An Agent Based Solution for Dispatching Items in a Distributed Environment

Christian Vecchiola, Alberto Grosso, Andrea Passadore, Davide Anghinolfi, Antonio Boccalatte, Massimo Paolucci, *DIST – Department of Communications Computer and System Sciences*, *University of Genova*

Abstract—This paper describes Herald, an agent based toolkit for dispatching and processing items in a distributed environment. Herald is suitable for scenarios where the process could be modeled as a tree: starting from the root node the collection of items is distributed along the nodes where they can be processed, forwarded to other nodes, and duplicated if necessary. Herald assigns a specific software agent to each node of the tree which participates into the dispatching process according to the knowledge base of the multi-agent system. Herald works as a general infrastructure for simulating, testing and executing dispatching algorithms that can be easily integrated into the system by changing the decision making process of the agents composing the architecture. A prototypal implementation, based on the AgentService programming framework, is then presented as a proof of its applicability in industrial scenarios.

***Index Terms*—Agent Oriented Software Engineering, Agent-based coordination, software agents for logistics.**

I. Introduction

The problem of distributing items on a somehow hierarchical structure is common to many different application contexts, for example logistics [1], routing [2], and scheduling [3]. In all the previously cited contexts a collection of elements (packets, orders, or simply items) has to be dispatched according to a certain strategy. Moreover, the distribution takes place on a structure exposing a sort of hierarchical organization: routing algorithms generally operate on graphs while in the case of scheduling orders are allocated to production area and then assigned to specific machines. The nature of items and the use of dispatching strategies, normally being aware of the structure, are what specializes the problem in each context. For these reasons, providing a general solution to the dispatching problem is limiting: it would not be possible to consider the specific issues of each scenario for effectively optimizing the dispatching process. A better idea could be providing a general framework for creating a dispatching system that is easily customizable for each specific context. In order to provide such a flexible structure the use of the agent-oriented technology could be an interesting approach. Multi-agent systems are flexible and dynamic software systems [4]. Software agents natively adopt high-level interaction patterns [5] and this is a relevant aspect for application interoperability and component coordination. By using software agent we can either provide the general structure of the dispatching or leave room for the specializations.

In this paper we will present Herald a toolkit for dispatching items in a distributed environment: the system is composed by a collection of agents, which manages the dispatching process, and a collection of additional components used to integrate the multi-agent system with the existing software. The main idea behind Herald is not to provide a ready to used product but a toolkit which is customizable to different scenarios with little effort. Herald provides a collection of agents implementing the dispatching infrastructure and describes a methodology to customize the toolkit for the different application contexts. The strength of Herald resides in exploiting the flexibility of agents for implementing the hierarchical dispatching infrastructure according to the structure required by the real application scenario. The multi-agent system, developed with the AgentService programming framework [6], is the core component of Herald along with a base class library defining the common data structures defined by these agents. Developers have to provide an application-based version of items, the elements to be dispatched, and custom dispatching strategies if required by the application scenario. Developers can also integrate external components for driving the dispatching activities since the systems allows callbacks at each stage of the process.

This paper is organized as follows: in Section II we will introduce the key elements of agent-based dispatching; in Section III we will present a selected collection of the most representative work in the field; Section IV describes in detail the architecture of Herald while in Section V we will present a practical application of the toolkit by describing the prototype developed in collaboration with Siemens A&D for a real scheduling scenario. Conclusions and final remarks follow.

II. Agent-based Dispatching

The relatively large number of solutions based on multi-agent systems, demonstrates the usefulness of an agent based infrastructure for the development of distributed and hierarchical applications managing logistic, routing and dispatching issues.

