Virtual Enterprise Design – BDI Agents vs. Objects

Iyad Rahwan[1], Ryszard Kowalczyk[2], Yun Yang[1]

[1] School of Information Technology, Swinburne University of Technology
P.O.Box 218, Hawthorn, VIC 3122, Australia
{iyad, yun}@it.swin.edu.au
[2] CSIRO Mathematical and Information Sciences
723 Swanston St, Carlton, VIC 3054, Australia
Ryszard.Kowalczyk@cmis.csiro.au

Abstract. Current research identifying architectures for a virtual enterprise has moved from information modelling to role modelling. Thus, a high level of autonomy results from the distribution of responsibilities, capabilities, and knowledge among different business units in the virtual enterprise at the design stage. Current trends tend towards using object-oriented technology as an effective abstract system design and implementation methodology. We argue that applying the software agent paradigm to the virtual enterprise provides various advantages on both the design and operational levels. We further show that the Belief Desire Intention agent architecture has additional abilities of mapping real world business unit autonomy and interaction. We also introduce the Belief Desire Intention agent paradigm capability of facilitating highly flexible (agile) enterprise design and implementation.

1 Introduction

A significant enhancement of inter-enterprise transactions has been provided by the emerging information technologies (IT). This has raised a need for establishing a new paradigm defining architectures, standards, and design and implementation policies for inter-enterprise systems, which realize the capabilities of IT and maximize their utilization in all possible ways. The Virtual Enterprise (VE) paradigm emerged in order to address this need and produce efficient procedures for dealing with the above requirements. Although there is no clear standardized definition of the VE, there is a general agreement between currently proposed definitions. For example, the NIIIP project [1] defines the VE to be "a temporary consortium or alliance of companies formed to share costs and skills and exploit fast-changing market opportunities". While for Walton and Whicker [2] "the Virtual Enterprise consists of a series of co-operating 'nodes' of core competence, which form into a supply chain in order to address a specific opportunity in the market place". In other words, the virtual enterprise is an organization of enterprises sharing resources and working cooperatively towards the achievement of mutual benefits. Terms such as e-commerce, supply chain management, and virtual corporation correspond to closely related concepts. In a VE, there is no centralized control, neither is there a hierarchy of enterprise management levels. Instead, the cooperation of independent, self-interested units results in convergence towards an overall welfare.

In the next section, we outline the problem with current virtual enterprise design, and hence implementation approaches. Then, in section 3, we present an overview of agent technology and different agent architectures. Section 4 discusses the problem with current approaches towards designing the virtual enterprise, particularly the object-oriented approach. Section 5 outlines the major advantages of the agent approach. In Section 6, we introduce the specific capabilities of Belief Desire Intention (BDI) agents in modelling and implementing virtual enterprises. Next, we show how the BDI approach corresponds well to a set of well-known agile (flexible) enterprise design guidelines in section 7. Finally, a number of conclusions are drawn, and future research is outlined.

2 Virtual Enterprise Problems

There is a great diversity in the way different organizations do business. Different companies have different priorities, business process definitions, ontologies representing business documents and procedures, and different tools for modelling their overall strategies and plans. Aspects such as the explicit representation of coordination strategies (by adoption of workflow technologies), the execution of the coordination strategies (workflow engine), and cooperative system organization (negotiation, coalition formation) represent major issues to be resolved [3]. The behaviour of each company regarding its participation in a VE can be explicitly configured and stated through a *plan* and other general *profile characteristics* [3]. The Workflow Process Definition Language following the workflow reference architecture proposed by the Workflow Management Coalition [4] is one approach to represent dynamic behaviour. It is based on explicit representation of the workflow of business processes as well as all possible exceptions and the way each one of them should be handled. Another method to represent the dynamic behaviour of each VE node is the use of Petri Nets [3].

The problem with using such approaches down to the detailed level is that an explicit representation covering all possible cases introduces a scalability problem. This makes exception handling module design an extremely complex task, especially within the VE context. In this paper, we propose the incorporation of agent technology in order to create dynamic VE systems while reducing the complexity of the configuration process. Instead of explicitly specifying how each situation must be handled at the elementary design stage, an overall process design is implemented, and responsibilities are given to different autonomous units which are capable of solving their own internal problems. The different units are then provided with a coordination mechanism, allowing for the dynamic nature of the system to emerge during the system implementation, deployment and operation phases. We do not assume static, pre-negotiated intra- and inter-organizational workflow. In contrast, as proposed in [5], we view the establishment of a VE as a problem of dynamically expanding and integrating workflows in decentralized, autonomous and interacting workflow systems. Workflow techniques are best suited for implementing an abstract business process model which describes the overall process steps that have to be performed to achieve a specific business goal according to well defined business rules and the respective responsibilities of process participants [6]. The various process steps can then be realized by *intelligent* business components (implemented by intelligent software agents) that perform specific business transactions.

