Exploiting the Environment for Coordinating Agent Intentions

Tom Holvoet[1] and Paul Valckenaers[2]

[1] K.U.Leuven - Dept. Computer Science, DistriNet
Celestijnenlaan 200A, B-3001 Leuven, Belgium
Tom.Holvoet@cs.kuleuven.be
[2] K.U.Leuven - Dept. Mechanics, PMA
Celestijnenlaan 300B, B-3001 Leuven, Belgium
Paul.Valckenaers@mech.kuleuven.be

Abstract. One large and quite interesting family of MAS applications is characterized (1) by their large scale in terms of number of agents and physical distribution, (2) by their very dynamic nature and (3) by their complex functional and non-functional requirements. This family includes a.o. manufacturing control, traffic control and web service coordination. BDI-based agent architectures have proven their usefulness in building MASs for complex systems - their explicit attention for coping with dynamic environments is one obvious explanation for this. For the family of applications mentioned above, the complexity of the software for the individual agents using traditional BDI-approaches, however, is overwhelming.

In this paper, we present an innovative approach to BDI agents which alleviates agent complexity through so-called "delegate MASs", which use the environment and its resources to obtain BDI functionality. Delegate MASs consist of light-weight agents, which are issued either by resources for building and maintaining information on the environment, or by task agents in order to explore the options on behalf of the agents and to coordinate their intentions. We describe the approach, and validate it in a case study of manufacturing control. The evaluation in this case study shows the feasibility of the approach in coping with the large scale of the application and shows that the approach elegantly achieves flexibility in highly dynamic environments.

1 Introduction

The term "coordination and control applications" can be coined to refer to a large family of application which share a number of characteristics. First, in the applications, one can distinguish (1) an underlying physical or software environment, and (2) a software system that is connected to this environment. The underlying environment contains fixed entities or "resources" capable of performing particular operations, as well as mobile entities which can move in the environment. Second, the coordination and control software system is able to observe and direct the entities in the underlying environment. Third, the purpose of

D. Weyns, H.V.D. Parunak, and F. Michel (Eds.): E4MAS 2006, LNAI 4389, pp. 51–66, 2007.

the application is to execute "tasks". Executing a task requires moving through the environment and performing operations by using resources. The purpose of the software system is to manage the underlying environment by controlling entities that live in the environment, and coordinating the collective behavior of these entities. Fourth, the underlying system evolves several orders of magnitude slower than the coordination and control software. This allows the software to observe the environment and to plan ahead. Fifth, the environment itself is highly dynamic. Resources may crash, new resources may be added, connections between resources may be added, lost, or their characteristics (e.g. throughput, speed) may change. Members of this family of coordination and control applications include manufacturing control, traffic control and web service coordination, but also supply chain management and multi-modal logistics. As an example, in manufacturing control, the environment is the physical world equipped with resources such as machines and conveyor belts, and the tasks are the client orders for fabricating particular products. The software in manufacturing control is responsible for controlling the resources and for guiding the orders through the factory floor.

Centralistic software approaches tend to break when the underlying system is large scale in terms of physical distribution and number of entities. Based on the characteristics and requirements described above, a decentralized, multi-agent system approach is suitable for modeling and developing the software for these applications. Both the mobile entities (partially fabricated products, vehicles, client software) and the fixed entities or resources in the environment (machines or conveyor belts, roads and intersections, web services) are obvious candidates to be represented as agents - which we call task agents and resource agents respectively.

Research related to BDI-approaches [1,11] is particularly interesting and broad - we refer to just of few related topics here [12,8,6,2,3]. Building a realistic BDI agent involves many aspects, including - but not limited to:

- **knowledge engineering** information (beliefs) must be gathered from the environment and from the task and resources agents, and must be kept up-to-date according to some policy;
- **deliberation** based on the world model, the agent needs to decide what state of affairs it will intend to bring about;
- **means-ends reasoning** either through on-line planning or using plan libraries, a plan is devised to reach the intention;
- **direct communication** for many aspects, distributed communication protocols are necessary - most notably for coordinating the behavior of the task agents, but also to inform resources of agent intentions, exchange state with other task agents, and so on;
- **advanced concepts** including joint plans, joint intentions, learning may be quite useful.

Experience with BDI-agents in small-scale and toy applications, as discussed in a lot of literature contributions for over ten years, yield agents which were conceptually clean, yet these agents quickly become quite complex. For the application

domains in coordination and control for large-scale systems in highly dynamic environments, the complexity of BDI-based models of agents and the expected computational effort is simply overwhelming.

