Re-organization in Warehouse Management Systems

Huib Aldewereld, Frank Dignum, and Marcel Hiel
Utrecht University - Institute of Information and Computing Sciences
Utrecht, The Netherlands, {huib,dignum,hiel}@cs.uu.nl

Abstract

Warehouse Management Systems (WMS) are traditionally highly optimized to a specific situation and do not provide the flexibility required in contemporary business environments. Agents have been advocated for their flexible and adaptive nature, but require organizational structure to ensure that the system performs as required. Due to changes in the environment a different organization may be more productive making re-organization essential. In this paper, we present an architecture and methodology for easing the redesign of a WMS. The method applied is based on heuristics for re-organization given the environment, the main objectives of the organization and the current situation.

1 Introduction

Warehouse Management Systems (WMS) are traditionally centralized, monolithic software systems that are highly optimized for a specific situation. These systems thus guarantee very efficient operation given some fixed constraints. However, these systems usually have trouble to achieve flexibility (such as handling priority orders) and robustness (such as machine failures). In modern WMS a good balance between efficiency, flexibility and robustness is of utmost importance.

Agents were introduced to tackle the problem of flexibility (e.g., see [2; 7]). However, only introducing agents is not sufficient. An argument against the usage of agents is that the control is hard to guarantee and that therefore the requirements of robustness and efficiency cannot be guaranteed. The main problem is that efficiency, flexibility and robustness are aspects that pertain to the system as a whole. Moreover, we cannot optimize all three at the same time, but have to balance the three aspects (e.g., more robustness usually means less efficiency). Thus, when an agent approach is used it does not mean that each agent has to manage the balance between the three aspects. However, the distributed nature of this type of solution makes it very hard to guarantee an overall balance between the three aspects. In order to prevent anarchy and regulate the agents within such a system, we propose to use agent organizations [6]. An agent organization is used to specify exactly those aspects of the system that need to be guaranteed by the agents together. Individual agents can have their own goals and ways of interacting with other agents. However, because they are designed to fit the agent organization their autonomy is limited by the overall objectives of the organization. E.g., suppose that we have 4 types of tasks occurring equally frequent and four machines that can perform all tasks. For efficiency sake we probably want each machine to specialize in one type of task (which might avoid reset time etc.). However, for robustness sake we want all machines to perform all tasks. We could choose a balance of having two machines perform two tasks such that if one fails the other takes over and work on that task does not completely halt. How the machines subsequently divide their tasks they might decide themselves (using agent based solutions for this situation).

Unfortunately, although organizations provide structure and stability for regulating a multi-agent system (MAS), the environment might change in ways such that the organization no longer guarantees the right performance. For example, after the introduction of new hardware or product changes, the WMS may provide a suboptimal solution. In order to fully use the flexibility provided by agents and maintain the robustness of an organization, being able to re-organize is essential.

However, in order to perform a successful re-organization, it should be done at the right moment and changing the organization in a way to perform better. In this paper we will show how agent organizations can be used to implement a WMS and how successful re-organizations can be carried out in this environment.

The alternative to re-organization is to make the agents themselves adaptive. The distinctive difference between adaptive agents and re-organization is that in the organization the knowledge concerning the global (organization-wide) objectives is explicit. The advantage of having this knowledge explicit is that it is easier to adjust when modification is necessary.

This paper is structured as follows: In Section 2 we first introduce a motivating example. Subsequently we show how agent organizations are used to control the warehouse in Section 3 and which criteria are used to measure its success. In Section 4 we show how re-organizations can be defined using change patterns and illustrate how they are used to keep the organization successful under changing circumstances. We conclude in Section 5 with some observations on our use case and future work.

