Contents lists available at SciVerse ScienceDirect

Computers in Industry

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

A model driven development approach based on a reference model for predicting disruptive events in a supply process

Erica Fernández [a,*], Enrique Salomone [a], Omar Chiotti [a,b]

[a] INGAR-CONICET, Avellaneda 3657, Santa Fe 3000, Argentina
[b] CIDISI-UTN FRSF, Lavaisse 610, Santa Fe 3000, Argentina

ARTICLE INFO

Article history:
Received 22 December 2010
Received in revised form 28 September 2011
Accepted 7 February 2012
Available online 13 March 2012

Keywords:
SCEM systems
Disruptive event
Event management
Monitoring system

ABSTRACT

Due to the impossibility of predicting with certainty the occurrence of disruptive events, buffers defined to obtain a robust schedule could not absorb all the changes. Then, local modifications of the schedule are usually performed to avoid a new planning task. For this task, obtaining disruptive event information in advance can help to make better decisions. As a result, ability to predict disruptive events that affect the execution of the supply process an order represents is required. With the objective of satisfying this requirement, this work proposes a model driven development approach based on a reference model to automate the generation of the monitoring model of a supply process able to anticipate the occurrence of a disruptive event by monitoring variables that can explain it.

The approach proposes both a reference model to represent the monitoring model independently of the implementation platform, and a specific model to represent the monitoring model with the particular language of the implementation platform. An engine based on transformation rules allows automating the generation of a platform dependent monitoring model from an instance of a platform independent metamodel. The monitoring component of a SCEM system has been developed, which implements the transformation engine as a Bayesian Network model, and uses an appropriate tool to execute it. For an empirical validation of the model three case studies are presented.

1. Introduction

Logistic planning systems, as enterprise resource planning systems, distribution requirements planning systems and advanced planning and scheduling systems, generate as output a schedule of both orders and critical resources, where each order represents a supply process (production or distribution) [1,2]. As a result of the uncertainty inherent in any supply process [3], a schedule may be affected by disruptive events [4] that could produce negative effects that propagate throughout the supply chain [5,6].

In this work, a *disruptive event* is defined as a significant change in planned values of the order attributes, which could be advance or delay in the start or end date, and changes in the amount specified. They can be produced by changes in the planned values of the resource attributes associated with an order and/or in the expected value of the environmental variables. For example, equipment breakdowns, breakage of materials, change of material specification, weather conditions, port congestion, etc.

The paradigm of robust planning recognises and explores the inherent uncertainty in supply chains [7], defining levels and locations of buffers (material, capacity and time) to achieve a robust schedule most likely to remain stable during implementation. These buffers can absorb some variability that may occur during the execution of a supply process an order represents. In the absence of robustness, any deviation can affect the schedule, so that a new planning task is necessary. This can be costly and time consuming, since all the business units involved in the supply process should agree on a new plan.

The benefits of having a robust schedule are indisputable, but despite the effort in term of resource buffers done to provide robustness, operation managers know that it is not easy to effectively use these buffers in a systematic way maintaining the execution adherence to planned targets. The objective is to repair a schedule through limited and localised modifications. To perform this task, obtaining disruptive event information in advance can help to make better decisions. As a result, ability to predict disruptive events that will affect the execution of an order is required.

A model to predict disruptive events has to be able to get information about internal and external changes that can affect a

* Corresponding author at: INGAR-CONICET, Avellaneda 3657, S3002GJC Santa Fe, Argentina. Tel.: +54 342 4 535 568; fax: +54 342 4 553 439.
E-mail addresses: ericafernandez@santafe-conicet.gov.ar (E. Fernández), salomone@santafe-conicet.gov.ar (E. Salomone), chiotti@santafe-conicet.gov.ar (O. Chiotti).

0166-3615/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.compind.2012.02.002

supply process execution, use it to infer changes that will affect order specification, and predict a disruptive event that can affect the schedule execution when it has enough evidence that this will occur.

