 1. Here, an $IDEF_0$ function includes decisions that are made on the basis of control input. Decide Action chooses a method or activity (Method 1 or Method 2), based upon the Input and the Control Input.

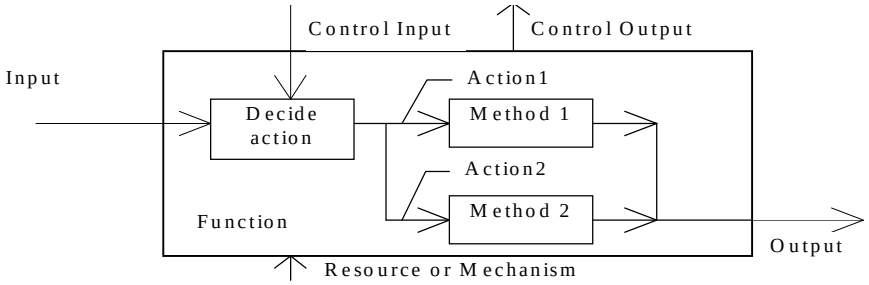

Figure 1: Detailed Function in $IDEF_0$

3.0 APPROACH

3.1 Object Oriented Software Engineering and Enterprise Modelling Methodologies

The three views of the IDEF methodology all have counterparts in object oriented software engineering. If the $IDEF_0$ function model is reexpressed in terms of an OMT functional model, active objects and passive objects (data stores) can be distinguished. With this approach, active input and output objects are identified, and mechanisms can also be modelled as active objects.

When use case diagrams and $IDEF_0$ diagrams are completed to the same level of detail, IDEF functions appear directly as a use case. In an $IDEF_0$ diagram, use case actors would appear as resources or mechanisms. The objects that appear within the use case are the same as those found in the $IDEF_1$ information model. Use case extensions are the same as multi-level (parent and child) $IDEF_0$ diagrams. When actors and objects manage a given use case, they generate control information and pass it on as input to other components of the system. In this way, policy or decision making actors and control objects carry out enterprise activities with control output.

3.2 Agents and Objects

3.2.1 The Difference Between Agents and Objects

Agents are real time, autonomous objects that carry out pro-active behavior. Each object sequentially executes methods, whereas a strong agent encompasses automated reasoning that allows it to "intelligently" choose one alternative. Objects encompass data and behavior; agents have beliefs, goals, plans and intentions. A reflective agent performs automated reasoning to select a plan; an object does not reason. An object simply responds to messages and events and executes its encoded behavior, step by step.

Because agents are concurrent, the use of inheritance in agent software requires avoidance of the inheritance anomaly (Matsuoka and Yonezawa, 1993). Synchronization behavior that works in an instance of a superclass will not necessarily provide accurate synchronization in an instance of a subclass; superclass synchronization behavior must often be redefined at the subclass level, negating any benefit of inheritance. agent behavior that doES not deal with synchronization can be inherited.

Objects communicate via unstructured messages, but agents employ KQML or a similar language to communicate through structured, meaningful messages. Objects that recognize and respond to KQML or other structured messages can cooperate with agents. Likewise, agents that can employ unstructured messaging or an object messaging convention can cooperate with objects. The items discussed above can be summarized by the following:

Agents and objects:
Compound - agents and objects can have properties or attributes that are objects or agents themselves.
Encapsulation - objects and agents can hide or protect data and behavior
Identity - objects and agents have identity that is not determined solely by their attribute values.
State - objects and agents have state. Their behavior is influenced by their history.
Behavior - objects and agents can act and react through receiving and sending messages
Objects:
Inheritance - an object type can have subtypes but synchronization behavior can not be inherited.
Unstructured messages - objects communicate through unstructured messages
Agents:
Reasoning - agents perform automating reasoning to "intelligently" choose between options
Pro-active behavior - agents have beliefs, knowledge, goals, plans, and intentions
Concurrency - agents act in a distributed manner
Structured messages - agents communicate with each other through structured messages

3.2.2 Objects as Beliefs, Sensors, and Effectors

The first relationship between agents and objects is at the belief level. Passive objects that merely store attributes frequently appear in belief databases (Georgeff and Ingrand, 1990). The Ontolingua project (Gruber, 1992) has produced a software system that can translate frame based knowledge representations into a KIF beliefs database, and vice versa. (A frame, as an object, exhibits the static aspects but differs in other ways.) Ontolingua can directly translate the static properties of objects that appear in an agent's belief database into KIF and other predicate calculus representations.

At a higher level, objects with encapsulation and dynamic behavior can be utilized by agents to initiate and to carry out pro-active behavior. In this regard, active objects can be sensors and effectors for agents. Objects in the environment impact the sensors and are impacted by the effectors, and all interaction between the objects is accomplished through message passing. Sensor objects message an agent with information about the environment that may impact the agent's beliefs and knowledge. These changes to the agent's beliefs and knowledge may in turn cause triggering events to fire, instantiating a plan into an intention. Effector objects carry out an agent's intentions, impacting objects in the environment.

3.2.3 Actors, Agents, and Use Cases

Use cases depict how humans and/ or organizations enter and interact with an object oriented system. One use case represents one mode of operation for the system, with each actor as a user(s) in a certain role. Agents enter applications to augment or automate human reasoning. For a full requirements analysis, use cases must be specified for each scenario of the application. At this level of detail each use case maps to an actor's (or agent's) plan. Conditional extends relationships --- when one use case is inserted into another one if certain circumstances exist --- correspond to context conditions that determine when an agent's plan is to be intended.