2 Motivating Example Scenario

A warehouse stores and collects products for customer orders. These products are typically packed and/or placed in boxes or containers, generally referred to as Transport and Storage Units (TSU). Figure 1 illustrates our example hardware configuration for a WMS. This figure contains three types of components, namely miniloads, conveyorbelts and workstations. The miniloads are storage units where TSUs are kept, the conveyorbelts are responsible for transporting TSUs between miniloads and workstations, and the workstations represent places where operators pick products from TSUs for fulfilling the orders. TSUs are not kept at the workstations, but requested from the miniloads when an order arrives and returned immediately after picking the required amount of products to fulfill the order. In our configuration, thus, we have three miniloads, one conveyorbelt and two workstations. The boxes on the conveyorbelt represent moving TSUs.[1] The squares with arrows represent buffers where the direction of the arrow indicates the direction of movement.

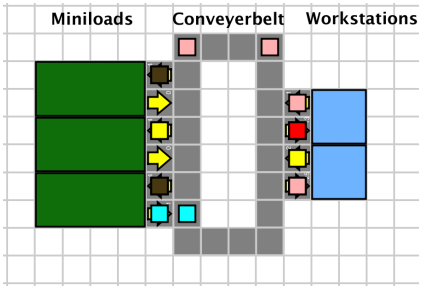

Figure 1: Example warehouse configuration

Assume that this WMS performs adequately. However, due to a successful marketing campaign one particular product becomes very popular. At a certain point, all components work at optimal efficiency, however, because of that popular product the number of orders becomes larger than can be handled per day and therefore the number of outstanding orders becomes larger and larger. Moreover, because of this delay customers become unsatisfied due to the long waiting time.

In order to deal with this situation management decides that the warehouse should be extended, however due to limited space only two workstations can be added but no miniloads. As the average miniload is calculated to be able to support one or two workstations (in processing time), expectations are that number of orders handled will improve. After placing the workstations the performance (throughput) increases but not as much as expected and the number of outstanding orders still increases.

In the next section we will first describe the basic set-up of the agent organization that controls the warehouse logistics and how this supports the balance between efficiency, robustness and flexibility of the WMS. In the rest of the paper we will show how the changing environment renders the organization inefficient. In order to determine whether the performance can be increased, re-organization is to be considered, which is explained in the sections after that.

[1]The color of the TSUs indicates the product family, which is unimportant in the discussion in this paper.

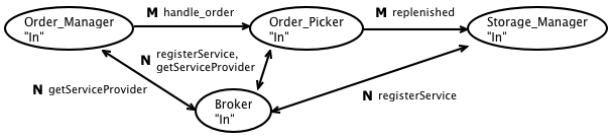

Figure 2: Organization for Planning

3 Agent-organizations in Warehouse Management Systems

In [6] we showed how agent organizations are used to balance flexibility, robustness and scalability. Here we only highlight the most important aspects of this agent organization model. In our architecture, we use agents to control every component (thus miniload, conveyorbelt and workstation). Every agent is responsible for, and optimizes the efficiency of, its component. The agent organization structures the interactions between agents and provides answers to design questions, such as: who talks to who?, what is the role of the agent within the organization?, and what are the objectives that this agent seeks to achieve?

In the remainder of this section, we describe our layered approach for modeling agents after which we present performance indicators that can be used to evaluate a warehouse management system.

3.1 Layers

Warehouse management systems are typically thought of in three layers of operation, namely the plant (execution), scheduling and planning. This distinction in layers is made to create a separation of concerns which makes it easier to create a modular design and thereby support decoupling of the different aspects of the WMS.

This separation is reflected in our model (for details see [6]) based on the MASQ meta-model [8]. Each of the layers of the warehouse management system, that is, the plant, scheduling and planning layers, correspond to a separate interaction space which defines the particular protocols that the agents use on that layer. Because each space incorporates its own implemented business rules and interaction protocols this improves modularity and maintainability.

Planning Space: "Planning is the process of generating (possibly partial) representations of future behavior prior to the use of such plans to constrain or control that behavior" [1]. In our domain this means that orders are assigned to be handled by certain components (without an explicit timing). The interaction necessary for the assignment of orders is modeled as an agent organization using the OperA framework [3; 4].