The objective of this paper is to present an alternative to the approaches of reactive monitoring of an order, trying to anticipate the occurrence of disruptive events by monitoring variables that can explain (predict) them. To this aim, due to each supply process associated with a production or distribution order requires a particular prediction function, this work proposes a model driven development approach based on a reference model to automate the generation of the monitoring model able to perform the prediction function each supply process requires. The model driven approach for software development can increase development productivity and quality by describing important aspects of a solution through user-friendly abstractions [8].

The approach proposes both a *reference model* as a platform independent metamodel, which allows describing the monitoring model without any knowledge of the final implementation platform; and a *specific model* as a platform specific metamodel, which allows describing the monitoring model with full knowledge of the final implementation platform. Based on appropriate transformation rules an engine that allows automatically generating an instance of a specific model from an instance of the reference model has been developed. The specific model instance defines a platform dependent monitoring model that can be executed using an appropriate tool. Based on the reference model and on a specific model, particularly a Bayesian Network model, the monitoring component of a supply chain event management (SCEM) system has been developed.

Based on the abstract language provided by the reference model, the user could represent the monitoring process of a supply process without the need of known the specific language of the implementation technology. The monitoring model represented in terms of the reference model can be transformer into different technological languages automatically, as needed or available tools.

This paper is organised in the following way. Section 2 presents a component-based architecture of a SCEM system. Section 3 presents a model driven development approach for automating the generation of a monitoring model for predicting disruptive events. Section 4 presents an empirical validation of the approach through three case studies. Section 5 discusses related works, and Section 6 presents conclusions and future work.

2. Architecture of the SCEM system

The supply chain event management (SCEM) is defined as a business process whereby significant disruptive events are recognised in time, reactive actions are quickly triggered, the flow of information and material are adjusted and key employees are immediately notified [9,10]. Information systems that implement SCEM processes, called SCEM systems, have as objective to manage the changes caused by disruptive events, allowing short-term logistical decisions, avoiding a new planning task [11].

Fig. 1 graphically represents a component-based architecture of the SCEM system composed of the *Monitoring System*, which performs the monitoring of resources attributes and environmental variables function, and disruptive event notification function; and the *Control System*, which performs the simulation, control and exception notification functions.

In this work, a *schedule* is defined as a set of orders, where each order represents a supply process (production or distribution) that assigns materials to a place, states the required resources, the time period during which each resource is required and its required capacity. The execution of a schedule implies performing the operations defined in the supply process each order represents. As it has been defined in the previous section, a disruptive event will imply a change in the amount specified in the order and/or in the time in which the order must be fulfilled.

When a disruptive event cannot be absorbed by the buffers of a schedule, an exception takes place. In this work an *exception* is defined as a deviation from the schedule that prevents the fulfilment of one or more orders that requires a new planning task.

Monitoring system: This component has the ability of generating the model for monitoring a supply process associated with an order for capturing significant changes either in the planed value of resource attributes or in the expected value of environmental variables. Once captured, the monitoring model performs a prediction function to infer changes that could be produced on planned values of order attributes, and analyses if these changes can produce a disruptive event. The prediction function is

