Submitted to: 3-rd Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 1994,
Morgantown, West Virginia

The Information Agent: An Infrastructure Agent Supporting Collaborative Enterprise Architectures

Mihai Barbuceanu and Mark S. Fox
Enterprise Integration Laboratory,
Department of Industrial Engineering,
University of Toronto
4 Taddle Creek Road, Toronto, Ontario, M5S 1A4
email: {mihai, msf}@ie.utoronto.ca, tel: 1-416-978-6823, fax:1-416-978-3453

Abstract: *We introduce the Information Agent as a component of the information infrastructure supporting collaborative enterprises. We discuss the functions of the Information Agent and illustrate them in the context of the Integrated Supply Chain Management system. We introduce a simple structure for the Information Agent, composed of an Agent Program and a Problem Solver and we present our choices for these components. We show how the problem-solving component can be designed and implemented using description logic representation systems and argue for the advantages of this approach. For the Agent Program, we adopt a speech-act based agent communication language (KQML).*

Keywords*: enterprise information architecture, agent programming, description logics, agent communication language, conflict management, change management.*

1.0 Introduction

The productivity of complex enterprises is increasingly limited not by labor or capital, but by information. The enterprise's ability to process information and make decisions is what determines its "agility", that is its capacity to permanently monitor market demand, quickly respond by introducing new products, services and technologies and quickly restructure their business methods. In enterprises achieving this level of responsiveness interactions among individuals and groups will be managed by dialogue and negotiation, relations will predominantly become peer-to-peer rather than hierarchical and knowledge will be more important than rank. To achieve this sort of organization, collaborative information systems are essential ingredients that help with providing access to information, support decision making and aid in action execution. The use of collaborative information systems will result in Computer Integrated Enterprises that will be able to rapidly respond to a changing market place, integrate decisions and the performance of activities.

As part of this broad enterprise integration program, the information architecture topic is of particular importance: If an information system is a system which *acquires, stores, retrieves, processes, communicates and presents* information relevant to an organization in such a way that the information is accessible and useful to those who wish to use it, the enterprise information architecture provides the underlying *information infrastructure* on the basis of which integrated cooperative enterprise information systems are built. This infrastructure consists of both the *conceptual framework* - approaches, theories and methods - and the actual *services and tools* supporting the design and implementation of enterprise information systems.

Based on [Pan and Tenenbaum 91] we can list several fundamental requirements that an agile and collaborative enterprise information architecture must address:

- *Information access:* accessing the wealth of information resources and services available through corporate networks, in ways allowing data and information to be integrated and inferred from many sources and presented in customized formats according to the interests and needs of each individual.
- *Consistency maintenance:* detecting various forms of inconsistency among models and data and taking action to remove them.
- *Monitoring and automation:* keeping track of where and how to get the information such that interested people and software can be notified of decisions and events that affect them in order to take appropriate action.
- *Cooperative work:* people and computers working together as teams across time (e.g. through various stages of a product lifecycle) and space (e.g. design and production engineers working concurrently to resolve a problem). Team members will need to share knowledge and information and be alerted to potentially conflicting decisions.
- *System integration:* independently developed software packages must be easily integrated into the framework so that they operate seamlessly and are easily used and maintained. Additionally, the framework must provide an incremental path for migrating the enterprise from a purely people-based operation to a cooperative environment uniting people and computers.

We approach these basic requirements by first noticing that an information architecture of the kind we are interested in spans multiple levels from the hardware/software platform to the high level knowledge-based problem solving activities. According to [Roboam and Fox, 1992] we can distinguish the following layers within an EIA: (i) the *network communication layer*, concerned with providing services for interconnecting heterogenous equipment and resources, (ii) the *data related support services layer*, that provides the capabilities to request and send information from/to the nodes of the corporate network, (iii) the *information distribution layer*, providing services for information exchange among the nodes of the corporate network, (iv) the *organization layer*, concerned with modeling the structure, goals, roles etc. that define the position of nodes in the corporate network, (v) the *coordination layer*, concerened with the high level knowledge based problem solving protocols used by nodes of the decentralized system to coordinate their problem-solving activities and (vi) the *market layer*, supporting strategies for dealing with the enterprise's environment (subcontractors, suppliers, consumers, etc.).

The development of an information architecture along these lines is of course a long range goal. This paper reports on our results concerning the *information distribution layer* of the architecture. This service is critical for the cooperativeness and agility of the distributed information system. In the approach presented here, it is accomplished by a class of agents called *information agents*. The concept, functions and implementation technology of this kind of agent forms the object of this paper. Section 2 describes our notion of information agent and its functions, presents its two major components - the *Agent Program* and the *Problem Solver* - and illustrates them with the discussion of the Information Agent in the context of a complex, distributed supply chain management system whose development is an important ongoing project in the Enterprise Integration Laboratory at the University of Toronto. Section 3 presents the description logic technology used to implement the Information Agent. Section 4 discusses some issues specific to the Information Agent notion - such as consistency management, semantic query management, time management and change management - and shows how extensions of description logic technology helps cope with them. Section 5 tackles the Agent Program component, while Section 6 presents concluding remarks and mentions other possible application areas.

2.0 The Information Agent

In order to make sense of the wealth of information available through corporate networks, we propose a new kind of component node for these networks: the Information Agent (IA). IA-s are infrastructure elements of the corporate network. IA-s are sophisticated knowledge and data management systems that allow other agents from the computerized enterprise to be selectively aware of relevant information by providing communication and information services supporting:

- Persistent storage of information to be shared among the multiple functional agents in the corporate network.
- Deductive capabilities allowing new information to be inferred from existing information.
- Automatic, content-based routing and distribution of information to the agents that need it.
- Automatic retrieval, processing and integration of information that is relevant to agents.
- Checking and maintaining various forms of consistency of the information.
- Performing information access control functions such as determining who is allowed to see and change the available information.

2.1 Information Agents in the Corporate Network

IA-s operate in a corporate network populated by other agents (human and software). Besides IA-s, corporate neworks will contain a variety of functional agents representing: (i) legacy software modules whose functionality needs to be integrated (e.g. database systems), (ii) personal assistants, that is software encapsulating human users and able to support them in their activity, (iii) knowledge-based problem-solving and decision making software, etc.

