Computers in Industry 70 (2015) 89–101

Contents lists available at ScienceDirect

Computers in Industry

journal homepage: www.elsevier.com/locate/compind

Agent-based monitoring service for management of disruptive events in supply chains

Erica Fernández [1], Carlos M. Toledo *, María R. Galli [2], Enrique Salomone [3], Omar Chiotti [4]

INGAR(CONICET-UTN), Institute of Development and Design, National Council of Scientific and Technological Research, Avellaneda 3657, S3002GJC Santa Fe, Argentina

ARTICLE INFO

Article history:
Received 6 February 2014
Received in revised form 15 November 2014
Accepted 23 January 2015
Available online 21 March 2015

Keywords:
Multiagent system
Disruptive event
Supply chain
Monitoring
Model-driven development

ABSTRACT

Schedules of supply chains are generated with buffers to absorb the effect of disruptive events that could occur during their execution. Schedules can be systematically repaired through specific modifications within buffers by using appropriate decision models that consider the distributed nature of a supply chain. To this aim, information of disruptive events at occurrence or in advance allows decision models to make better decisions. To detect and predict disruptive events along a schedule execution, a service-oriented monitoring subsystem that uses a reference model for defining monitoring models was proposed. This subsystem offers services for collecting execution data of a schedule and environment data, and assessing them to detect/anticipate disruptive events. Because of the distributed nature and the complexity of these services functionalities, this paper presents an agent-based approach for their implementation. This technology allows dealing with supply chain monitoring by structuring monitoring subsystem functionalities as a set of autonomous entities. These entities are able to perform tailored plans created at execution time to concurrently monitor different schedules. A case study is described to try out the implemented prototype system.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

In an integrated supply chain, the overall performance largely depends on keeping the coordination of the schedules for producing and distributing the goods. These schedules are typically represented by production and distribution orders, where each order represents a particular instance of a generic supply process.

During the execution of the scheduled orders, significant changes may occur either in the specification of the orders or in the availability of the involved resources. These unplanned changes, called disruptive events, can produce negative effects that are propagated throughout the supply chain affecting schedules and their coordination [1–3].

The robust planning paradigm advises the definition of schedules with buffers (material, resource capacity, or time) that are capable to absorb the effect of disruptive events [4]. Some decision models were proposed to systematise the use of these buffers [5]. These models consider the distributed nature of a supply chain for repairing schedules through limited and specific modifications within the provided buffers [6]. To perform these modifications, the mentioned decision models require being notified on the occurrence or alerted about the possible occurrence of disruptive events by performing a continuous monitoring of the schedule execution.

Predictive monitoring is able to anticipate a disruptive event when there is enough evidence of its occurrence [7]. By collecting environment data (such as weather conditions or port congestion) and changes in the expected availability of resources (such as equipment breakdowns or breakage of materials), the predictive monitoring should be able to anticipate a possible change in an order specification. *Reactive monitoring* is able to detect a disruptive event when it has occurred. To this aim, it collects observed information on changes in resource availability and order specifications, assessing those changes to detect disruptive events.

* Corresponding author. Tel.: +54 342 453 4451/453 5568x3042; fax: +54 342 455 3439.
E-mail addresses: ericafernandez@santafe-conicet.gov.ar (E. Fernández), cmtoledo@santafe-conicet.gov.ar (C.M. Toledo), mrgalli@santafe-conicet.gov.ar (M.R. Galli), salomone@santafe-conicet.gov.ar (E. Salomone), chiotti@santafe-conicet.gov.ar (O. Chiotti).
[1] Tel.: +54 342 460 1579x268.
[2] Tel.: +54 342 453 4451/453 5568x3008.
[3] Tel.: +54 342 453 4451/453 5568.
[4] Tel.: +54 342 453 4451/453 5568x3007.

http://dx.doi.org/10.1016/j.compind.2015.01.009
0166-3615/© 2015 Elsevier B.V. All rights reserved.

Several approaches for reactive monitoring [8–13], for predictive monitoring [14,15] and for both reactive and predictive monitoring [5,16–19] were proposed. From the point of view of data collecting capability, they can be classified into approaches for order monitoring [5,11,16], approaches for resource monitoring [9,14,15,18] and approaches for both order and resource monitoring [8,9,12,13,19].

Monitoring systems are conceived as an extension of traditional tracking and tracing systems [20,21] with capability to collect data and to process these data in order to detect and/or anticipate disruptive events. Monitoring system capabilities rely on the monitoring process that defines the set of task to be performed for anticipating/detecting disruptive events [22].

Given the diverse nature of supply chain operations, monitoring processes are usually domain-specific and quite dependent on the type of resources and supply processes being monitored. To address this diversity in a systematic way, Fernández et al. [22] proposes a domain-independent metamodel as a reference model to generate monitoring processes for any kind of resources or supply processes. This reference model defines an abstract language that, unlike [13], allows the specification of models for reactive and predictive monitoring based on orders, resources, and environment data. By a set of transformation rules, monitoring models can be automatically transformed into monitoring processes to be performed by a monitoring system.

By using the proposed abstract language, users can represent the monitoring process of a supply process without being aware of the implementation technology. Each monitoring model, represented in terms of the reference model, could be also automatically transformed into different technological languages. However, the development of a monitoring system that implements this approach is still an unresolved issue which involves a complex challenge.

Based on the reference model proposed by [22], this paper presents an approach for implementing a monitoring subsystem as a part of an integral service-oriented architecture [23,24] for a Collaborative Management of Disruptive Events in Supply Chains system [25]. The monitoring subsystem can be hired by any enterprise involved in the supply chain. To this aim this subsystem provides a monitoring service with two main functionalities: collection of data could affect supply process executions; and collected data processing to detect and/or anticipate disruptive events.

This proposal introduces three novel aspects not addressed in previous works. First, the monitoring system is conceived as a multiagent system composed of autonomous agents with the ability to concurrently monitor a set of orders and resources involved in a schedule. Since agent-oriented paradigm provides suitable support for distributed systems and web service implementation [26,27], the proposed agents are able to work remotely collecting execution and environment data. Second, the proposed monitoring system allows the definition of new processes by just designing high-level models based on the reference model abstract language and dynamically generate, using a model-driven development approach, the executable instance of the monitoring process. That is, agents, plans, and assessment functions are created by transformations of a high-level conceptual model. Third, by further using transformation rules, the system is able to translate the predictive models declared at the high level language, into implementations with a specific tool (for instance, Bayesian network).

The remainder of this paper is structured as follows: Section 2 discusses related works. The multi-agent based architecture proposed for monitoring subsystem is presented in Section 3. Section 4 describes the implementation of monitoring subsystem. Section 5 describes a case study used for trying out the prototype system, and, finally, conclusions and future work are presented in Section 6.

2. Approaches for monitoring systems

The research related to supply chain monitoring still has not a body of disciplinary knowledge. It is supported by contributions from various disciplines and applied to different domains, which hinders to arrange an historical development of community awareness. In addition, recent works do not imply a necessary evolution of monitoring features, but rather they frequently refer to particular applications with different decision tools to detect/anticipate disruptive events in a domain.

With the purpose of reviewing related research work in a systematic way, we classify monitoring approaches taking into account their prediction ability (reactive/predictive) and the scope of event sources they are observing (orders/resources). Another aspect that this classification takes into account is the generality of approaches considering their capability for using different monitoring models that belong to different application domains. Based on this classification, relevant literature related to monitoring systems is reviewed. A summary of approach features is presented in Table 1.

Winkelmann et al. [8], Basal et al. [9], Liu et al. [10], Oztemel and Tekez [11], and Mahdavi et al. [12] present approaches focused on reactive monitoring. Winkelmann et al. [8] present a conceptual language for modelling monitoring processes by a set of rules based on arithmetic ratios of order specifications and material resource parameters. Basal et al. [9] present an approach based on key performance indicators assessed at regular intervals to detect material resource changes by monitoring the crude oil inventory. Liu et al. [10] present an approach that distinguishes task status-related events, events produced by a task, and external events, and define a set of rules relating them. Each rule is associated to a coloured Petri Net pattern in order to generate the monitoring process to detect disruptive events. Oztemel and Tekez [11] define several software agents responsible for performing different activities for monitoring manufacturing orders. For each activity, these agents can have different monitoring models that use information collected through a predefined network of sensors. Mahdavi et al. [12] develop an agent-based system for quality control of cement production processes. The system implements a model that receives the result of quality tests at each state of the supply process and uses a rule-based control mechanism for detecting disruptive events and correcting the process.

Table 1
Approach comparative.

Monitoring features	[8]	[9]	[10]	[11]	[12]	[14]	[15]	[16]	[17]	[5]	[18]	[19]	[13]
Reactive monitoring	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Predictive monitoring						✓	✓	✓	✓	✓	✓	✓	
Monitoring orders	✓		✓	✓	✓			✓	✓	✓		✓	✓
Monitoring resources	✓	✓	✓		✓	✓	✓				✓	✓	✓
Generality of the approach													✓

Approaches such as Kurbel and Schreber [14] and Kwangmyeong and Injun [15] considered the predictive monitoring of orders and/or resources. Kurbel and Schreber [14] develop an approach for anticipating disruptive events that could affect resource availability. Resource attributes are monitored in different supply process steps (milestones) and compared with target values to anticipate disruptive events. Similarly, Kwangmyeong and Injun [15] present an active data acquisition language for predictive monitoring of resources. It is applied to predict tools breakage by monitoring their attributes (axis displacement, tool bending load, and tool compression load).

Although described approaches have had good performance in domain they were implemented, decision models for systematising the use of buffers require information of disruptive events either at occurrence or in advance. For providing this information, support to reactive and predictive monitoring of schedules is required [28,29].