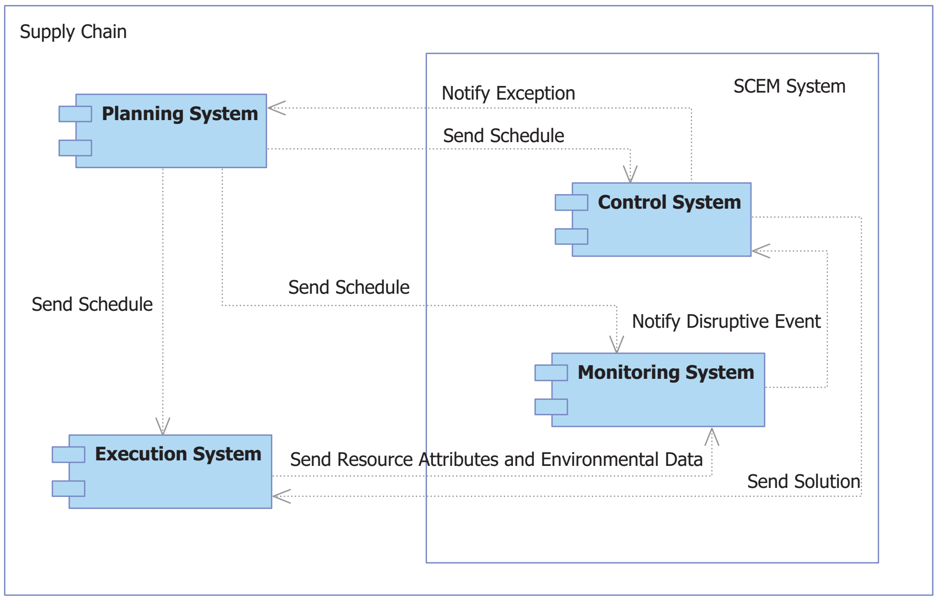


Fig. 1. Model of main components of the SCEM system.

developed using the supply process structure and statistical data. In this way, the monitoring system can proactively notify a disruptive event that can affect an order to the control system. This component is the objective of this work.

Control system: This component receives the disruptive event notification sent by the monitoring system, searches for alternatives as a response to disruptive events, analyses the effect for the subsequent orders and evaluates the impact of each alternative (simulation ability). It analyses the local absorption of the disruptive event as first alternative; if it is feasible, it modifies the schedule and sends the solution to the Execution System; otherwise, it analyses alternatives based on neighbourhood absorption of the disruptive event. If it can find at least one feasible alternative, it modifies the schedule (control ability) and sends the solution to the Execution System; otherwise, it notifies the *exception* to the Planning System for re-planning.

The control system performs these functions using a model based on a reference model for disruptive event management that is able to describe ongoing execution schedules of any kind, systematically capturing the planned buffer on resources and orders in a way that is suitable for analysing feasibility in presence of disruptive events. From an instance of this reference model a constraint satisfaction problem (CSP) for feasibility check and restoration can be automatically derived through a model-to-model transformation. Based on this CSP, a mechanism for automatic repair of disrupted supply processes is derived. This mechanism is able to detect whether a disruptive event generates a significant disruption in the schedule, and if so, it searches for a feasible solution using explicit and implicit (given by the interaction between orders and resources) planned buffers. This solution reduces the impact and propagation of the schedule disruption. This component is beyond the scope of this work.

3. Model driven development approach for predicting disruptive events

To automate the process of generating a monitoring model able to perform a prediction function to infer if observed changes during the execution of a supply process could produce a disruptive event, in this work we propose a model driven development approach based on the principles of the Model Driven Architecture framework [12]. The approach proposes a *reference model* as a platform independent metamodel, which allows describing the monitoring model of a supply process without any knowledge of the final implementation platform. The reference model allows representing all entities required for monitoring a supply process associated with an order.

The approach also proposes to define a *specific model* as a platform specific metamodel, which allows describing the monitoring model of a supply process with full knowledge of the final implementation platform. The specific model allows representing the prediction function of disruptive events of a monitoring model based on a particular modelling language such as Bayesian Networks, Petri Nets, decision trees, etc.

A *transformation* defines how a source metamodel can be transformed into a target meta-model. Through a transformation tool, an instance of a specific model (target) can be automatically generated from an instance of the reference model (source). The specific model instance defines a platform dependent monitoring model able to infer changes in the planned values of any attribute i of an order, ΔO_i, through a causal relationship function that relates it with changes in the planned values of any attribute j of any resource k, $\Delta R_{j,k}$, and changes with regard to the expected of any environmental variable h, ΔE_h. This causal relationship function