3 Software Agents for Modelling

Agent-based computing has been considered 'the new revolution in software' [7]. The following definition is adapted from a definition proposed by [8]: "an agent is a software system (or system component) that is *situated* in an environment, which it can *perceive*, and that is capable of *autonomous* action in this environment in order to meet its design objectives."

Jennings and Wooldridge [9] proposed that an *intelligent* agent is an agent that is capable of *flexible* autonomous behaviour, where flexible means:

? ***Responsive***: able to perceive the environment and respond in a timely fashion.
? ***Proactive***: exhibit goal-directed behaviour and take initiative when appropriate.
? ***Social***: able to interact/communicate when appropriate with humans and other agents.

From this point and on, we will use the term agent to refer to intelligent agent.

One of the interesting and most important aspects of agents is that they facilitate *cognitive* modelling (based on behavioural aspects fulfilling the purpose), as opposed to *role* modelling (based solely on purpose). If we are able to define a framework that best describes an agent, and how it interacts with its environment and with other agents, we will have achieved a significant contribution towards the abstraction of system design, and gain better mapping of real world problem solving into our computer systems. Moreover, the autonomy and pro-activity of agents facilitate a system in which entities take action without the need for centralized control. More advantages of using the agent approach are discussed in section 3.

A number of proposals have been made describing different internal architectures of agents and their implications on the performance of agent systems.

? **Reactive Agents.** In a reactive agent, the behaviour modules are finite state machines. Behaviours are implemented as rules of the form: situation $\rightarrow$ action [8].
? **Deliberative Agents.** A deliberative agent has explicit goals. It reasons about which, when, and how to achieve them [8]. A deliberative agent has an internal state, allowing flexible, history-sensitive, non-deterministic behaviour.
? **Hybrid Agents.** Hybrid agents combine the best of reactive and deliberative features. BDI Agents, discussed next, are one example of Hybrid Agents.

The Belief Desire Intention (BDI) model combines reactive and deliberative approaches in a uniform way. BDI agents have internal *mental* states [10]. A mental state is comprised of three concepts:

? **Beliefs**: The information the agent currently believes true. This information could be about the agent internal state or about the environment. We should emphasize the difference between *knowledge* and *belief*. While knowledge is assumed to be always *true*, beliefs are considered to be *true* but possibly *false*. This captures the dynamic nature of the agent's information about itself and the environment.

? **Desires (goals)**: Desired future states.

? **Intentions**: Commitments to action. The notion of intentions is closely related, but not identical, to the notion of plans. In a certain situation an agent might have a number of possible plans. The selected plan, and the commitment to taking action become an intention. In other words, an intention is an instance of a plan.

The control cycle of BDI agents is described in *Figure 1*. First an event or goal is selected for processing. The agent then finds plans that are applicable to the current situation. Appropriate plans are chosen, resulting in the creation of intentions. The agent then executes the enabled intention, starting with the first step. This may result in an action being executed, or an event being posted, hence invoking sub-plans, etc.