Since the operative management of resources is a critical aspect of the business activity of an enterprise, there exist several industrial solutions, involving multi-agent systems too. On the other hand, the research activity in this sector is lively and ready to exploit every new approach. The aim is to provide systems with an intrinsic intelligence, then denoting a particular adaptiveness to the environment changes and unexpected events. Multi-agent systems seem to satisfy these requirements. The typical socio-technical functions that we find in an enterprise can be easily modeled by means of a software agent playing a management role. For this reason an agent denotes a natural predisposition to collaborate with peers in order to achieve a common goal, through cooperation protocols. It could have skills and behaviours that can implement different solving strategies [7]. As we will see in the following, multi-agent systems represent a basis on which different and original solutions can be implemented.

A. Industrial solutions

In the panorama of industrial solutions regarding the management of resources, two strengthened products are on the market: Magenta and LS/ATN (Living System Adaptive Transportation Network).

Magenta [8] is a MAS framework for the development of ad hoc applications focused on the design, planning, scheduling, and management of enterprise resources. Typical examples of Magenta applications are the supply chain management, enterprise resource planning, transportation logistics, crew scheduling and knowledge management. Magenta integrates three crucial technologies: multi-agent systems, semantic web and J2EE. One of the main features of Magenta is the strong support for modeling the relevant entities of the enterprise through an ontology representation. Ontologies can be modified and updated online, during the execution of the application.

LS/ATN [9] is a comprehensive solution for optimization and dispatching of full and part truck loads including tracking and real-time event handling. It is produced by Whitestein Technologies Inc. and it is oriented to the European logistics companies. LS/ATN is based on the LS/TS (Living System Technology Suite) agent development framework. The implementation of the agent-based solution takes into account the geographically dispersed nature of transportation. For this reason agents represent the geographical regions and they exchange objects representing the cargo loads. The transportation operations are allocated to the different dispatching regions which are managed by an agent region manager; a broker agent named agent distributor deals with incoming transportation requests. The optimization process involves two steps: a first local optimization within the region (it involves the broker agent and the agent region manager) and the global optimization through the collaboration of different agent region manager.

B. Research solutions

Different proposals involve the agent technology, in order to implement original solutions: from the imitation of social insects to genetic algorithms. A more pragmatic approach is the identification of relevant entities involved in the dispatching process (machines, raw materials, tools, management units, etc.), establishing a direct correspondence among these entities and software agents. The significant observation is that every approach can be implemented by a multi-agent platform.

There is an immediate similarity between agents and ants: the power of an ant colony is not the single individual but the cooperation of every insect. In the solution presented in [10] the agents-ants collaborate in order to provide a heuristic scheduling solution in a parallel machine environment. The proposed solution is applied in a complex context regarding a manufacturing company.

Another solution [11] to the dispatching problem involves genetic algorithms and an agent community particularly reactive and adaptive to the environment changes. The agents do not have a predefined set of rules or instruction to reach their goal. In particular, an agent is defined by the knowledge about the environment condition (i.e. processing resources and the status of other agents), the agent status (position, process plan status, completed and remaining operations), and a tuning vector (to weight the decision rules). There are two types of agent: the part agent able to select the machine which will process the part and the workstation agent, which select the part to process on the basis of different parameters as: processing time, deadlines, and setup times. These agents adapt their actions to the plant status using a multi-criteria decision-making algorithm that encompasses multiple weighted dispatching rules, in particular using fuzzy set concepts to implement a trade-of among different decision rules. The performance of each agent is evaluated by a performance indicator. Cyclically, every agent is replaced by another one with different tuning parameters. The “natural selection” detects the best agent; this approach leads up to two considerations: in case of stationary conditions of the plant, the dynamic optimization acts as an online strategy that improves the whole MAS performance. In case of unexpected perturbations (e.g. a machine failure), the genetic adaptation allows searching of more effective agents for the new operating context.

A more conventional solution [12] follows the usual methodology to create agents representing the main entities involved in the dispatching of resources. A set of highly specialized agents is provided in order to adapt the system to the perturbations and disturbances. A system editing agent (SEA) is in charge of the composition of the whole system, as the agent instantiation and the centralized database filling. It is an interface between the human user and the multi-agent platform. The manufacturing management agent (MMA) is associated to a production area, receives orders (composed by a set of operations), instantiates a job order agent (JOA) monitoring its activity. The JOA is able to evaluate the feasibility of an order and communicates with the production area. A logistic management agent (LMA) holds the logistic data of a production area, while the logistic agent (LA) helps the JOA and the agents associated to the machines (machine agent, MA), tools (tool agent, TA), and jigs (jig agent, JA) to evaluate the feasibility of an order or operation.

