Resource, Service and Product: Real-Time Monitoring Solution for Service Oriented Holonic Manufacturing Systems

Octavian Morariu[2], Cristina Morariu[1], and Theodor Borangiu[2]

[1] Cloud Computing Research Department, Cloud Troopers Intl.
Cluj Napoca, Romania
cristina@cloudtroopers.ro
[2] University Politehnica of Bucharest, Dept. of Automation and Applied Informatics
Bucharest, Romania, 060042
{octavian.morariu,theodor.borangiu}@cimr.pub.ro

Abstract. Service orientation of holonic manufacturing systems represents a major milestone in increasing efficiency, flexibility and standardization for manufacturing enterprises. SOA governance assures the capability for dynamic composition of services at runtime without human intervention, allowing the system to automatically align itself to the business drivers. In this context there is a need for accurate and real time monitoring of the shop floor activities during the manufacturing process. This paper presents a shop floor monitoring solution based on distributed multi-agent system architecture capable of real time data collection and presentation for production tracking. The solution provides a monitoring portal where system administrators can track key performance indicators in real time. The paper discusses the strategies for handing the monitoring data in real time and also long term, focusing on the consolidation of information in persistent data structures.

Keywords: Shop-floor monitoring, production tracking, simulation, monitoring portal, real time data, alerts, multi agent systems.

1 Introduction

Holonic manufacturing systems evolution in the last decade was driven by the technological advances in the underlying areas and by the business environment changes. This class of manufacturing systems has gained capabilities that allow real time decision making based on unpredictable events that provides a high degree of flexibility and adaptability to change as described by Borangiu [1][2][3]. In this context there are three inter-related vectors that define the effectiveness of the manufacturing system: robustness, composability and observability. In any complex manufacturing system there is a risk that a failure of a module has the potential of stopping the entire system. In other words, a partial system failure can lead to a complete failure if it cannot be detected, isolated and handled in a timely manner. In Service Oriented

T. Borangiu et al. (eds.), *Service Orientation in Holonic and Multi-Agent Manufacturing and Robotics*, Studies in Computational Intelligence 544,
DOI: 10.1007/978-3-319-04735-5_4,

manufacturing systems, the impact of partial system failure is litigated by promoting a loosely coupled architecture, assuring isolation both between individual modules and the underlying runtime platform. In practice, there are two approaches that are used together for isolation of modules: data isolation and execution isolation. Data isolation refers to the internal state and information stored in a module. Execution isolation refers to the runtime platform independence for each module. Recent research done by Morariu et al. [4][5] has identified ways to assure this isolation by promoting loosely coupled systems, connected through service busses. Lack of one or both types of isolation can have catastrophic effects on the entire system even in case of a minor module failure. In order to achieve robustness, manufacturing systems need to become compositions of loosely coupled modules. This raises another important problem of how to locate and compose these modules into a functioning manufacturing system. Traditionally the composition was static and it was the responsibility of the system architect, due to intricacies of document and protocol standards and poor discovery and publishing services. With the emergence of standards in manufacturing domain like ISA 95/99, MIMOSA and others, the support for runtime and dynamic service composition has become available. Another critical aspect of any manufacturing system is the possibility to know what it is doing, the current state it is in, and the state history from the beginning of the execution to the current state. In other words, in order to know whether the manufacturing system is working correctly, it must provide means to be observed, or monitored in real time. However, the notion of observability as an abstract term in service oriented architecture (SOA) goes beyond just monitoring the system by inspecting the state of its services. For manufacturing systems, observability can have more concrete meanings, like energy consumption at shop floor level, resource utilization, time per operation, average make-span etc.

In 2010, Gartner Research published the Application Performance Monitoring (APM) Conceptual Framework [6] which defines five directions for APM: end-user experience, run-time application architecture, business transactions, component monitoring and reporting. While these directions are generally valid for generic APM solutions targeted for end user applications, for Manufacturing Systems Performance Monitoring (MSPM) a derived schema is more relevant. This is more evident if the manufacturing system design is based on SOA design patterns in conjunction with meta-heuristic algorithms or local intelligence for autonomous decision making. In these conditions the overall performance of the system is not a simple composition of the performance measured or estimated on each individual sub-module. For these reasons we propose a derived conceptual model for manufacturing systems performance monitoring, as illustrated in Fig. 1.

The MSPM framework has four main directions: service monitoring, resource monitoring, product monitoring and analytics/reporting.

Service Monitoring focuses on the actual web services exposed and used by the system sub-modules. The monitoring is done at the HTTP/SOAP protocol layer and the data consists in service status, response time for requests and overall workload.