Among approaches that considers reactive and proactive monitoring, work of Zimmermann et al. [16], Xu [17] Zimmermann [5], Ribeiro et al. [18], and Heinecke et al. [19] can be highlighted. Zimmermann et al. [16] present an ontology-based process for monitoring orders with high probability of undergoing disruptions. The ontology includes milestones (control points), which are defined on orders to be monitored. Based on milestones, reactive monitoring of orders and predictive monitoring to anticipate changes of order specifications that could produce a disruptive event can be performed. A milestones approach also is proposed by Xu [17], which monitors order delivery by tracking work orders and controlling inventory levels. A disruptive event is detected when significant deviations occur in inventory levels during production process. Milestones are defined according to the local production process for a specific product. Zimmermann [5] develop an agent-based system for monitoring of orders during schedule execution. The monitoring process implemented by the system integrates data collected from supply chain members to detect a disruptive event and propagate its impact. Ribeiro et al. [18] also present a agent based approach by developing a service-oriented architecture for monitoring internal and external resources related to shop floor manufacturing. Each service is implemented by a software agent instantiated at design time, which represents a device with its specific data and its working and fault diagnosis models. Heinecke et al. [19] present a dynamic model to identify different states associated with a supply process through ratios of order specifications and material resource parameters. The model is used to simulate the knock-on effects of a disruptive event.

The above mentioned monitoring approaches are able to monitoring order and/or resources in a reactive and predictive way and provide information to domain-specific decision models that use buffers for repairing schedules, but their monitoring processes are usually domain-specific and quite dependent on the type of resources and supply processes being monitored. To address this diversity in a systematic way, Darmoul et al. [13] present a conceptual framework to support the reactive monitoring of orders and resources, the identification of disruption effects, and the determination of control actions. This framework defines a metamodel to represent reactive monitoring models for different type of supply processes. Fernández et al. [22] also propose a domain-independent approach by developing a metamodel as a reference model to generate monitoring processes for any type of resource or supply process. This reference model defines an abstract language that, unlike of the previous framework, allows specifying monitoring models for reactive and/or predictive monitoring of orders, resources, even considering environment data.

This paper uses this reference model to implement an agent-based monitoring system that provides information to decision models in a reactive and predictive way. In order to use domain-independent monitoring models, this system implements a set of transformation rules, which automatically transform monitoring models into decision support tools and monitoring processes to be performed by the monitoring system.

3. An agent-based monitoring subsystem

3.1. Monitoring Process

Def. 1. A schedule is a sorted set of orders and resources that specifies the time period during which each resource is required by each order, and its required capacity and states. It is defined by the tuple $Sch = (R, O, S, E, C)$ where: R is a set of resources r. O is a set of orders o. S is a set of order specification $s_o =< o$, quantity, startTime, endTime $>$ that states the quantity to produce/supply and times in which the order starts and ends. E is a set of states $e_{o,r,t} =< o, r, e, t >$ that specify the required state e of resource r for order o at time t. C is a set of capacities $c_{o,r,t} =< o, r, c, t >$ that specify the required capacity c of resource r for order o at time t.

Def. 2. A milestone is a control point that defines a state or time in which a set of variables will be observed. It is defined by the tuple $m = (mt, v, OV_m)$ where: mt is the milestone type (state or time); v is the state or time value; and OV_m is the set of observed variables. Observed variables can be an order specification s_o; the capacity c_r or a state e_r of a resource; or an environment variable $z \in Z$ that may affect an order or a resource.

Def. 3. A specific-platform monitoring process for each order o or resource r involved in a schedule Sch is defined by the tuple $MoPr = (MWf, EvFu)$ where: MWf is the monitoring workflow; and $EvFu$ is a set of evaluation functions composed by reactive evaluation functions ($reacEvFv$) and/or predictive evaluation functions ($predEvFu$), which allows assessing in a reactive and/or predictive way the possible occurrence of a disruptive event (Fig. 1).

Def. 4. The monitoring workflow is defined by the tuple $MWf = (CF, M, A, D)$ where: CF is an executable language conditions for activating each milestone $m \in M$, A is a set of actions to be performed in each milestone, and D a set of decisions to be made based on result of functions $EvFu$.

Def. 5. A reactive evaluation function $reacEvFv$ is a function able to detect a disruptive event during the execution of a schedule Sch, comparing the value of each observed variable Θs_o, Θc_r or Θe_r of the current milestone with its planned values for calculating its variation $\Delta s_o = \Theta s_o - s_o$, $\Delta c_r = \Theta c_r - c_{o,r,t}$ or $\Delta e_r = \Theta e_r - e_{o,r,t}$, and

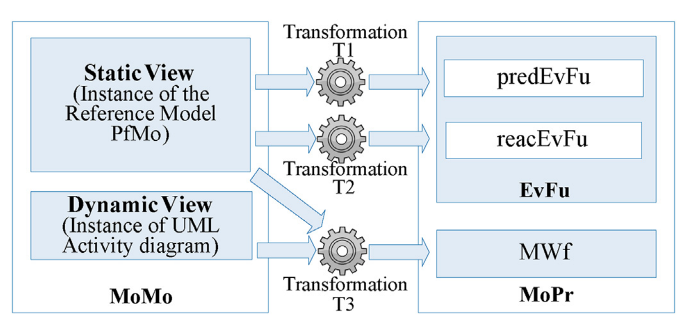


Fig. 1. Monitoring model and monitoring process: MDD transformation.

then comparing these variations with a threshold value δ_{s_o}, δ_{c_r} or δ_{e_r}. This function is defined as: if $(\Delta s_o \geq \delta_{s_o} \vee \Delta c_r \geq \delta_{c_r} \vee \Delta e_r \geq \delta_{e_r})$ then {disruptiveEvent = Yes} $\forall o$, r.

Def. 6. A predictive evaluation function *predEvFu* is a cause-effect relationship function able to infer changes Δs_o, Δc_r or Δe_r in the planned values of observed variables in upcoming milestones from values of observed variables in the current milestone. This function is defined as: $\Delta s_o = f(\Delta c_r,\ \Delta e_r,\ \Delta c_r(z \forall z),\ \Delta e_r(z \forall z) \forall r) \forall o$; $\Delta c_r = f(\Delta s_o, z \forall o, z) \forall r$; $\Delta e_r = f(\Delta s_o, z \forall o, z) \forall r$.

Def. 7. An independent-platform monitoring model of a supply process order $o \in O$ or a resource $r \in R$ is defined by the tupla *MoMo* = (*StaticView*, *DynamicView*) where: *StaticView* is an instance of the reference model *RfMo*. It is generated by users using the abstract language provided by *RfMo*. *DynamicView* is an instance of a UML Activity Diagram (UML) [30] (Fig. 1). It includes a set of predefined actions *A*, a reduced set of control nodes *CN* (start/end and split/merge), a set of subprocesses *SP*, and a control flow $CF \subset (A \cup CN \cup SP)^n$. Predefined actions *A* are: `Milestone[_]:`, `CollectData(_,_)`, `EvaluateDisruptiveEvent(_)`, `DisruptiveEvent`, and `EndMonitoring`; which allows defining a milestone, collecting the value of the observed variables, assessing in a reactive and/or predictive way the possible occurrence of a disruptive event, notifying a disruptive event, and finishing the monitoring process.

Def. 8. A reference model for Monitoring Orders and Resources is defined by the tuple *RfMo* = (*Et*, *Rt*) where: *Et* is a set of entity types that are instances of Eclass of Ecore meta-metamodel; and *Rt* is a set of relationships allowed between instances of entity types, these relationships are also instances of the Ereference class of Ecore meta-metamodel [22].

A model-driven development approach (MDD) based on the reference model *RfMo* is used to automate the process of generating a specific-platform monitoring process *MoPr* from the independent-platform monitoring model *MoMo*. To this aim, the following transformations are defined.

Def. 9. Transformation T1 is defined as a set of transformation rules tr_1 that allow generating predictive evaluation function from the static view of monitoring model, $T1(StaticView) \xrightarrow{tr_1} predEvFu$. Transformation T2 is defined as a set of transformation rules tr_2 that allow generating reactive evaluation functions from the static view of monitoring model, $T2(StaticView) \xrightarrow{tr_2} reacEvFu$. Transformation T3 is defined as a set of transformation rules tr_3 that allow generating the monitoring workflow from the static and dynamic views of the monitoring model, $T3(MoMo) \xrightarrow{tr_3} MWf$ (Fig. 1).

The relationship between the value of order specifications s_o, capacity c_r or state e_r of a resource *r* and environment variable *z* is generally subjected to uncertainty. Because of this, a probabilistic model is required to capture and propagate changes based on the structure of the supply process. To this aim, a specific model based on a Bayesian Network is transformated by *T*1.

Def. 10. A Bayesian Network allows representing uncertain knowledge and reasoning based on probability theory [31]. It is defined by the tuple *BNet* = (*X*, *ED*, *P*) where: *X* is a set of random variables called nodes, each $x \in X$ can take exclusive and exhaustive values in a continuous or discrete range; *ED* is a set of relations between *x* called edges; and P is a set of a joint probability distribution associated to each random variable *x*. *BNet* satisfies Markov condition if each variable *x* is conditionally independent of the set of all its descendents given the set of all its parents. If *BNet* satisfies Markov condition, then the joint probability distribution associated to a random variable *x* is equal to the product of its conditional distributions of all nodes given the values of their parents. That is $P(x) = \prod_{\forall x \in X} P(x/Parents(x))$.

3.2. System goals

The two main functional requirements identified for the monitoring subsystem are:

Requirement 1. Data collection, which implies system ability to collect the value of the set of observed variables OV_m associated to a milestone *m*.

Requirement 2. Identification of a disruptive event that may affect a schedule *Sch*, which implies system capability for processing collected values of the set of observed variables OV_m to detect and/or anticipate a disruptive event.

In order to accomplish these requirements, a Multi-Agent Monitoring Subsystem (MAMS) was developed [32]. It provides services for monitoring a schedule *Sch*, registering a new hiring enterprise, searching for previously defined monitoring models *MoMo*, and retrieving the reference model *RfMo* for defining a new monitoring model *MoMo*.