When two or more actors appear in a use case, they collaborate or cooperate via their own message passing and negotiation. In an IDEF representation, resources or individuals collaborate either by appearing in the same functional block or by exchanging input and output, with one function's output serving as control input to another function, and perhaps vice versa. In the agent base system, the actors and resources become the agents and their negotiation and information exchange becomes a coordination protocol. Use cases can be abstracted and specialized, as can coordination protocols; both use cases and coordination protocols are in fact objects or classes of objects.

3.3 An Agent Oriented Methodology

The key results are summarized in Figure 2 where Figure 1 is modified to indicate the corresponding agent representation. The inputs and outputs are the same, but the interior to the functional block now shows an agent. The agent has beliefs, goals, plans, and an interpreter, with passive objects shown as beliefs. Instantiated plans become intentions via the interpreter. Three kinds of intentions are indicated. Task 1 involves coordination with other agents, while task 3 invokes an effector. Task 2 impacts the agent's own beliefs. A sensor and an effector are also shown, with the sensor taking input from an active object and the effector impacting an active object. A use case extension diagram is depicted, and this translates into the agent's plans.

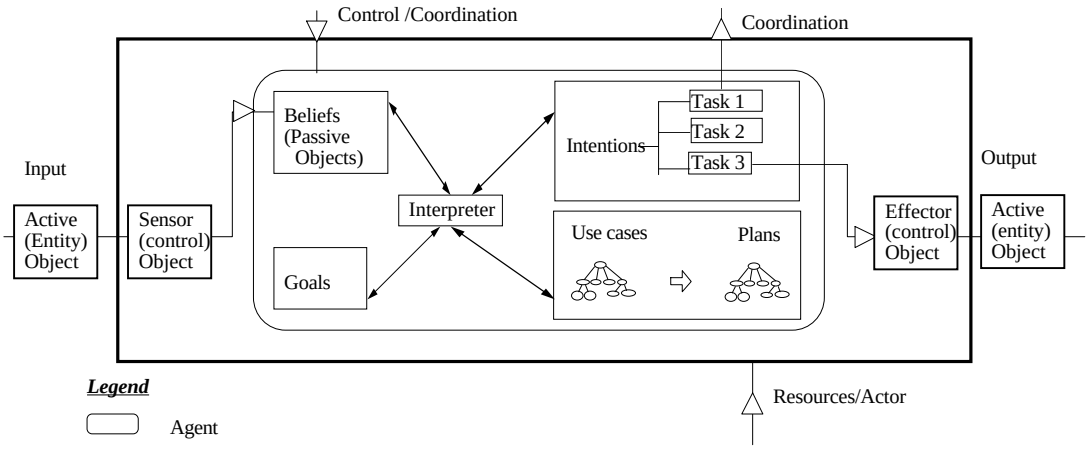

Figure 2: The Correspondence between Enterprise Activities, Use Cases and Agents

The correspondence between IDEF, use case and agent system components is summarized in the following table.

IDEF Model	Use Case (OOSE)	Agent Oriented System
Function with Control Output	Use Case and/ or Use Case Extension	Goal and Plan
Functional Input	- Entity Object - Actor Input	- Coexisting Object via Sensor - Belief
Control Input	- Entity Object via Control Object - Actor Input - Conditional Use Case Extension	- Coexisting Object via Sensor - Belief - Context or Invocation Condition
Control Output	- Control Object to Entity Object - Control Object to Actor	- Goal or Subgoal - Effector to Coexisting Object
Functional Output	- Control Object to Entity Object - Control Object to Actor	- Effector to Coexisting Object
Resource	Actor	Agent
More than one resource per function or resource information exchange	- More than one actor per use case - Use case event trace - Use case abstraction and specialization (inheritance)	- Agent collaboration - Coordination protocol - Coordination protocol abstraction and specialization (inheritance)
Information Model	Objects	- Beliefs and Coexisting Objects

4.0 CASE STUDY

The methodology is illustrated in a case study from discrete parts manufacturing. Customers place orders for parts, and the parts are selected for processing and grouped into batches. A batch is assigned to a machine for processing, and the cost of a batch is determined on the basis of material used and wasted, and on handling and processing. There are high quality and normal quality batches. Normal parts can be formed on a high quality machine, but not vice versa.

4.2 IDEF Model

$IDEF_0$ diagrams are provided in Figures 3, 4, and 5. In Figure 3, forming is not shown, and the system has five functions: Create Parts, Select Parts and Sheets, Create Batch, Assign Machine, and Determine and Evaluate Cost. Input and output are indicated, along with resources and control. Control feedback is an important element of the system, as costing impacts the policies used for selection, batching, scheduling (assigning) machines, and costing itself. These feedback paths are two way, with selection, batching and scheduling policies also impacting the cost evaluation policy.