Partially inspired by the recent trend to exploit the environment as a design abstraction for managing complexity in MAS [14], we propose a particular BDI-based approach that aims to avoid most of the internal agent complexity. Rather than creating and maintaining complex world and agent models themselves, the agents delegate this to the environment. In the approach, we introduce the concept of "delegate MASs" which are issued by the different agents. Delegate MASs consist of light-weight agents which perform particular activities on behalf of a task or resource agent. These light-weight agents can explore the environment and bring relevant information back to their responsible task agent, can evaluate optional paths, and can put the intentions of their task agent as information in the environment. This allows delegate MASs of different agents to coordinate by aligning or adapting the information in the environment according to their own tasks.

This paper is structured as follows. Section 2 describes the basic software architecture of our approach, which consists of task and resource agents and their environment. In Sect. 3, we refine this architecture by proposing how the agents deal with beliefs, desires and intentions through delegate MASs. As the contribution of this paper is on the approach rather than on the application domains, we restrict the examples and illustrations to manufacturing control. A concrete case study in manufacturing control is described in Sect. 4. Section 5 makes an evaluation and concluding remarks on the proposal, and points out directions for future work.

2 Basic Software Architecture for Coordination and Control Applications

We describe the basic components in the software architecture of our approach, i.e. task agents, resource agents, and the environment.

The environment. The environment of the targeted applications is a dynamic directed graph. The nodes in the graph represent the resources in the environment and the edges represent connections between different resources. The environment contains the mobile entities and allows these entities to move from resource to resource. A mobile entity that resides on a node in the graph can communicate with the resource on this node. When a mobile entity is on a particular node, the corresponding resource can perform an action on the task of this mobile entity.

The lower part of Fig. 1 shows an example of a simple factory of six resources, connected through unidirectional "left-to-right" connections.

Resource Agents. A resource agent represents a resource in the environment and contains an information processing part for controlling the resource. The resource agent lives in a virtual world that represents the underlying system,

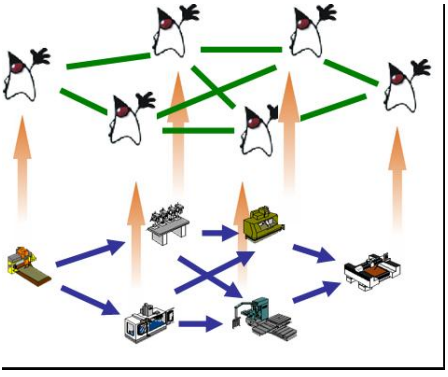

Fig. 1. A simple factory consisting of six resources, which are connected left to right

but which allows bidirectional communication for each connection (see upper part of Fig. 1). The resource itself offers processing capacity and functionality to the resource agent. In manufacturing control systems, a resource agent is an abstraction of the production means such as machines, conveyors, tool holders, material storage or even personnel.

A resource can abstractly be described as a set of capabilities. A capability specifies the operations that the resource is able to perform.

Resource agents need to be able to make schedules based on requests from task agents. Resource agents must also be able to answer "what-if questions": a task agent may ask a resource agent when and according to what quality standards a particular operation could be performed if the task would arrive at a future time. This allows task agents to evaluate the total time to completion and the expected quality of the finished task for a particular plan.

Task agents. A task agent represents and controls a task in the coordination and control application, and resides on a (physical or virtual) mobile entity in the environment. A task agent is responsible for performing its task by guiding its mobile unit through the environment, and communicating with resource agents in order to perform operations on the unit. A task has to be performed correctly and in time. Every task agent is aware of the goal of its task, and has available the schemes or plans that can be followed in order to reach this goal. For manufacturing control, the task agents represent (unfinished) client orders and are associated with the pallets with partially fabricated products. The production schemes describe possible sequences of operations on the product in order to obtain the final product. A task agent may represent customer orders, make-to-stock orders, prototype-making orders, orders to maintain and repair resources, etc.

Task agents are obvious candidates to be modeled as BDI agents. Task agents need to deal with the observations of the environment and its entities (beliefs), consider possible options on how to proceed (desires), choose a particular option (intention) and communicate this with the other task agents. This allows the agents to coordinate their behavior by accommodating their intentions.

Coordination is necessary as actions of one task agent may obviously influence the situation of the other task agents. If the situation in the environment is such that another option becomes substantially more favorable, the agent can reconsider and adopt a new intention.

Agent Interactions. Both resource and task agents control the entities in the environment, and obviously need to interact to achieve the goal of the application. A typical interaction amongst these agents goes as follows.

When a new "task" enters the system, a task agent is created and connected to the appropriate unit. The agent is aware of the initial state of the task and investigates possible next operations that can be performed. The agent searches and selects a combination of a next processing step and a suitable resource that has the capability to execute the step.