The social structure of the agent-organization, which specifies the roles and the relations between these roles, is shown in Figure 2. The arrows between the roles indicate dependency relations. From the figure it is clear that there is a role taking care of incoming orders and that the agents fulfilling the "Order_Picker" role (in our case the workstations) will

Presence	Stock Management	Broker Communication
Order_Picker	createReplenishCFP	registerDeliveryService
	createDeliveryProposal	requestReplenishProviders
Storage_Manager	createReplenishProposal	registerReplenishService
Order_Manager	createDeliveryCFP	requestDeliveryProviders

Table 1: Planning Capabilities per Goal and Presence

Keeping plant active	Scheduling Transfer of TSUs	For Planning
getNextAction	handleIncomingMessage	getLeadTime
processAction	sendTSUPlacementRequest	scheduleTSU
isOutgoingTSUScheduled	sendTSUPlacementReply	
handleIncomingTSU		

Table 2: Scheduling Capabilities per Goal

provide the order. In order for the Order_Picker to get the necessary products for the order they ask the agents fulfilling the "Storage_Manager" role (in our case the miniloads) to provide the products from storage. Finally the Broker maintains knowledge about which roles exist in the system and which service(s) they can provide.

Through this organization structure we already convey that the incoming orders determine the logistics (we did not incorporate a product reception role). We also do not connect the customers directly with the storage manager. This means that the order picker decides whether an order should be processed and never the storage manager. The underlying reason to model it this way is that the workstations are known to form a bottleneck. We therefore want those to be in charge of the workload. If they are optimally used the whole system performs optimally.

Besides these basic points we can also get information from the types of dependencies between the roles. There are different types of dependencies possible, each resulting in a different type of interaction; bidding [**M**arket], delegation [**H**ierarchy], and request [**N**etwork]. The protocol that is used between the agents depends on the kind of interaction type specified in the organization model. In our case, market relations are implemented using the Contract Net Protocol, and network relations are implemented as request/inform messaging. The use of market relations and the Contract Net Protocol as implementation for service requests means that, given an appropriate bidding mechanism, balance of the workload of the components is achieved automatically. Thus we provide for robustness and flexibility within the organization structure. Note that the efficiency of the resulting logistic process depends on the decision mechanisms of the agents.

Another aspect that can not directly be seen from Figure 2 is that all miniloads can communicate with all workstations, but they do not communicate amongst each other. This makes the configuration very robust and flexible, but potentially less efficient. It also contributes to an quadratic growing amount of communication.

The required capabilities of the agents on planning are summarized in table 1.

Scheduling Space: In our domain scheduling encompasses three goals, namely (1) supplying the hardware with actions to perform, (2) transferring TSUs from one component to another, and (3) provide information and the means for planning such that plans can be created and executed. For each of these objectives a number of capabilities are required. Table 2 lists the capabilities that we distinguish.

As can be seen from the above, in our (simple) scenario scheduling has little independence from planning and thus we do not provide the social structure for this part as it is completely in line and enforcing the objectives of the planning. Mainly, scheduling ensures that the hardware, at all times, knows what to do next. After the plant gives a notification that the current action is finished, the scheduling component of the appropriate agent supplies it with the next action (getNextAction). Furthermore, scheduling maintains a list of TSUs in the component, as well as those that are planned for movement (handleIncomingTSU and isOutgoingTSUScheduled). Maintaining this list after executing an action is done by the processAction capability.

As planning is dependent on information from scheduling, for example in calculating the time it would cost to process a TSU (leadTime), scheduling components provide capabilities to get this information, such as the getLeadTime capability. Furthermore, the scheduling components present the capabilities for scheduling a TSU to be processed by the plant.