All the proposed solutions show how a hierarchical structure of cooperative agents is helpful in the resolution of dispatching problems. The variety of agents which belong to the tree hierarchy suggests that a toolkit for the implementation of a custom project is an interesting approach. All the analyzed solutions propose specific algorithms for driving agents' behaviors, i.e. swarm intelligence and genetic algorithms; on the other hand, the aim of the toolkit proposed in this paper is to provide an high level infrastructure for developing distributed systems where different kinds of algorithms can be applied.

III. The Herald Toolkit

In the Middle Age the term herald was used to identify those people which acted like factotums at the king's court. Among the other tasks, heralds were in charge of dispatching messages, managing negotiations, and accomplishing missions. As the ancient herald, the Herald toolkit is aimed to resolve logistic and decision problems.

A. *Introduction*

Herald is a toolkit that implements agent based software systems and aims to act as a generic distribution network. Herald proposes an approach which is independent from the specific nature either of the items dispatched or of the problem domain. Hence, it proposes a general infrastructure that needs to be further customized in order to be effectively applied to real-life scenarios. The core features of Herald are its agent based architecture and the protocol adopted to dispatch and assign items to the processing units. External software components customizing the dispatching infrastructure are integrated into the system and their activities mainly cover the collection of the items to dispatch, the selection of the dispatching policies, and the management of results. The external components are also responsible of activating the multi-agent system and controlling the dispatching process if needed. In this section we will discuss the architecture of the system and the details of the protocol proposed by Herald.

B. *Architecture*

The core of Herald is based on a protocol execution engine constituted by a collection of software agents which create the dispatching process by interoperating with external software modules driving the protocol. Figure 1 gives an overview of the organization of the components constituting an implementation of Herald.

The architecture proposed by Herald is based on the assumption that the dispatching process takes place within a hierarchical structure that can be easily represented by a tree: the process originates from one root node and at each node items are distributed among the child nodes. Such a structure is re-created within the multi-agent system by three different types of agents representing the different roles which nodes have in the dispatching process. Additional agents take care of the interaction with external world. The multi-agent system is implemented with the AgentService programming framework which provides advanced features for the creation of multi-agent systems and their integration with non-agent based software.

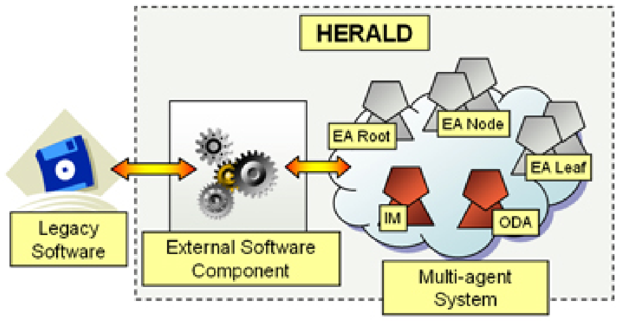

Fig. 1 – View of the System

1) *Multi-agent System*

The dispatching process is mostly performed by the collection of agents which constitute the multi-agent system designed in Herald. Herald proposes two different kinds of software agents: logical agents and physical agents. Physical agents represent physical entities existing in the hierarchy of the real structure, while logical agents are mostly related with the elaboration of the input and the output data of the whole process.

There are three different types of physical agents – also called entity agents – they are:

- EA Root (Entity Agent Root): commonly, there is only one agent of this type since it abstracts the root of the tree. This agent receives the whole collection of items and distributes them among its child nodes. It can also perform additional operations specific to the role of the root in the real scenario.
- EA Node (Entity Agent Node): these agents represent all the intermediate levels of the hierarchy and dispatch items among to their children nodes.
- EA Leaf (Entity Agent Leaf): they represent the leaf of the tree structure. These agents process the items and send up to the hierarchy the data collected during their activity on the items.