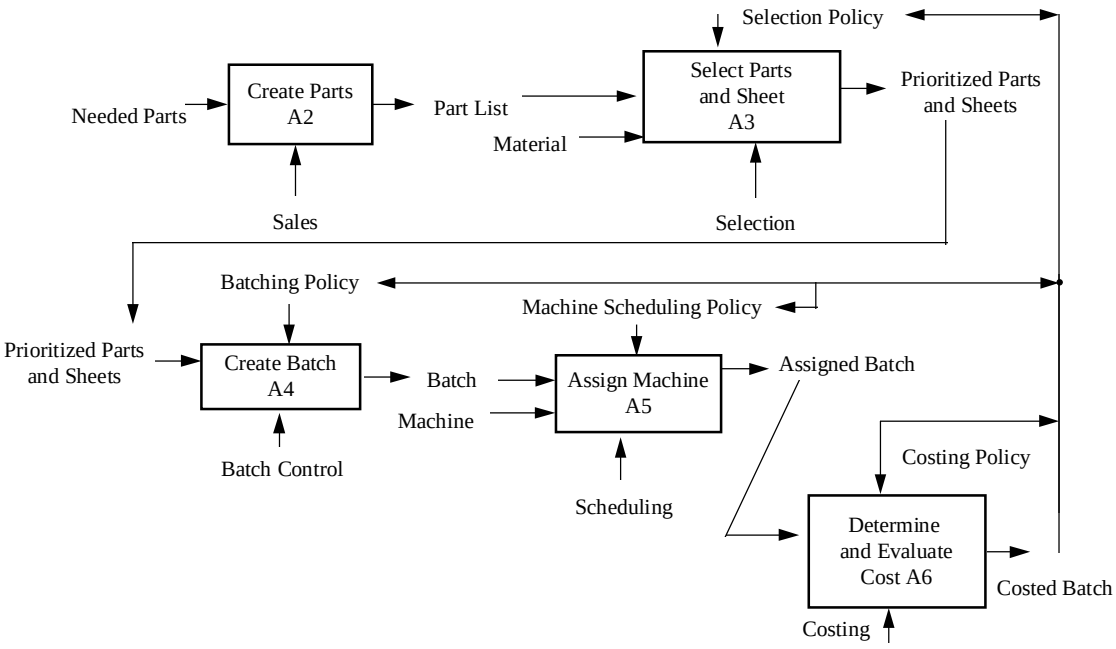

Figure 3: Level 1 $IDEF_0$ Function Model Overview of Case Study

Determination of the part selection policy is addressed in a level 2 $IDEF_0$ diagram in Figure 4. Part selection policy is determined on the basis of parts and material. Information supplied from the control output of costing is used, along with strategic knowledge. The selection policy is used to control part and sheet selection; the selection policy decision may also impact costing via control input. In CIMOSA, the selection policy functional output of Figure 4 would be portrayed as a control output and the set selection policy function would be an enterprise activity.

Selection policy determination monitors part due dates, qualities, and sizes, and input from costing. Simple selection policies are to select the first or the largest parts with the earliest due date. Alternatively, parts may be selected because they match the material that is presently available, and special orders may occur. Lastly, the selection policy may be based upon collaboration with costing, giving precedence to parts and sheets that are *cost effective*. Costing might recommend a sheet that is available but also large enough for cost effective handling, or suggest a part that is high quality, large, and due early because these parts are behind on cost trends. If selection is able to satisfy these requests, this collaboration with costing involves rejecting parts and sheets that have already been selected. If selection is unable to carry out these recommendations, this result must be fed back to costing as a control input. These possibilities are depicted in Figure 5, a level 3 $IDEF_0$ model. Here detailed aspects of the function are shown, fitting the format depicted in Figure 1.

Level 3 diagrams for the Set Batching Policy and Set Scheduling Policy functions have also been produced but are omitted from this discussion.

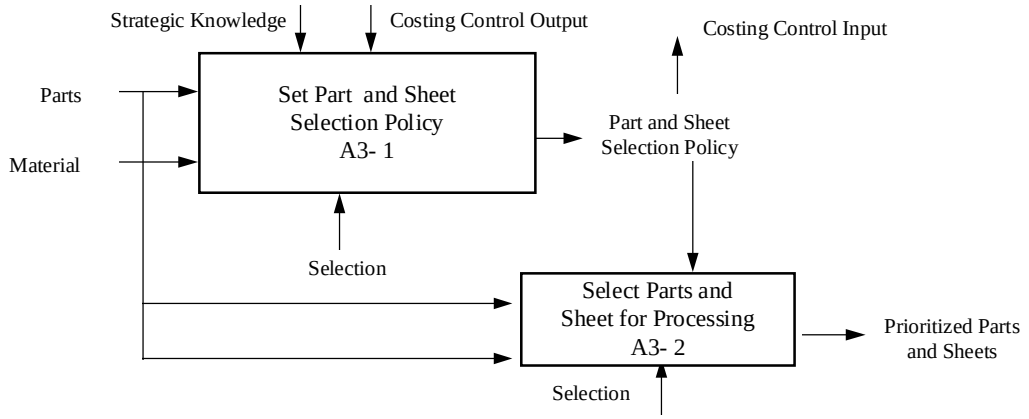

Figure 4: Set Selection Policy $IDEF_0$ Level 2 Diagram

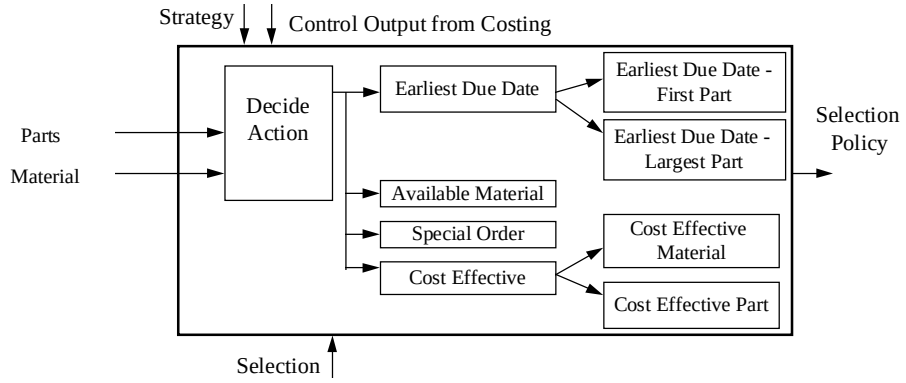

Figure 5: $IDEF_0$ Level 3 Diagram of Set Selection Policy

The level 3 diagram for the Evaluate Cost function A6-2 is provided in Figure 6. The control output, cost evaluation, is used as a control input by costing itself and by the functions in Figure 5. Control input from Selection, Costing, and Scheduling is utilized, along with strategic knowledge. Each decision that Costing can make is shown. Costs may meet objectives, an individual batch may have too much waste or costly processing, or long term cost trends may be problematic. For too much waste, Selection may be able to select a smaller sheet, or Batching may be able to fill the batch. For costly processing and/ or handling, Batching may remove parts, Scheduling may assign a new

machine, or Selection may be able to find a larger sheet. Lastly, for long term cost trend problems, Selection may change its policy.