3.2 Performance

Organizations have to try to achieve three global objectives, namely efficiency, robustness and flexibility. An organizational structure maintains a certain balance of these objectives with respect to its environment. However, if the environment changes the balance might shift into an unfavorable direction. The questions are then: what criteria should be measured in an organization in order to decide to re-organize and when to change? In general, efficiency can be linked to throughput (i.e., the amount of orders processed per day). Robustness is more difficult to measure. It should be measured by resilience to failure of machines. One can compare the throughput of the warehouse when one machine fails with the throughput of the warehouse where it is configured optimally without that machine. Finally flexibility depends on environment parameters. In this case one compares the performance of the warehouse in case all events were known on forehand with the case where some events happen unexpectedly.

It is clear that the measurements get more and more difficult when going from efficiency to robustness to flexibility. E.g., a warehouse might be very robust with respect to failing miniloads but very sensitive to failing workstations. In practice one looks at one or more bottlenecks in the warehouse and checks the robustness with respect to their failure. The same holds for flexibility. A warehouse might be very good at handling priority orders as long as the orders required 'standard' products. However, it may perform poorly when the priority orders require combinations of products that are rare. Again, in practice one only measures those cases that are expected.

Because in our scenario overall efficiency of the warehouse is the most important objective we use simple throughput as the performance criterium. Figure 3 illustrates a graph of the throughput for our running example. The figure contains four

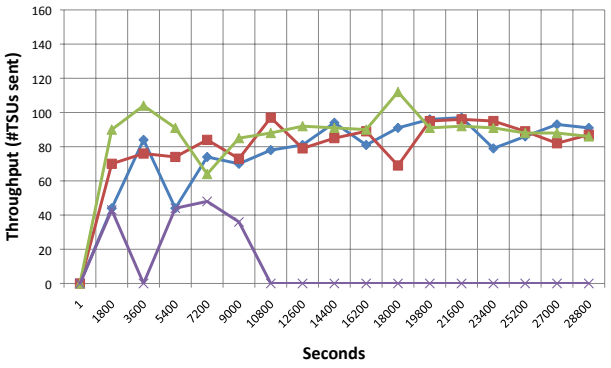

Figure 3: TSU Throughput for four workstations

lines, one for each workstation. One of these four lines climbs in the beginning, but then quickly drops to zero. After placing the two additional workstation the miniloads are not quick enough to handle all the requests on time, therefore the fourth workstation is without work most of the time. It is clear that the organization is not performing optimally in the new configuration. In the next section we will investigate what type of changes we can make to the organization in order to improve the performance.

4 Developing Change Patterns for Re-organization

It is impossible to know all the changes that may affect a warehouse in advance. Therefore an exhaustive overview of changes and corresponding action for adaptation is hard to provide. However, we can provide some general heuristics for each of the main warehouse objectives: efficiency, robustness and flexibility. We will first give these general heuristics and subsequently describe in more detail how the re-organization of our scenario is realized.

Efficiency When throughput should be increased first one finds the bottleneck components. Basically the agents indicate their performance as a percentage of their maximal performance. The agents with the highest percentage will be the bottleneck. The next step is to add more components of that type (e.g., in our scenario we will add more workstations). Now two steps should be taken with respect to communication. First it should be checked whether the components fulfilling the same role were communicating amongst each other already. If not, it should be checked whether they should start such communication now. This communication is added when the products leaving one component might be interesting for another component and thus a kind of ”sideway” logistic step is added.
Finally, it should be checked whether a new role should be added in between other roles for communication efficiency. In our case all miniloads communicate with all workstations. If there are too many of each type each component is all the time communicating and processing information about possible work, most of which would be useless. In that case a new intermediary can be created that makes the decision on division of work based on communication with both sides (we reduce the amount of communication channels from $n * m$ to $n + m$).

Robustness When robustness has to be increased we have to create alternative potential workflows. There are two ways to create alternative workflows. The first is by adding more components of a certain type. This is handled in the same way as indicated above for increasing efficiency. The second way to create alternative workflows is to connect more agents fulfilling different roles. In the extreme case every agent and associated component is connected to every other agent/component. Of course many connections are useless, because the associated components cannot exchange their products (in a meaningful way), but it guarantees that every possible workflow is indeed covered by the organization. Our example warehouse looks very robust, because all miniloads are connected with all workstations. However, they all use the same conveyorbelt. To make things more robust separate connections could be used for every pair of miniload and workstation. The decision to create such alternative paths is made on the basis of the expected rate of failure of each component and the costs to create and maintain an alternative path.