When the selected processing step is executed by the resource, the resource reports on the new state of the task. This may or may not be the expected outcome of the operation. Based on this state, the task agent investigates possible next steps based on the task schemes, and selects a combination of a processing step and a suitable resource again. This process is repeated until the task is finished.

3 Delegate Multi-Agent Systems for BDI Through the Environment

In the previous section, we identified the core abstractions and concepts for modeling a coordination and control application as a multi-agent system. In this section, we describe the functionality that is required for the task agents to be able to achieve their goals, and explain how we achieve this functionality through delegate MASs.

3.1 Required Functionality

Make feasibility information available. As routing tasks through the environment is an essential feature in coordination and control applications, the environment must provide a means to inspect *feasible paths.* A feasible path describes a sequences of resources that can be reached by following this path. Feasibility information reflects physical or topological constraints in the environment. If there is a path from one node H to a destination node D via node V, this must be observable as a feasible path.

Task agents need to explore relevant paths. Task agents need to explore the feasible paths that *correspond* to their task schemes. A feasible path corresponds to a task scheme if following this path routes the task agent along the resources that are necessary to reach its final goal. A task agent needs to consider all possible schemes (i.e. sequences of operations) which can bring the current task toward its goal, and match these plans with the feasibility information. The

feasible paths that match a suitable scheme represent the different options that the task agent has to achieve its goal.

Exploring a path means to evaluate the path in a "what-if mode" in order to judge timing and quality if this path would be followed by the task agent.

Intentions. Based on the options that are available to a task agent and their evaluation in a what-if mode, the task agent chooses one path as its intention. Adopting an intention obviously has implications on the resource agents that will be visited as part of the intention. The task agent needs to communicate with those resource agents and inform them of when they will arrive and which operation the resource will need to perform. The resource agents need to book these reservations.

3.2 How: Delegate MASs

A typical approach would be to use direct communication protocols, knowledge engineering and means-ends reasoning to achieve this functionality. Here, we use delegate MASs for obtaining feasibility information, exploration and propagation of intentions toward the resource agents. To some degree, delegate MASs are inspired by food foraging in ant colonies. Food foraging ants execute a simple procedure. In absence of any signs in the environment, ants walk around randomly in search for food. When an ant discovers a food source, it drops a smelling substance - a pheromone - on its way back to the nest while carrying some of the food. This pheromone trail evaporates over time, and disappears if no other ant deposits fresh pheromone. Another ant in search for food will use pheromones in the environment as a source of information to direct its own behavior. Pheromones indicate possible routes to a food source, ants are urged by instinct to follow this trail to the food source. When the ant finds the food source, it will return with food, while depositing pheromone itself. When the ant discovers that the food source is exhausted, it starts a randomised search for food again. As the pheromone trail is no longer maintained, it disappears over time.

These simple behavior patterns result in an emergent behavior of the ant colony that is highly ordered and effective at foraging food while being robust against the uncertainty and the complexity of the environment. An important capability of this type of collective behavior is illustrated: global information - about where to find food in a remote location - is made available locally - in which direction must the ant move to get to this food. The following interesting principles are recognized: (1) make the environment part of the solution to handle a complex environment without being exposed to its complexity - ants are quite simple agents; (2) place relevant information as signs in the environment ensuring that locally available data informs about remote system properties; (3) limit the lifetime of this information (evaporation) and refresh the information as long as it remains valid - this allows the system to cope with changes and disturbances.

We exploit these principles in our approach and define three types of lightweight agents, which each represent a different delegate MAS and which share a common environment for indirect communication. To distinguish them from

task and resource agents, we call the light-weight agents "ant agents" or ants further on. Delegate MASs consist of ant agents that reside in a virtual software environment which reflects the application environment, and in which ant agents can navigate. The responsibilities of the delegate MASs are restricted in that they are managed by the basic agents. Individual ant agents have a particular activity to perform autonomously, yet which information they distributem or how the information they gather is used, is the responsibility of the issuing basic agent.

Feasibility Ants. Feasibility ants form a delegate MAS that is issued by resource agents. Their purpose is to roam the environment and, at each node they pass, drop information on feasible paths that start from this node.

Resource agents which have not seen a feasibility ant passing by for a particular period will create new feasibility ants themselves at a certain frequency. The behavior of a feasibility ant is as follows.

A feasibility ant communicates with the resource agent at its current node, and asks for its capabilities. The ant observes the environment locally and finds out from which other nodes the current node can be reached. As the system environment is a directed graph, this means that the ant makes a list of all nodes that are predecessor of the current node. A clone of the feasibility ant is sent to each node in this list. The ants can be seen as moving upstream. When arrived at the new node, the ant asks the local resource agent for its capabilities, and merges this information with information of previous nodes. Now this ant knows that from this node, a sequence of operation that requires a capability from the current node followed by an operation that can be performed at its previous node is currently feasible. This information is dropped at the local information space of the current node, as a kind of road sign. Then the ant clones itself for every node upstream, and the process is repeated. A feasibility ant dies if there are no nodes upstream. Cycles can be dealt with by accumulating this information - as such the information may not only contain sequences but also iterations of resource capabilities. The information that is stored in local information spaces is time-stepped, and, if not refreshed by another feasibility agent in time, that information disappears. This is necessary to accommodate dynamic changes in the environment both on the topology and the resources in the topology.