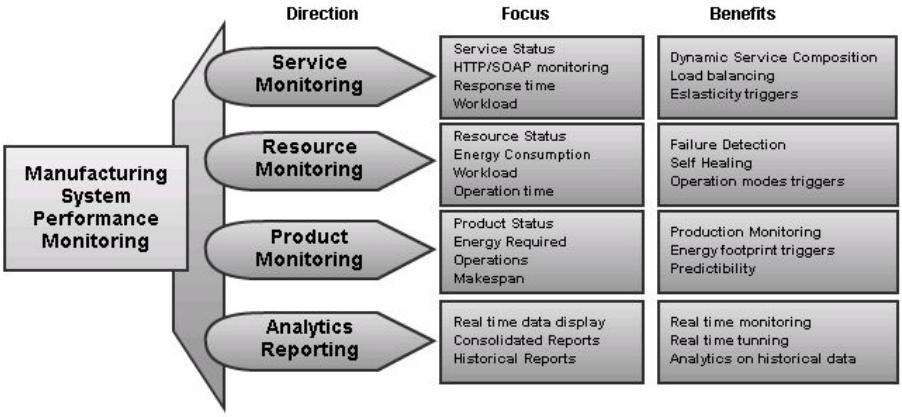


Fig. 1. Manufacturing System Performance Monitoring Directions

Using this information in real time allows the system to automatically reconfigure and re-compose complex services in case of an individual service failure. In case of an increased workload, a threshold mechanism can be used to balance the load to a secondary service provider. The data gathered here is used to monitor the information flow in the system. Resource Monitoring refers to the actual robots on the shop floor that are executing operations during the manufacturing process. The focus here is on the resource status, the energy consumption and on the resource utilization. These factors can generate triggers that can change the manufacturing system behaviour, as changing from a hierarchical operational mode to a heterachical operational mode or can change the scheduling strategy to avoid a defective resource. *Product Monitoring* is used to keep track of the products that are in production at any given time. The data collection focuses on energy consumption, real time status and make-span for each intelligent product. The data gathered here is used to monitor the material flow in the manufacturing system. The concept of intelligent product considered here was first described by McFarlane et al. [7] and developed further by Meyer [8]. Finally the *analytics and reporting* direction represent the mechanisms to consolidate the collected data around relevant key performance indicators (KPIs) for the manufacturing system. The most important KPIs and their impact on manufacturing system performance and integration were studied by Ahmad et al. [9], Cai et al. [10] and Zheng et al. [11]. The MSPM solution should be able to provide both real time reports that would allow dynamic tuning of the system and long term historical reports that would bring more predictability in the manufacturing processes.

Holonic manufacturing systems operating in an online scheduling mode, or also known as heterarchical operation mode, are affected by the risk of myopia, where intelligent products and agents have a limited information horizon. The implementation of advanced monitoring solutions can reduce this risk by augmenting the real time information available in the system with relevant real time KPIs.

This paper presents a MSPM solution for holonic manufacturing systems that aligns to these requirements, containing a MAS based architecture for real time data collection and a web based monitoring portal for real time data visualization and reporting.

2 Monitoring Solution for Holonic Manufacturing Systems

Multi agent systems have been used previously for control of manufacturing systems and implement communication and complex behaviour of the actors involved as described in the survey done by Monostori et. al [12]. The monitoring solution proposed in this paper is divided in two functional modules: the data collection module, responsible for gathering real time data from monitored targets and consolidate it in a persistent storage, and the monitoring portal which provides a user rich user interface for data visualization. The general architecture of the monitoring solution is illustrated in Fig. 2.

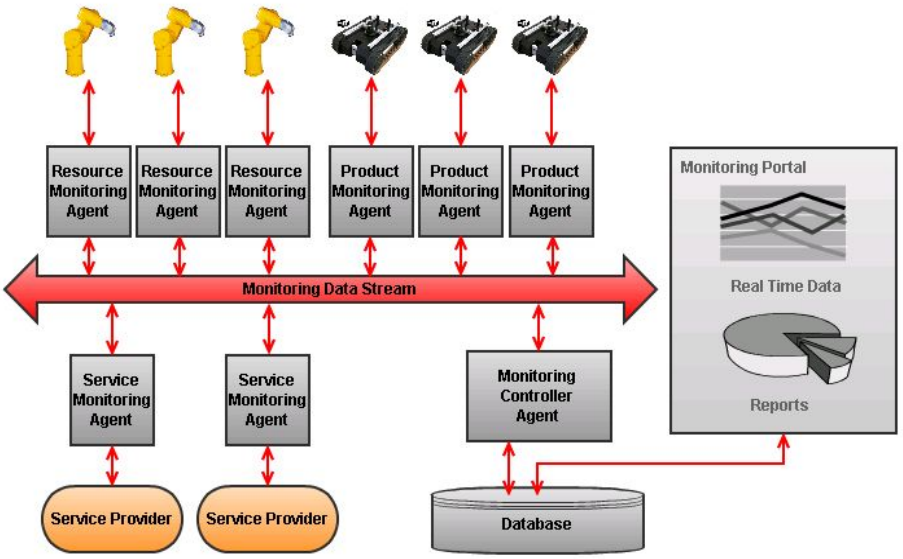


Fig. 2. Monitoring Solution Architecture

2.1 Data Collection Agents Design

There are four types of agents developed for this solution: resource monitoring agent, product monitoring agent, service monitoring agent and a controller agent. Along with these agents, there are three more components important in the architecture: the monitoring data stream, the monitoring controller agent and the monitoring portal. The following section provides some implementation details for each of these components.