Tree structure having more than one intermediate level are modeled by introducing the required number of *EA Node* agents representing each node in these levels.

During the dispatching process items can be manipulated at each level: items can be aggregated, duplicated, or split. The nature of these operations strictly depends on the application context and Herald gives the opportunity to external software components to control the dispatching process.

The multi-agent system is completed by two logical agents that are the Item Manager Agent and the Output Data Agent:

- Item Manager (IM) Agent: the IM Agent collects the items that have to be processed and distributed along the hierarchy. As happens, for the EA Root it can pre-elaborate the collection of items but its main role is to represent the access point to the dispatching process by the external software;
- Output Data Agent (ODA): the ODA is activated at the end of the dispatching process; it receives the mapping between the items and the EA Leaf agents composing the system. Additional information, specific to the application context, can decorate this mapping.

All the agents defined in the system mostly manipulate items (split, duplicate, or change properties): these are abstract entities that can be further customized with additional properties.

2) *AgentService*

Even though the practical implementation of Herald does not strictly require a specific agent programming framework, its canonical model adopts the AgentService framework for which we have developed all the required class libraries and software agents. AgentService [6] is an agent programming framework built on top of the Common Language Infrastructure [13], whose .NET Framework is the most popular implementation. The framework provides the programmers with the following features:

- definition of autonomous, independent, and persistent agents;
- concurrent execution of agents and their multi-behavior activity;
- persistent shared data structures within a single agent;
- transactional agent communication based on message exchange;
- access to the FIPA service components (AMS, DF, MTS).

One of the key features of AgentService is its agent model that is not particularly tied to specific agent architectures, but is flexible enough to implement different ones. Within AgentService, an agent is constituted by a set of knowledge objects and a set of behaviour objects. Knowledge objects define the agents' knowledge base while behaviour objects define the activities that an agent can perform and the services it offers to the others. Knowledge objects are shared among the different behaviours that are scheduled in a concurrent manner. The definition of a new type of agent leads to the definition of a template which specifies the behaviour and knowledge objects that characterize it. It also leads to the definition of these behaviour and knowledge types.

AgentService supports mobile agents [15] and provides programming tools for managing ontologies and interaction protocols [16]. AgentService also allows an easy integration of multi-agent systems with external applications by using agent avatars: agent avatars are application stubs that are seen as software agents inside multi-agent system. By using avatars external software modules can interact with agents and access platform services as if they were like real agents.

3) *External Software Components*

External software components connect the multi-agent system with the external world and the problem domain. These components have the following responsibilities:

- they provide the collection of items to the MAS;
- they activate the system;
- they control the execution of the protocol;
- they elaborate results.

These components manage all the customization aspects of the system: they create the items, they add the required additional properties, and they provide the custom dispatching policies when needed. Within a default installation of Herald external software components constitute the software environment into which the multi-agent system is created and executed. By using an agent avatar they interoperate with the MAS, drive the dispatching protocol, and get the results.

C. *Dispatching Protocol*

Herald uses a flexible dispatching structure in which the decisions concerning the distribution of items can be taken in cooperation with external software entities providing either the complete set of allocations or just simple indications.