A given IA will service a number of agents (both functional and other IA-s) by providing them with a layer of shared information storage and the above mentioned services for managing it. Agents periodically volunteer some of their information to the IA (and keep it up to date) or just answer the queries sent to them by the IA. The IA uses its own knowledge together with the supplied information to determine which information needs of other agents can be satisfied. It processes the information in order to determine the most relevant content and the most appropriate form for the needs of these agents. In the process, it may uncover various forms of inconsistency among the supplied information and take action to remove them. IA-s will also communicate with each other in order to accomplish their functions.

IA-s are not meant to replace the direct communication channels established among agents during their usual interactions. Rather, they support these interactions by providing shared access to information and the basic information management services listed above. IA-s will be particularly useful in cases when:

- The existence of a consistent form of the shared information is vital for the participating agents to achieve collaborative behavior.
- Information from many sources needs to be aggregated, perhaps in a continous fashion, to answer queries or produce reports.
- Information has to be distributed among many agents and the content of the information is the only basis available for the distribution activity.
- Changes in the state of the modeled enterprise need to be propagated over the models and activities of various agents.
- Inconsistencies arising during agent interaction need to be quickly detected and resolved.

2.2 Architecture of the Information Agent

Essentially, an IA of the kind we are envisaging is composed of two components: a problem-solver and an agent program, as shown in figure 1.

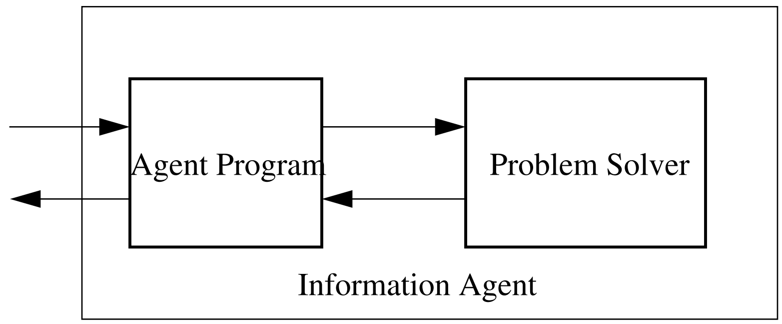


FIGURE 1. Architecture of the Information Agent.

The Problem Solver must provide a collection of knowledge representation and knowledge processing services that, according to the previously listed requirements, should support at least: (i) a fairly general inferential capability, in order to cope with the requirements for deductive query processing and content-based information routing, (ii) mechanisms for checking and enforcing the consistency of information, (iii) mechanisms for change management, in order to cope with the unexpected and unforseeable nature of the events happening in real corporate networks.

The Agent Program has the purpose of turning the Problem Solver into an autonomous agent. The Agent Program must provide an interface to the external environment, allowing external messages to reach the IA and the IA to send out its own messages. The Agent Program must adopt a protocol for information exchange and must translate from this protocol into the language of the Problem Solver and viceversa.

Given this general characterization of the IA architecture, different IA-s can be obtained by making different decisions about the nature of the two components. Our own decisions were to use:

- A description-logic language (also called terminologic logic, or term-classification language) to implement the problem-solver. As we will explain later on, this is one of the few classes of languages able to handle the above posted requirements.
- The KQML [Finin et. al. 92] agent communication language as the language used by the Agent Program to communicate with the external world. KQML is a speech-act based language, produced as a result of the DARPA Knowledge Sharing Initiative, and was chosen on the grounds of its expressiveness and adequacy to our task.

2.3 An Example: the IA in the Supply Chain

The IA is applied in the the TOVE virtual entreprise [Fox et. al., 1993]. The TOVE Project includes two major undertakings: the development of an Enterprise Ontology, and a Testbed. The TOVE Enterprise Ontology provides a generic, reusable ontology for modelling enterprises. An ontology is comprised of a reference data model composed of objects, attributes and relations (also called the terms of the ontology), and formal definitions of and constraints on the terms in First Order Logic. The TOVE ontology currently spans knowledge of activity, state, time, causality, resources, cost and quality. The ontology's data model is implemented on top of C++ using the Carnegie Group's ROCK knowledge representation tool and the axioms are implemented in Quintus Prolog. The TOVE Testbed provides an environment for analysing enterprise ontologies. The Testbed provides a model of an enterprise - a lamp manufacturing plant -, and tools for browsing, visualization, simulation and deductive queries.

Consider now the integrated supply chain management (ISCM) system of the TOVE virtual entreprise shown in figure 2. The ISCM is composed of a set of cooperating agents, where each agent performs one or more supply chain management functions, and coodinates its decisions with other relevant agents. There are two types of agents: functional agents and information agents. Functional agents plan and/or control activities in the supply chain. Information agents support other agents by providing information and communication services.

The dynamics of the environment make cooperative behavior an important factor in integrating supply chain agents. In order to optimise supply chain decisions, an agent cannot make a locally optimal decision, but must determine the affect its decisions will have on other agents, and choose an alternative that optimises the entire supply chain.

An agent is responsible for a set of functions or activities in the supply chain. Each agent stores information and knowledge locally and it may access information and knowledge throughout the network. We assume that the agents are in a heterogenous environment, hence their interactions are made through message-based transactions.

IA-s provide each agent with automated information acquisition and distribution. When an agent requests information, an IA will find it. When an agent creates information of interest to others, an IA will distribute it to those agents that wish to know. IA-s provide the "right information in the right way" to the decision makers.

At the core of the information architecture and of the supply chain management system lies a generic reusable enterprise model [Fox 92]. In order to support the integration of supply chain agents, it is necessary that a shareable representation of knowledge be available that minimizes ambiguity and maximizes understanding and precision in communication.

Here is now a brief review of the functional agents existing in the supply chain:

Order Acquisition agent. This agent is responsible for acquiring orders from customers, negotiating with customers about prices, due dates, etc., and handling customer requests for modifying or canceling respective orders. This agent captures the order information directly from customers and communicates these orders to the logistics agent. When a customer order is changed, it is communicated to the logistics agent.