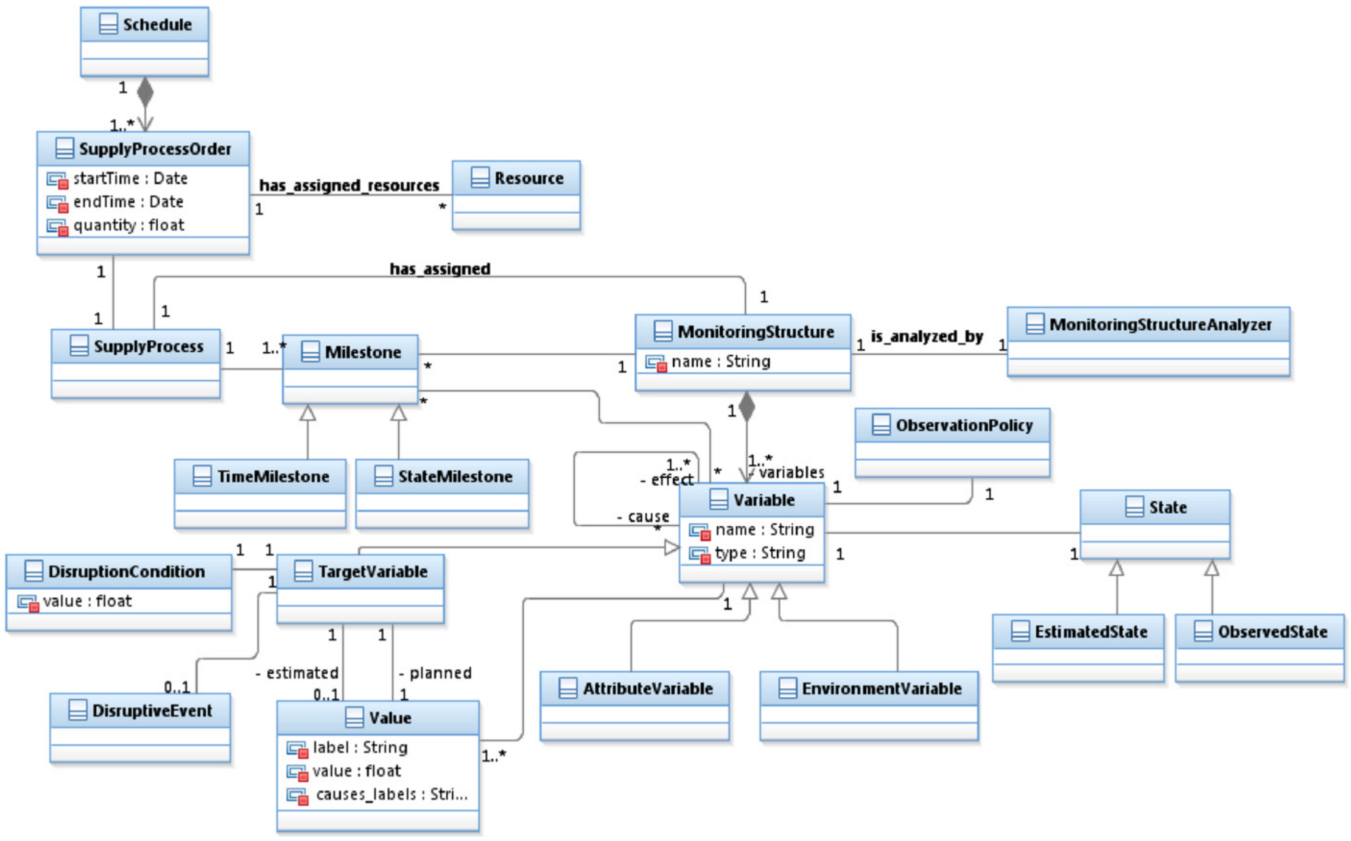


Fig. 2. Reference model.

can be defined as follows:

$$\Delta O_i = f(\Delta R_{jk}, \Delta R_{jk}(\Delta E_h, \text{ for all } h) \text{ for all } j, k) \quad (1)$$

where

$\Delta R_{j,k}$ = change of the attribute j of the resource k
ΔE_h = change of the value of the environment variable h

The specific monitoring model will notify (report) a disruptive event when it has enough evidence that it will occur. This model can be defined in the specification language a tool for processing it requires. Usually XML specifications are required [13].

In this work we present a transformation tool to automatically derive from an instance of the reference model a specific monitoring model based on a Bayesian Network. The reference model, the Bayesian Network model, the relationships between both models and the transformation rules are described following.