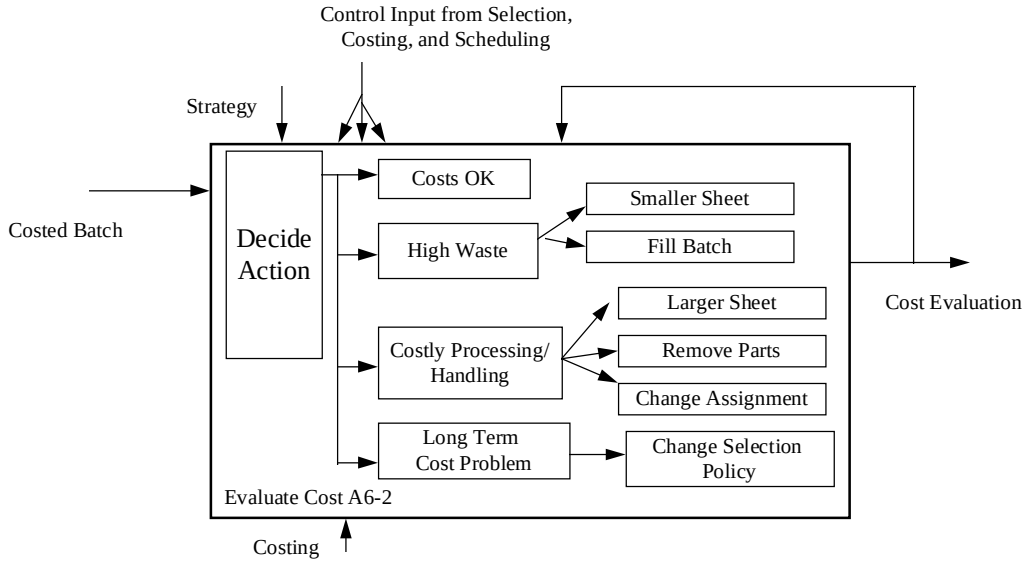

Figure 6: Level 3 $IDEF_0$ Diagram for Evaluate Cost

4.3 Use Case Model

Use case representations are in Figure 7. The stick figures are actors while each ellipse is a use case. *Uses* relationships are shown by dashed double headed arrows; an actor uses a case if it appears in the scenarios. *Extends* relationships are indicated by a solid and single headed arrow, and one use case extends another if it adds scenarios under certain condiations. If an actor *uses* a base use case it also *uses* the extensions. Figure 7 depicts the nine functions (five level 1 and four level 2 in IDEF terms) and the five different resources. The level 2 IDEF functions' use cases extend the level 1 functions' use cases. Selection sets selection policy and selects parts, Batching sets batching policy and creates the batches. Assigning machines and setting scheduling policy are used by Scheduling. As the Costing actor gives input for policy selection, it *uses* these cases. Likewise, four actors collaborate for the cost evaluation.

Figure 8 shows use cases extending Set Selection Policy in the same way that the functions in the level 3 IDEF diagram are extensions of higher level functions. The Set Part Selection Policy use case is extended by Earliest Due Date, Special Order, Available Material, and Cost Effective. Earliest Due Date is refined to First Part and Largest Part, while Cost Effective is extended to Cost Effective Part and Material. The level 3 use case extensions are conditional. Use case extensions for batching, scheduling, and costing policy determinations are similar.