2.1.1 Resource Monitoring Agent (RMA)

The resource monitoring agent (RMA) is implemented as a JADE agent with a cyclic behaviour. The agent does a cyclic poll at 5 seconds interval on the resource in order to gather information about the resource status, energy usage and the operations performed. The information is serialized and sent to the Controller Agent over the monitoring data stream.

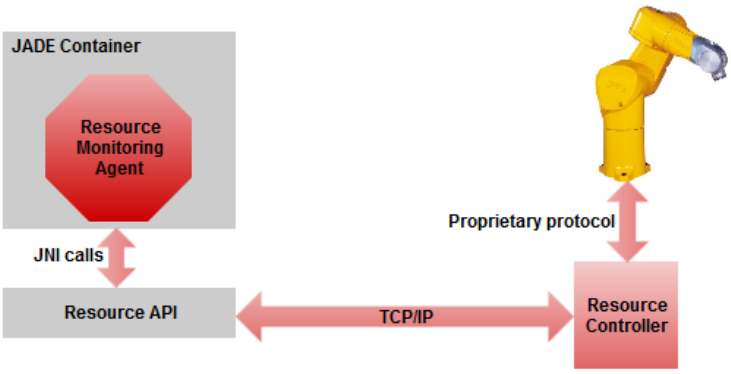


Fig. 3. Resource monitoring agent architecture

RMA typically interacts with the shop floor resource through a proprietary API provided by the resource vendor (Fig. 3). These APIs are provided most of the times in the form of native libraries, specific for each operating system and platform supported. For this reason, a generic implementation of the RMA cannot be provided, as the integration will differ for each resource type. However, the general architecture of the RMA, validated in the pilot implementation, consists of a JNI (Java Native Interface) interface that provides a wrapper over the native library and allows integration for the polling mechanism of the agent. The actual metrics available are highly dependent on the API provided by the vendor, but typically these include: resource real time status, resource power consumption, current operation and operation duration.

2.1.2 Product Monitoring Agent (PMA)

The product monitoring agent (PMA) is implemented as a JADE agent with a consumer behaviour. The agent runs in a shared JADE container and is notified after each operation performed. The agent records the operation data and sends these details on the monitoring data stream.

On the product pallet carrier (containing an Intelligent Embedded Device, IED) a small footprint monitoring agent runs, implemented specifically for the native embedded platform. This monitoring agent acts as a service client, creating and sending the notifications to the PMA over SOAP protocol using the build in WiFi network connection (Fig. 4). The monitoring data depends on the IED characteristics, but typically includes: operation execution time, time spent on the conveyor, cumulated travel time, current position and current action.

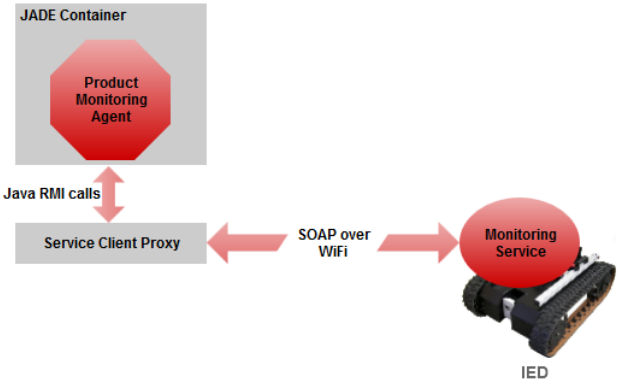


Fig. 4. Product monitoring agent architecture

2.1.3 Service Monitoring Agent

The service monitoring agent (SMA) is implemented as a Web Filter for the Web Container in the application server (Fig. 5). The filter calls the JADE API to send a FIPA INFORM message to the monitoring data stream including the timestamp, the URL of the service and the payload used to invoke it.