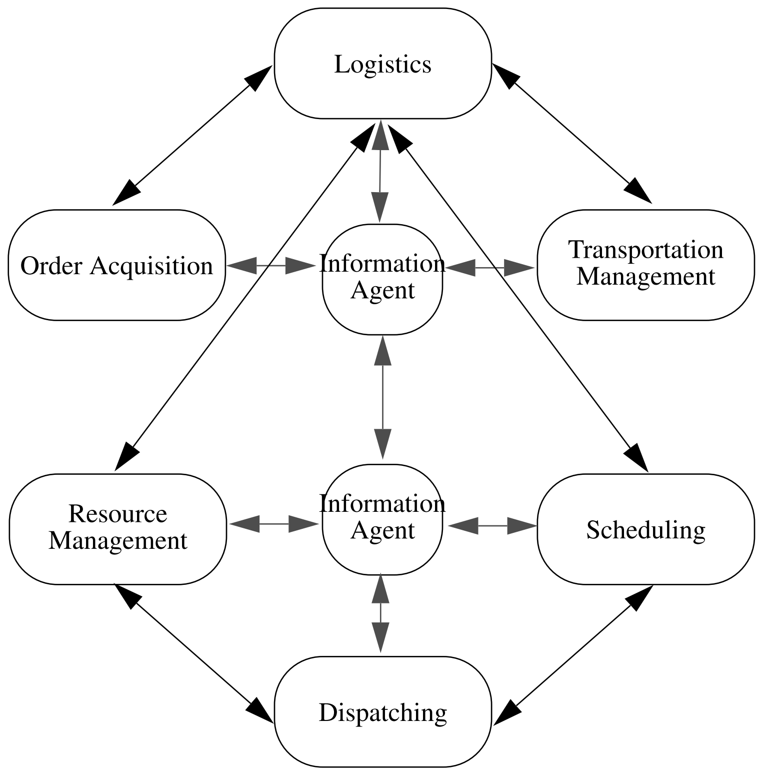
When schedules violate constraints imposed by the customer (such as due date violation), the order acquisition agent participates in negotiating with the customer and the logistics agent for a feasible schedule.

Logistics agent. This agent is responsible for coordinating multiple-plants, multiple-supplier, and multiple-distribution center domain of the enterprise to achieve the best possible results in terms of final deliveries to customers. It manages the movement of products or materials across the supply chain from the supplier of raw materials to the customer of finished goods. The inputs to the logistics agent are customer orders, deviations in factory schedules which affects customer orders, and availability of resources. The outputs of the agent are customer orders (assigned to respective plants), and purchase orders for "critical" components.

Scheduling agent. This agent is responsible for scheduling and rescheduling activities in the factory, exploring hypothetical “what-if” scenarios for potential new orders, and generating schedules that are sent to the dispatching agent for execution. The inputs to the scheduling agent are requests for new orders from the logistics agent and the deviations of the current schedule from the dispatching agent. Its output is an executable schedule.

Resource agent. This agent dynamically manages the availability of resources so that the schedule can be executed. It estimates resource demand and determines resource order quantites. It is responsible for selecting suppliers while minimizing costs and maximizing delivery. The inputs to the resource agent are the schedule from the scheduler, the availability or unavailability of resources from suppliers, the arrival of resources from the factory floor, and the consumption of resources from the dispatcher. The outputs of the resource agent include the arrival of resources, the availability of resources, and the orders sent to suppliers.

FIGURE 2. The ISCM agents



Dispatching agent. This agent implements the schedule created by the scheduling agent. It monitors the factory operations and decides what to do next. Its decisions are constrained by the schedule. It communicates deviations in the schedule to the scheduling agent for repair. The inputs to the dispatching agent are the schedule from the scheduling agent, the status of the factory floor, and the availability of resources. The outputs are the deviations from the current schedule and the starting of activities.

Transportation agent. This agent is responsible for the assignment and scheduling of transporation resources in order to satisfy inter-plant movement requests specified by the Logistics Agent. It will be able to consider a variety of transportation assets and transportation routes in the construction of its schedules.

Let us now consider the "MRP-level" IA that would service the following agents from the Supply Chain: Resource Manager, Scheduler, Dispatcher and, through another IA, Order Acquisition. Given the above functions and paths of interaction among the functional agents, this IA can help the functional agents accomplish their goals by providing services such as the following:

- Assembling typical MRP reports relating orders with sums of component parts and quantities as well as with the time intervals for which these are needed. This information is produced by aggregating information received from Order Acquisition, Scheduling and Resource Management.

- Aggregating costs based on the activities required to carry out orders. This is done using information from Scheduling and Dispatching as well as cost-computation theories from the overall common-sense model of the enterprise [Fox 93].
- Change management functions, determining the consequences and checking conflicts when changes of the enterprise model are envisaged. These changes span the range from changes of the instance network modeling various levels of the enterprise (e.g. removing, moving, replacing machines) to changes in the generic vocabulary in terms of which the enterprise models have been constructed (e.g. changing the definition of an activity, resource, etc.).
- Monitoring activities, like monitoring the state of orders, estimating delays, monitoring machines or other agents.
- Gathering, aggregating and analysing various statistical data (usage of resources, time spent on various activities, etc.).

This functionality is achieved by having the IA rely on knowledge about: parts of the enterprise model (generic terms and instances), rules for model change management (at both the “schema” and actual model level), theories about inferring properties (such as costs of orders, time delays of orders, collecting information along *part-of* relations, etc.). The IA will receive information from the other agents concerning: orders (status, dates, components), the factory floor model (changes), manufacturing processes (changes, status), resources (status, amounts, activities) and schedules.

In the remainder of the paper we will discuss our choices for the two components of the information agent.

3.0 Using Description Logic Technology

We have chosen to develop the Problem Solver componemt of the IA based on description logic representation languages. This section reviews the basic features of description logic languages and presents the motivations for this choice and its consequences with respect to the requrements imposed over the information agent.

3.1 Description Logic Languages

Description logic languages (or terminological languages) integrate aspects of object-oriented and logic representations. They express knowledge in a modular fashion, using inheritance and hierarchical organizations and rely on well-defined declarative semantics to describe the meaning of constructs. Well-known examples of such languages include KLONE [Brachman and Schmolze 1985], LOOM [McGregor and Bates 1987], BACK [Luck et. al 1987], NIKL [Schmolze 1985] or CLASSIC [Borgida et al. 1991]. Recently, some of these languages have been successfully applied to "industrial" applications, such as the suite of configuration tasks performed in CLASSIC.

For the work reported herein, we use our own description logic language called MODEL [Barbuceanu 1993]. The language is object centered in that it concentrates on describing, organizing and manipulating classes of objects in the domain of discourse. There are tree kinds of formal entities in MODEL.