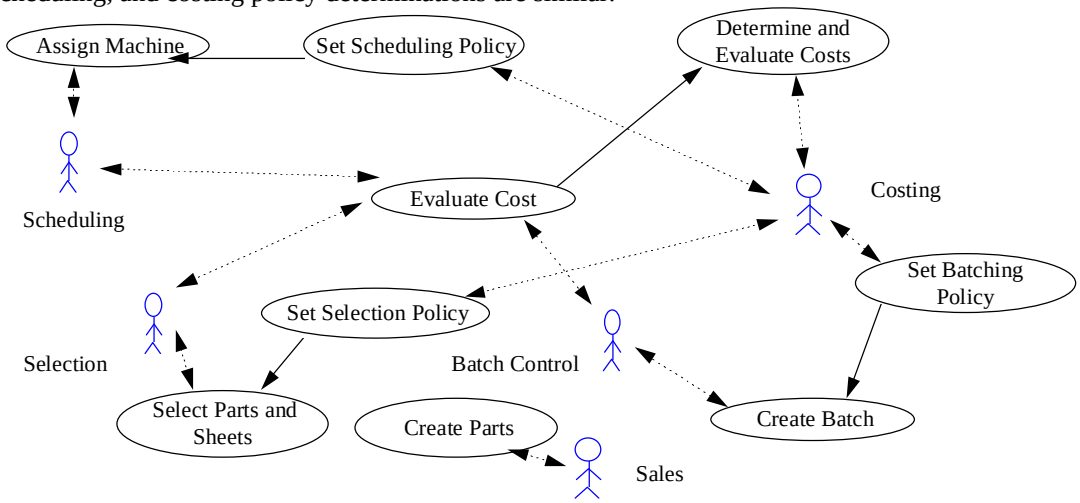

Figure 7: Use Cases for the Application

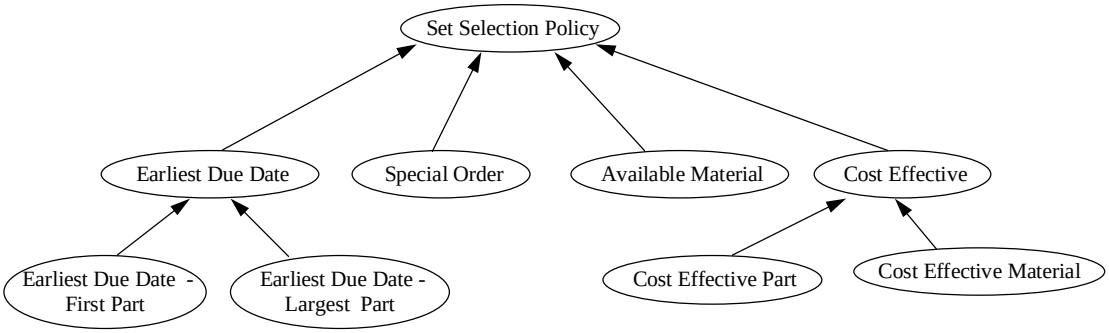

Figure 8: Use Case Extensions of Set Part Selection Policy

The dynamic model analyses object interactions. In the Set Selection Policy use case, the actor sets the selection policy. This policy is then reviewed by the Costing actor during cost evaluation, and the policy may be accepted or rejected, or a new policy may be suggested. Similar steps occur in Set Batching and Set Scheduling Policy. The similarity of the event traces and the state diagrams signifies that these three use cases can be derived from one abstract Set or Select Policy use case. The evaluate costs use case, however, is derived from an abstract Evaluation use case in that the costing actor reviews the policies and provides feedback. The two abstract use cases are shown in Figure 9 by ellipses with double lines. The actual use cases specialize (single headed arrow with double lines) the abstract cases, inheriting all of the features.

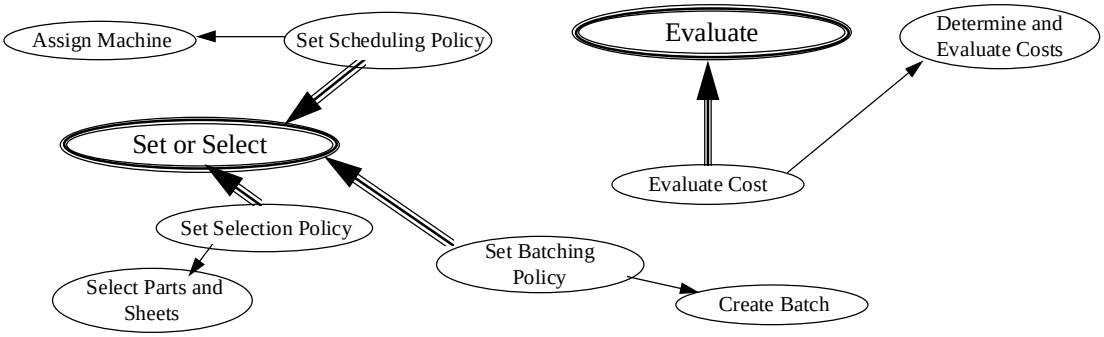

Figure 9: Abstract Use Cases with Inheritance

The actor that sets policy and the actor performing evaluation both use Select and the Evaluate, and the interaction follows Figure 10. In the event trace, a vertical line is an object or an actor, the arrows are messaging, and the boxes depict activities. One activity, evaluate, is shown for the evaluator, but several outcomes are messaged to the selector. In the interactions, the selector responds back with the results, leading to new activities for the evaluator. For example, the selector performs select, messaging the evaluator to evaluate. There are several outcomes: the evaluator may request information, suggest an alternative, reject the selection, or accept it. In these alternatives (except accept), the selector responds by either satisfying or rejecting the evaluator's recommendation.

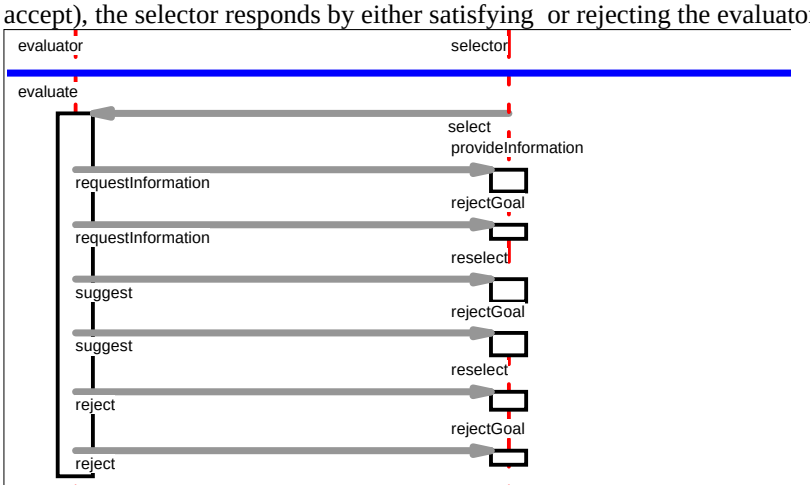

Figure 10: Event Trace of Interaction between Evaluator and Selector in Set or Select Use Case

4.5 The Agent Oriented System

4.5.1 The Agents

The agent oriented system for the case study consists of four agents that are needed to carry out automated reasoning to set policies and formulate decisions: one each for setting selection, batching, scheduling, and costing policies. Four of the actors from Figure 7 are manifested as the four agents. Once the agents have been identified, their goals and plans must be established. The major goal of the Selection agent is to set selection policy. The agent's plans arise from the IDEF level 3 diagram in Figure 5 and the use case extensions in Figure 8. Each of these maps directly to a plan, as shown in Figure 11. The agent has six different plans: earliest due date - first or largest part, special order, part material selection, cost effective part and cost effective sheet.