Def. 11. An agent is defined by a tuple *AgX* = (*BS*, *PL*, *GO*, *AC*) where: *BS* is a set of beliefs b_s; *PL* is a set of plans p_l; *GO* is a set of goals g_o; and *AC* is a set of actions a_c to be performed.

Based on functional requirements 1 and 2, a set of goals/sub-goals were defined to implement the monitoring service. Main actions $a_c \in AC$ and data needed for achieving these goals were identified (Table 2).

From goals, a set of BDI (Belief-Desire-Intention) agents were modelled (Fig. 2). In order to achieve the global goal for detecting and/or anticipating a disruptive event that could affect a schedule *Sch*, a cooperative architecture was defined where each agent is responsible for achieving a set of sub-goals [33].

3.3. Agent specification

Agents were identified by a goal analysis taking into account: 1 – activities to be performed; 2 – natural distribution of the architecture; 3 – agent participation of a cohesive set of sub-goals limiting their involvement to a different stage of the monitoring service; 4 – distribution of workload, avoiding complex coordination mechanisms.

As the internal logic of agents is dynamically created by a MDD transformation from *MoMo* tailored for each $o \in O$ or $r \in R$ of *Sch*, state machine diagrams are used to specify main states undergone by an agent during its life cycle. Following, each agent is described.

Agent Order predictive monitoring (AgOpm) monitors $o \in O$ of *Sch* to predict a disruptive event (Fig. 3). Given a monitoring model *MoMo*, AgM creates an instance of AgOpm at execution time for each order *o* to be monitored. AgM transforms *MoMo* into agent plans *PL* by using rules tr_3 which are performed by T3tool, i.e. $T3(MoMo) \xrightarrow{tr_3} PL$. Once created, AgOpm receives the *StaticView* of *MoMo*. Once started, AgOpm creates Decision Support Tool (DStool) and Transformation T1 Tool (T1tool). Then, AgOpm uses T1tool for creating the predictive evaluation functions from *StaticView* of *MoMo*, i.e. $T1(StaticView) \xrightarrow{tr_1} predEvFu$, which are used to initialise DStool.

Table 2
Agent goals.

Goal 1: Anticipate a disruptive event in an order $o \in O$ of a schedule *Sch*		
Sub-goal	**Actions**	**Involved Data**
Get information of a schedule *Sch*	Receive message	Service contract identification
Get model to be monitored	Receive message	*MoMo*
Make an agent responsible for monitoring	Create agent	
Get agent's plans $pl \in PL$ defined by the monitoring workflow *MWf*	MDD Transformation	*MoMo*, Transformation T3
Get evaluation functions *predEvFu*	MDD Transformation	*StaticView*, Transformation T1
Get values of observed variables OV_m for the current milestone m	Send message / Read raw data; Receive data from the Control subsystem	Sensors to collect current values of OV_m.
Anticipate a disruptive event	Inference process	*predEvFu*
Notify a disruptive event	Send agent/SOA message	Disruptive event
Identify the next milestone m and its set of observed variables OV_m		*MWf*
Goal 2: Anticipate a disruptive event in a resource $r \in R$ of a schedule *Sch*. (Sub-Goal, actions and involved data, are the same as that for Goal 1)		
Goal 3: Detect a disruptive event in an order $o \in O$ of a schedule *Sch*		
Sub-goal	**Actions**	**Involved Data**
Get information of a schedule *Sch*	Receive message	Service contract identification
Get model to be monitored	Receive message	*MoMo*
Make an agent responsible for monitoring	Create agent	
Get agent's plans $pl \in PL$ defined by the monitoring workflow *MWf*	MDD Transformation	*MoMo*, Transformation T3
Get evaluation functions *reacEvFu*	MDD Transformation	*StaticView*, Transformation T2
Get values of observed variables OV_m for the current milestone m	Send message / Read raw data; Receive data from the Control subsystem	Sensors to collect current values of OV_m.
Detect a disruptive event	Compare current and planned values	*reacEvFu*
Notify a disruptive event	Send agent/SOA message	Disruptive event
Identify the next milestone m and its set of observed variables OV_m		*MWf*
Goal 4: Detect a disruptive event in a resource $r \in R$ of a schedule *Sch*. (Sub-Goal, actions and involved data, are the same as that for Goal 3)		
Goal 5: Provide a monitoring model. sub-Goal: Retrieve a monitoring model *MoMo*; Actions: Semantic search; Involved Data: Keywords		
Goal 6: Provide the reference model. Actions Send the reference model *RfMo*		
Goal 7: Hiring monitoring service. sub-Goal: Negotiate contract and Assign contract to monitoring agent; Involved Data: Enterprise information		

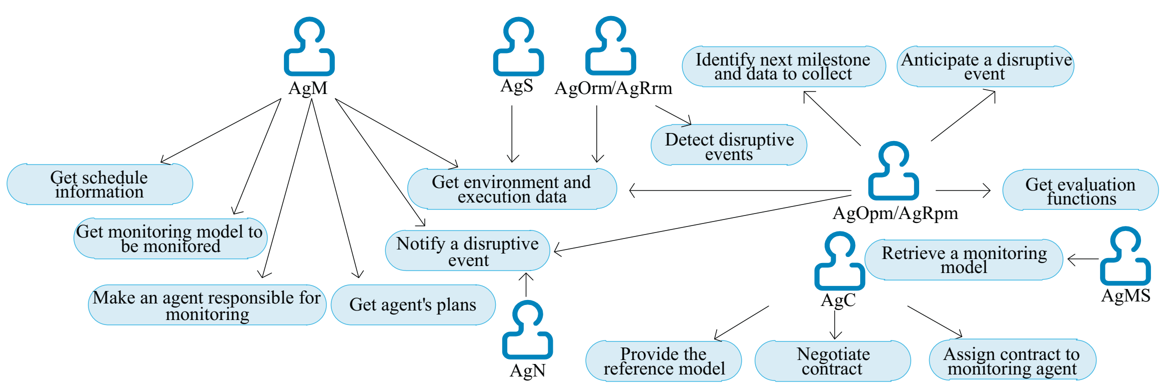


Fig. 2. MASM agents and goals.

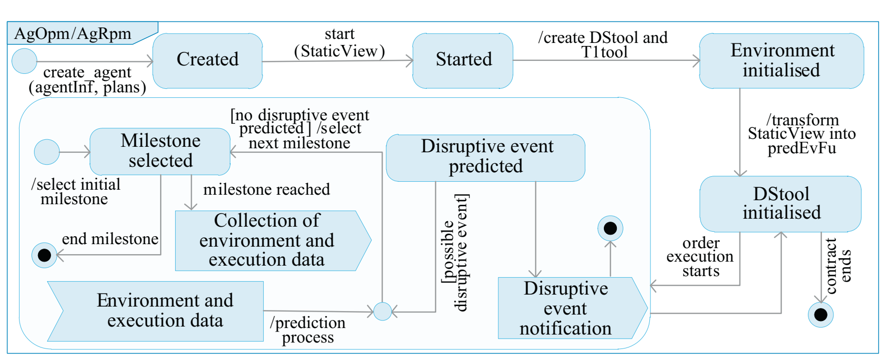


Fig. 3. Predictive monitoring: events and states.

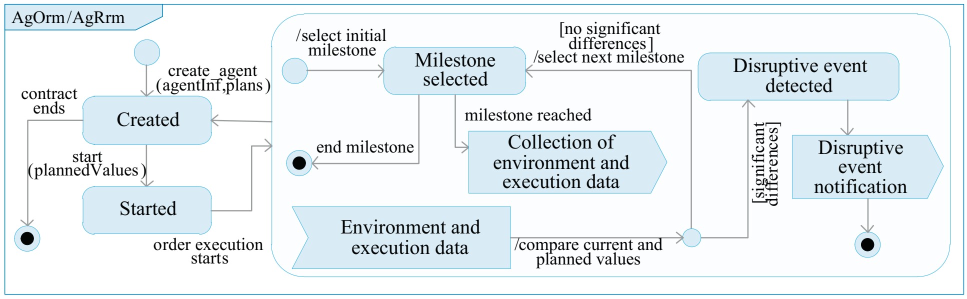


Fig. 4. Reactive monitoring: events and states.

When the execution of $o \in O$ starts, AgOpm selects an initial milestone m. When m is reached, AgOpm requests the collection of current values of $OV_m \in m$ to AgM. The DStool uses these values to perform an inference process. If a disruptive event is predicted, it is notified to AgM which in turn notifies the hiring enterprise and the monitoring ends. Otherwise, AgOpm selects the next milestone m. If it is an end milestone AgOpm returns to state `DStool_initialised` until receiving a new order to be monitored. This life cycle is repeated until the contract ends.

Agent Resource predictive monitoring (AgRpm) monitors state e_r or capacity c_r of resource $r \in R$ to predict a disruptive event. At milestone m, AgRpm uses *predEvFu* to infer how current values resource state Θe_r or capacity Θc_r can affect the value of state or capacity of r at future milestones. The state machine diagram for AgRpm is the same as that for AgOpm (Fig. 3).

Agent Order reactive monitoring (AgOrm) monitors changes in order specification s_o of $o \in O$ to detect a disruptive event (Fig. 4). Given a monitoring model *MoMo*, AgM creates an instance of AgOrm at execution time for monitoring o. AgM transforms the *MoMo* into agent plans *PL* by using rules tr_3 which are performed by T3tool, i.e. $T3(MoMo) \overset{tr_3}{\rightarrow} PL$. Once created, AgOpm receives *StaticView* of *MoMo*. Once started, AgOpm creates Transformation T2 Tool (T2tool). Then, AgOrm uses T2tool for creating reactive evaluation functions from *StaticView*, i.e. $T2(StaticView) \overset{tr_2}{\rightarrow} reacEvFu$, which are used to initialise DStool.