<concept definition>->(concept <concept name> [:primitive]
(:and <concept>$^{+}$) [<annotations>] [<methods>])

<concept>-><concept name>
(:and <concept>$^{+}$)
(:all <role name> <concept>)
(:the <role name> <concept>)
(:some <role name> <concept>>)
(:atleast <positive integer> <role name>)
(:atmost <non negative integer> <role name>)
(:oneof <constant>$^{+}$)
(:fills (role name> <constant>$^{+}$)
(:lt <role name> <number>)
(:le <role name> <number>)
(:gt <role number> <number>)
(:ge <role number> <number>)

<annotations>-><annotation>$^{+}$

<annotation>->(:has <role name>(<annotation name><constant>$^{+}$)$^{+}$)
(:self (<annotation name> <constant>$^{+}$)$^{+}$)

<annotation name>-><symbol>

<methods>-><method>$^{+}$

<method>->(:method <role name>(<message><method expr>$^{+}$)$^{+}$)
(:method self (<message> <method expr>)$^{+}$)$^{+}$)

<message>-><symbol>

<method expr>-><symbol>|<Common LISP expr>

<constant>-><Common LISP constant>

Note: (:the R C) $=_{def}$ (:and (:all R C)(:atleast 1 R)(:atmost 1 R))
(:some R C) $=_{def}$ (:and(:all R C)(:atleast 1 R))
:lt = less-then, :le = less-equal, :gt = greater-then, :ge = greater-equal

a - syntax of concepts

<role definition>-> (role <role name> (:and <role name>$^{+}$)
(:domain <concept>)
(:range <concept>)
(:inverse <role name>))

b - syntax of roles

<instance-assertion>->(<concept> <instance name>)

<role-assertion>->(<role name><instance name> <constant>)

<instance name>-><symbol>

c - syntax of asserting instances and roles

Legend: $^{+}$ = at least one occurence, [...] = optional element

FIGURE 3. Syntax of MODEL entities.

Concepts are complex aggregates composed of a limited set of description-forming operators. Concepts correspond to one-place predicates being applied to one individual object at a time. *Roles* describe properties of or relations among objects. Roles correspond to two-place predicates and thus relate two individual objects at a time, one of them belonging to the domain of the role and the other to its range. *Instances* directly represent objects in the domain of interest. Instances satisfy concepts (e.g. `M1` is a `Machine`) and have roles filled with instances of other concepts (e.g. `A2` is the `next-activity` of `A1`). An instance is an atomic, named individual which belongs to the class of objects described by a concept. Instances are factual or assertional entities, while concepts are intensional descriptions.

Concepts can be either primitive or defined. Defined concepts contain necessary and sufficient conditions for an individual to be recognized as belonging to the concept. Primitive concepts contain only necessary conditions and thus are not fully specified. Several primitive concepts can be explicitly declared as being disjoint. This ensures that there will be no individuals "shared" by such concepts.

The syntax of concepts, roles and instances is shown in figure 3. Concept definition (figure 3-a) has four main sections. The first is the structured definition of the concept, that is a conjunction of terms formed with a vocabulary of descriptors shared by several terminological languages. Among these descriptors we mention the following. The `:all` descriptor restricts the concept type of role fillers, `:atleast` and `:atmost` restrict the cardinality of the filler set, `:fills` imposes that a number of fillers will fill the role in any instance, and `:oneof` specifies a concept as a choice from an enumerated set of individual instances. Roles with numerical values can be restricted by imposing that their values be less than (`:lt`), less equal (`:le`), greater than (`:gt`) or greate equal (`:ge`) than given numbers.

The second section holds meta-information about the concept. Meta-information may be specified for each role in part (the `:has` operator) or for the concept as a whole (the `:self` operator). Meta-information is structured in an annotation-values format. Values of annotations are inherited from superconcepts. The last two sections contain MODEL specific enhancements related to method attachment and message passing.

Figure 3-b shows the syntax of roles. The domain of a role is the concept the role is applicable to. The range of a role is the concept all individual values of a role must belong to. The inverse of a role is another role which holds the inverse of the relationship denoted by the role. Figure 3-c shows the syntax of asserting instances and role fillers. An instance is an atomic, named individual which belongs to the class of objects described by a concept. Instances are factual or assertional entities, while concepts are intensional descriptions.

3.2 Terminological services

MODEL terminological services are related to the definition and validation of concepts and the construction of conceptual taxonomies. MODEL terminological services include completion, subsumption checking and classification.

Completion. All T-box and A-box operations use concepts and roles in the so-called complete form. Essentially, in the complete form concepts and roles have been checked for consistency, inherited propertied have been unified and all contained information has been explicitly represented.

Subsumption. A concept `a` subsumes a concept `b` iff all individuals described by `b` are also described by `a`. The set of individuals described by a concept is called the extension of the concept. Given concepts in the complete form, a procedure proving subsumption on the basis of the structure (not names) of concepts can be written as a recursive rolewise comparison of the concepts.

Classification. With concepts in complete form and with a subsumption procedure, the classification service has the simple job of building a lattice of concepts where each concept is linked to two sets of concepts. The first is the set of the most specific subsumers (*mss*), comprising all concepts which subsume the given concept such that none of the *mss* concepts subsumes another *mss* concept. The second set is the set of most general subsumees (*mgs*), comprising all concepts subsumed by the given concept such that none of the *mgs* concepts is subsumed by another *mgs* concept. The minimal element of the lattice is concept `nothing` and the maximal is the concept `anything`. Classification also detects equivalent concepts, the current policy being to forbid the existence of such concepts.

3.3 Assertional services

Terminolgical services describe, verify and organize the generic terms in which domains can be described. Assertional services help create actual domain descriptions expressed with these terms and enforce the language semantics upon these descriptions. Assertional services deal mainly with instances of concepts. Instances are individuals belonging to the classes described by concepts. Instances have unique names and are by definition disjoint from each other (remember that concepts can be proven equivalent). Instances are created as belonging to the class of a given concept, but during their lifetimes can be recognized as belonging to other concepts as well. This recognition process, automatically triggered whenever an instance is changed (e.g. by asserting or retracting a role filler) is based on the content of instances and not on the their names (hence it is properly described as a proving process). The process is best viewed as a constraint propagation activity in which instances are made to conform to the various constraints applied to them. Assertions about instances and their fillers can be retracted, the consequences of retractions being automatically enforced.

Assertional services operate with a propositional representation of the knowledge base. This representation manipulates propositions that describe instances, such as `(milling-machine M1)` and propositions describing roles of instances, such as `(weight M1 200)`.