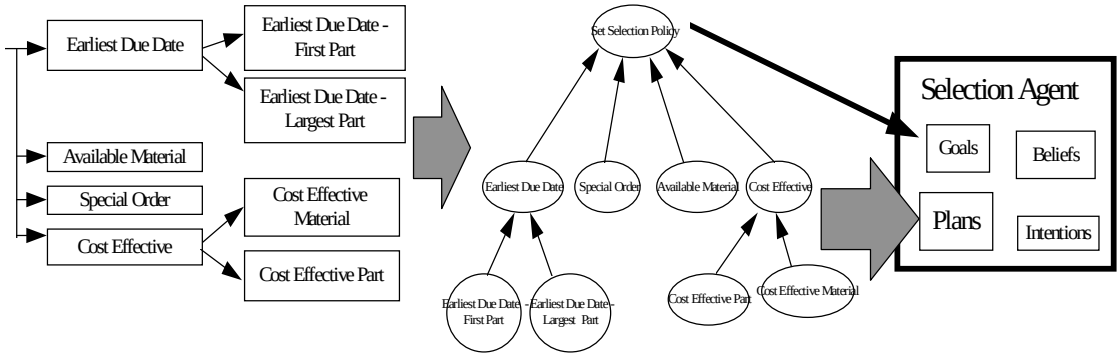

Figure 11: The Selection Agent's Plans

4.5.3 Coordination Protocol or Script

The Selection agent does not work alone; it collaborates with the Costing agent. This is evident from the fact that both the Selection and the Costing actors use the Set Selection Policy use case, and from the control output from Costing that enters the IDEF level 3 diagram. A plan with collaboration is not intrinsically different from a plan without it. All plans have a set of tasks that are meant to achieve the goal; plans with collaboration follow a script for these tasks. As stated in section 3.3, this script can be derived from the event trace for the use case; further, the script is a specialization of the script for the abstract use case.

(Dunskus et al., 1995) discusses the conflicts that can arise between a selector single function agent and an evaluator. If an evaluator gives a negative result, it is a critic. Alternatively a positive evaluator is a praiser, and an evaluator that can make a suggestion is a suggestor. Rather than make these distinctions here, we show that these possibilities are extensions or specializations of one coordination protocol. This approach follows (Kuwabara et al., 1995). There are then three possibilities: an evaluator may not have enough information to make an assessment, it may make a positive evaluation, or it may make a negative evaluation or a rejection. If a negative evaluation is made, the evaluator may be able to make a suggestion, adding a fourth possibility.

State diagrams for the coordination protocols or scripts followed by an evaluator agent are shown in Figure 12. In Figure 12a, the evaluator is only able to request information. In Figure 12b, the evaluator can also be a praiser in that it can accept a selector's findings, or it can be a rejector or a critic. If the evaluator can also be a suggestor, a further orthogonal addition would be made to the state diagram. This script maps directly to the role of the evaluator in the event trace and sequence diagram shown in Figure 12. The corresponding script for the selector agent has states provideInformation, reselect, and rejectGoal, following the activities and messaging in Figure 10.

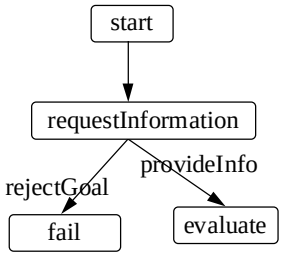

Figure 12a: Evaluator Script

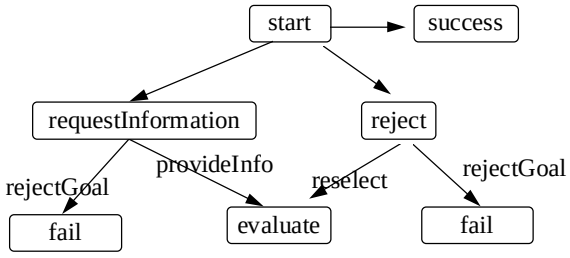

Figure 12b: Evaluator Script with Rejection and Acceptance

4.5.4 Invocation Conditions

The plans, whether they involve cooperation or not, are enacted or intended based upon strategic knowledge and on context or invocation conditions. The context conditions can be derived from the use case extension conditions and the conditions in the determine policy function of Figure 5. Use case extension conditions are discussed in (Jacobsen, 1992). A use case extension is indicated by a probe in the sequence diagram for the original or base use case. When the case reaches the probe, it checks whether the extension condition is true. If it is, the case follows the extension to its completion.

The sequence diagram for the use case extension Set Selection Policy is given in Figure 13. A probe is inserted to indicate when an extension should be followed. The extension returns to the base use case at the second probe. Probes mark the entrance and exit locations for two extensions (four probes are required for the extensions in Figure 9). If a part is from a special order, this use case extension is inserted. If the Costing agent has indicated that a more cost effective approach is required, this extension is followed. When more than one probe appears in a use case, strategic knowledge must be employed to determine which extension take precedence.

The Selection agent's behavior is enacted by stating a new goal to determine selection policy. This goal is stated if the context has changed, corresponding to a new input (a part, or a material) to the Set Selection Policy function. Once this new goal is posted, the agent decides what plan should be intended based on the conditions that match the probes.