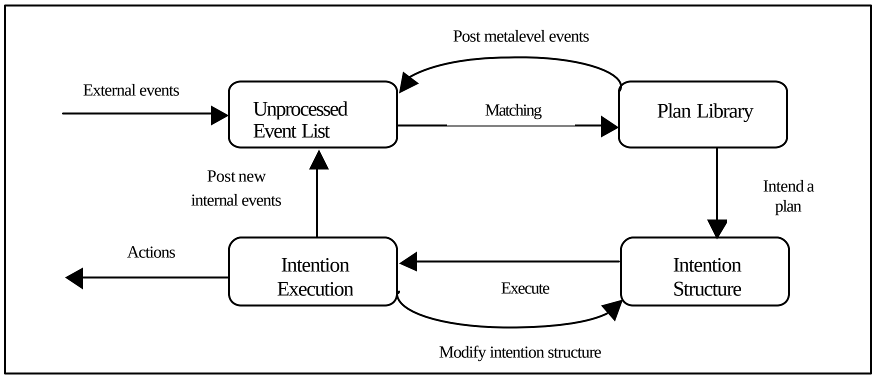

Figure 1. BDI Control Cycle

Due to its highly abstract philosophical origins, the BDI approach has proved valuable for the design of agents that operate in a dynamic environment, and that respond flexibly to changing circumstances despite incomplete information about the state of the world and other agents in it [11]. The BDI reasoning model resembles the kind of reasoning that we appear to use in our everyday lives [8]. Another major attraction of this model is that it provides a clear functional decomposition, similar to, but more uniform than that provided by hybrid layered architectures.

There is an ongoing argument around the best way to represent and generate plans in different agent models. For example, some argue that rather than explicitly providing a library of plans and choosing from it, a rule-based agent has an implicit representation of different plans, and plans emerge at run time through following those rules. The BDI cycle "provides for a spectrum of behaviours ranging from purely deliberative, to highly reactive, depending on the structure of the plans provided and on decisions such as how often to take account of changes in the environment while executing plans" [11]. In other words, the BDI model provides an effective, highly flexible paradigm for developing intelligent software agents that can be configured on various levels of intelligence and deliberation to accommodate the nature of the problem.

Since this paper is in the context of facilitating VE, there is an inherent necessity for having multiple agents in the systems. Agents, in this case, will not only be able to reason autonomously and effectively about achieving local objectives, but they will also be able to gain the benefits of interacting with other agents in the VE. *Interaction protocols* are "recipes" for structured communication between agents [12]. Ultimately, an interaction protocol consists of: 1) a set of messages encoded in an Agent Communication Language *ACL*; and 2) a set of rules governing conversations (sequences of messages).

4 Approaching the Virtual Enterprise Problem

Enabling the VE requires the adoption of a new technical infrastructure, as well as an effective design methodology. As systems scale up, and the complexity increases, there is a need for a design methodology, which is highly abstract yet, has the ability to be directly implemented. The most dominant methodologies currently used adapt the object-oriented (OO) system design principles. These methodologies provide a higher level of abstraction of the problem. In the object-oriented paradigm, the system is comprised of several, possibly distributed, objects interacting with each other through predefined interfaces. The interface describes the object's services which can be invoked by other objects. In other words, an object is a *passive* system component which is able to respond to predefined requests and react accordingly. An object system is usually governed by a central process/object responsible for the coordination between objects and the flow of control between them towards the achievement of the desired result.

The first drawback of objects is related to the level of abstraction they provide. Business *objects* make a major contribution to modelling *information* in the enterprise. While business *agents* extend this capability to model *behaviour* in the enterprise. This fact means that the behaviour of agents can be modified *dynamically*, due to learning or influence of other agents or the environment. Moreover, agents can dynamically cooperate to solve problems.

Another important difference between the object and agent paradigms is related to the level of autonomy. The object paradigm uses the notion of *encapsulation* to give objects control over their own internal states. An object can declare that a certain instance variable (or method) is private (only internally accessible) or public (accessible both internally and by other objects). While an object may exhibit autonomy over its *state*, by making instance variables accessible only via certain methods (controlled channels); it does not exhibit autonomy over its *behaviour* [8]. That is because once a method has been declared public (at design time), the object has no control over whether that method is executed. Any other object could *invoke* this method if it wants to. In a multi-agent system, an agent *requests* an action to be performed by another agent rather than *invoking* this action. And this request could be accepted or refused by the serving agent (i.e. the decision lies in the hands of the serving agent rather than the client. Decision could depend on various factors such as the current domain state or the identity of the client). Objects are considered to provide good mapping of real-world problem solving. But that limitation affects this mapping capability dramatically, since the actual problem description needs to be conceptually modified to accommodate the limited capabilities objects offer.

The major reason for the apparent failure of object orientation to deliver on reuse is insufficient attention to the issue of domain understanding and the representation of this understanding in an unambiguous and precise way [13]. The OO paradigm is intended towards the implementation of software systems rather than rich business concept representation (such as rules, constraints, goals and responsibilities). Goal seeking behaviour, policies and trust are all enterprise concepts which do not naturally fit into a computational object model. Based on this argument, A. Wood et al [14] showed that conventional object modelling constructs are not sufficient for modelling enterprises.