This process can and must be fine-tuned for every application in order to avoid flooding. Hop limits, limited cloning budgets and probabilistic choices of upstream nodes are a few mechanisms that can be used, but which mechanisms are useful and effective depends on the concrete application.

Exploration Ants. Using the feasibility information available locally, a task agent is able to find out which paths are physically or virtually feasible for achieving the goal of its task. A task agent generates exploration ants at a certain frequency which explore feasible paths. These exploration ants are scouts that each explore a feasible route through the underlying system and evaluate this route. This evaluation typically concerns completion time and quality criteria on the final state of the task, but can also include a cost e.g. for the usage of fragile, expensive or critical resources. To make the evaluation, an exploration ant follows a path

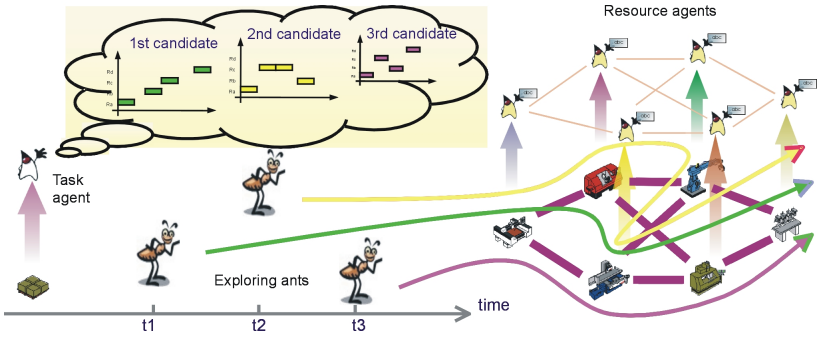

Fig. 2. Exploration ants, issued by a task agent, scout feasible paths by roaming the graph environment

through the environment, and interacts with the resource agents at the different nodes by asking the resource agent *what* the timing, quality or costs would be *if* a task in a particular state would arrive at a particular moment in time. The exploration ant collects this information, and then proceeds to the next node in the path, in which this behavior is repeated. When arrived at the end of the path, the exploration ant returns and reports back to its base, i.e. the task agent that created the exploration ant. Figure 2 illustrates this process for the simple factory shown in Fig. 1. The task agent on the left hand side creates three exploration ants for scouting feasible paths.

The information that a task agent gathers in this way from all its exploration ants is filtered out in order to withhold the paths that are valid options for the task at hand. An option is valid if, besides yielding the required goal, the goal is reached in due time and with an acceptable quality and cost.

The list of candidates get refreshed regularly as exploration ants are sent out regularly. Candidates that are not refreshed are removed over time, assuming that these candidates have become invalid or infeasible because of changes in the environment.

Intention Ants. Exploration as described above requires the resource agents to possess an adequate estimate of their future workload. To serve this purpose, task agents generate intention ants, which propagate the intention of task agents through the environment.

The process goes as follows. When a task agent has constructed a set of valid paths to follow, the task agent selects one candidate path to become its intention. The criteria used for this selection depends on the requirements of the task, and is application-specific. Then, the task agent creates intention ants, at a certain frequency, to inform the resource agents that are involved in this intended path.

The intention ants follow the selected path, and virtually execute the routing and processing of their selected candidate solution. On their virtual journey, the intention ants acquire travel, queuing, and processing times from the resource agents on their path. Any changes, which occurred since the exploration,

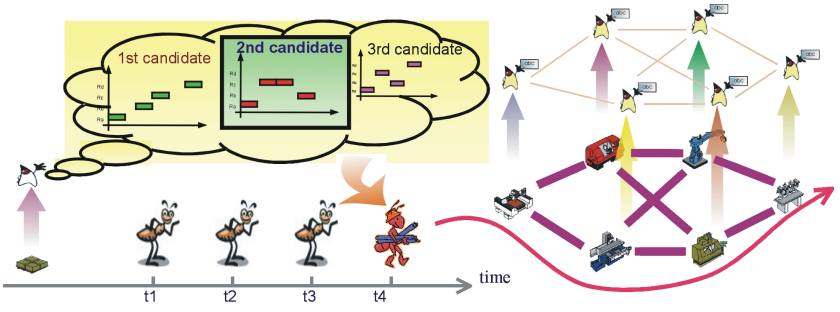

Fig. 3. Intention ants, issued by a task agent, communicate the intention of the task agent through the environment

immediately become visible when these resource agents provide the information. In contrast to the exploration ants, intention ants inform the resource agent that their order agent is likely to visit them at the estimated time and to perform a particular operation. In this way, intention agents make a (evaporating) booking on the resource, and the resource agent adjusts its load forecast (local schedule of the resource) to account for this visit. As a consequence, resource agents are able to predict their workload and performance more accurately to their visiting exploration and intention ants. Similar as exploration ants, intention ants report back to their task agent to inform the agent about the schedule and performance of the bookings.