When the execution of $o \in O$ starts, AgOrm selects an initial milestone m. When m is reached, AgOrm requests the collection of current values of $s_o \in OV_m$ to AgM. AgOrm uses *reacEvFu* for analysing these values. If a disruptive event is detected, it is notified to AgM which in turn notifies the hiring enterprise and the monitoring ends. Otherwise, AgOrm selects the next milestone. If it is an end milestone, AgOrm returns to state `Created` until receiving a new $o \in O$. This life cycle is repeated until the contract ends.

Agent Resource reactive monitoring (AgRrm) monitors a resource $r \in R$ of *Sch* to detect a disruptive event. Once the execution of *Sch* starts, each time a milestone m is reached, AgRrm uses *reacEvFu* for detecting if a disruptive event has occurred. The state machine diagram for AgRrm is the same as that for AgOrm (Fig. 4).

Agent Sensor (AgS) collects the current value of OV_m that Control Subsystem cannot provide. AgS supplies current values when they are required (preplanned milestones) or when it detects sensed values have significantly changed (milestones dynamically created).

Agent Notifier (AgN) enables additional interactions with the hiring enterprise. AgN reports, in a simple and human-readable format, on the monitoring task progress and a disruptive event. In order to renew monitoring contracts, AgN also facilitates Monitoring Subsystem to communicate with the hiring interface in order to access its services.

Agent Monitoring (AgM) fulfils a monitoring contract. Once AgC and the hiring enterprise agreed a monitoring contract, an instance of AgM responsible for this contract is created. Monitoring information and *MoMo* are supplied to AgM (Fig. 5). AgM creates: its workspace, T3tool, and the interface for communicating with Control Subsystem of the hiring enterprise. When a monitoring request from the hiring enterprise is received, depending on the monitoring requirement (order/resource and reactive/predictive), AgM creates specialised agents AgOpm, AgRpm, AgOrm, and/or AgRrm by using T3tool. Once created, AgM sends *StaticView*, reaching state `Specialised_agent_ready`. From this state, AgM can fulfil monitoring requests that belong to the contract.

During the monitoring, AgM handles the data exchange among AgOpm, AgRpm, AgOrm or AgRrm and Control Subsystem, AgN and each AgS. AgM receives requests from AgOpm, AgRpm, AgOrm or AgRrm for collecting current values of OV_m and forwards them to Control Subsystem and/or each AgS. When Control Subsystem and/or AgSs reply with requested values, AgM receives and forwards them to AgOpm, AgRpm, AgOrm, or AgRrm that requested them. If AgM receives a disruptive event notification, it is sent to Control

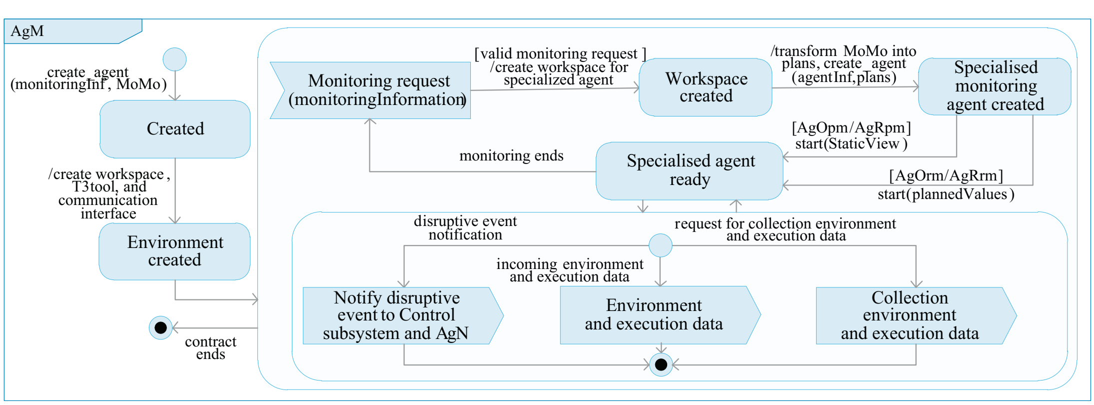


Fig. 5. Monitoring: events and states.

Subsystem and to AgN at the same time. Once monitoring ends, AgM waits for a new monitoring request, until it arrives or the contract ends.

Agent Model Searcher (AgMS) stores and retrieves a *MoMo* using a keyword searching strategy. Monitoring service requires defining a *MoMo* for each $o \in O$ or $r \in R$ of *Sch*. To this aim, the hiring enterprise can either generate each *MoMo* based on an already stored model in the model base, or define it from scratch by modelling *StaticView* and *DynamicView*. For *StaticView*, the hiring enterprise must request *RfMo* and generate an instance of it with the particular characteristics of its $o \in O$ or $r \in R$. For *DynamicView*, the hiring enterprise must model *DynamicView* in a UML activity diagram. Except for exclusive contracts, each new *MoMo* is automatically stored in the model base for future uses.

Agent Coordinator (AgC) responds to incoming service requests. It is created at system startup (Fig. 6). AgC creates: the workspaces, the hiring interface, a model base to store *RfMo*, a model base to store each *MoMo*, and a sensor base to store pre-programmed AgSs. After creating AgMS, AgC reaches state `AgMS_created`. From this state, AgC can receive requests for enterprise registration, provision of *RfMo*, *MoMo* searches, and monitoring service hires.

For an enterprise registration, AgC generates a new account and responds with an identification to be used by the hiring enterprise for future service access. To respond to a request from the hiring enterprise that requires *RfMo*, AgC looks for *RfMo* in the model base and sends it. For *MoMo* searches, AgC asks AgMS for a *MoMo*, who searches for it in the model base using a keyword-based strategy.

For hiring the monitoring service, AgC negotiates contract terms such as monitoring type (order/resource and reactive/predictive), *MoMo* exclusivity (*MoMo* is not stored for future uses), hiring time (time lapse of the contract), values of OV_m that will be provided by Control Subsystem of the hiring enterprise, and values that will be collected by AgSs. Once it is agreed, state `Service_contracted` is reached. AgC creates an instance of AgN and a set of AgSs from the sensor base, and afterwards it creates an instance of AgM, which will perform the monitoring task defined in the contract. Based on the contract and data that will be collected by AgSs, AgC specifies two interfaces: the observed variables interface to be implemented by Control Subsystem, and the monitoring interface that is implemented by AgM. Following, contract terms, interface specifications, AgSs, and AgN are sent to the hiring enterprise. Once any of these service requests are finished, AgC returns to state `Incoming_request` to wait for new service requests.

4. Multi-agent design and implementation

4.1. Architectural design and implementation

The architecture was implemented using JaCaMo agent programming platform [34,35]. JaCaMo encompasses the three main abstraction levels of a multi-agent system: agent, organisation, and environment. It allows programming BDI agents governed by Moise organisational model, and supporting Agent & Artefact metamodel [36]. Moise describes organisational collaborations and addresses collective behaviour by individual behaviour constraints [37]. Agent & Artefact metamodel allows implementing environment-based coordination mechanisms, and non-autonomous services and tools [38].

JaCaMo was selected since it is an operational programming platform that includes integrated support for programming artefacts, which facilitated the integration of non-goal oriented functionalities such as: model-to-model and model-to-code transformation engines to implement T1tool, T2tool, T3tool, and DStool. Besides this facility, JaCaMo provided support for implementing web services from WSDL (Web Services Description Language) specification [39,40]. Although JaCaMo's features facilitated MASM architecture implementation, it could be implemented by using any platform that allows implementing BDI-agents such as Jadex [41], Jack [42], among others.

Agents were implemented by means of beliefs, rules, goals, and plans, which represent their informational, motivational, and deliberative states [43]. The organisation was modelled by agent roles, missions, and normative rules [37]. The environment was specified by artefacts and workspaces. An artefact defines a non-autonomous first-class entity that represents a tool that agents can instantiate, share, use, and perceive at runtime. A workspace defines the environment topology acting as a logical container grouping agents and artefacts [44]. CArtAgO-WS artefacts were used for defining web services through environment artefacts using WSDL specifications of provided services [45]. CArtAgO-WS includes two specialised artefacts: WSInterface and WSPanel. WSInterface provides necessary functionalities for interacting with web services, and WSPanel allows creating, configuring, and controlling new web services [46].

The MAMS is composed of five workspaces named Hiring, Monitoring, Order & Resource Monitoring -performed in Monitoring Subsystem-, Sensor -performed in places where data are collected-, and Notifier -performed in the hiring enterprise (Fig. 7). These workspaces were defined for grouping agents related to similar monitoring tasks and agents that share environment

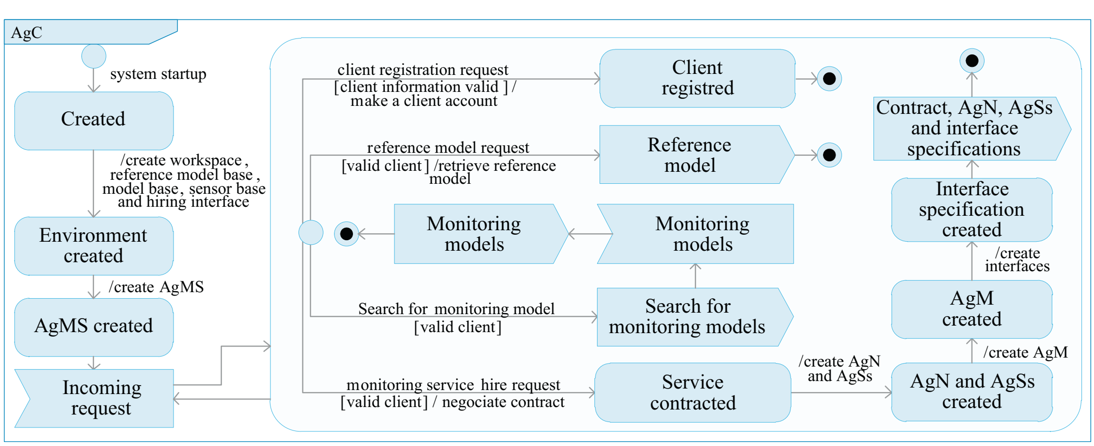


Fig. 6. Coordinator: events and states.