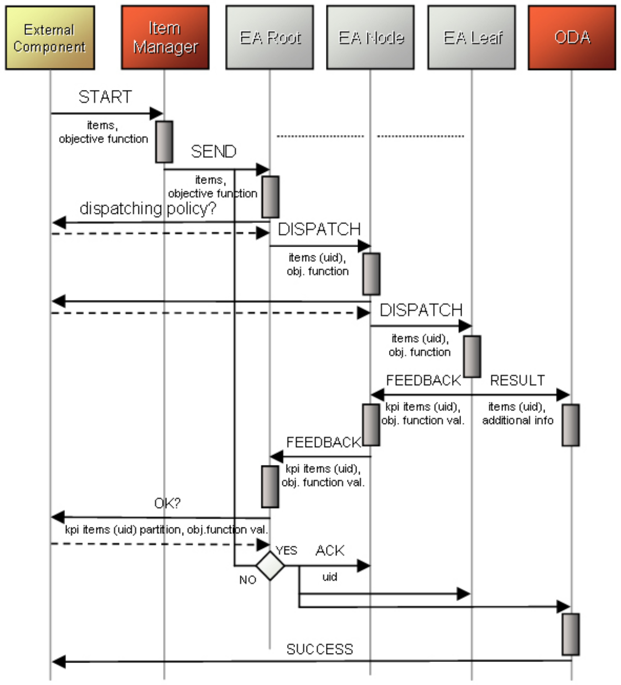
External Component
Item Manager
EA Root
EA Node
EA Leaf
ODA
START
items, objective function
SEND
items, objective function
dispatching policy?
DISPATCH
items (uid), obj. function
DISPATCH
items (uid), obj. function
FEEDBACK
RESULT
kpi items (uid), obj. function val.
items (uid), additional info
FEEDBACK
kpi items (uid), obj. function val.
OK?
kpi items (uid) partition, obj.function val.
YES
NO
ACK
uid
SUCCESS

Fig. 2 – Protocol execution step

This gives a high degree of flexibility since it allows developers to modify the course of actions with the highest level of detail and it makes Herald suitable for many different scenarios. The only requirement of Herald is the hierarchical structure of the distribution process which strongly characterizes the architecture of Herald.

As previously said in section III.B.3 the process is activated by an external software component which has to create the collection of agents described in section III.B.1. Once the multi-agent system is activated by the external software components the following steps apply:

1. the external software component creates/retrieves the collection of items that have to be dispatched and send them to Item Manager agent along with additional ordering criteria and an objective function that has to be minimized or maximized;
2. the Item Manager pre-elaborates the items, sorts them according to the suggested criteria and communicates to the EA Root the collections of sorted items;
3. the EA Root creates a unique dispatching identifier that will be tagged to the collection of items while they flow through the child nodes. If necessary, the EA Root performs additional operations on the items (i.e. aggregation of items) before sending them to child nodes. The EA Root asks to the external software module for a dispatching strategy of the collections of items: if the external software component gives suggestions (i.e. by providing a mapping from items to child nodes or by explicitly giving a partition algorithm to apply) these suggestions are applied otherwise the default partitioning is applied. Then the items are sent to the child nodes according to the partition previously performed;
4. each EA Node which does not receive an empty list of items executes the dispatching. As happened for the EA Root the EA Node can, if required, manipulate items by aggregating or splitting them according to the requirements of the possible final targets (the subset of EA Leaf agents that can be reached from this node) of the items;
5. the EA Node asks to the external software component indications about the partition of items by sending the request along with some context information (i.e.: the identifier of the node and the collections of items). If there are any suggestions they are applied otherwise the default distribution takes place. The items are then sent to child nodes;
6. steps 4 and 5 are repeated for each level of the tree until leaf nodes are reached;
7. each EA Leaf that receives a collection of items elaborates them and eventually computes some key performance indexes as suggested by the external software component and its contribute for the objective function;
8. each EA Leaf sends a feedback – composed by the key performance indexes and the value of the objective function – to the parent EA Node and the list of assigned items along with additional information to the ODA;
9. each EA Node aggregates the feedbacks received from the child nodes and compute the value of the objective function for the node, then sends back to the parent node the data;
10. the EA Root aggregates the feedback received by the child nodes, computes the final value of the objective function and, according to these data, decides if the current partition of the items is acceptable, by eventually asking to the external component. In case of successful partition the EA Root broadcasts an acknowledge message to all the entity agents and the ODA by specifying the dispatching identifier. If the partition is not acceptable the entire process starts again from step 3 by using different criteria;
11. the entity agents – and the ODA – receiving an acknowledge message keep track of the partition related to the acknowledged dispatching identifier and delete all the other partitions;
12. the external component is notified by the ODA the successful termination of the dispatching process.