MODEL provides an *assertion language* for creating instances and filling their roles, a *query language* for retrieval and a *retraction language* for removing fillers and instances. The most important services provided by the assertional mechanisms are the following.

Truth-maintenance. Assertional services work on a propositional representation managed by a boolean constraint propagation truth maintenance system [McAllester 90]. This allows linear-time handling of assertions and retractions.

Methods. MODEL is extended with method attachment and message passing in a manner similar to object oriented languages. Methods are procedures that can be can be attached to each role in part or to the concept as a whole. Because concept are automatically classified into a generalization-specialization lattice based on their content, attached methods can make systematic use of the objective generalization / specialization relations provided by the lattice.

Rules. A rule is an implication stating that if an instance is recognized as belonging to an antecedent concept, then a consequent concept should be asserted about the instance. Rules account for an important component of "reasoning" in terminological languages. Whenever assertions are made, existing instances are reclassified and rules are being fired. This asserts new concepts, triggers new classifications and possibly new rule firings, a.s.o. until a quiescent state is reached.

Patterns and events. These provide a general rule-like mechanism allowing arbitrary collections of concepts as antecedents, variable binding and arbitrary actions in the consequent. Patterns are conjunctions formed by propositions which can also accept variables in place of constants. Events define arbitrary computations to be executed for each tuple of variable bindings discovered to satisfy a pattern.

Demons. Demons are a limited form of forward reasoning, allowing arbitrary propositions to be asserted whenever a specified set of propositions becomes true.

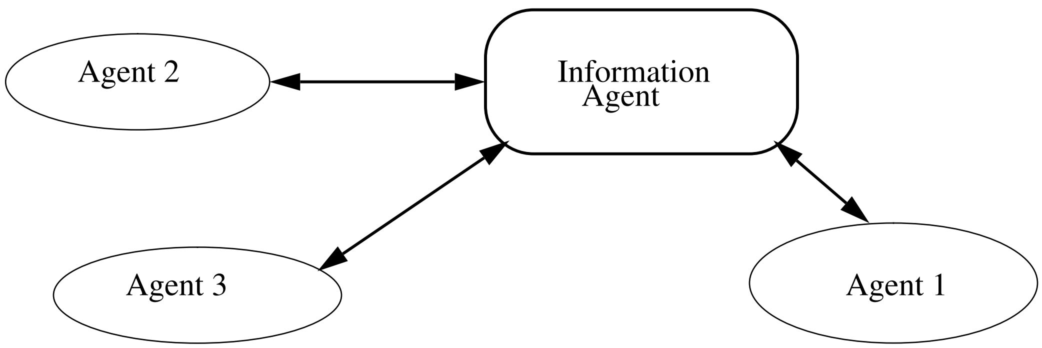
3.4 Building IA-s with Description Logics

Developing IA-s as knowledge and data management systems based on description logic technology takes advantage from many features of the technology, some of the most important being the following:

- The common, shared vocabulary agents use to communicate as well as the knowledge of the IA can be represented as concepts, roles and rules, with the advantage of automatically *checking consistency* and *non-redundancy* and automatically *organizing* this knowledge into the classification lattice.
- Information supplied to and by the IA is represented as propositions about instances. The advantage is that the assertional services will work with *partial information* about these entities.
- The continous classification process allows the IA to provide *descriptive/intensional answers* to queries. As classification is a theorem-proving process, answers to queries are constructed through deductive mechanisms.
- Asserting constraints (concepts) upon entities (instances) provides in fact a very flexible *view mechanism* that accomodates *view updates*. All of these features are currently beyond the reach of conventional database technologies.

In the current implementation, agents supply information to the IA in the form of propositions that can be asserted, retracted or queried. These propositions construct instances of concepts from the shared ontology. These instances are usually incomplete and are classified under the most specific concepts that describe them given their current content. Agents also specify their *topics of interest*, a form of persistent queries encoded as concepts or patterns. The IA continously classifies the information it receives (possibly triggering rules or events in the process). This provides the functionality of deductive query processing and automatic retrieval of relevant information. The information classified under concepts representing agents' topics of interest is then supplied to the agents that expressed the interest (accomplishing the function of automatic distribution of information). The truth-maintenance mechanisms support tracing contradictions, according to the discussion following in section 4. "Domain" knowledge, e.g. for detecting changes in the modeled enterprise that need to be propagated over the models and activities of other agents, is encoded in rules, events and demons as explained further in section 7.

To illustrate part of how the IA works, consider the situation in fig. 4-a where an IA services three functional agents in an organization and assume that the topics of interest of two agents are as given in figure 4-b.



a - Information Agent servicing functional agents

Topic of interest of Agent-2:

```
(concept heavy-component
   (:and component (:gt weight 5000))),
```

that is "any component whose weight is greater than 5000"

Topic of interest of Agent-3:

```
(concept weight-change
    (:and change (:the changed "weight")
                 (:gt difference 100))),
```

that is "any change in weight such that the difference between the new and the old value is at least 100"

b - Topics of interest

FIGURE 4. The Information Agent at work.

Suppose first that Agent-1 informs the IA about a certain part `p-111,` by sending it the following information:

```
(part p-111)
(part-of p-111 c-12)
(part-of p-111 c-13)
(part c-12)
(part c-13)
(weight c-12 2700)
(weight c-13 3400).
```

Given its prior knowledge about parts and components, the IA will infer that `(component p-111) and (weight p-111 6100), hence (heavy-component p-111),` thus informing Agent-2 about some information relevant to its interests. The processes required for this inference use terminological knowledge, e.g. any `part` is a `component` and "aggregation" knowledge (for weights, cost or other similar properties) expressed with patterns and events.

Later on, Agent-1 may modiy the `weight` of `c-12:`

```
(retract (weight c-12 2700))
(assert (weight c-12 1000))
```

As a result, the `weight` of `p-111` becomes 4400, `p-111` ceases to be a `heavy-component` of interest to Agent-2 and the IA informs Agent-2 about that. Also, using an event that builds an `weight-change` instance whenever a `weight` is changed, the IA realizes that it has two `weight-change` instances, both with a "`difference`" of 1000 and sends both to Agent-3 since both match its interest.

4.0 Some IA specific issues

After discussing the general solutions for developing IA-s with description logic technology, in this section we tackle a number of issues that are important for IA-s and require extending the standard description logic technology in order to be treated correctly. These issues are related to query management, consistency management, time management and change management.