3.1. Reference model

The UML [14] class diagram in Fig. 2 presents the reference model. Logistic planning systems generate as output a *Schedule* of production and distribution orders. A *SupplyProcessOrder* requires executing a *SupplyProcess* (for example, freight transport, cheese production in dairy industry, castings production in moulding industry). Its main attributes are: start time (*startTime*), end time (*endTime*) and material amount (*quantity*). The first two attributes define the time period during which the supply process (production or distribution) will be performed. An order defines the resources (material, transport equipment, machinery, etc.) (*Resource)* required to be executed.

Each supply process has a set of milestones associated. A *Milestone* defines a supply process state or time where a set of variables will be observed to evaluate the progress of execution of an order. That is, a *Milestone* defines a control point. By default, the monitoring system has an initial milestone (*startTime*) and a last milestone (*endTime*) to evaluate modifications of the planned values of the supply process order attributes. Usually, these two milestones can be enough when the total time (difference between last and initial milestones) of the supply process is of short duration, which does not allow introducing complex cause-effect relationship among variables to improve the predictive capabilities of a disruptive event. In this case, the model tries to predict if a disruptive event may occur in the last milestone, after analysing the values of attributes of resources and environment variables in the initial milestone.

If the total time of the supply process is of long duration the order execution may follow different stages. Different milestones can be defined to identify those stages, which allow the monitoring system to assess if a disruptive event may occur in the last milestone, after analysing the values of resource attributes and environment variables in successive points (milestones) during the progress of an order (execution of the supply process the order represents). In this case, a monitoring network structure based on a *cause_effect* relationship among *Variables* is defined. A variable has two attributes: *name attribute* and *type attribute* (which represents the type of variable). These variables represent *AttributeVariable* or *Environment Variable* affecting a resource or a supply process order specification.

Each *SupplyProcess has_assigned* a *MonitoringStructure* based on a *cause_effect* relationship among *Variables* associated with all milestones, which allows predicting if a disruptive event at the last milestone can occur. The monitoring structure has a set of milestones. A *Milestone* can be a *TimeMilestone* (absolute time or related to another milestone) or a *StateMilestone* (state to be reached). Each *Milestone* defines a point where a set of variables will be observed. Each *Variable* of the monitoring structure has one *State* that can be: *ObservedState* or *EstimatedState*. When the *Variable* is observed and its value is given, the state of the variable is *ObservedState*. When the state is *EstimatedState*, the *Variable* value is estimated from the value of other variables using the *cause_effect* relationship network. To perform this task, the *MonitoringStructure is_analysed_by* a *MonitoringStructureAnalyzer*. A variable has an observation policy. An *ObservationPolicy* defines the mode, the recurrence and the updating time of the observed variable.

The state of a variable can be changed on the same milestone. After the value of a variable whose state was estimated is known (evidence), the variable changes to observed state. The branch of the monitoring structure (cause_effect relationship sub-net) that predicted its value is no longer necessary and can be eliminated.

The *Monitor* is responsible for observing the variables at each *Milestone*. It starts with the initial milestone, gets the value (*Value*) of each observed *Variable* and inserts them to the *Monitoring-Structure*. A *Value* has three attributes: *label attribute*, *value attribute* and *causes_labels attribute* (which represents the labels of the causes associated with the value). The *MonitoringStructure-Analyzer* using the *MonitoringStructure* (cause_effect relations net) evaluates the impact of these variable values on current and next milestones until the last milestone. Particularly, it defines an estimated *Value* for the *TargetVariable*. The *TargetVariable* has a planned *Value* that is an order parameter (for example, amount planned, end time planned, etc.). Following, to predict if a disruptive event can occur, the *Monitor* uses the *TargetVariable* comparing its planned and estimated values. Based on a *DisruptionCondition*, it predicts if a disruptive event can occur, if so, it reports the *DisruptiveEvent* and the monitoring process ends; if not, the monitoring process follows. To this aim, the *Monitor* defines the next *Milestone* where variables have to be observed. The monitoring structure is initially defined for each supply process, but it is dynamically explored each time a milestone is reached. That is, the *Monitor*, depending on the results generated by the *MonitoringStructureAnalyzer*, can extend its monitoring strategy to another milestone including other observed variables, eliminating those that are not necessary or exploring different branches of the monitoring structure.