Figure 3 illustrates this for our simple factory example. Based on the information that was gathered by the three exploration ants, the task ant decides that the path explored by the last exploration ant (which explored candidate path 2) fits the task requirements best. To confirm this and align this decision with the involved resource, an intention ant tries to walk the same path and make bookings on its way.

One important note to make is that, as a consequence of this process, the actual intention of a task agent, as it is distributed to the different resource agents, is only the path that the task agent intends to follow. This intention is then aligned with the schedules and performance of the involved resources. The task agent decides on the path to follow in an intention, the environment and its resources decide on the resulting schedule and performance, which may or may not correspond with the beliefs of the task agent based on the information from the exploration ants. As such, this process reliefs the task agent from massive communication using complex protocols to ensure e.g. a two phase commit for reserving all resources.

The intention information at the resource agent - the booking - evaporates. Task agents must create intention agents to refresh their intention at a frequency that is sufficiently high to maintain their bookings at the resources.

While refreshing, a task agent observes the evolution of the expected performance of its current intentions through the reports on the estimated performance that intention ants bring back. This performance estimate is compared to the

estimates of the candidate solutions that are found and refreshed by the exploration agents. When the estimated performance of the current intention drops significantly below the estimated performance of other candidate solutions, the task agent may revise its intention. When the task of a task agent reaches the point where a decision needs to be executed, the task agent triggers the action in the underlying system in accordance with the intention.

Task and Resource Agents. Let us now list the responsibilities of task and resources agents. These lists should guide a developer of a coordination and control application to define a concrete software architecture for these agents.

A resource agent is responsible for (1) answering what-if questions from exploration ants, (2) making schedules based on requests for bookings by intention ants - the schedules must obviously respect resource constraints as well as follow a predefined policy for re-scheduling, e.g. when high-priority tasks make a booking, this may reject earlier reservations, (3) keep an up-to-date view on the resource that the resource agent is managing (e.g. observe operation quality and status), and (4) send out feasibility ants if no feasibility ants have contacted this resource agent for a while. These guidelines should suffice to produce a concrete architecture according to the requirements of a concrete application. The resource agent could execute each of these responsibilities in sequence, or one may opt to define concurrent execution of some of these responsibilities.

The responsibilities of the task agents are the following. First, a task agent must have knowledge about its task, about the initial state of the task, and about task schemes. Task schemes describe one or several plans to achieve the goal of the task, and given any intermediate state of the task, the task scheme should provide one or more sub-plans to fulfill the task. Second, a task agent must manage its beliefs. Beliefs on feasibility are readily available from the information space in the environment. Beliefs about explored paths are gathered by exploration ants. Third, this information needs to be filtered out, yielding valid paths - the agents desires. Fourth, based on the beliefs about the task and the options, a task agent then chooses or revises its intention. Fifth, at appropriate times and frequency, exploration ants as well as intention ants are sent out. Finally, a task agent is responsible for interacting with resource agents in order to perform operations on its task.

Again, these responsibilities are either executed as a sequence, or one may choose for concurrent execution of some of these responsibilities.

4 A Case Study in Manufacturing Control

We have applied the approach (in simulation) on several artificial toy examples, and on one realistic case. In a research and development project, in collaboration with an industrial partner, we investigated the approach for coordination and control of a factory that produces parts of weaving machines (see Fig. 4 for a screen shot of the factory in a simulation tool). The factory that is modeled in