Figure 2 gives a graphical representation of the steps described above. We can observe that the protocol simplifies the introduction of on-line decisions. On-line decisions are taken while the system is running and they can modify its course of action. This is accomplished by letting the entity agents interact with the external software components which should maintain an updated view of the state of the problem domain. Finally, keeping separate the general infrastructure of the protocol and all those aspects customizing the algorithm for a specific problem we are able to obtain a very flexible structure. Such structure can be easily applied to different scenarios and problem domains. In each customization what really changes is the external software component driving the protocol and what this module provides to the multi-agent system which remains almost the same.

Another important aspect of this system is the ability of executing multiple dispatching processes in parallel without increasing the number of agents. Each Entity Agent instantiates a behaviour for executing a new incoming dispatching strategy: item partitions belonging to different strategies are identified by different unique dispatching identifiers. Thanks to the natural and effective multi-threading management system of the AgentService framework the execution of multiple dispatching processes comes with no additional cost. The ability of executing dispatching processes in parallel allows implementations of Herald to try more different solutions for the same problem at the same time and then select the best one.

D. Application Scenarios and Customization

There are many different domains in which there is the need of using a dispatching policy for some kind of items. If we have a network the routing of packets is a common application of dispatching policies. Other examples involve logistics and scheduling: in the case of logistics there is the need to select the best route for a given item while in the case of scheduling we have to process a collection of orders in a given time constraint. The execution of orders eventually results in a set of tasks that have to be assigned to a set of machines according to a given algorithm. These are only the most evident domains in which the problem of dispatching items has to be managed. Not all the possible instances of these domains are eligible as case studies for Herald. In particular those exposing a structure that is inherently a graph and that cannot be reduced as a tree cannot be considered. Fortunately there are many cases in which the original hierarchy is a tree or can be reduced to a tree: these are the instances eligible for Herald and now we will see what is required to customize Herald for a practical application.

The first thing that needs to be customized is the item: in order to fully represent the entities of the problem domain, items need to be enriched with additional properties. The Herald Toolkit provides a library defining all the data structures required to apply the dispatching process: by sub-classing the Item type developers can enrich the item class with all the required properties. In order to fully exploit the personalization applied to items all the data structures which operate on item have to be specialized: in particular, algorithms for distributing items and those evaluating the key performance indexes have to take into account the value of additional properties. Herald defines interfaces and delegates for these objects and developers just have to adhere to these type contracts while implementing the specialization. The last component that needs to be implemented is the external software module which drives the protocol and connects the multi-agent system with the problem domain. The implementation of this component is a common activity when designing multi-agent systems with AgentService: developers are normally required to create a batch which sets up the multi-agent system and interact with it if necessary. The implementation of a protocol driver for Herald does not take any additional burden.

In the next section we will see a practical example of the customization described here by describing a case study where Herald as been effectively applied.

IV. Case Study

A. The Problem Context

In order to test its feasibility we implemented a prototype of Herald in the field of production scheduling as a result of collaboration with Siemens Automation & Drive (A&D). Siemens A&D is a leading firm in the field of MES (Manufacturing Execution System) production and scheduling. In order to satisfy the customers' needs Siemens A&D offers the SimaticIT Production Suite which is a suite of cooperating applications which takes care of production process by starting from the Enterprise Resource Planning (ERP) and by reaching the machine level. The Production Suite controls many different tasks: low level plant monitoring, resources allocation, logistics, and scheduling. In this context we developed a prototype based on Herald for scheduling a collection of production orders into a production plant. The Agent Based Detailed Production Scheduler, which is the name of the prototype, relies on the AgentService programming framework for the design and the implementation of software agents constituting the architecture defined by Herald.