As mentioned in the previous section, the passiveness of objects requires the availability of "controller objects" which are responsible for the coordination of control among other objects. As the size of an OO system grows, "the number of messages between objects grows non-linearly and controller objects themselves need to be coordinated or they can become performance bottlenecks" [13].

It is important to denote that many of the advantages of agents mentioned above could indeed be implemented using object tools. But our argument is based on the fact that those attributes are not components of the basic object-oriented model.

5 The General Agent Approach

Following the definition of the VE mentioned earlier, the VE creation could be viewed as a Cooperative System design problem. A Cooperative System is "a system in which a set of autonomous agents (computational and human) interact with each other through sharing their information, decision making capabilities and other resources, and distributing the corresponding workload among themselves, in order to achieve common or complementary goals" [3]. There is an apparent similarity between the definition of the VE and that of a Cooperative Agent System from two perspectives: the problem addressed, and the approach adopted towards solving it.

Both object and agent paradigms address *change* and *complexity* but to different levels. We are not proposing the disposal of the widely used distributed object foundation. Instead, agent technology can be built on top of the distributed object infrastructure to provide a natural merging of object orientation and knowledge-based technologies [13]. Agents' ability to provide reasoning capability within the primitive application component logic facilitates direct mapping and encapsulation of business rules within the organization.

The motivations behind the agent solution could be summarized by the following points adapted from [15]:

1. The need for a higher-level design approach, which is capable of mapping effectively to real world problem solving capabilities as opposed to approaches where the actual problem description might need to be conceptually modified to accommodate the limited capabilities of the approach.
2. The domain involves an inherent distribution of data, problem solving capabilities, and responsibilities.

3. The need to maintain the autonomy of organizational business units and sub-components.
4. The need for accommodating complex interactions, such as negotiation, coordination, cooperation, and information sharing. This calls for more sophisticated social skills provided by agents.
5. The fact that a detailed solution to a problem cannot be designed from start to finish, as the business environment is highly unpredictable and a rigid one-time design does not accommodate such changes. Instead, a general workflow definition of business processes could be implemented, while autonomous, adaptive components deal with specific business transactions and their internal problems.

6 Advantages of The BDI Agent Approach

In addition to the advantages of using the agent paradigm mentioned in the previous section, there are BDI Agent-specific features that are central to our argument. We will show that each element of the BDI model offers an advantage in the scope of capturing significant aspects of virtual enterprises.

Beliefs: Going back to the object-oriented model, there is an overall agreement concerning the benefits of the encapsulation of information within enterprise objects. This advantage is still realized by the BDI Agent model through the agent's beliefs (its *current* knowledge about itself, its environment, and other agents). Furthermore, the BDI architecture offers a higher level of abstraction by explicitly allowing beliefs to have a direct impact upon the agent's behaviour. This allows for the agent's behaviour to be dynamically modified as its knowledge about the domain changes. For example, one of the enterprise concepts that do not naturally fit into a computational object model is the specification of *policies* that govern the behaviour of enterprise objects [14]. An agent's set of beliefs, on the other hand, could include specifications of these policies, as well as having the ability to accommodate *changes* in such policies as part of its nature.

Desires: There is a rising need for capturing the goal directed behaviour in enterprise business units. For example, if the coordinating process in an enterprise were based on precise specifications of sub-process activities, sub-processes would not be allowed to dynamically change their activities, as this will cause coordination failure. Capture of intentional information concerning higher level business objectives of processes and the mapping of specific activities to those objectives allows dynamically changing systems to maintain coordination across a useful range of situations [16]. This way, activities and strategies that constitute the business unit task are allowed to autonomously evolve as required by changes in their local domain.

Intentions: Intentions reflect the reasoning ability which an agent pursues before it takes decisions about its actions (i.e. about what plan to commit next). The ability to choose from different possible plans maps well to a business unit's different strategies for achieving its task. If one plan fails, another could be tried until no other plans are available. This way, no error reporting to the higher-level coordination process or service requestor is required until all possible strategies are consumed without success.

7 Agile Enterprise Design

In the previous sections, we showed that the BDI agent architecture has interesting capabilities of modelling the behaviour of business units. In this section, we will take a further step into showing how the BDI agent paradigm may provide powerful facilitation of highly adaptable (*agile*) enterprise design. We will use a set of agile enterprise system design principles in order to show how the BDI agent paradigm could support such an enterprise through attributes central to its description.