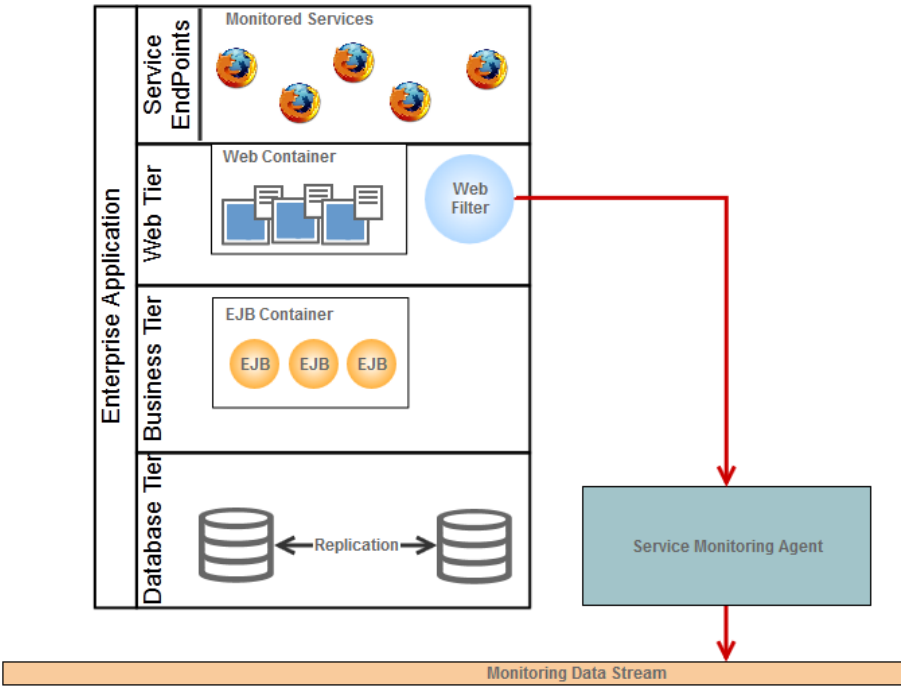


Fig. 5. Service monitoring agent architecture

Web Filter component definition was introduced in Java Servlet specification version 2.3. The filter intercepts requests and responses and has full access to the information contained in these requests or responses. Filters are useful for many scenarios where common processing logic can be encapsulated. Historically filters have been used for access management (blocking requests based on user identity), logging and tracking users of a web application, data compression, localization, XSLT transformations of content, encryption, caching, triggering resource related events, mime-type processing and many others. The implementation of a Web Filter is governed by the following interfaces: Filter, FilterChain, and FilterConfig in the javax.servlet package. The actual filter is a implementation of the Filter interface. The filters are invoked in a chained fashion by the servlet container. The Filter interface declares the doFilter method, which contains the actual processing of the request/response objects.

In our implementation, the doFilter method sends a message to the Monitoring Agent containing the service being invoked by the user. The information sent by the Web Filter to the monitoring agent has the following structure:

Table 1. Web Filter message structure

ID	The ID of the thread in which the web filter is called
Client ID	Client ID invoking the service
Service URL	URL of the service invoked
EndPoint	The service endpoint being invoked
Parameters	The complete payload used to invoke the service
Time	Time elapsed between request and response

2.1.4 Monitoring Data Stream, Controller Agent and Monitoring Portal

The *Monitoring Data Stream* is implemented as a message queue at JADE agent platform level. This queue allows asynchronous communication between the data producers, in this case the monitoring agents, and the data consumers: the controller agent and the monitoring portal.

The *Monitoring Controller Agent* is a JADE agent that implements a cyclic behaviour and consumes the monitoring messages sent to the monitoring data stream. The agent aggregates the data based on the monitoring target and saves it in the persistent storage. Monitoring Portal is a web based application that allows real time data display with AJAX and Partial Page Rendering, and report generation based on the data saved in the database. The integration and the dataflow between these components is illustrated in Fig. 2.

2.2 Data Storage Strategy

The strategy for the data storage is common for all the metrics recorded. For each metric collected by the target monitoring agents, the data is stored in two database tables using the following strategy:

- Short term storage (Staging): consists in a rolling table with a fixed number of rows, having a timestamp as a primary key. This table contains the "last N time intervals" of that particular metric and is used to display real time data in the web application.
- Long term storage: consists in a table containing averaged data for each metric for given time intervals (i.e. one hour). This data is generated from the short term storage, at each rollover by the controller agent. The long term storage table is used for reporting and analytics purposes (Fig. 6).

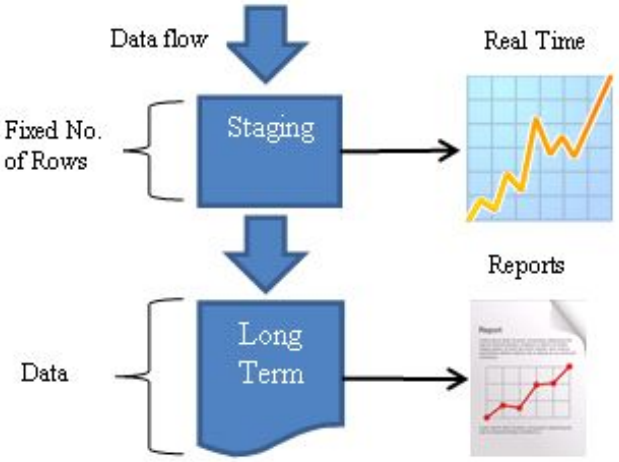


Fig. 6. Data Flow

The granularity in both short and long term storage is configurable globally and for each metric individually, allowing higher or lower history in real time views and detailed reporting. The agent uses an incoming message queue where messages from the other agents are placed in order to be processed. There are typically two types of messages for each metric. In case the target agent is sending real time data, the data will be stored in the staging table and the agent will rotate the staging table and promote aggregate metrics in the long term table. The other scenario is when the agent is sending directly aggregated metrics, which are added directly to the long term storage tables. Because of the large number of target agents and large amount of data to be managed, the monitoring controller agent (or main agent) is using a database connection pool where connections are re-used. This avoids the need to re-authenticate for each connection and improves the overall performance.

3 Agent Interaction and Scalability

The agent interaction is based on FIPA standard messaging protocol implemented by JADE [13]. JADE is a software framework designed to make the development of agent applications easier and in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. The goal of JADE is to simplify development while ensuring standard compliance through a comprehensive set of system services and agents.