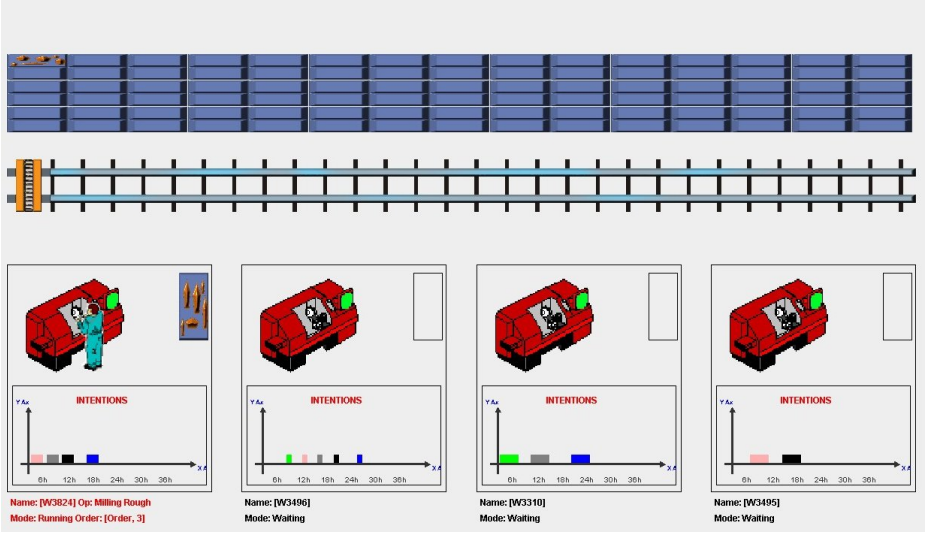

Fig. 4. A weaving machine factory - CNC machines and a warehouse, connected to a shared tram system

this case study has a particular topology, which consists of several workstations (machines, in the middle of the picture) and one warehouse (top of the picture), and a shared transportation unit (on the rails). The warehouse contains storage slots, which store containers with product parts. The warehouse is automated in that it is capable of managing these slots itself. Besides the machine workstations, an "input station" is responsible for entering new orders into the system, and one "output station" delivers finished product outside of the system. The transportation unit is a tram which can visit all workstations and the warehouse. This tram carries containers between the different workstations and between the workstations and the warehouse.

A schematic representation of this topology illustrates the directed graph of this environment, see Fig. 5. Conceptually, all workstations and the warehouse are bi-directionally connected to the tram resource.

A crucial requirement for the coordination and control software for this case is the optimization of usage of the transport system (the tram). During periods of heavy demand for transportation (rush hour), the tram is a bottleneck and causes workstations and operators to idle, which is expensive.

For this case, we conducted two sets of experiments. The experiments focus on the flexibility of the approach in dealing with unpredictable timings of machine operations. Experiments with changing topologies are promising results as well, but are not reported here.

In a first set of experiments, we applied a straightforward detailed design of our approach, in which the production schemes reflect current practice. The production schemes are simple deterministic lists of sequential processing steps,

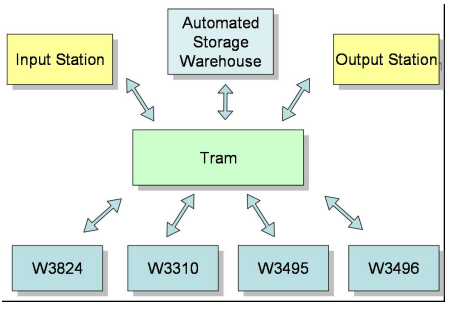

Fig. 5. The graph environment of the manufacturing control case - a star topology

which includes a visit to the warehouse between each two successive processing steps on workstations. The order is stored in the warehouse until the next machine where the order needs to be processed, becomes available. It may seem illogical to visit the warehouse in between each two operations. The reason why this is current practice is the inability of the current system to cope with flexibility for dealing with unpredictable timings of operations. When an operation is finished on one machine, and the workstation that is going to be used for the next step is available, a visit to the warehouse is unnecessary, and the order could be delivered immediately. If, however, because of variation in processing time, the workstation for the next operation is not available, even though it was predicted to be available at this time, the transport cannot be performed. The current industrial system (in reality) does not have up-to-date forecasts, and the effects of direct workstation-to-workstation transports on the system performance are therefore unclear and represent an unacceptable risk for the company.

The results obtained from this first set of experiments are quite satisfactory compared to the current characteristics of the system as it is operational today (numbers not available by non-disclosure). Results on the performance of the transport system and the bottleneck machines (W3824 and W3310) in the factory are reported in Table 1. The rightmost column displays aggregated results for all workstations.

In a second set of experiments, we aimed to test the effect of adding flexibility to the system by allowing more flexibility in the task schemes. In these experiments, visiting the warehouse is no longer mandatory but optional between each two processing steps. Besides this change in production schemes, the task agents and resource agents were identical to the first set of experiments. Our approach

Table 1. Results from the basic controller

	Tram	W3824	W3310	Total
avg. wait time	76	41	145	287
total wait time	4315	370	435	8928
max wait time	891	84	365	1587
utilization rate	8%	91%	89%	

Table 2. Results from the enhanced controller

	Tram	W3824	W3310	Total
avg. wait time	109	31	129	274
total wait time	4374	280	387	8488
max wait time	838	57	345	1587
utilization rate	6%	93%	90%	

can benefit from this extra flexibility in the task schemes as it relies on forecasts (intentions) that may be revised, e.g. in the case where processing a particular operation takes longer than predicted. Table 2 shows the results for these experiments. The main effect of this extra flexibility on the tram is a reduction of its load by 25% and the number of transports by 30%. This results in reduced waiting times for all workstation and especially for the bottleneck workstations. The utilization rate on the bottleneck workstations increases, implying that the overall throughput of the manufacturing system increases proportionally. This improvement is significant, even for small percentages, since it directly affects the financial return rate of the investments.