An instance of this reference model is a self-contained description of the monitoring structure of an order, which can be automatically transformed into a specific model.

3.2. Specific model: Bayesian Network model

The relationship between the value of the attributes of resources, environment variables and the attributes of an order is generally subject to uncertainty. Because of this, to capture and propagate changes based on the structure of the supply process a probabilistic model is required. To this aim, in this section we present a specific model based on a Bayesian Network [15], due to it allows both representing uncertain knowledge and reasoning based on probability theory. In this case Petri Nets could not be used due to their inability to represent probabilities.

Each node of a Bayesian Network is composed of a random variable X, which can take values x_i in a continuous or discrete range. The values are exclusive and exhaustive and they have an associated probability $P(x_i)$. Then, each node is represented by $X : (x_i, P(x_i))$ and direct conditional dependences are the directed edges in the graph.

Formally, we have a joint probability distribution P of the random variables in some set V and directed acyclic graph $G = (V, E)$ where V is a finite, nonempty set whose elements are called nodes and E is a set of ordered pairs of distinct elements of V.

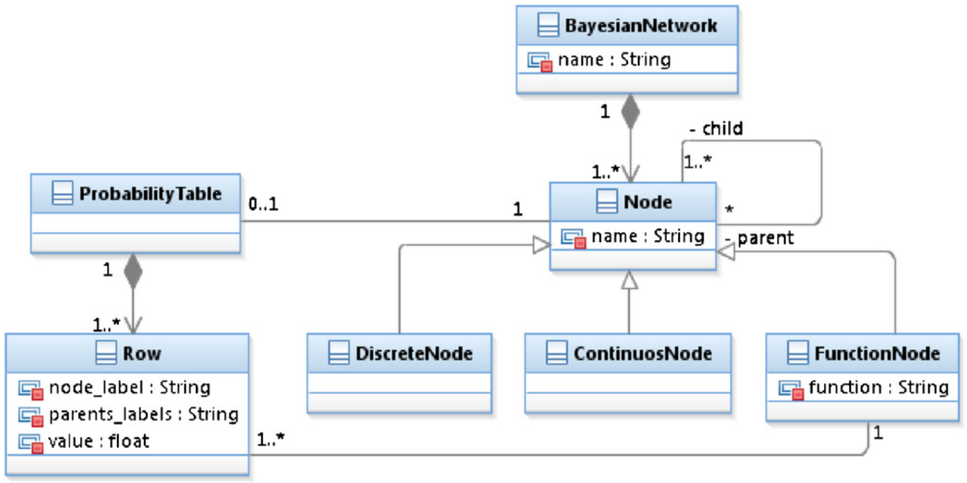


Fig. 3. Bayesian Network model.

Elements of E are called edges. We say that (G, P) satisfies the Markov condition if for each variable $X \in V$, $\{X\}$ is conditionally independent of the set of all its descendents given the set of all its parents. If (G, P) satisfies the Markov condition, then P is equal to the product of its conditional distributions of all nodes given the values of their parents.

$$P(x_1, x_2, \ldots, x_n) = \prod_i P(x_i | Parents(x_i)) \quad (2)$$

The UML class diagram of Fig. 3 represents the specific model. A *BayesianNetwork* is composed of a set of nodes connected with directed links with a probability function (*ProbabilityTable*) attached to each node. The network is a directed acyclic graph. A *Node* can represent either a discrete random variable (*DiscreteNode*) with a finite number of states or a continuous random variable (*ContinuousNode*) with a Gaussian distribution. A Bayesian Network may have a *FunctionNode* representing a real-valued function that depends on (some or all of) the parents of the node. Function nodes are not involved in the inference process and evidence cannot be specified for them, but the function associated with the node can be evaluated using the results of inference. The inference process implies computing the conditional probability for some nodes given the information (evidence) on other nodes.