4.1 Query management

Description logic languages provide powerful query services. This power stems from the representation of a query as a concept, the meaning of such a query being the set of instances belonging to the concept. Such queries can be answered by first classifying the concept in the concept taxonomy (remember this is a theorem proving activity) and then collecting (and usually filtering) all instances (and their roles) of all subsumed concepts. This approach is essentially deductive and relies on interpreting the semantics of the query and of the knowledge-base.

4.1.1 Conjunctive patterns

While the above standard approach can handle queries aimed at sets of propositions related to a single concept, it can not help when one is interested in collecting propositions related to several concepts among which arbitrary relations exist (specified by means of predicates and paths along the relations among concepts). For example, suppose an agent is interested in all pairs of machines that are scheduled for maintenance activities during adjacent time intervals. The conjunctive pattern mechanism we have extended our language with allows such queries to be easily formulated and answered. For example:

```
:and
((machine ?x)(machine ?y)
(maintenance-time ?x ?t1)
(maintenance-time ?y ?t2))
:such-that
(meets ?t1 ?t2).
```

This pattern can be used in a query that would return all desired pairs of machines. The pattern is made of proposition patterns and arbitrary predicates applied on the matching propositions.

4.1.2 Making closed-world assumptions

Closed world assumptions are often used when computing answers to queries. Consider the usual situation when agents question IA-s about information relevant to them. The IA may supply the requested information by listing a collection of data matching the request or may give a null answer. The problem is what does the null answer represent: a “don’t know” answer, or a definite negative answer. The fact that the IA has no information about something can be caused either by the fact that although the agent should know about the issue, it has not received anything yet, or by the fact that no agent will ever specify anything about that topic to that particular IA. It thus becomes necessary to be able to know in advance which is the range of information that the IA will be given. This can be achieved by having the agents volunteering information to the IA specify at an initial moment the topics they will supply information about. In this way, any IA will know if a given query falls within the range of information the agent should have (in which case the closed world assumption may be considered) or if it is outside its competence, in which case the correct answer must be “I am not competent about this issue”. If agents supply both the topics of interest and the topics of volunteered information, then the issue can be settled from the beginning by imposing as a minimal consistency condition that for each topic of interest there must exist at least one source of volunteered information. In the current IA implementation this is solved by using the KQML `subscribe` and `advertise` performatives, which force0 agents to specify in advance the range of information they are interested to acquire or they will supply.

4.2 Consistency management

Consistency management is the process of discovery and removal of contradictions in the information maintained by the information agent. The IA operates by receiving information volunteered by the agents it services, performing inference to discover new information and finally determining (1) which of the supplied or derived information matches the information interests of other agents (2) whether the supplied and derived information is in any significant sense contradictory. The latter issue is of particular importance: if computerized models are to be introduced in real open environments where everything changes and contradictions abound, then a primary function of these models should be the early detection and removal of inconsistencies in a manner affecting as little as possible the normal operation of the real system.

How is this objective achieved in the IA? Once information has been volunteered to the IA, the IA will reason based on it. The reasoning process may:

- discover new information as implied by the definitions of the terms used. For example, if p-111 is a *pipe*, then it is also a *part*, a *structural component* and a *component*.
- uncover contradictory assertions of the "p & q-> false" type. For example, the above inference yields *component(p)* and a previous derivation might had resulted in *process(p),* with background knowledge telling that *process(?x)&component(?x)*=> *false*.
- with reasoning about time taking place, uncovered contradictions may refer to future states of the world. For example, knowing that a machine is dedicated to perform an operation during the [t1, t2] time interval, turning off the machine at t3<t2 will result in an inconsistent state in the future.

We assume that the reasoning process involves axioms describing both the definitions of terms (as in the first example above) and the various possible interactions among entities (as in the last example above). The axioms in the latter class may be part of more general ontologies of change complementing the existing ontologies relating activities, states, causality and resources.

Inconsistencies can arise at two levels. The first is the level of the terminologies employed by the problem-solving agents. First, each of the terminologies of agents must be locally consistent, that is no problem-solving agent must have definitions of the form C=A&B, where A&B=>*false*. Second, if two or more agents share concepts, their local definitions must be equivalent. Since each local concept definition is formed by logical operators applied on other local concepts, this condition propagates constraints on the local definitions that must be satisfied. In order to be able to define and enforce these conditions, we need to formulate the concept consistency problem in a formal language for which formal verification mechanisms can be applied. Fortunalely, a description logic language provides the necessary means to do this.

The second level at which we study consistency is the level of assertions formed with concepts. Any assertion must be consistent with the terminology on one hand and with possibly other contingent (non-definitional) constraints imposed e.g. by the domain models on the other hand.

Changes at both of these levels may require verification of consistency and modification of the models. If changes are performed at the terminology level (e.g. changing the definition of a "*machine*") then both the terminology and the assertion base must be checked for consistency and possibly reorganized. This is similar to the problem of schema-modification in a database. If changes are performed only at the level of the assertions, then only the assertion base will have to be verified and reorganized. Again, the particular language used for describing the terminology and models will impose the particular techniques for carrying out the consistency preserving operations.

4.2.1 Terminology-level modifications

The general approach to terminology-level modifications is to devise a typology of allowed modifications and to develop techniques for dealing with the allowed classes of modifications. Examples of classes of modifications include: generalizing/specializing the domain or range of a role, generalizing/specializing the type of a role value, modifying cardinality restrictions of role values, etc. For each of these modifications, verifications and repair activities can be defined. For example, if the domain of a role has been generalized, then the role may appear in new concepts and instances; if it has been specialized, existing concepts and instances can become inconsistent.

4.2.2 Assertion-level modifications

The assertional information in the IA is of two types: supplied by problem-solving agents agents and derived by the IA. Whenever a contradiction occurs, the IA will be able to trace the agent-supplied assertions from which the contradiction was generated. In general, these assertions can be described as a CNF formula: (a1 & a2 ...)V ... V(p1 & p2 ...). After tracing the sources of the contradiction, the next activity is removing the contradiction. This amounts to denying one of the conjuncts, so that the derivation of the contradiction would no longer be valid. Each conjunct represents information directly supplied by an agent, hence it must be decided which of the agents can be "contradicted".

In order to do that, it is vital to consider the consequences of retracting an assertion upon the rest of the assertion base. When an assertion is retracted, believed assertions will become unbelieved and unbelieved assertions may become believed. The assertions whose belief status changed might have been used by other agents that might have made significant internal decisions based on them, and retracting these decisions may have costs associated (e.g. rescheduling activities, modifying purchase orders, etc.).