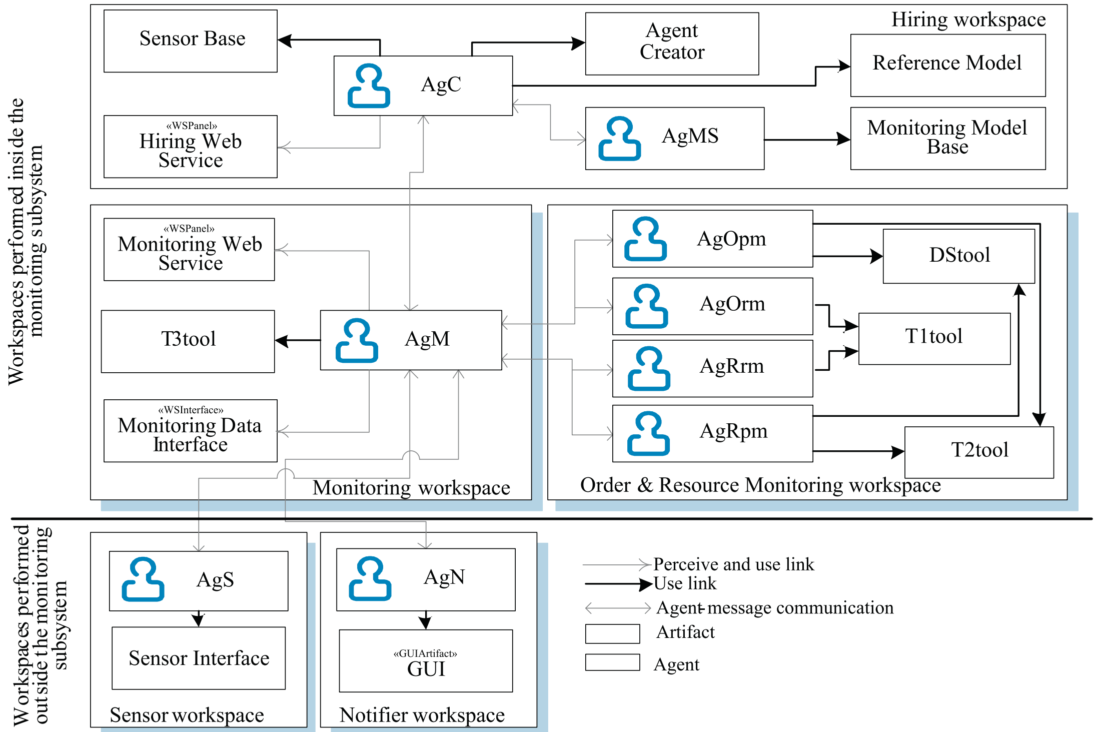


Fig. 7. JaCaMo implementation.

artefacts, and for allowing agents' execution in the hiring enterprise.

Hiring workspace groups AgC, AgMS, and artefacts: Hiring Web Service that implements the hiring interface using CArtAgO-WS; reference model that stores *RfMo* codified in Ecore [47]; Monitoring Model Base, a software interface to the model base that stores each *MoMo*; Sensor Base, a software interface to a sensor base that stores a set of pre-programmed AgS, each AgS includes a tailored interface that allows communicating with physical sensors, software systems, or a human interface; and Agent Creator, responsible for modifying the code of AgS to add communication parameters that allow them to communicate with the proper AgM.

Monitoring workspace groups AgM and artefacts: Monitoring Web Service that implements the monitoring interface using CArtAgO-WS; Monitoring Data Interface, implemented using WSInterface-like, allows AgM to communicate with the web services implemented by Control Subsystem; and T3tool, responsible for generating agent's plans. Using T3tool, a *MoMo* is transformed into the metamodel of AgentSpeak plans using ATL,[5] i.e. $T3(MoMo) \xrightarrow{tr_3} AgentSpeakPlans$ and following it is transformed into AgentSpeak plans by using Xpand[6] to be interpreted by JaCaMo agents [48].

Order & Resource Monitoring workspace (O&RMWs) groups AgOpm, AgOrm, AgRpm, AgRrm, and artefacts DStool, T1tool and T2tool. Currently, T1tool allows generating predictive evaluation functions *predEvFu* as Bayesian Networks, i.e. $T1(StaticView) \xrightarrow{tr_1} BNet$. DStool was implemented by means of Hugin API.[7] DStool could be implemented for processing any decision algorithm to anticipate a disruptive event (for example, based on Petri nets, decision trees, etc.). To this aim, appropriate transformation rules for T1tool must be developed. T1tool implements a transformation engine using ATL API.

Through MAMS life cycle, workspaces and agents are created and destroyed (Fig. 8). The hiring workspace and its artefacts are initially created. Then, AgC and AgMS are created. When a monitoring service is contracted, a new AgM is created together with its monitoring workspace and artefacts. Then, workspaces for AgS and AgN and their artefacts are created. AgS and AgN are created from WSDL specification and *MoMo*. These workspaces will exist in the system until the contract ends. When an order execution starts, artefacts DStool and T1tool are created if a predictive monitoring is to be performed.

MAMS organisation was defined by functional, structural, and deontic specifications of Moise model. Each modelled agent adopts a role in the organisation (structural specification). Goals, decomposed in sub-goals (Table 2), were distributed to agents in the form of missions (functional specification). Since MAMS was designed as a cooperative system, all agents must collaborate to achieve the global goal, and therefore each agent committed to a mission is obligated to accomplish it. It is specified by an obligation norm (deontic specification).

In order to exemplify the operation of MAMS architecture, the following subsection presents details of monitoring processes.

4.2. Agent interaction design and implementation: monitoring process

Once the monitoring service has been hired, the *Sch* to be monitored is accepted by Monitoring Subsystem. As an example, the order predictive monitoring process and the interaction between Control and Monitoring Subsystems are depicted in Fig. 9. Table 3 details message sequence. For simplifying interactions description, exceptions and failure control operations are not represented.

[5] ATL Transformation Language, Eclipse project – http://www.eclipse.org/atl/.
[6] Xpand Transformation Language, Eclipse project – http://www.eclipse.org/modeling/m2t/?project=xpand.
[7] Hugin Expert, Hugin Decision Engine Tool – http://www.hugin.com/.

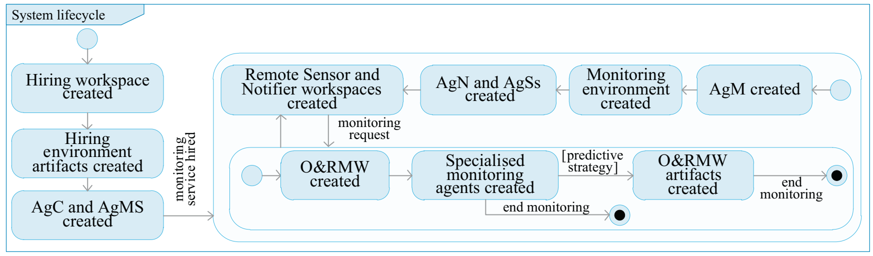


Fig. 8. System life cycle.

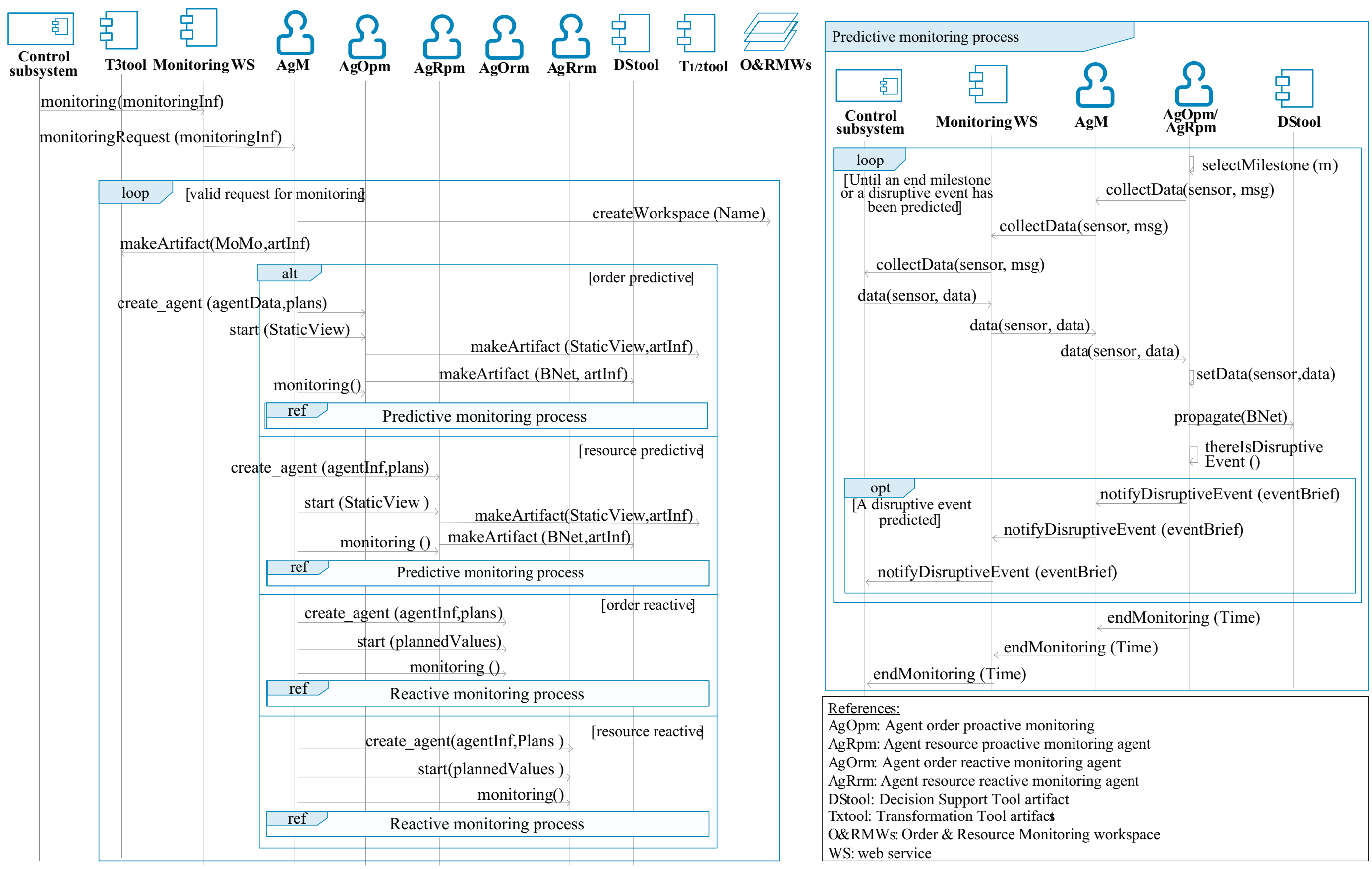


Fig. 9. Monitoring process: simplified interaction diagram.