A *DiscreteNode* or *ContinuousNode* has a *ProbabilityTable* that contains a set of rows (*Row*). A row has the following attributes: *node_label* (which represents the label of a state or the label of a value), *parents_labels* (which represents the labels of its parent states) and *value* (which represents a value). A *FunctionNode* has a set of rows that allow defining the function associated with the node.

An instance of this specific model is a self-contained description of the monitoring structure of a supply process order defined as a Bayesian Network.

3.3. Relationships between both models

The monitoring structure of a supply process associated with an order is created by instantiating the reference model. Then, this instance must be mapped to the specific model, particularly in this case to a Bayesian Network model. To carry out this mapping, each *EnvironmentVariable* or *AttributeVariable* of the monitoring structure has to be represented as a *DiscreteNode* or *ContinuousNode*, and its *ProbabilityTable* has to be defined based on historic data. Fig. 4 shows the relations between the main concepts of both models.

3.4. Transformation between both models

To define the model transformations, we have used ATL (ATLAS Transformation Language) [16], which is inspired by the QVT (Query View Transformation) requirements [17] and builds upon the OCL (Object Constraint Language) formalism [18]. ATL is developed over the Eclipse platform.

In order to carry out model transformations, the MDA (Model Driven Architecture) by the OMG proposes an approach where all metamodels must be conformed to MOF (Meta Object Facility) [19] and QVT. The Eclipse Modelling Framework (EMF) for building Java applications is based on simple model definitions and implements a core subset of the MOF API (Application Programming Interface).

An ATL transformation is composed of rules that define how source model elements are matched to create target model elements. Source and target meta-models are expressed in XMI (XML Metadata Interchange Format) file.

Following, the main transformation rules are presented:

Rule1: This rule implements the logic for transforming the *MonitoringStructure* class to *BayesianNetwork* class.

```
rule MonitoringStructure2BayesianNetwork {
        from mS:ReferenceModel!MonitoringStructure
        to bN:BayesianNetwork!BayesianNetwork (
                name <- mS.name
        )
}
```

Rule2: This rule implements the logic for transforming variables to nodes (*DiscreteNode*, *ContinuousNode* or *FunctionNode*) and for transforming the causes of the variables to the node parents.

The following helper must be defined:

```
helper context ReferenceModel!Variable def : getCauses() : String =
        self.causes->collect(e | e.name)-> iterate(causeName; acc : String = '' |
                acc +
        if acc = '' then
                causeName
        else ' and '
                + causeName
        endif);
```

This helper is called on a variable *(context ReferenceModel!Variable)* and gives a string (: String) and returns all causes associated with a variable.

For each of the three *TypeNode* (*DiscreteNode*, *ContinuousNode* or *FunctionNode*) the following helper and rule must be defined:

```
helper context ReferenceModel!Variable def: isType () : Boolean =
        if self.type = ' TypeNode ' then
                true
        else
                false
        endif;
```