To achieve such a goal, JADE offers the following list of features to the agent developer:

- FIPA-compliant Agent Platform, which includes the AMS (Agent Management System), the DF (Directory Facilitator), and the ACC (Agent Communication Channel). All these three agents are automatically activated at the agent platform start-up.
- Distributed agent platform. The agent platform can be distributed on several hosts (provided that the required ports are opened in the firewalls). Only one Java Virtual Machine is executed on each host. Agents are implemented as one Java thread and Java events are used for effective and light-weight communication between agents on the same host. Parallel tasks can be executed by one agent, and JADE schedules these tasks in a more efficient way than the Java Virtual Machine does for threads.
- A number of FIPA-compliant DFs (Directory Facilitator) can be started at run time in order to implement multi-domain applications. The concept of domain is a logical one as described in FIPA97 Part 1.
- API provided to allow registration of agent services with one or more domains (i.e. DF).
- JADE provides an integrated transport mechanism and interface (API) to send/receive messages to/from other agents
- FIPA97-compliant IIOP protocol is used to connect different agent platforms.
- Light-weight transport of ACL messages inside the same agent platform, as messages are transferred encoded as Java objects, rather than strings, in order to avoid marshalling and un-marshalling issues. When the sender or the receiver does not belong to the same platform, the message is automatically converted to/from the FIPA compliant string format. In this way, this conversion is hidden to the agent developer.
- JADE provides a library of FIPA interaction protocols.
- Automatic registration of agents with the Agent Monitoring Service (AMS).
- FIPA naming service, build in the platform, provides agents with a GUID (Globally Unique Identifier).
- Graphical user interface to manage several agents and agent platforms from the same agent. The activity of each platform can be monitored and logged.

The above characteristics make JADE a suitable platform for implementing a distributed data collection framework as it combines the flexibility and portability of Java with a very low memory footprint and CPU profile. This helps reducing the overhead of running JADE agents inside virtualized workloads in the cloud. The JADE container uses typically 32 Mb of memory for the Java heap, which is enough for most types of data collection.

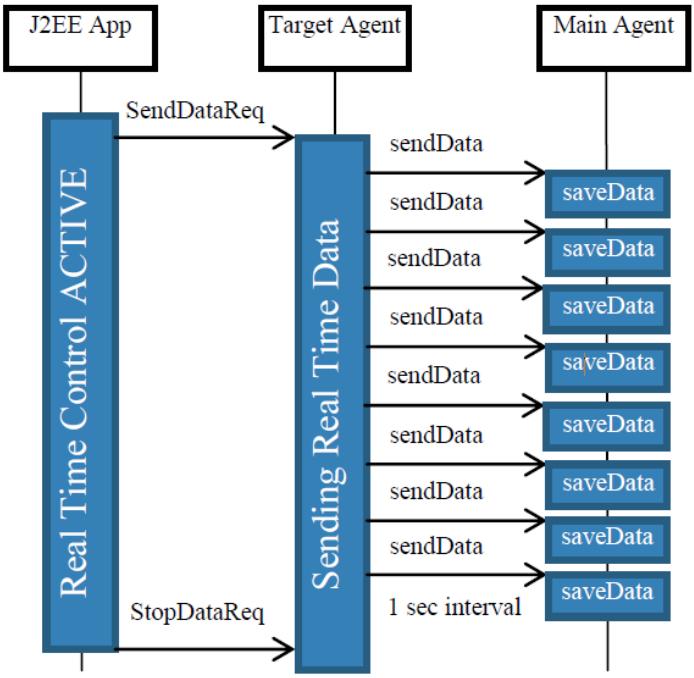


Fig. 7. Interaction diagram

The agent interaction in terms of real time data dialogue is presented in Fig. 7 and is similar for all data collection agents.

From Fig. 7 it can be seen that the normal operation of a data collection agent (target agent) is to send aggregate data to the main agent. When the end user accesses the web application and displays a page that requires real time data from a specific resource, the web application sends a message to the data collection agent. The data collection agent starts sending real time data to the main agent, which in turn stores it in the database. At this point, the database sends the change notification to the web application and the real time data is displayed to the user. Similarly when the real time data display control (graph, table, etc.) becomes inactive, the agent is notified to stop sending real time data. This approach reduces the network overhead of sending high granularity data, especially in large deployments, involving many agents.

In this agent architecture, scalability is implemented at the monitoring controller agent (or main agent) layer. This agent can aggregate data from other agents of the same type in a tree like model. This concept is illustrated in Fig. 8.

Each agent has a start-up switch that determines if the agent is the primary agent or if it is an intermediary agent used for data aggregation. Currently only the primary agent is storing data in the database. However, this agent is having an active/active setup in separate JADE containers, so that it assures high availability.