5. A case study: cheese production

In order to depict the use of MAMS, this section presents a cheese production process as a case study to exemplify the *MoMo* specification and its transformation in a specific DStool used by AgOpm to predict a disruptive event.

Process data were collected from a regional dairy industry. *Sch* and execution data were obtained from the database of planning and execution systems. In the cheese production process, milk acidity can affect cheese quality. High acidity can produce sandy cheese, bitter cheese, or increase the curdling rate causing surface cracks. Low acidity can produce insipid cheese.

To illustrate this case study, the scenario in which the monitoring service is hired for monitoring an $o \in O$ that requires producing 1 Tn of soft cheese (quantity = 1) is described. Fig. 10 shows *StaticView* of *MoMo* for the cheese production process. It presents the set of milestones m_1=(state, process_starts, OV_1), m_2=(state, curdle_finishes, OV_2), m_3=(time, 48hs_after_process_has_started, OV_3), m_4=(time, 72hs_after_process_has_started, OV_4); m_5=(time, 120hs_after_process_has_started, OV_5) and m_6=(state, process_ends, OV_6) where: OV_1=(acidity), OV_2=(time_of_curdle), OV_3=(texture), OV_4=(surface_cracks), OV_5=(taste) and OV_6=(quality). Acidity and time_of_curdle are parameters of resource *milk* $\in R$; whereas texture, surface_cracks, taste, and quality are parameters of resource *cheese* $\in R$. The target variable is estimated_quantity. It has a planned value of parameter quantity for $o \in O$ (quantity = 1). A disruptive event can be predicted by comparing planned and estimated values, based on a disruption condition.

Each $var_i \in OV_m$ is defined by $var_i = < name, type, sensor >$ where: type specifies if the variable takes discrete or continuous values; and sensor describes the sensor type required to collect data. Based on this parameter, AgM selects a suitable AgS from artefact Sensor Base.

T1tool of MAMS, which implements the set of ATL rules described in [22], allows generating *predEvFu* as a Bayesian Network (Fig. 11) from *StaticView* of *MoMo* (Fig. 10).

Table 3
Monitoring process: messages details.

Message	Source	Recipient	Message type	Details
monitoring (monitoringInf)	Control Subsystem	Monitoring WS	SOAP	monitoringInf:<contract id,*MoMo*,type>
monitoringRequest (monitoringInf)	Monitoring WS	Monitoring	Artefact perception	monitoringInf:<contract id,*MoMo*>
makeArtefact(*MoMo*,artInf)	Monitoring	T3tool	Artefact operation	artInf:<name,class,parameters,id>
createWorkspace (Name)	Monitoring	O&RMWs	Workspace primitive	
create_agent(agentInf, plans)	Monitoring	AgOpm/vAgRpm	Agent primitive	agentInf:<name,pathfile,Jason class>plans: <agent plans codified in AgentSpeak>
start(*StaticView*)	Monitoring	AgOpm/AgRpm	Agent plan	*StaticView* of *MoMo*
start(*PlannedValues*)	Monitoring		Agent plan	Set of planned values for observable variables
		AgOrm/AgRrm		
makeArtefact (*StaticView*,artInf)	AgOpm/AgRpm	T1tool/T2tool	Artefact operation	artInf:<name,class,parameters,id>
makeArtefact (*BNet*,artInf)	AgOpm/AgRpm	DStool	Artefact operation	artInf:<name,class,parameters,id>
monitoring()	Monitoring	AgOpm/AgRpm AgOrm/AgRrm	KQML message	
selectMilestone(m)	AgOpm/AgRpm		Agent plan	
collectData (sensor, msg)	AgOpm/AgRpm	Monitoring	KQML message	sensor that must collect data, msg: message to show for manual entry
data(sensor,data)	AgOpm/AgRpm		KQML message	sensor that collected data and data value
setData(sensor,data)	AgOpm/AgRpm		Agent plan self addition	
propagateNet(*BNet*)		DStool	Artefact operation	
notifyDisruptiveEvent(eventBrief)	AgOpm/AgRpm		Agent plan self addition	eventBrief: (time, reason, sensor data, etc.)
notifyDisruptiveEvent(eventBrief)	AgOpm/AgRpm	Monitoring	KQML message	eventBrief: (time, reason, sensor data, etc.)
notifyDisruptiveEvent(eventBrief)	Monitoring	Monitoring WS	Artefact operation	event brief (time, reason, sensor data, etc.)
notifyDisruptiveEvent(eventBrief)	Monitoring WS	Control subsystem	Webservice request	eventBrief: (time, reason, sensor data, etc.)
endMonitoring(Time)	AgOpm/AgRpm	AgM	KQML message	Time when the disruptive event ocurred
endMonitoring(Time)	AgM	Monitoring WS	Artefact operation	Time when the disruptive event ocurred
endMonitoring(Time)	Monitoring WS	Control subsystem WS	Webservice request	Time when the disruptive event ocurred

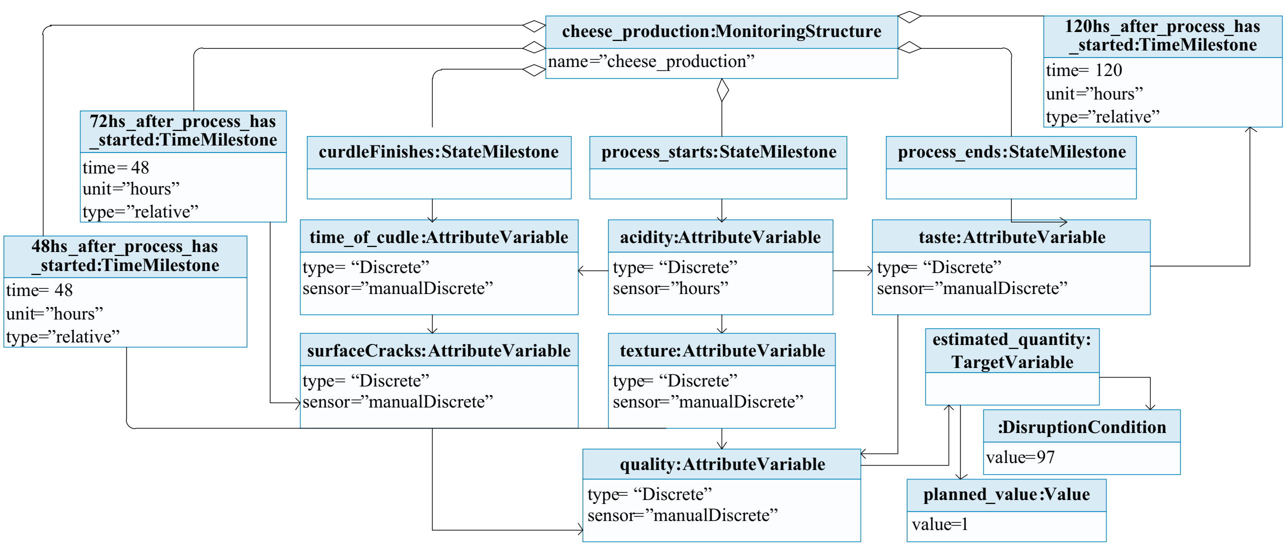


Fig. 10. *StaticView* of *MoMo* for cheese production.

BNet consists of discrete nodes X, each one representing a variable to be observed OV_m in a milestone m (Table 4). Domain values of OV_m are represented in the fifth column. The target variable of *StaticView* is transformed into node estimated_quantity, which is a function node with a disruption condition. This function is defined in the fifth column. Value 0 indicates that no cheese with the specified quality could be produced; and value 1 indicates that 1 Tn of cheese with the specified quality could be produced.

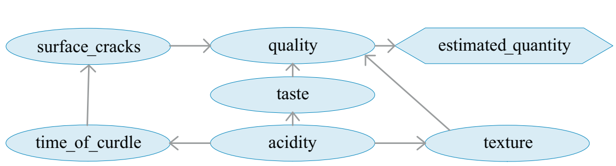


Fig. 11. Cheese production: Bayesian network.

MWf of the cheese production process was generated using T3tool. Based on *MWf*, AgOpm decides on the actions to be taken, milestones to be activated, and data to be collected (Fig. 12).

5.1. Result analysis

Table 5 summarises results reported by Monitoring Subsystem for six scenarios. For each scenario, the second column shows successive milestones defined by AgOpm until reaching a conclusion. The third and fourth columns show OV_m in each milestone m and their collected values. These values were evidences taken into account by AgOpm for calculating the joint probability P(quality==bad) or P(quality==good). Results are represented in the fifth column. While these evidences were insufficient to predict an outcome, AgOpm proposed a new milestone to collect evidence to enable it to predict that an event

Table 4
Relationship between milestones of the cheese monitoring model and nodes in the Bayesian network.