4.5.4 Agent Beliefs, Sensors, and Effectors

Information must either be retained within the agent as beliefs (passive objects) or obtained from coexisting objects via sensors. Additionally, achieved goals bring about changes to beliefs or modifications manifested via effectors. Parts and material are the objects that concern the selection agent. The agent's sensors monitor part and material features, raising alarms similar to the probes in Figure 14 when part or material data meet certain criteria. Sensors interact with the various objects via messaging to preserve object encapsulation. The agent's effectors impact instances of parts and sheets, resetting or newly defining their priorities.

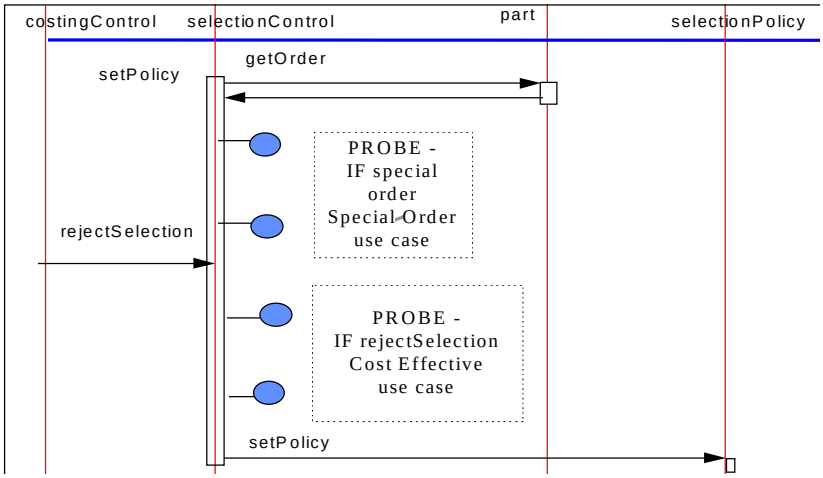

Figure 13: Set Selection Policy Sequence Diagram

Sensors and effectors for setting selection policy are depicted in Figure 14. Sensors monitor part and material features, in addition to watching for special orders. Based on its own knowledge and the control output from Costing, the Selection agent may reject a costly batch or sheet, placing the parts onto the part list and the sheet in inventory. These actions are carried out by effectors. The Part Selection Filter is responsible for prioritizing the parts, according to the agent's intentions, and it is another effector. The Sheet Selection Filter acts similarly.

.

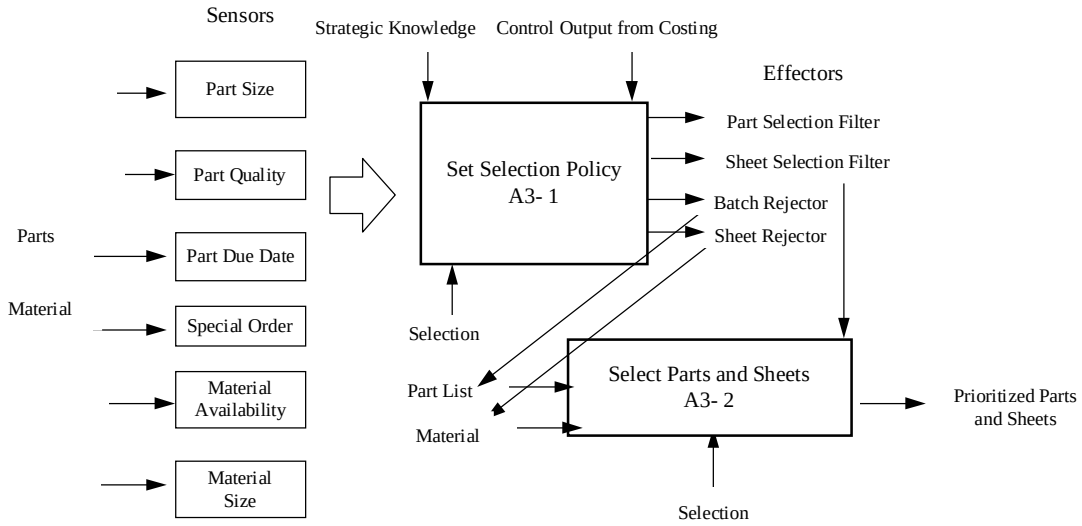

Figure 14: Sensors and Effectors for the Selection Agent

5.0 SUMMARY AND FUTURE WORK

This paper has described steps toward a methodology for developing agent based systems for enterprise integration. The methodology is based upon IDEF and CIMOSA modelling frameworks for enterprise integration and upon object oriented techniques, in particular the use case approach. With this methodology, agents can be identified, along with their plans, goals, beliefs and sensors and effectors that allow them to deal with objects that exist in the outside environment. Agent collaboration through scripts has also been addressed, and the methodology for the analysis and design of the collaboration is based on use cases and use case abstraction.

The agent oriented system for the case study is presently under development in dMARS and C++. Whereas the intial system will utilize passive objects as beliefs, an integrated agent and object design and implementation will be carried out. When this work is finished, a more complete statement of the agent development methodology will be formulated.

In addition to the case study implementation, several areas have been identified for further work:
1. *Additional forms of collaboration and conflict* - agent collaboration is often much more comprehensive than that discussed in this paper; each agent can collaborate with every other agent.
2. *Opportunistic behavior* - pro- active behavior and context conditions do provide for opportunistic behavior, but additional capability is needed.
3. *Synchronization behavior* - the object models must address concurrency and synchronization.
4. *Class hierarchies and designs* - for agent sensor, effector, and encapsulator objects
5. *Structured and unstructured messaging* - between agents and objects

6.0 ACKNOWLEDGEMENTS

This project was partly funded by the Cooperative Research Centre for Intelligent Decision Systems under the Australian Government' s Cooperative Research Centres Program.