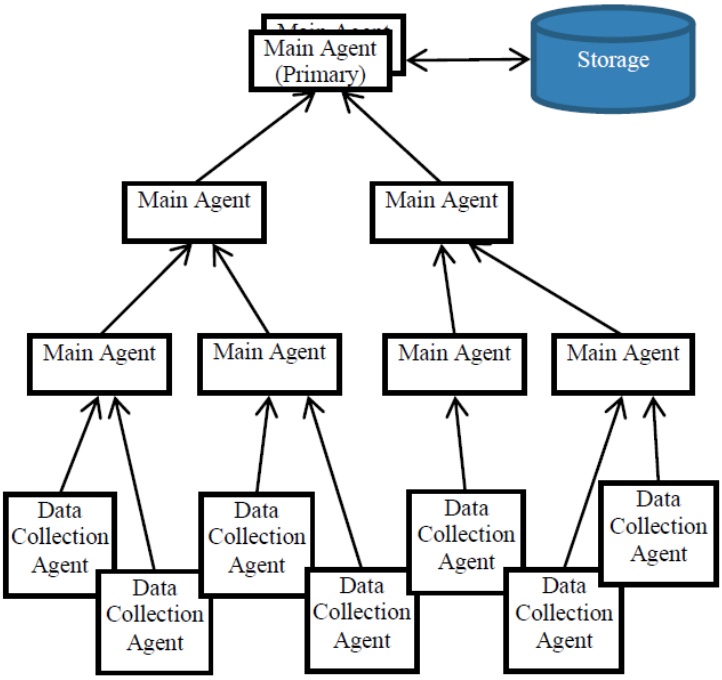


Fig. 8. Agent Model Scalability

The implementation contains an additional monitor agent that sends a FIPA inform message to each agent registered with JADE DF service. This assures that any agent failure can be detected by the monitor and handled accordingly.

4 Implementation Details and Experimental Results

The data collection agents are implemented using a JADE platform, in Java programming language. The generic code structure of a data collection agent is consisting in two behaviours: a ticker behaviour and a receiver behaviour. The first is triggering the polling mechanism that reads the target (resource/product/service) metrics. The second behaviour is listening for FIPA messages and processes them as required. The behaviours are defined by overriding the agentSpecificSetup() method:

```
protected void agentSpecificSetup() throws AgentInitializationException {
                    super.agentSpecificSetup();
  //registering with JADE DF Service
  ServiceDescription sd = new ServiceDescription();
  sd.setType("RESOURCE_MONITORING_AGENT");
  DFAgentDescription dfTemplate = new DFAgentDescription();
  dfTemplate.addServices(sd);
```

```
    SearchConstraints sc = new SearchConstraints();
    sc.setMaxResults(new Long(10));
    ACLMessage subscribe = DFService.createSubscriptionMessage(
                                    this, getDefaultDF(), dfTemplate, sc);
    //sending a subscription message to the JADE DF service
    //this will allow monitor agent to find us
    send(subscribe);
    //Defining ticker behaviour
    Behaviour ticker = new TickerBehaviour(this, 10000) {
      @Override
      protected void onTick() {
          try{
              collectDataAndSendMessage();
          }catch(Exception e){
             handleException();
          }
       }
    };
    //adding the behaviour to the agent
    addBehaviour(ticker);
    //defining message processor behaviour
    Behaviour messageProcessor = new CyclicBehaviour(this) {
      @Override
      public void action() {
         ACLMessage msg= receive();
         if (msg!=null){
            processMsg(msg);
         }else{
            //wait until a new message arrives in the queue
            block();
         }
      }
    };
    //adding the behaviour to the agent
    addBehaviour(messageProcessor);
}
```

The methods collectDataAndSendMessage() and processMsg() are implemented differently for each agent, depending on what target is the agent connecting to. However, these methods are defined in the DataCollectionAgent interface, which is implemented by every data collection agent. The UML class diagram showing the two interfaces implemented by the data collection agents is shown in Fig. 9.

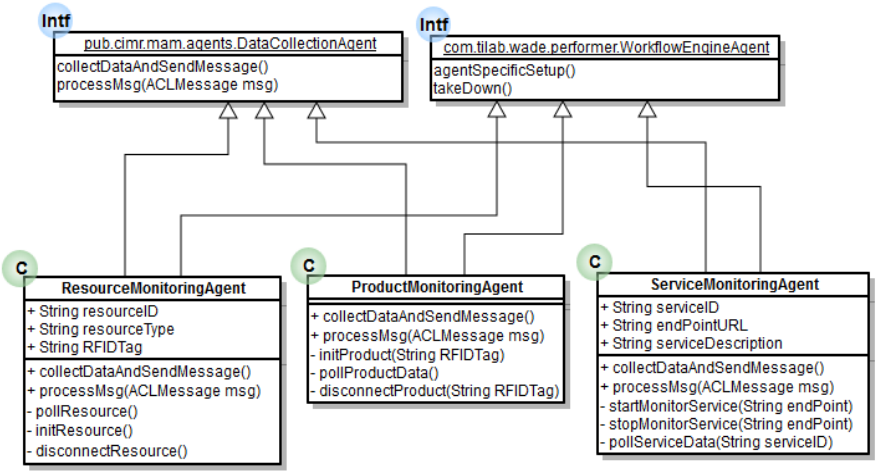
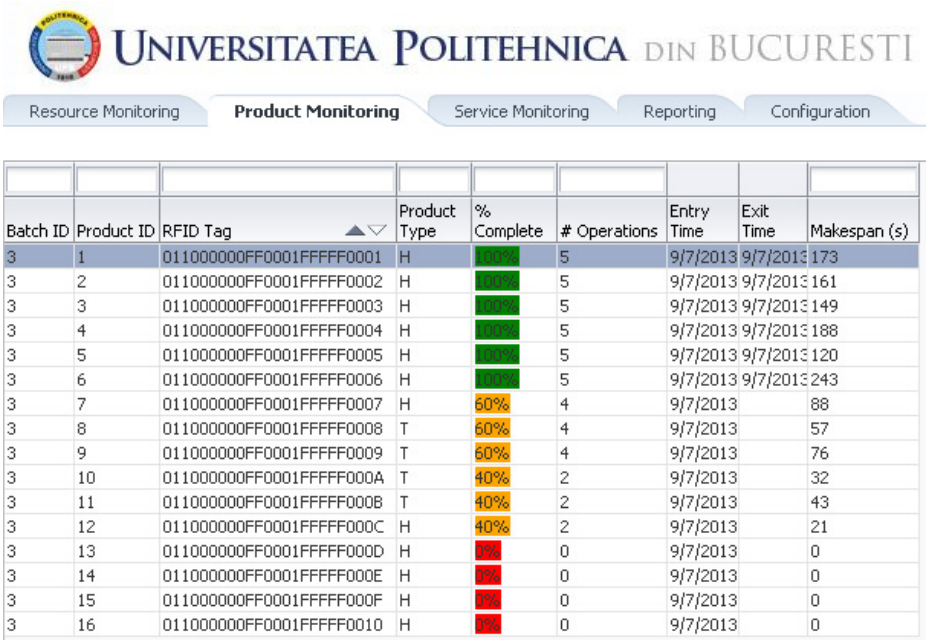