StaticView		*predEvFu*: BNet		
m	OV_m	Node X	Type	Value domain
process_starts	acidity	acidity	Discrete	normal, low, high
curdle_finishes	time_of_curdle	time_of_curdle	Discrete	normal, low, high
48hs_after_process_has_started	texture	texture	Discrete	no_granulated, granulated
72hs_after_process_has_started	surface_cracks	surface_cracks	Discrete	no, yes
120hs_after_process_has_started	taste	taste	Discrete	good, insipid, bitter
process_ends	quality	quality	Discrete	good, bad
Target variable		estimated_quantity	Function	if (P (quality==bad) >value) estimated_quantity = 0 else estimated_quantity = 1

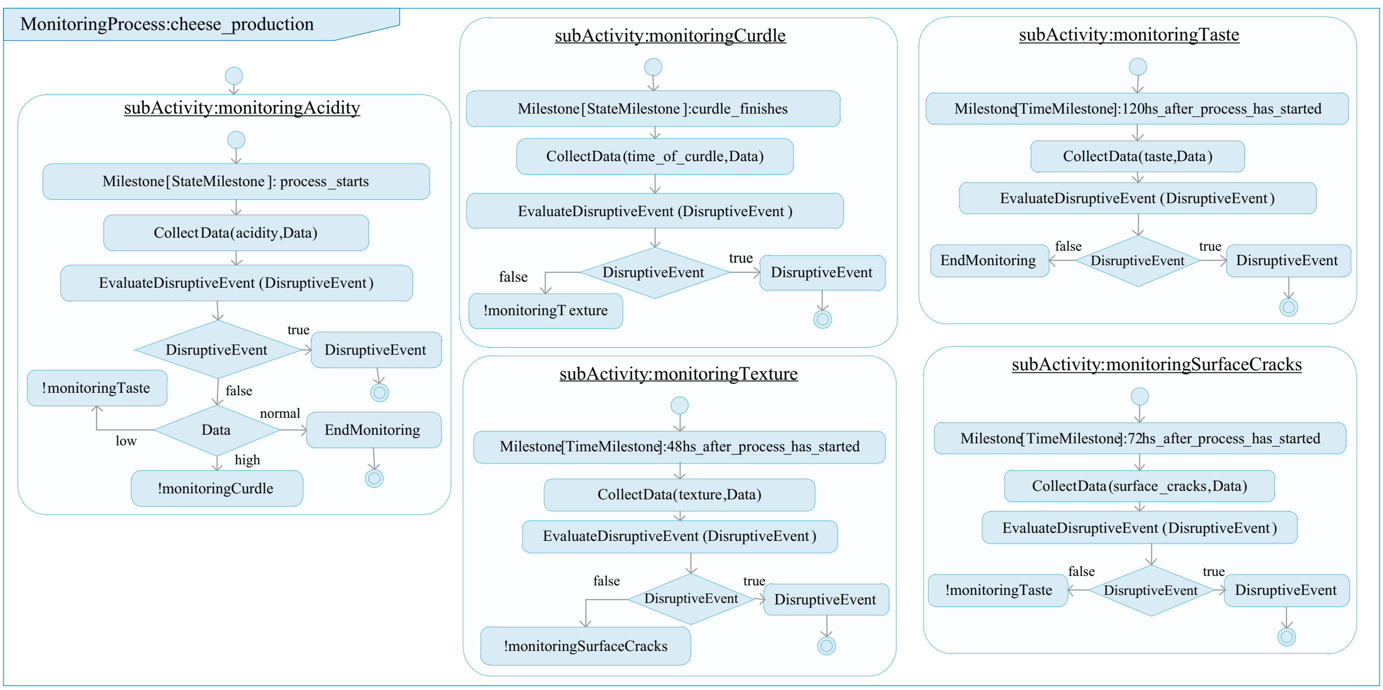


Fig. 12. Cheese production: workflow of the monitoring process.

Table 5
Cheese production: monitoring cases.

Scenario	m	OV_m	Collected value	X:(Q==bad) Y:(Q==good)	Threshold	Result predicted	Actual result [Tn]	Time [H]
1	process_start	acidity	high	P(X)==0.95	0.97	insufficient evidence		
	curdle_finishes	time_of_curdle	low	P(X)==100	0.97	estimated_quantity ==0	0	238
2	process_start	acidity	high	P(X)==0.95	0.97	insufficient evidence		
	curdle_finishes	time_of_curdle	normal	P(X)==0.84	0.97	insufficient evidence		
	48hs_after_process_start	texture	granulated	P(X)==100	0.97	estimated_quantity ==0	0	192
3	process_start	acidity	high	P(X)==0.95	0.97	insufficient evidence		
	curdle_finishes	time_of_curdle	normal	P(X)==0.84	0.97	insufficient evidence		
	48hs_after_process_start	texture	no_granulated	P(X)==0.60	0.97	insufficient evidence		
	120hs_after_process_start	taste	bitter	P(X)==100	0.97	estimated_quantity ==0	0	120
4	process_start	acidity	high	P(X)==0.95	0.97	insufficient evidence		
	curdle_finishes	time_of_curdle	normal	P(X)==0.84	0.97	insufficient evidence		
	48hs_after_process_start	texture	no_granulated	P(X)==0.60	0.97	insufficient evidence		
	120hs_after_process_start	taste	good	P(Y)==100	0.97	estimated_quantity ==1	1	120
5	process_start	acidity	low	P(X)==0.60	0.55	estimated_quantity ==0	1	240
6	process_start	acidity	low	P(X)==0.60	0.97	insufficient evidence		
	120hs_after_process_start	taste	good	P(Y)==100	0.97	estimated_quantity ==1	1	120

could occur ($P(x) > threshold$). The seventh and eighth columns show predicted and current results, and the ninth column presents the number of hours left for the end of the cheese production process when the result was predicted by AOpm.

An analysis of scenarios summarised in Table 5 allows concluding:

- From values of observable variables collected in milestone process_start, in Scenario 6, the next milestone evaluated by AgOpm was 120hs_after_process_start, which was different from milestone curdle_finishes evaluated by AgOpm in Scenarios 1–4. This shows that AgOpm can select different plans depending on the collected value of observable variables in milestones.

- When threshold value was high enough, AgOpm predicted right results, but when the threshold value was not high enough, AgOpm predicted wrong results (Scenario 5).
- High value of threshold prevented AgOpm from predicting wrong results, but reduced the ability to early anticipate disruptive events. In Scenario 1, for example, AgOpm was able to predict the result 238 hours before the end of the cheese production process, but in Scenario 3, AgOpm was able to predict the result only 120 hours before the end of the cheese production process.

6. Conclusion and future work

Agent technology made it possible to profit from the concurrent programming for designing a subsystem able to manage the distributed nature of disruptive events, generate goal-oriented plans from monitoring models, and address the complexity of monitoring services by structuring their functionalities as a set of autonomous entities. Using the conceptual and engineering background provided by agent technology, operations for providing monitoring services and protocols carried out when an enterprise hires the monitoring service were implemented.

A novel aspect of this approach is the tailored monitoring process that defines the planning strategy of agents responsible for monitoring orders and resources is dynamically generated at execution time. For performing this task, the monitoring subsystem uses a model-driven development approach to generate the tailored monitoring process from a monitoring model defined by using the abstract language provided by a reference model.

Another novelty in this approach is the fact that Monitoring Subsystem can decide on the most appropriate tool to perform the monitoring task, keeping implementation details hidden to the hiring enterprise. For carrying out this task, the monitoring subsystem can be provided with appropriate sets of transformation rules that allow it to generate monitoring model implementations for different technological platforms. While the developed prototype only implements Transformation T1 Tool able to transform the static view of the monitoring model into evaluation functions defined as Bayesian Networks, new transformation T1 tools can be included into MAMS.

JaCaMo platform allowed implementing web services in a native way. Agents were programmed through a set of beliefs, rules, goals, and plans able to be automatically tailored for monitoring schedules of different supply processes in a transparent and concurrent way. The multi-agent environment was specified by artefacts and workspaces. Artefacts provided tools as first-class entities to perform model-to-model and model-to-code transformation, executing inference processes to predict disruptive events, and storing/retrieving previously generated monitoring models. Workspaces distributed the subsystem in containers, grouping artefacts and agents to be executed in appropriate processing nodes.

Finally, a case study performed in a real environment allowed illustrating the operation of the monitoring subsystem to proactively detect and inform probable disruptive events in a factory. The ability of MAMS to anticipate disruptive events was analysed through a set of scenarios, which demonstrated satisfactory results in predicting disruptive events well before schedule execution ends.

In future work, new algorithms to predict disruptive events will be developed. The aim is to have a repository of algorithms for predicting disruptive events, from where they can be automatically selected depending on the supply process to be monitored. This selection must be transparent for hiring enterprises which define high level and platform independent monitoring models as instances of reference model.

Acknowledgements

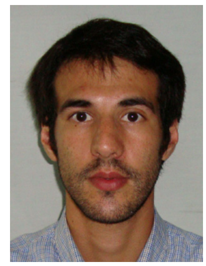
Financial support for this research study was granted by ''Desarrollo de Herramientas de Software para Implementar procesos Colaborativos'' PIP 11220110100705 of Consejo Nacional de Investigaciones Científicas y Técnicas and ''Modelos de negociación de holguras para reparar un programa de abastecimiento en ejecución'' 25/o146 of Universidad Tecnológica Nacional.