An agile enterprise is one that is broadly change-proficient [17]. In other words, an agile enterprise manages and applies knowledge in order to accommodate and cause change to its own advantage. Regardless of the strategies chosen, effective implementations of such an enterprise employ a common set of principles that promote proficiency at change. Designing agile systems, be they entire enterprises or any of their critical elements like business practices, operating procedures, supply-chain strategies, and production processes, require designing a sustainable proficiency at change into the very nature of the system [18].

R. Dove [18] identified ten key design principles which are responsible for the high adaptability in a number of industrial applications. These principles have emerged from observations of both natural and man-made systems. *Table 1* shows these principles and the corresponding BDI agent architecture features which facilitate the design and application of each principle.

Table 1. Correspondence between the BDI agent mode l and agile enterprise design principles

Design Principles	**Corresponding BDI Agent Model Features**
Self Contained Units: System of separable self-sufficient units not intimately integrated. Internal workings not important externally.	An agent is an autonomous, self-sufficient unit capable of performing a task proactively and independently. It can interact with other agents without central control. An agent encapsulates capability implementations resulting in service abstraction.
Plug Compatibility: System units share common interaction and interface standards, and are easily inserted or removed.	Agents use an *interaction protocol* implemented using an agent communication language (ACL) such as KQML to enforce message syntax. Message semantics could be realized by ad-hoc or industry standards such as EDIFACT.
Facilitated Re-use: Unit inventory management, modification tools, and designated maintenance responsibilities.	Agents are modular, and belong to classes from which any number of agents could be instantiated. Modification could be done by either changing the agent's state (beliefs) or by replacing the agent by another with more sophisticated model to support extra or more efficient functionality.

Non-Hierarchical Interaction: Non-hierarchical direct negotiation, communication, and interfacing among system units.	Agents interact with each other without centralized control through direct messaging and negotiation. Bidding for internal jobs could be done among groups of agents, providing granularity of interactions. Because BDI agents introduce the notion of planning, sophisticated negotiation and collaboration techniques could be incorporated within the business unit itself.
Deferred Commitment: Relationships are transient when possible; fixed binding is postponed until immediately necessary.	Individual business unit agents are assigned job fulfilment in real time rather than pre-specifying a complete detailed workflow of system processes. This could be done through interaction with other agents as imposed by the situation. Since there is no heavily centralized control, new agents can be easily added to the system to facilitate unsupported functionalities, allowing the system to grow dynamically. Another level of deferred commitment is present in the fact that agents perform online planning according to situations.
Distributed Control and Information: Units respond to objectives; decisions made at point of knowledge; data retained locally but accessible globally.	Each agent has a private representation of its own objectives (desires), which are directed towards the overall system performance. This enables agents to *decide*, locally and in real time, what to do next. Information distribution advantages are similar to those proposed by object models (encapsulation), but the beliefs notion of BDI agents allows for more flexible knowledge representation (for example, allowing for true and false beliefs to be included).
Self-Organizing Relationships: Dynamic unit alliances and scheduling; open bidding; and other self-adapting behaviours.	Since a BDI agent has its own cognitive model, modifications in its beliefs can cause change in behaviour. Plan generation, and hence decision-making, are dependant on the agents own dynamic model of the environment. Automated coalition formation [19] allows both static and dynamic formation of BDI teams. Coalition formation could be done in many different ways to enable the achievement of mutual goals or the exchange of benefits.
Flexible Capacity: Unrestricted unit populations that allow large increases and decreases in total unit population.	An instantiation of any number of agents is possible as needed. Agents could be added to perform new business functionalities or represent business units. Agents could reside on different machines or be mobile in order to achieve scalability.
Unit Redundancy: Duplicate unit types or capabilities to provide capacity fluctuation options and fault tolerance.	Agent systems allow easy recovery. If after consuming all possible strategies, an agent fails to achieve its task, it could report this to another agent which is capable of dealing with such situation by either finding another way of performing the task or choosing an alternative task.
Evolving Standards: Evolving, open system framework, capable of accommodating legacy, common, and completely new units.	Agents could be designed so that they interact with legacy systems by using technologies such as those used to integrate object systems (eg. CORBA). Moreover, it is possible to upgrade an agent to a version with more functionality, enabling the system to evolve.