Fig. 9. UML class diagram

Each data collection agent is implementing both interfaces and implements the defined methods. Also, the agents provide implementation of specific methods and attributes used to connect and manage the lifecycle of the monitored target: resource, product or service.



Batch ID	Product ID	RFID Tag	Product Type	% Complete	# Operations	Entry Time	Exit Time	Makespan (s)
3	1	011000000FF0001FFFFF0001	H	100%	5	9/7/2013	9/7/2013	173
3	2	011000000FF0001FFFFF0002	H	100%	5	9/7/2013	9/7/2013	161
3	3	011000000FF0001FFFFF0003	H	100%	5	9/7/2013	9/7/2013	149
3	4	011000000FF0001FFFFF0004	H	100%	5	9/7/2013	9/7/2013	188
3	5	011000000FF0001FFFFF0005	H	100%	5	9/7/2013	9/7/2013	120
3	6	011000000FF0001FFFFF0006	H	100%	5	9/7/2013	9/7/2013	243
3	7	011000000FF0001FFFFF0007	H	60%	4	9/7/2013		88
3	8	011000000FF0001FFFFF0008	T	60%	4	9/7/2013		57
3	9	011000000FF0001FFFFF0009	T	60%	4	9/7/2013		76
3	10	011000000FF0001FFFFF000A	T	40%	2	9/7/2013		32
3	11	011000000FF0001FFFFF000B	T	40%	2	9/7/2013		43
3	12	011000000FF0001FFFFF000C	H	40%	2	9/7/2013		21
3	13	011000000FF0001FFFFF000D	H	0%	0	9/7/2013		0
3	14	011000000FF0001FFFFF000E	H	0%	0	9/7/2013		0
3	15	011000000FF0001FFFFF000F	H	0%	0	9/7/2013		0
3	16	011000000FF0001FFFFF0010	H	0%	0	9/7/2013		0

Fig. 10. Production tracking in monitoring portal

Fig. 10 presents a screenshot from the monitoring portal showing the production tacking page. The page contains a table presenting a real time view of the products in the product batch. The information presented in this table contains the ID of the product, the product type (in our case T shaped products and H shaped products), the % complete, the number of operations performed, the entry time and exit time and the make span.

Fig. 11. Real time resource utilization

Fig. 11 presents a screenshot from the resource monitoring portal. This page shows real time information about each shop floor resource, including: resource identifier, the picture of the resource (descriptive purpose), the resource utilization computed for the current product batch, the current operation executed by the resource, the current product in the corresponding workstation and the operation being performed. The last three columns are updated in real time based on data sent by the product monitoring agents.

The reporting engine is based on Pentaho [14] reporting solution and is offering a set of predefined reports, including: Resource Operation History, Product Operation History, Resource Utilization Trend and Product MakeSpan Trend. At the same time, the system allows plugin like creation of new reports, using the Pentaho report designer tool.