References

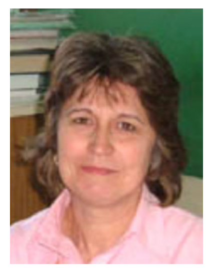
[1] P.R. Kleindorfer, G.H. Saad, Managing disruption risks in supply chains, Production and Operations Management 14 (2005) 53–68.
[2] H. Lee, V. Padmanabhan, S. Whang, The bullwhip effect in supply chains, Sloan Management Review 38 (1997) 93–102.
[3] N. Radjou, L. Orlov, T. Nakashima, Adapting to supply network change, in: The TechStrategy Report, 2002.
[4] H. Van Landeghem, H. Vanmaele, Robust planning: a new paradigm for demand chain planning, Journal of Operations Management 20 (2002) 769–783.
[5] R. Zimmermann, Agent-based supply network event management, in: Whitestein Series in Software Agent Technologies, 2006.
[6] A. Guarnaschelli, O. Chiotti, E. Salomone, Service oriented approach for autonomous exception management in supply chains, in: Software Services for e-World, 2010, 292–303.
[7] Y. Li, M. Kramer, A. Beulens, J. van der Vorst, A framework for early warning and proactive control systems in food supply chain networks, Computers in Industry 61 (2010) 852–862, Trends and Challenges in Production and Supply Chain Management.
[8] A. Winkelmann, S. Fleischer, S. Herwig, J. Becker, A conceptual modeling approach for supply chain event management, in: Proceedings of the 17th European Conference on Information Systems (ECIS 2009, 2009.
[9] M. Bansal, A. Adhitya, R. Srinivasan, I. Karimi, An online decision support framework for managing abnormal supply chain events, in: L. Puigjaner, A. Espuña (Eds.), European Symposium on Computer-Aided Process Engineering, volume 20 of Computer Aided Chemical Engineering, Elsevier, 2005, pp. 985–990.
[10] R. Liu, A. Kumar, W. van der Aalst, A formal modeling approach for supply chain event management, Decision Support Systems 43 (2007) 761–778, Integrated Decision Support.
[11] E. Oztemel, E.K. Tekez, A general framework of a reference model for intelligent integrated manufacturing systems (remims), Engineering Applications of Artificial Intelligence 22 (2009) 855–864, Artificial Intelligence Techniques for Supply Chain Management.
[12] I. Mahdavi, B. Shirazi, N. Ghorbani, N. Sahebjamnia, IMAQCS: design and implementation of an intelligent multi-agent system for monitoring and controlling quality of cement production processes, Computers in Industry 64 (2013) 290–298.
[13] S. Darmoul, H. Pierreval, S. Hajri-Gabouj, Handling disruptions in manufacturing systems: an immune perspective, Engineering Applications of Artificial Intelligence 26 (2013) 110–121.
[14] K. Kurbel, D. Schreber, Agent-based diagnostics in supply networks, Issues in Information Systems 8 (2007).
[15] K. Kwangmyeong, C. Injun, Active data acquisition for proactive exception handling in manufacturing, International Journal of Advanced Manufacturing Technology 43 (2009) 365–378.
[16] R. Zimmermann, S. Käs, R. Butscher, F. Bodendorf, An ontology for agent-based monitoring of fulfillment processes, in: Whitestein Series in Software Agent Technologies and Autonomic Computing, 2005, 323–345.
[17] H.H. Xu, A web-based system for proactive management of supply exceptions, Journal of Manufacturing Systems 29 (2010) 91–101.
[18] L. Ribeiro, J. Barata, A. Colombo, Supporting agile supply chains using a service-oriented shop floor, Engineering Applications of Artificial Intelligence 22 (2009) 950–960, Artificial Intelligence Techniques for Supply Chain Management.
[19] G. Heinecke, J. Koeber, A. Kunz, S. Lamparter, Modeling the basic cause-effect relationship between supply chain events and performance, in: 3rd International Conference on Dynamics in Logistics, 2012.
[20] N. Szirbik, A. Aerts, H. Wortmann, D. Hammer, J. Goossenaerts, Mediating negotiations in a virtual enterprise via mobile agents, in: Proceedings ofAcademia/Industry Working Conference on Research Challenges, IEEE, (2000), pp. 237–242.
[21] M. Kärkkä inen, T. Ala-Risku, K. Främling, Integrating material and information flows using a distributed peer-to-peer information system, in: Collaborative Systems for Production Management, Springer, 2003, pp. 305–319.
[22] E. Fernández, E. Salomone, O. Chiotti, A model driven development approach based on a reference model for predicting disruptive events in a supply process, Computers in Industry 63 (2012) 482–499.
[23] M. Papazoglou, W. Van Den Heuvel, Service oriented architectures: approaches, technologies and research issues, VLDB Journal 16 (2007) 389–415.
[24] T. Erl, SOA: Principles of Service Design, Prentice Hall, 2008.
[25] A. Guarnaschelli, E. Fernández, E. Salomone, O. Chiotti, A service-oriented approach to collaborative management of disruptive events in supply chains, International Journal of Innovative Computing, Information and Control (IJICIC) 8 (2011), Special Issue: On intelligent and Innovative Computing in Business Process Management.

[26] A. Ricci, C. Buda, N. Zaghini, An agent-oriented programming model for soa & web services, in: 5th IEEE International Conference on Industrial Informatics, volume 2, IEEE, (2007), pp. 1059–1064.
[27] D. Tapia, R. Alonso, C. Zato, O. Gil, F. De la Prieta, Analysis and design of a soa-based multi-agent architecture, Trends in Practical Applications of Agents and Multiagent Systems (2010) 183–190.
[28] A. Cauvin, A. Ferrarini, E. Tranvouez, Disruption management in distributed enterprises: a multi-agent modelling and simulation of cooperative recovery behaviours, International Journal of Production Economics 122 (2009) 429–439.
[29] A. Guarnaschelli, O. Chiotti, H.E. Salomone, An approach based on constraint satisfaction problems to disruptive event management in supply chains, International Journal of Production Economics 144 (2013) 223–242.
[30] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Pearson Education, 2005.
[31] F. Jensen, N. Thomas, Bayesian Networks and Decision Graphs, Springer Science and Business Media, 2007.
[32] M. Wooldridge, An Introduction to Multiagent Systems, Wiley, 2009.
[33] M. d'Inverno, M. Luck, Understanding Agent Systems, Springer, 2004.
[34] O. Boissier, R.H. Bordini, J.F. Hübner, A. Ricci, A. Santi, Multi-agent oriented programming with JaCaMo, Science of Computer Programming 78 (2013) 747–761.
[35] C. Toledo, R. Bordini, O. Chiotti, M. Galli, Developing a knowledge management multi-agent system using JaCaMo, in: L. Dennis, O. Boissier, R. Bordini (Eds.), Programming Multi-Agent Systems, volume 7217 of Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, 2012, pp. 41–57.
[36] A. Omicini, A. Ricci, M. Viroli, Artifacts in the A&A meta-model for multi-agent systems, Autonomous Agents and Multi-Agent Systems 17 (2008) 432–456.
[37] J. Hübner, R. Bordini, G. Picard, Using Jason, MOISE+, and CArtAgO to develop a team of cowboys, in: K. Hindriks, A. Pokahr, S. Sardina (Eds.), Programming Multi-Agent Systems, volume 5442 of Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, 2009, pp. 238–242.
[38] D. Weyns, A. Omicini, J. Odell, Environment as a first class abstraction in multiagent systems, Autonomous Agents and Multi-Agent Systems 14 (2007) 5–30.
[39] T. Erl, A. Karmarkar, P. Walmsley, H. Haas, U. Yalcinalp, K. Liu, D. Orchard, A. Tost, J. Pasley, Web Service Contract Design and Versioning for SOA, Prentice Hall, New York, 2009.
[40] F. Curbera, F. Leymann, T. Storey, D. Ferguson, S. Weerawarana, Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More, Prentice Hall PTR, Englewood Cliffs, 2005.
[41] A. Pokahr, L. Braubach, W. Lamersdorf, Jadex: a BDI reasoning engine, in: Multi-Agent Programming, Springer, 2005, pp. 149–174.
[42] P. Busetta, N. Howden, R. Rönnquist, A. Hodgson, Structuring BDI agents in functional clusters, in: N.R. Jennings, Y. Lespérance (Eds.), Intelligent Agents VI. Agent Theories, Architectures, and Languages, volume 1757 of Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, 2000, pp. 277–289.
[43] R. Bordini, J. Hübner, M. Wooldridge, Programming Multi-Agent Systems in AgentSpeak Using Jason, Wiley, 2007.
[44] A. Ricci, M. Piunti, M. Viroli, Environment programming in multi-agent systems: an artifact-based perspective, Autonomous Agents and Multi-Agent Systems (2010) 1–35.
[45] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, Web Services Description Language (WSDL) Version 1.2. W3C Working Draft, Technical Report, W3C, 2003.
[46] M. Piunti, A. Santi, A. Ricci, Programming SOA/WS systems with BDI agents and artifact-based environments, in: Proceedings of MALLOW Federated Workshops on Agents, Web Services and Ontologies, Integrated Methodologies (AWESOME-09), Torino, 2009.
[47] D. Steinberg, F. Budinsky, E. Merks, M. Paternostro, EMF: Eclipse Modeling Framework, Addison-Wesley, 2008.
[48] R. Gronback, Eclipse Modeling Project: A Domain-Specific Language Toolkit, Addison-Wesley Professional, 2009.

Erica Fernández received a PhD in Information System Engineering from UTN (Universidad Tecnológica Nacional). She is an assistant professor of the UTN since 2007. Her research mainly focuses on information technology applied to supply chain management and cost management.



Carlos M. Toledo received a PhD in Information System Engineering from UTN (Universidad Tecnológica Nacional) and is a research fellow from the CONICET. He is an assistant professor of Information Systems Engineering at UTN and he has been working at INGAR(CONICET-UTN) since 2009. His current research interests include knowledge management, ontologies, semantic web, and multiagent systems.



María R. Galli received a PhD from Universidad Nacional del Litoral in 1989. She has been working at CONICET as a researcher since 1992. She is a professor of Information Systems Engineering at since 1988. Her current research interests include knowledge management, ontologies, semantic web, and multi-agent systems

Enrique Salomone has obtained his PhD from Universidad Nacional del Litoral (UNL) in 1993. He is a professor at UNL and member of the CONICET since 1997. His current research interest include information technology for supporting for supply chain management.

Omar Chiotti received a PhD from Universidad Nacional del Litoral in 1989. He has been working at CONICET as a researcher since 1991. He is a professor of Information Systems Engineering at Universidad Tecnológica Nacional since 1986. His current research interests include e-collaboration, knowledge management, and multi-agent systems.