Flexibility To increase flexibility the general approach is to create components with more capabilities. In this way each component is better capable to handle more situations. In the warehouse domain a relatively cheap way to increase the capability of a component is to create an input and output buffer. This creates the possibility for the component to have more flexibility on deciding what task to perform next (of course this is a limited form of flexibility, but an often occurring need). Once components get more capabilities it is also possible to create more alternative workflows, thus increasing the robustness of the system.
When components have more capabilities, of course, they also need to have the decision mechanism on how and when to use the capabilities. Besides that they need the right information to make the right decisions. In general this means that new communication channels have to be created to get the information to all the components that need it. In the most extreme case all components can perform all tasks and exchange all possible information with all other components. This is clearly not very cost efficient, but extremely flexible and robust also.

In general the organization should balance efficiency, robustness and flexibility and all the required levels should be attained at a minimum cost (in terms of resources, communication, and complexity). A re-organization makes sense if the added expected gain in performance of the system is higher than the additional costs. Here “performance” is meant in a wider sense than just throughput, but rather behaviour of the system with respect to all criteria over a period of time. Crudely stated: if the gain in profit by the re-organization is higher than the costs (calculated over a certain period of time), re-organization makes sense.

Given these general heuristics for re-organization, we now turn to the concrete example. We describe how to capture the experiences of redesign such that they can assist developers in future projects. In particular, we focus on change patterns. Change patterns, and design patterns in general, provide for a structured documentation thereby making the transference

	Concept	Operator	Description
Organization	Role	addRole(r_i)	add role r_i
		removeRole($name$)	remove role r_i
	Dependency	addDependency(d_i, r_{from}, r_{to})	add dependency d_i between role r_{from} and role r_{to}
		removeDependency(d_i)	remove dependency d_i
	Objective	addObjective(o_i, d_i)	add objective o_i to dependency d_i
		removeObjective(o_i, d_i)	remove objective o_i from dependency d_i
	Player	addPlayer(p_j, r_i)	add player p_j to perform role r_i
		removePlayer(p_j)	remove player p_j
Agent	Agent	addAgent(a_i)	add agent a_i
		removeAgent(a_i	remove agent a_i
	Presence	addPresence(p_i, a_i, l_j)	add presence p_i to agent a_i for layer l_j
		removePresence(p_i, a_i)	remove presence p_i from agent a_i
	Capability	addCapability(c_k, p_i, a_i)	add capability c_k to presence p_i of agent a_i
		removeCapability(c_k)	remove capability c_k from presence p_i

Table 3: Change Operators

Name	Communication Between Pickers
Change Type	Re-organization of planning layer
Trigger	A popular product & nr orders received > nr orders handled (per day)

Change Template	//Organizational model
	addDependency(d6,order_picker,order_picker);
	addDependencyObjective(replenish,d6);
	
	//Planning Implementation (for all players p)
	//to PickerPlannerPresence
	addCapability(registerReplenishService,getPlanBody(p),p);
	addCapability(createReplenishProposal,getPlanBody(p),p);
	
	//Scheduling Implementation
	addCapability(scheduleTSU,getScheduleBody(s),s);

Table 4: Change Pattern Communication Between Pickers

of knowledge between developers easier. Making this knowledge explicit thereby reduces the time required for (re-)design and (re-)implementation.

4.1 How to re-organize?

Following a model-based approach, we use models to get an overview of what can be done. More specific, we use a model-management approach that was used to describe the evolution of services [5]. In this model-management approach, with a *model*, a complex structure is meant that represents a design artifact. The usage of models implies manipulation and transformation of one model to another model. The key idea behind model-management is to develop a set of algebraic *operators* that generalizes the transformation operations. In our framework, these operators consist of adding and removing concepts (and relations) in a model. Operators are commonly stored in a script. A script is a sequence of operations that (automatically) transforms one model into another.