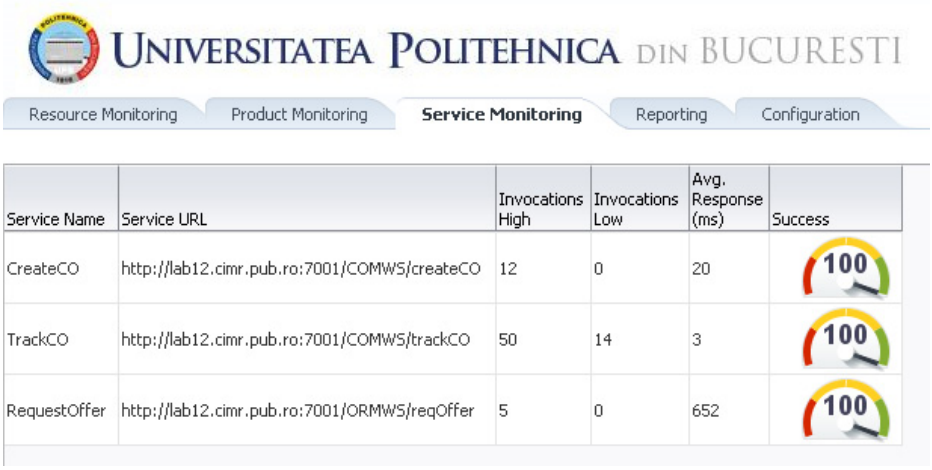


Fig. 12. Real time resource utilization

Finally, Fig. 12 presents the service monitoring module. In the pilot implementation the service monitoring agent is monitoring three web services: create customer order (CreateCO), track customer order (TrackCO) and request offer. The table shows real time data including the invocations high and low for the time unit (default is 1 day, specifically the last 24 hours), average response time in ms and the success rate.

5 Conclusions

Real time monitoring in a holonic manufacturing system provides valuable information that can be used for tuning the system and for long term reporting. Service monitoring is also an important aspect specifically targeted to illustrate the information flow in the SOA oriented manufacturing system. It helps identifying the number of invocations and the success rates together with the service performance. Real time monitoring of services provides useful clues when diagnosing information flow bottlenecks in the system. Product monitoring provides real time information about material flows and together with resource stock monitoring represent direct integration points for supply chain applications in regards to the manufacturing system.

The real time data collection provides the opportunity to create a trigger mechanism for various conditions, like resource breakdown or stock depletion. At the same time, due to the fact that all monitoring data is concentrated centrally in the Monitoring Data Stream, complex triggers can be defined that can consider multiple input signals, like energy consumption for a specific operation, in conjunction with the actual target product where the operations are performed. Currently the prototype of the monitoring application supports only basic resource monitoring and service monitoring, but the trigger mechanism is considered for future developments.

References

1. Borangiu, T.: A service-orientated arhitecture for holonic manufacturing control. In: Rudas, I.J., Fodor, J., Kacprzyk, J. (eds.) Towards Intelligent Engineering and Information Technology. SCI, vol. 243, pp. 489–503. Springer, Heidelberg (2009)
2. Borangiu, T., Gilbert, P., Ivanescu, N., Rosu, A.: An Implementing framework for holonic manufacturing control with multiple robot-vision stations. Journal of Engineering Applications of Artificial Intelligence (2009) ISSN 0952-1976
3. Borangiu, T., Raileanu, S., Anton, F., Parlea, M., Tahon, C., Berger, T., Trentesaux, D.: Product-driven automation in a service oriented manufacturing cell. In: Proceedings of the Int. Conf. on Industrial Engineering and Systems Management, IESM 2011, Metz, May 25-27, pp. 978–972 (2011) ISBN 978-2-9600532-3-4
4. Morariu, C., Morariu, O., Borangiu, T.: Manufacturing Service Bus Integration Model for Implementing Highly Flexible and Scalable Manufacturing Systems. Information Control Problems in Manufacturing, IFAC Papers Online 14(1), 1850–1855 (2012)
5. Morariu, C., Borangiu, T.: Manufacturing Integration Framework: A SOA Perspective on Manufacturing. Information Control Problems in Manufacturing, IFAC Papers Online 14(1), 31–38 (2012)
6. Cappelli, W.: Magic Quadrant for Application Performance Monitoring. Gartner Research (2012)
7. McFarlane, D., Sarma, S., Chirn, J.L., Wong, C.Y., Ashton, K.: The intelligent product in manufacturing control and management. In: 15th Triennial World Congress, Barcelona, Spain (2002)
8. Meyer, G.G., Främling, K., Holmström, J.: Intelligent products: A survey. Computers in Industry 60(3), 137–148 (2009)
9. Ahmad, M., Nasreddin, D.: Establishing and improving manufacturing performance measures. Robotics and Computer-Integrated Manufacturing 18(3), 171–176 (2002)
10. Cai, J., Xiangdong, L., Zhihui, X., Jin, L.: Improving supply chain performance management: A systematic approach to analyzing iterative KPI accomplishment. Decision Support Systems 46(2), 512–521 (2009)
11. Zheng, L., Jing, X., Hou, F., Feng, W., Na, L.: Cycle time reduction in assembly and test manufacturing factories: A KPI driven methodology. In: IEEE International Conference on Industrial Engineering and Engineering Management, IEEM 2008, pp. 1234–1238 (2008)
12. Monostori, L., Váncza, J., Kumara, S.R.: Agent-based systems for manufacturing. CIRP Annals-Manufacturing Technology 55(2), 697–720 (2006)
13. Bellifemine, F., Poggi, A., Rimassa, G.: JADE: a FIPA2000 compliant agent development environment. In: Proceedings of the 5th International Conference on Autonomous Agents, pp. 216–217. ACM (2001)
14. Bouman, R., Dongen, J.V.: Pentaho Solutions: Business Intelligence and Data Warehousing with Pentaho and MySQL. Wiley Publishing (2009)
15. Shen, W., Wang, L., Hao, Q.: Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 36(4), 563–577 (2006)