Further improvements are being investigated in this case. The excess capacity on the tram, outside rush hours, can be used to prepare the work during periods of high demand. The availability of an up-to-date prediction is essential for this enhancement since it both informs the system whether there is an opportunity to rearrange the storage and tells the system which rearrangement is likely to lower the workload during upcoming periods of high demand.

Related work on MAS for Coordination and Control. The research presented in this paper builds upon previous research on the PROSA reference architecture [13]. The PROSA architecture recognizes four types of agents, being Product agents, Resource agents, Order agents and Staff agents. Order agents represent specific client orders (comparable to our task agents), resource agents are similar to our resource agents, product agents represent a service which knows the product recipes (i.e. the possible sequences of operations that need to be performed on an order for producing the intended product), staff agents provide a service for the other agents by offering predefined pragmatic solutions for particular problems. The basic architecture of the approach presented here provides a cleaner conceptual model of the core agent types (product and staff agents are not considered as basic agent types, as they do not exhibit pro-active behavior but merely provide services to agents), and most of all allowed us to clearly relate the ant-based mechanisms to BDI-based architectures. This relation will foster this research in two ways. First, it is aimed to provide a more convincing case of our research to the community that is centered around BDI-based approaches. Second, it opens the pathway for studying our approach in terms of well-know and well-studied concepts (both core concepts as well as derivative concepts) in the field of BDI-based approaches.

The main contribution of a delegate MAS design is its ability to generate short-term forecasts; these forecasts account for recent updates on the state

of the underlying system and the control system entities themselves. Moreover, feasibility ant agents propagate constraints in the underlying system to wherever they may be relevant.

In comparison, known scalable MAS coordination and control developments are myopic [4,9]. These approaches decide about task allocations when the preceding task is about to finish or is already finished. These systems use interaction protocols e.g. variations on the well-known contract net to implement decision-making mechanisms. A "utility function" in such interaction protocols needs to capture all future implications of the decisions. As a consequence, these designs have proven to be very successful in dynamic but homogeneous environments. For instance, [4] is capable of controlling a homogeneous collection of CNC machine tools (Computer Numerically Controlled machines)in a factory with a flexible transport system but fails to handle a mix of hard automation (low cost and very fast) with flexible but expensive CNC equipment and fails to handle transport systems that have limited flexibility (routing from resource A to resource B is not always feasible). In a delegate MAS, feasibility ant agents account for such heterogeneity and the forecasting functionality permits the coordination and control system to account early enough for the often erratic constraints in such production systems.

Early attempts to account for the complete sequence of production steps that are required to execute a task, suffer from combinatorial explosions. In such developments, resource agents, when they are unable to finish a task, recursively subcontract the remainder of this task to other resource agents before entering their bid [7]. Recently, advanced machine learning techniques have been applied to select candidate subcontractors and eliminate the combinatorial explosion [5]. However powerful, such solutions require software maintenance when the model of the underlying system changes (e.g. to account for storage and transportation) by an expert in such machine learning technology. Likewise, researchers have developed MAS control system that incorporate planning systems [10]. Again, changes to the model of the underlying system are likely require challenging maintenance efforts. In contrast, models in delegate MASs have a one-to-one correspondence to the corresponding entity in the real world; their is no modeling effort required to fit the coordination technology; delegate MAS designs stay close to reactive agent designs in that the world almost remains its own best model.

5 Evaluation and Conclusion

Developing a real-world coordination and control application will never be easy - the problem domains are too complex and the environment too dynamic for this. In this paper, we do not claim that other MAS approaches to these applications (e.g. classical BDI-based or other approaches), technically cannot be used to tackle these applications. Instead, we want (1) to emphasize the enormous complexity that is involved in the agent software for such realistic applications, and (2) to make a strong case that creative architectural alternatives are worth investigating. In the approach we propose, we stick to the basic philosophy of

belief-desires-intention agents, but exploit the environment and delegate MASs, inspired by ant behavior, to realize beliefs, desires and intentions. This innovative approach yields quite interesting results for the targeted application domains, both in terms of the reduction of the agent software complexity and in overall performance in a highly dynamic environment.