For this reason, the decision to retract an assertion must consider (1) the "authority" of the agent that has initially asserted the proposition and (2) the consequences of the retraction in terms of costs incurred upon other agents that used the information (we call this the "deniability" of the information). In order to deal with these issues, we can consider several alternatives, based on whether the authority and deniability can be determined at run-time or before. For authority measures, we can assume that each agent volunteering information can also associate a measure of its authority with respect to the supplied information (or else some statically defined default may be used). For deniabilities, let us consider both possibilities. First, if deniabilities can be determined only at run time, the IA will have to engage conversations with agents to find out what deniabilities are. Second, if deniabilities can be determined before execution, we can imagine attaching them to the topics of interest agents advertise. In this case, the IA will avoid the seemingly expensive process of questioning each interested agent about the costs of modifying a belief status. In the latter case it becomes possible to have an authority/deniability-based approach to belief revision that does not come with run-time penalties. Finally, if authorities and deniabilities can not be determined at all, we may try an approach in which the changed beliefs are those that affect least the rest of existing beliefs in the system (i.e. leave the rest of beliefs as close to the initial state as possible).

Given this range of possibilities, we have implemented an authority/deniability belief revision mechanism that tries to delay the moment the authority and deniability of a piece of information has to be acquired as much as possible in order to avoid any dialogue or negotiation on this theme. Authority is specified with each piece of information the moment it is supplied to the IA. In the current version, authority is specified as a numeric value. Deniability is specified at a moment close to the information use time. In the current system this is either the time topics of interest are advertised or the time information is requested, that is when the query is formulated. Deniability is represented as a boolean value (true means OK to deny, false means better not deny it). A query such as

```
:and ((machine ?x)(weight ?x ?y))
:deniability nil
```

is taken to mean that any answer to it should be considered undeniable, hence the contradiction removing mechanism should better not retract it later. The actual algorithm used to resolve contradictions works by retracting one of the terms that caused the contradiction using the authority/deniability approach discussed and is shown in figure 5. The algorithm is incorporated into the truth-maintenance system that supports the operation of the language's A-Box.

When a contradiction *p&q->false* is encountered:

1. Compute the sets of premises each term (*p* and *q*) is derived from:

set1 = support-set (*p*)

set2 = support-set (*q*)

2. Merge the two sets *set1* and *set2* into a single one, *S*, and sort it in increasing order of authority of the component propositions.

3. For each premise *p* in *S*, compute the number of undeniable propositions that would have to be retracted if premise *p* is retracted.

4. Retract the first premise in *S* that is associated with the smallest number of undeniables.

5. Check if the initial contradiction can be rederived (by checking if there exists another derivation for the retracted term). If so, repeat the above procedure. Else exit.

FIGURE 5. Authority/deniability algorithm for contradiction removal.

4.3 Managing time

Explicit reasoning with time is an important capability that an IA must provide. Recently, there have been some proposals to extend description logics with temporal reasoning. One general attempt of this sort is [Schmiedel 90], that extends description logics with temporal logic [Shoham 87] and Allen's interval calculus[Allen 83]. This general framework allows formulating temporal concept definitions, that is concept definitions interpreted wrt time intervals (e.g. a `former-manager` is someone who has been a manager sometime in the past) and proving subsumption among such concepts (e.g. `(operator at 'june 1990')` subsumes `(operator at '1990'))`.

As algorithms for temporal subsumption are not very well studied at present, we have decided, at least in the short run, to support temporal reasoning in a more limited manner that is both efficient and useful. The approach is to let the terminological component unaffected (thus we do not have temporal concept definitions) but to intoduce time in the assertional component. As a result, the proposition database maintained by the assertional component is indexd by time intervals and the truth maintenance process infers time intervals for each proposition. For example, the assertional component represents the fact that John is a manager during 1993 as `(manager John (at 1993))` and that Bill operates machine M1 during december 1993 as `(operates M1 Bill (at december 1993))`. The justifications propagated by the truth-maintenance system also propagate time intervals. Such a justification is, for example:

```
(failure-free M1 (at december 1993))
  <- (operates M1 Bill (at december 1993)).
```

The usefulness of this approach is that the query service can now answer queries that mention time intervals. Thus, a query about who operated all failure-free machines during december 1993 can be stated as:

```
and:((failure-free ?machine (at december 1993))
  (operates ?machine ?who)).
```

4.4 Managing change

Agents continously supply information to the IA and the IA processes it and supplies derived or integrated information to interested agents. When the state of affairs described by the supplied information changes, interested agents must be notified. A typical example is selecting a machine for scheduling an operation. The scheduling agent may question the IA about all machines that are expected to be availabe during a specified period of time (e.g. for which no maintenance activities will be carried out). The scheduling agent will receive such a list and then will select one machine for the desired operation. Later on, some other agent may change the schedule for maintenance activities for the selected machine. The scheduling agent needs to be informed about that in order to take appropriate action.

This kind of change management can be supported with the mechanisms discussed up to now. The following is an interaction scenario that shows how the above situation can be handled using queries and demon posting:

1. The Scheduling agent queries the IA about the possible to use machines:

```
and:
   ((machine ?machine)
   (maintenance-schedule ?machine ?maint))
```

2. The IA answers with the list of machines and their maintenance schedules:

```
((M1 maintenance-schedule-M1)

(M2 maintenance-schedule-M2) ... )
```

3. The Scheduling agent decides to use machine Mj because of its particular maintenance-schedule. It advertises its decision by posting a demon that will be fired by the IA:

```
   (demon
   :if (maintenance-schedule Mj ((march 1993) (september
   1993)))
   :then (use OP-12 Mj)(time OP-12 (april 1993))(time OP-12
   (august 1993)))
```

4. Another agent (with adequate authority) later changes the maintenance schedule:

```
(assert
   (maintenance-schedule Mj ((march 1993) (august 1993))))
```

5. This makes the propositions inferred by the demon become false and hence they are retracted by the IA by sending denial messages to all agents that used these conclusions (including the Scheduling agent that initially posted the demon):

```
(deny
  (use OP-12 Mj)
  (time OP-12 (april 1993))
  (time OP-12 (august 1993))).
```

6. At this moment the Scheduling agent (and any other interested agent) knows that machine `Mj` is no longer available for operation `OP-12`.