B. The Multi-agent System

The structure of plant, which is defined by the S-95 standard [14], naturally resembles a hierarchical tree: the standard defines a production site as a collection of areas which are subsequently partitioned into production cells constituted by units. Production units host production machines which execute real production operations. Hence, mapping such a hierarchy into a tree structure – having the root node in the site and developing till the production units – has been a natural and seamless task. In order to complete the case study in the scope of this paper we will consider a simplified example of this hierarchy that is the sub-tree representing a production area. The hierarchy of the system is mapped onto collection of three different entity agents: EA Area, EA Cell, EA Unit. Moreover, a collection of specific logical agents, namely the WOM (Work Order Manager) and the OSA (Output Schedule Agent), are provided to retrieve data and present results of the scheduling process.

The EA Area agent is a specialization of the EA Root agent and basically dispatches the production orders to the EA Cell agents, waits for the KPI indexes, and eventually sends the acknowledge to terminate the process. The dispatching strategy is selected by the external component. EA Cell agents specialize the EA Node agent. EA Cell agents dispatch the entry received by the EA Area to the EA Units according to some dispatching strategy decided by the external component. They wait for the KPI from the units and assemble them before sending them to EA Area. EA Unit agents are EA Leaf agents: they process the entries, execute the scheduling algorithm, register the KPI, and send them back to EA Cell agent.

The Work Order Manager agent is the specialization of the IM agent. It receives the collection of orders to be scheduled and sets up the multi-agent system constituting the production area to which these orders have been dispatched. Finally, the Output Schedule Agent specializes the ODA and has been introduced to the system to organize and present the schedule data collected from the units in a more convenient format: the OSA uses these data and organizes them into a Gantt diagram.

C. The Protocol

Due to the requirements provided by Siemens A&D the entire dispatching process must be controlled by the Production Modeler (PM) which is the component of the Production Suite actively controlling the physical plant. For this reason the dispatching protocol is said to be PM driven: the Production Modeler is the component which starts the scheduling process and it is also the one which is constantly queried by physical agents during the process. Finally, the results collected by the OSA agent are returned back to the PM which evaluates the performance indexes computed by the multi-agent system and chooses the best schedule. For all these reasons, the PM represents the external component which is involved in the dispatching process as described in Herald. Thanks to the flexible structure of Herald and advanced features of AgentService the customization has been simple and quick.

We decided to centralize the interaction with the PM by using a specific software module implemented as an agent avatar and that we called PM Gateway: the PM Gateway represents the PM in the multi-agent systems and the other agents interact with the PM by exchanging messages with this agent. This design decision represents a slight variation of the architecture proposed by Herald but it is mostly an implementation issue which helped us while testing the system off-line. The use of the PM Gateway allowed us to deploy the multi-agent system by only changing the implementation of the agent avatar and without modifying the code of the other software agents.

V. Conclusions

In this paper we presented an agent-based toolkit for implementing flexible dispatching infrastructures. Flexibility is given by the adoption of agent technology as a core component of Herald and by the high degree of customization offered to end users who can drive the process at each stage. Software agents are adopted to replicate the hierarchical structure of the real life system where the dispatching process takes place. Users just have to develop the software interconnection layer between the multi-agent system and the software legacy system requiring the support of Herald. The features provided by the AgentService programming framework make the interaction with multi-agent system an easy task and allow developers to quickly customize the model provided by Herald to their own problem context.

As noticed in the introduction, even though the model proposed by Herald is based on a tree hierarchy for dispatching the items it remains general enough to be applicable to a wide range of scenarios. As a future improvements we consider the adoption of a more flexible holonic architecture. We developed a case study in the field of production scheduling in collaboration with Siemens A&D and the results have been interesting. The case study showed also that the use of parallel dispatching strategies is valuable since this features makes easier comparing the application of different strategies side by side during their execution.