8 Conclusions and Further Research

Automating the virtual enterprise is the next step beyond today's e-commerce. The most effective way of designing and implementing the virtual enterprise is that which offers capabilities for direct mapping to the behavioural nature of various business units. This is not currently fulfilled by existing paradigms, such as the object-oriented paradigm, which impose a need for mutating the problem in order to fit into the limited design capabilities the paradigm offers. Software agents can offer a significant advantage to the design and implementation of flexible, adaptive, and scalable virtual enterprises. Furthermore, Belief Desire Intention agent architectures can naturally accommodate a rich representation of various business units' knowledge, goals, and strategic plans. They can facilitate a highly adaptive (agile) enterprise design through attributes that are central to this particular agent architecture.

This paper is a step towards incorporating agent technologies into electronic commerce and virtual enterprises. There is a need for further investigation of agents' capability to offer additional features from the functional point of view, such as negotiation and dynamic planning capabilities. Among different proposed planning, negotiation and collaboration models, effective choices must be made which effectively model real world business practices. More work also needs to be done towards methodologies for the design and analysis of agent systems.

Acknowledgement*: Special thanks to David Kinny for his valuable contributions made in various discussions surrounding software agents and to referees for their comments.*

References

1. The NIIIP Reference Architecture, 1996, www. niiip.org.
2. Walton, J., Whicker, L.: Virtual Enterprise: Myth & Reality. J. Control (1996).
3. Camarinha-Matos, L.M., Afsarmanesh, H.: Cooperative Systems Challenges in Virtual Enterprises. Proceedings of the 2nd IMACS International Multiconference on Computational Engineering in Systems Applications (in CD), CESA'98, Nabeul-Hammamet, Tunisia, April (1998).
4. Workflow Management Coalition: Workflow Management Coalition, The Workflow Reference Model - Document Number TC00 - 1003, Issue 1.1, Brussels, Nov (1994).
5. Chrysanthis, P.K., Znati, T., Banerjee, S., Chang, S.K.: Establishing Virtual Enterprises by means of Mobile Agents. RIDE99, Sydney, Australia, (1999) 116-123.
6. Schmidt, M.T.: Building Workflow Business Objects. Proceedings of the OOPSLA 98. Workshop on Business Object Component Design and Implementation. Vancouver, (1998).
7. Ovum Report. Intelligent agents: the new revolution in software (1994).
8. Wooldridge, M.: Intelligent Agents. In: Grehard Weiss (ed.): Multiagent Systems. MIT Press (1999).
9. Jennings, N.R., Wooldridge, M.: Applications of Intelligent Agents. In: Jennings, N.R., Wooldridge (eds.): Agent Technology: Foundations, Applications, and Markets (1998) 3-28.

10. Rao, A., Georgeff, M.: BDI Agents: From Theory to Practice. Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), San Fransisco, USA, June, (1995).
11. Dignum, F., Morley, D., Sonenberg, E.A., Cavedon, L.: Towards socially sophisticated BDI agents. (To appear) In Proceedings of the Fourth International Conference on MultiAgent Systems, Boston, USA (ICMAS 2000).
12. d'Inverno, M., Kinny, D., Luck, M.: Interaction Protocols in Boa. Proceedings of ICMAS (1998).
13. Farhoodi, F., Fingar, P.: *Competing for the Future with Intelligent Agents*. Distributed Object Computing "DOC" Magazine, Part 1: Oct 1997, Part 2: Nov (1997).
14. Wood, A., Milosevic, Z., Aagedal, J.O.: Describing Virtual Enterprises: the Object of Roles and the Role of Objects. Proceedings of the OOPSLA 98. Workshop on Objects, Components and the Virtual Enterprise. Vancouver (1998).
15. Jennings, N.R., Norman, T.J., Faratin, P.: ADEPT: An Agent-Based Approach to Business Process Management. SIGMOND Record 27(4), (1998) 32-39.
16. Burg, B.K.B.: Using Intentional Information to Coordinate Interoperating Workflows. Proceedings of the OOPSLA 98. Workshop on Business Object Component Design and Implementation. Vancouver, (1998).
17. Dove, R., Hartman, S., Benson, S.: An Agile Enterprise Reference Model with a Case Study of Remmele Engineering. Agiligy Forum, December (1996), Report AR96-04.
18. Dove, R.: Design Principles for Highly Adaptable Business Systems, With Tangible Manufacturing Examples. Maryland's Industrial Handbook. McGraw Hill, (1999).
19. Sandholm, T., Lesser, V.: Coalition Formation Among Bounded Rational Agents. 14th International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada, (1995) pp. 662-669.