In our approach, we have two types of models, namely the organizational model, and a model which represents the implementation of the agents. We list the operators for each of these models in Table 3. These change operators represent all the possibilities for changing the organization and the agents. For example, if the organization grows substantially by incorporating many more agents, a manager role can be introduced through the addRole operator. As the organizational model is a specification, the operators used for this model can be automated. In other words, a new organizational model can be automatically derived by applying these operators to the old model.

Next to the organization model, the agent(s) implementation should also be changed to reflect the new organization. However, this might imply giving agents new goals, new protocols, etc. Because we do not desire to restrict the agent implementation we only require the agents to incorporate the concepts in Table 3. Here we provide only guidelines and requirements for how to structure the implementation such that these operators can be used to precisely determine where to update the implementation.

Following our model of an agent, we can either add or remove presences for agents in the different spaces. Furthermore, capabilities can be added to or removed from the different presences at both planning and scheduling. Note that these capabilities provide only the skeleton functionalities and that they still have to be programmed.

In order to enhance reusability of existing code and thereby reduce the time to implement the new WMS, the addition of new capabilities should be done mainly based on the existing capabilities. That is, the operators that are specified should preferably exist in other presences, such that the developer already has an idea of what the capability should do and what decisions are important for this capability.

4.2 Example Scenario: Communication Between Pickers

We return to our running example scenario. With the additional workstations the number of orders handled becomes larger but the new configuration did not solve the complete problem. The solution found by the warehouse manufacturer is that TSUs containing the popular product need not be returned to the miniload but can be send directly between workstations if needed. This cuts the traveling time back to the miniload as well the processing time of the miniload for getting the TSU out of the rack.

This change requires a re-organization where workstations do not only communicate with the miniloads, but also communicate with other workstations in order to request and propose replenishments. The pattern for this situation is shown in Table 4. The change template shows the operators needed for creating the new WMS. The first two operators apply to the organizational model and can be automatically applied. The other operators affect the implementation and cannot be automatically applied. These operators add capabilities to presences of agents on both the scheduling and planning space. On the planning space the capability should be added that pickers register themselves as replenish service providers at the broker. This causes them to be contacted if components (pickers) require replenishment. Furthermore, the createReplenishProposal should be added such that pickers can create proposals to answer to the call-for-proposal (cfp) of the Contract Net Protocol. In the scheduling space the TSU should be scheduled to go to a picker instead of back to the miniload. Therefore the capability should be added to schedule the TSU.

In our change pattern, the capabilities can be reused from other components. For example, the createReplenishProposal is already a capability of the miniloads, however, the reason-

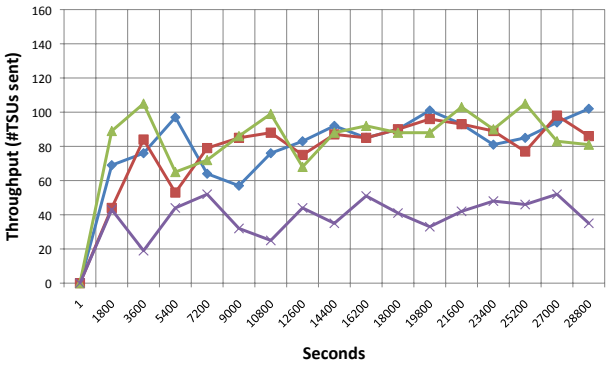

Figure 4: TSU Throughput with picker communication

ing to decide whether a TSU is available differs whether it is in the miniload or in a picker.