7.0 REFERENCES

Barbuceanu, M. and M. S. Fox, “COOL: A Language for Describing Coordination in Multi-Agent Systems,” First International Conference on Multi- Agent Systems, pp. 17 - 24, 1995.
Booch, G., “Object Oriented Analysis and Design with Applications, Second Edition”, The Benjamin/ Cummings Publishing Company, 1994.
Bravoco, R. R., and S. B. Yadav, “Requirement Definition Architecture - An Overview”, Computers in Industry, Vol. 6, pp. 237- 251, 1985a.
Bravoco, R. R., and S. B. Yadav, “A Methodology to Model the Functional Structure of an Organization”, Computers in Industry, Vol. 6, pp. 345- 361, 1985b.
Brenton, A., “The dMARS Plan Language Reference Manual”, Australian Artificial Intelligence Institute (AAII), 1995.

Cohen, D., and G. Ringwood, “Decision Table Languages and Guarded Definite Cleauses: Old Wine in New Bottles”, Queen Mary and Westfield College, Dept of Computer Science Report, 1995.
Cutkosky, M., R. Engelmore, et al., “PACT: An Experiment in Integrating Concurrent Engineering Systems”, IEEE Computer, 1992.
Dunskus, B. V., D. L. Grecu, D. C. Brown, and I. Berker, “Using Single Functions Agents to Investigate Conflict”, AI in Engineering Design and Manufacturing, special issue in Conflict Management in Multi-Agent Systems, to appear, 1995.
Finin, T., J. Weber, et al, “Specification of the KQML Agent- Communcation Language”, The DARPA Knowledge Sharing Initiative External Interfaces Working Group, February, 1993.
Genesereth, M., R. E. Fikes, et al, “Knowledge Interchange Format, Version 3.0, Reference Manual,” Report Logic- 92- 1, Computer Science Department, Stanford University, 1992.
Georgeff, M. P., F. Ingrand, et al, “Research on Procedural Reasoning Systems”, SRI International, 1990.
Gruber, T. R, “Ontolingua: A Mechanism to Support Portable Ontologies”, Reference Manual, June, 1992.
Gruber, T. R., J. M. Tenenbaum, and J. C. Weber, “Toward a Knowledge Medium for Collaborative Product Development”, in Artificial Intelligence and Design ‘92, etd. by J. S. Gero, Proceedings of the Second International Conference on Artificial Intelligence in Design, 1992.
Huntbach, M. M., N. R. Jennings, and G. A. Ringwood, “How Agents Do It In Stream Logic Programming”, Proc. First International Conference on Multi- Agent Systems, pp. 177- 184, 1995
Jacobsen, I., *Object- Oriented Software Engineering: A Use Case Driven Approach*, Addison Wesley, 1992.
Klein, M., “Supporting Conflict Resolution in Cooperative Design Systems”, in IEEE Transactions on Systems, Man, and cybernetics, Vol. 21, No. 6, November/ December 1991.
Klein, M., “Business Process (Re-)Engineering: Methodologies and Multi- Agent Technologies,” Tutorial at First International Conference on Multi-agent Systems, 1995.
Kuwabara, K., T. Ishida, and N. Osato, “AgenTalk: Describing Multiagent Coordination Protocols with Inheritance,” submitted to Tools for Artificial Intelligence Conference, 1995.
Kwok, A., and D. Norrie, “Intelligent Agent Systems for Manufacturing Applications”, J. of Intelligent Manufacturing, Vol 4, pp. 285- 293, 1993.
Matsuoka, S., and A. Yonezawa, "Analysis on Inheritance Anomaly in Object-Oriented Concurrent Programming Languages", in *Research Directions in Concurrent Object-Oriented Programming* (ed. Gull Agha, P. Wegner, and A. Yonezawa), The MIT Press, Cambridge, MA., 1993.
Olsen, G. R., M. Cutkosky, J. M. Tenenbaum, and T. R. Gruber, “Collaborative Engineering Based on Knowledge Sharing Agreements”, Proc. of the 1994 ASME Database Symposium, 1994.
Pan, J. Y. C., J. M. Tenenbaum, and J. Glicksman, “A Framework for Knowledge- Based Computer Integrated Manufacturing”, IEEE Transactions on Semiconductor Manufacturing, Vol. 2, No. 2, May 1989.
Pan, J. Y. C., and J. M. Tenenbaum, “An Intelligent Agent Framework for Enterprise Integration”, IEEE Trans. on Systems, Man and Cybernetics, Vol. 21, No. 6, November/ December 1991.
Pressman, R. S., *Software Engineering: A Practitioner’s Approach*, 3rd Edition, McGraw- Hill, 1992.
Rumbaugh, J., M. Blaha, et al., *Object Oriented Modeling and Design*, Prentice Hall, 1991.
Tenenbaum, J. M., J. C. Weber, and T. R. Gruber, “Enterprise Integration: Lessons from SHADE and PACT”, in Enterprise Integration Modeling, Proceedings of the First International Conference, Etd. by C. J. Petrie, MIT Press, 1992.
Wooldridge, M. J., and N. R. Jennings, “Agent Theories, Architectures, and Languages: a Survey”, 1994.
Wooldridge, M. J., and N. R. Jennings, “Agent Theories, Architectures, and Languages”, Tutorial, First International Conference on Multi- Agent Systems, 1995.