The use of a delegate MAS allows the coordination and control system to handle changes and disturbances as "business-as-usual". Indeed, feasibility ants discover the (dis)appearance of resources during refresh. Likewise, lost connections and (re)connections are discovered during such refresh by feasibility ant agents. Furthermore, disturbances such as a rush order pushing reservations by other tasks backwards, or a temporary equipment malfunction causing similar shifts for reservations, are detected during refresh of both intentions and candidate solutions by respectively intention and exploration ants. Stale information, which refresh activities fail to update or remove, disappears through the evaporation mechanism within the time needed for a few refresh cycles.

One important difference between an approach using delegate MAS and traditional BDI approach for coordination and control systems is that a delegate MAS design extends Brooks concept of having the world as its own best model, while traditional BDI approaches rely on maintaining world models. The basic idea of having the "world as its own best model" only discusses the present state of the world. In contrast, the delegate MAS approach in this paper extends this idea toward the future state, using exploration and intention ants, while keeping modeling efforts acceptable. Indeed, resource agents only need to be knowledgeable and intelligent about their own small section of the world.

There is one caveat to the approach: the task agents must behave in a socially acceptable manner. This means that agents do not change their intentions too easily and too frequently. Otherwise, minor disturbances such as a short breakdown of a resource may create an avalanche of tasks that shift to alternative resources. The perceived improvement must be higher than a threshold value before the current intention is replaced by the more-promising alternative.

To avoid too many agents revising their intentions at the same time, possibly yielding thrashing behavior, task agents change their intentions probabilistically. As a result, only a small fraction of task agents may change their intentions, and the other agents are able to observe the consequences before changing their intentions as well. Adopting this mechanism ensures that task agents will gradually shift toward alternative routes when a disturbance occurs until a new equilibrium is reached.

To serve the purpose of this paper, the approach was described at a high-level of abstraction as a generic software architecture, and contained mainly hints and guidelines for designers of real coordination and control applications, and a report on experiments in one manufacturing control case. A detailed design of the approach specific for manufacturing control is available. Other detailed designs are likely to follow when adopting this approach in other concrete application domains.

References

1. M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard, Cambridge, MA, USA, 1987.
2. L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A bdi agent system combining middleware and reasoning. In M. K. R. Unland, M. Calisti, editor, *Software Agent-Based Applications, Platforms and Development Kits*, pages 143–168. Birkhuser-Verlag, Basel-Boston-Berlin, 9 2005. Book chapter.
3. L. Braubach, A. Pokahr, W. Lamersdorf, and D. Moldt. Goal representation for bdi agent systems. In R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, editors, *Second International Workshop on Programming Multiagent Systems: Languages and Tools*, pages 9–20, 7 2004.
4. S. Bussmann, N. Jennings, and M. Wooldridge. *Multiagent systems for manufacturing control: A design methodology*, volume XIV of *Springer Series on Agent Technology*. Springer-Verlag, 2004.
5. B. Cs.Csji, L. Monostori, and B. Kdr. Reinforcement learning in a distributed market-based production control system, 2004.
6. F. Dignum, D. Morley, L. Sonenberg, and L. Cavedon. Towards socially sophisticated bdi agents. In *ICMAS*, pages 111–118. IEEE Computer Society, 2000.
7. A. Marcus, T. Vancza, and L. Monostori. A market approach to holonic manufacturing, 1996.
8. S. Parsons, O. Pettersson, A. Saffiotti, and M. Wooldridge. Intention reconsideration in theory and practice. In W. Horn, editor, *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI-2000)*. John Wiley & Sons, 2000.
9. H. Parunak, A. D. Baker, and S. J. Clark. The aaria agent architecture: From manufacturing requirements to agent-based system design. *Integrated Computer-Aided Engineering*, 8(1):45–58, 2001.
10. M. Pěchouček, A. Říha, J. Vokřínek, V. Mařík, and V. Pražma. Explantech: applying multi-agent systems in production planning. *International Journal of Production Research*, 40(15):3681–3692, 2002.
11. A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.
12. J. Thangarajah, L. Padgham, and J. Harland. Representation and reasoning for goals in BDI agents. In M. J. Oudshoorn, editor, *Twenty-Fifth Australasian Computer Science Conference (ACSC2002)*, Melbourne, Australia, 2002. ACS.
13. H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters. Reference architecture for holonic manufacturing systems: Prosa. *Computers in Industry*, 37(3):255–276, 1998.
14. D. Weyns, H. V. D. Parunak, F. Michel, T. Holvoet, and J. Ferber. Environments for multiagent systems state-of-the-art and research challenges. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *E4MAS*, volume 3374 of *Lecture Notes in Computer Science*, pages 1–47. Springer, 2004.