The result of applying this change pattern is shown in Figure 4. As can be seen when comparing Figures 3 and 4, the fourth workstation increased its throughput by roughly 40 percent, due to the communication between pickers. This shows that the problem has been solved using the heuristic mentioned in the beginning of this section. First the management located the bottleneck of the old warehouse (the workstations) and added more components of that type. This increased performance slightly. Second, the flow of product TSUs was improved by enabling horizontal communication between the pickers. Note also that the robustness of the warehouse went up as well (but only with respect to situations where workstations fail).

5 Conclusion

Although in Warehouse Management Systems efficiency is of prime concern it is clear that this has to be balanced with robustness and flexibility of the system. We have shown how agent organizations can be used to balance efficiency, robustness and flexibility of a system. However, due to changing circumstances in the environment an organization might not keep the right balance over time. Thus re-organization of agent-organizations is essential in an evolving environment. In order for the re-organization to have a positive effect we first have to check which criteria could be used to measure the performance of the organization with respect to each of the aspects. For efficiency this is reasonable unique and can be captured by throughput given a set of resources. Robustness and flexibility have to do with responses to resource and communication failures and "unexpected" events. So, the criteria should actually be detailed with respect to the kind of failures and "unexpected" events. We have given some general criteria and subsequently described some general heuristics for re-organization scripts based on these criteria. In the example scenario it is clear that when the throughput of some of the newly added workstations is almost zero the organization is performing badly.

The heuristics that were used to change the Warehouse organization had an effect on organization efficiency; if a role forms a bottleneck, additional players of that role can be added to distribute the load. This change, however, shifted the bottleneck to the miniloads, which did not perform optimally due to lack of coordination between them. Several things could have been done to remedy the situation. The first would be to add additional miniloads, but this is costly (and not possible given the space limitations in our scenario). Secondly, a manager role could have been added to the pickers to streamline the distribution of work, and allow for the reuse of replenishment TSUs among the pickers. This would, however, add another layer in the organization (complicating communication by adding an extra step in the chain) while the decisions to be taken by that manager are relatively simple (and can be taken by the miniloads themselves). Therefore the option we chose (which is less drastic and more flexible) was to allow for horizontal communication between the picker agents. We saw that this change could be captured by change patters that guide the re-organization process. Of course, the next steps will be to further investigate heuristics that guarantee at least a certain amount of improvement in efficiency, robustness, and/or flexibility while keeping a right balance given the changing environment. And we will test our heuristics systematically in many different types of scenarios.

Acknowledgment

This work has been carried out as part of the FALCON project under the responsibility of the Embedded Systems Institute with Vanderlande Industries as the industrial partner. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Embedded Systems Institute (BSIK03021) program.

References

[1] Austin Tate in the MIT Encyclopedia of Cognitive Science. Planning. `http://cognet.mit.edu/library/erefs/mitecs/tate.html`.

[2] R.S Chen, K.Y. Lu, and C. C. Chang. Intelligent warehousing management systems using multi-agent. *Int. J. Comput. Appl. Technol.*, 16(4):194–201, 2003.

[3] Virginia Dignum. *A Model for Organizational Interaction: based on Agents, founded in Logic*. PhD thesis, Universiteit Utrecht, 2004.

[4] Virginia Dignum and Huib Aldewereld. OperettA: Organization-oriented development environment. In *LADS2010@MALLOW*, 2010.

[5] Marcel Hiel. *An Adaptive Service-Oriented Architecture - Automatically solving Interoperability Problems*. PhD thesis, Tilburg University, September 2010.

[6] Marcel Hiel, Huib Aldewereld, and Frank Dignum. Modeling warehouse logistics using agent organizations. In *CARE' 10*, LNCS. Springer-Verlag, to appear.

[7] Teruaki Ito and S. M. Mousavi Jahan Abadi. Agent-based Material Handling and Inventory Planning in Warehouse. *Journal of Intelligent Manufacturing*, 13:201–210, 2002.

[8] Tiberiu Stratulat, Jacques Ferber, and John Tranier. MASQ: Towards an Integral Approach to Interaction. In *AAMAS '09*, pages 813–820, 2009.