5.0 The Agent Program

The Agent Program has the purpose of turning the description logic problem-solver into an autonomous agent. This is an instance of the general problem of "agentification" [Shoham 93] in which an existing system has to be interfaced to a world of agents by making it behave in manner comprehensible by the other agents.

The Agent Program interprets a protocol for agent interaction and translates expressions of this protocol into the language of the problem-solver and viceversa. For the IA reported here, the interaction protocol is the one specified by the KQML [Finin et. al. 92] agent communication language. KQML is a speech-act based language. It organizes its speech-acts (called performatives) into several classes: informative performatives, query performatives, effector performatives, generator performatives, network performatives, and others.

The KQML implementation maintains several databases storing the agents the IA communicates with, the topics of interest of these agents, the sort of information these will supply to tho the IA, the status of communication with agents, the information already delivered to agents. The mapping of KQML performatives onto the description logic language is pretty straightforward as the query performatives are handled by the query services, the effector performatives are handled by the asserting and retracting services, and other more complex performatives (such as the generators) are supported on the basis of these.

The KQML `subscribe` and `advertise` performatives are used to circumscribe the topics of interest of agents and the topics they will supply information about. This helps the IA make close-world assumptions when appropriate.

The content language (the language in which information is transfered among agents) assumed by the Agent Program is a subset of the Knowledge Interchange Format (KIF).

6.0 Conclusions

Information Agents are a component of the information infrastructure supporting collaborative enterprises. One contribution of this paper is the identification of the Information Agent notion and of the functions it can play in a cooperative information system environment. We have proposed a simple structure for the IA, composed of an Agent Program and a Problem Solver and we have discussed our choices for these components. We have illustrated this notion in the context of the Integrated Supply Chain Management system under development in the Enterprise Integration Laboratory. The second contribution refers to the construction of Information Agents using description logic representation systems. We have shown that this approach provides leverage for building Information Agents that can automatically determine the relevance of information, automatically distribute it, maintain consistency at both the level of the terminology and of the assertion base, automatically organize the shared vocabulary cooperating agents use, apply deduction to respond queries, provide descriptive/intensional answers, reason with partial information.

As illustrated here, the Information Agent is a general tool for collaborative problem-solving whose applications span a wider range than discussed in this paper. Equally interesting applications of the concept can be found in concurrent design, where the ability to notify all affected parties of design changes so they may assess the impact is one essential feature that will be heavily exploited. Another possibility is in the area of computer-supported collaborative work, where the Information Agent can be used as a personal assistant that will be able to extract for its user the relevant information form the wealth of information available through electronic highways and take over a number of information transmission tasks such as informing about changes in calendars, notification of important events, etc.

At the moment, the literature mentions too few attempts at building similar systems to allow a comparative discussion. [Lovstrand 91] mentions a system for information distribution used in conjunction with the electronic mail, while [Pan and Tenenbaum 91] and [Tenenbaum et. al. 92] only mention more or less similar concepts for enterprise integration and concurrent engineering, without touching design and implementation issues.

7.0 Acknowledgements

This research is supported, in part, by the Manufacturing Research Corporation of Ontario, Natural Science and Engineering Research Council, Digital Equipment Corp., Micro Electronics and Computer Research Corp., and Spar Aerospace.

8.0 References

1. Allen, J., F., (1983) Maintaining Knowledge About Temporal Intervals, Communications of the ACM, 26 (11), pp 832-843, 1983.

2. McAllester, D., (1990) Truth Maintenance, Proceedings AAAI-90, pp. 1109-1116.

3. Barbuceanu, M. (1993) Models: Toward Integrated Knowledge Modeling Environments, Knowledge Acquisition 5, pp. 245-304.

4. Borgida, A., Brachman, R. J., McGuiness, D. L., Resnick. L. A., (1989) CLASSIC: A Structural Data Model for Objects, Proc. 1989 ACM SIGMOD International Conference on Management of Data, june 1988, pp. 59-67.

5. Brachman, R. J., Schmolze, J. G., (1985), An Overview of the KL-ONE Knowledge Representation System, Cognitive Science 9(2), April-June 1985, pp. 171-216.

6. Finin, T., et al. (1992) Specification of the KQML Agent Communication Language, The DARPA Knowledge Sharing Initiative, External Interfaces Working Group.

7. Fox, M. S., (1993) A Common-Sense Model of the Enterprise, Proc. of Industrial Engineering Research Conference, 1993.

8. Fox, M. S., Chionglo, J., Barbuceanu, M. (1993) The Integrated Supply Chain Management System, submitted.

9. McGregor, R., Bates, R., (1987) The LOOM Knowledge Representation Language, ISI IRS-87-188, USC/ISI Marina del Rey, CA

10. Lovstrand, L., (1991) Being Selectvely Aware with the Khronika System, Proc. of Second European Conference on Computer Supported Cooperative Work, Bannon, Robinson and Schmidt (eds), Sept 1991, Amsterdam, The Netherlands.

11. Luck, K., Nebel, B., Peltason, C., Schmeidel, A., (1987) The Anatomy of the BACK System, KIT Report 41, Fachbereich Informatik, Technische Universitat Berlin, Berlin.

12. Pan, J.Y.C., Tenenbaum, J. M., (1991) An Intelligent Agent Framework for Enterprise Integration, IEEE Transactions on Systems, Man and Cybernetics, 21, 6, pp. 1391-1408, 1991.

13. Roboam, M., Fox, M. S., (1992) Enterprise Management Network Architecture, A Tool for Manufacturing Enterprise Integration, Artificial Intelligence Applications in Manufacturing, AAAI Press/MIT Press, 1992.

14. Schmiedel, A., (1990) A Temporal Terminological Logic, Proceedings of AAAI-90, pp 640-645.

15. Schmolze, J.G., (1985) The Language and Semantics of NIKL, BBN Inc., Cambridge, MA.

16. Shoham, Y., (1987) Temporal logics in AI: Semantics and Ontological Considerations, Artificial Intelligence 33 (1), pp 89-104, 1987.

17. Shoham, Y., (1993) Agent-Oriented Programming, Artificial Intelligence 60 (1993) pp 51-92.

18. Tenenbaum, J. M., Gruber, T. R., Weber, J.C., (1992) Lessons from SHADE and PACT, Enterprise Modeling and Integration, C. Petrie (ed), McGraw-Hill 1992.