The Herald toolkit is actually developed in its core components and it has been tested on a real case study, we are now working on creating libraries of dispatching strategies and on providing a graphical user interface for using Herald as a standalone tool. In particular, in order to definitely improve Herald usability, a “drag and drop” interface for graphically model the hierarchical structure of the system and a tool for monitoring the dispatching process at runtime have to be implemented.

References

[1] M. Walliser, M. Calisti, T. Hempfling, S. Brantschen, F. Klügl, A. Bazzan, and S. Ossowski, “Agent-Based Approaches to Transport Logistics, Applications of Agent Technology in Traffic and Transportation, Birkhäuser Basel, pp. 1-15, March 30, 2006.

[2] I. Kassabalidis, A.K. Das, M.A. El-Sharkawi, R.J. Marks II, P. Arabshahi, and A. Gray, “Intelligent routing and bandwidth allocation in wireless networks”, Proc. NASA Earth Science Technology Conf. College Park, MD, August 28-30, 2001.

[3] D. Ouelhadj, C. Hanach, and B. Bouzouia, “Multi-agent system for dynamic scheduling and control in manufacturing cells”, Robotics and Automation, 1998, Proceedings, 1998 IEEE International Conference on V. 3, pp. 2128–2133, 1998.

[4] H. Nwana, “Software agents: An Overview”, Knowledge and Engineering Review, November, Vol. 11, No 3, 1996.

[5] K. Sycara, and D. Zeng, “Coordination of Multiple Intelligent Software Agents”, International Journal of Cooperative Information Systems Vol. 5, 1996.

[6] C. Vecchiola, A. Grosso, A.Gozzi, and A. Boccalatte, “AgentService”, Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering (SEKE04), Banff, Alberta Canada, KSI Publisher, 2004.

[7] M. Wooldridge, “Intelligent Agents”, Multiagent Systems: A modern Approach to Distributed Artificial Intelligence, Weiss, MIT Press, 1999.

[8] Magenta Technology, “Software Platform v2.1”, Magenta Technology Whitepaper, [Online document] 2005, Available at HTTP: www.magenta-technology.com/

[9] Whitestein Technologies, “LS/TS – Living Systems ® Technology Suite”, [Online document], Available at HTTP: http://www.whitestein.com/resources/products/whitestein_lsts_flyer.pdf

[10] C.A. Silva, J.M. Sousa, T.A. Runkler, and J.M. Sà da Costa, “A Multi-Agent Dispatching Heuristic for Manufacturing Systems Using Ant Colonies”, Proceedings of the European Network of Excellence on Intelligent Technologies for Smart Adaptive Systems (EUNITE 2002), 2002.

[11] B. Maione, and D. Naso, “Evolutionary adaptation of dispatching agents in heterarchical manufacturing systems”, International Journal of Production Research, Vol. 39, N. 7, pp. 1481-1503(23), 2001.

[12] S. Heinrich, H. Durr, T. Hanel, and J. Lassig, “An Agent-based Manufacturing Management System for Production and Logistics within Cross-Company Regional and National Production Networks”, International Journal of Advanced Robotic Systems, Vol. 2, N. 1, pp. 7-14, 2005.

[13] Standard ISO/IEC 23271:2003: Common Language Infrastructure, ISO, 2003.

[14] ISA, S95—Enterprise-Control System Integration, Part 1: Models and Terminology, Instrumentation, Systems and Automation Soc., 2000. ISA, S95—Enterprise-Control System Integration, Part 2: Object Model Attributes, Instrumentation, Systems and Automation Soc., 2001 [Online]. Available: http://www.isa.org.

[15] A. Boccalatte, A. Grosso, C. Vecchiola, “Implementing a Mobile Agent Infrastructure on the .NET Framework”, 4th International Conference in Central Europe on .NET Technologies, Plzen, 2006.

[16] A. Passadore, C. Vecchiola, A. Grosso, and A. Boccalatte, “Designing agent interactions with Pericles”, ONTOSE 2007, Second International Workshop on Ontology, Conceptualization and Epistemology for Software and Systems Engineering, Milan, Italy, Giugno 2007.