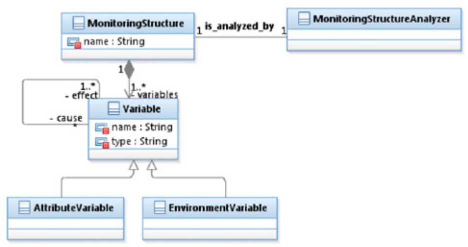
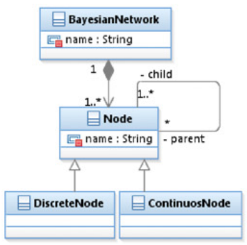
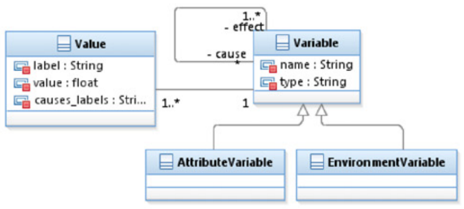
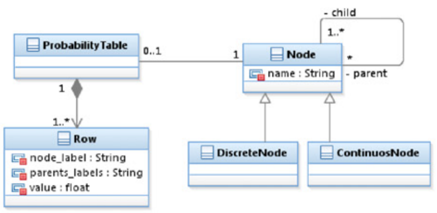
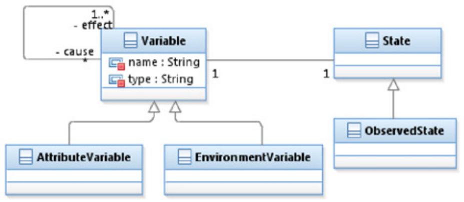
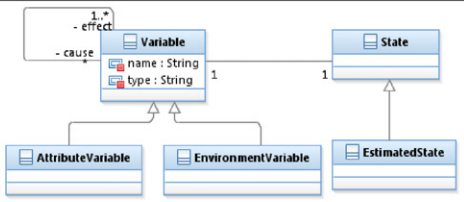
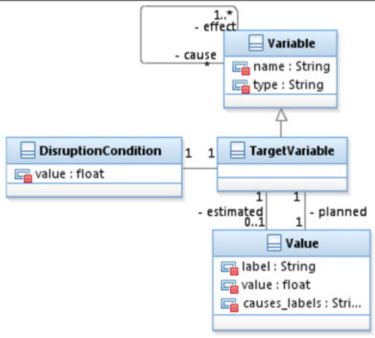
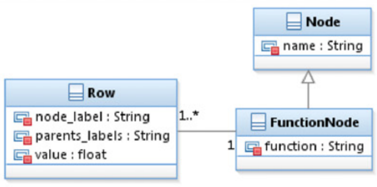
Reference Model	Specific Model: Bayesian Netwok Model
 *MonitoringStructure* *EnvironmentVariable* or *AttributeVariable* A link from *Variable A* to *Variable B* means A causes B *ControlStructureAnaliser*	 *BayesianNetwork* *DiscreteNode* or *ContinuousNode* A link from *Node A* to *Node B* means A is parent of B Inference Process of a Bayesian Network
 A *value* associated with *EnvironmentVariable* or *AttributeVariable* *label attribute* of the *Value* class *causes_labels attribute* of the *Value* class	 A *value* associated with a *Row* *node_label attribute* of the *Row* class *parents_labels attribute* of the *Row* class
	An *EnvironmentVariable* or *AttributeVariable* with *ObservedState* corresponds to an input (evidence) on the respective *Node*.
	An *EnvironmentVariable* or *AttributeVariable* with *EstimatedState*, *X*, corresponds to probabilistic information $X:(x_i, P(x_i))$ on the respective *Node*
 *TargetVariable* *DisruptionCondition* *Planned Value*	 It is implemented as a *FunctionNode* Probabilistic threshold of the *FunctionNode* and it is a value of the function. The value is associated with a row. A value of the function and it is associated with a row.

Fig. 4. Relations between both models.

This helper is called on a variable *(context ReferenceModel!Variable)* and gives a boolean (: Boolean) and tells whether the type of the variable is 'TypeNode' or not.

Finally, the following rule is added to create a 'TypeNode' node with its list of parents from variables:

```
rule Variable2TypeNode {
        from v: ReferenceModel!Variable (v.isType())
        to nD: BayesianNetwork!TypeNode (
                name <- v.name,
                parents<-v.getCauses()
        )
}
```

Rule3: This rule implements the logic for transforming values to rows. If the node is Discrete or Continuous a set of rows define its probability table. If the node is Function a row has its name, the planned value and its parent nodes.

```
rule Value2Row {
        from v: ReferenceModel!Value
        to row: BayesianNetwork!Row (
                node_name <- v.variable.name,
                node_label <- v.label,
                parents_labels <- v.causes_labels,
                value <- v.value
                )
}
```

Rule4: This rule implements the logic for transforming a disruption condition to the probabilistic threshold of the *FunctionNode*.

```
rule DisruptionCondition2Threshold {
        from dC: ReferenceModel!DisruptionCondition
        to row: BayesianNetwork!Row (
                value <- dC.value
                )
}
```

4. Empirical validation of the approach

4.1. Methodology

In order to validate the model driven development approach for predicting disruptive events three supply processes have been studied. A case study offers the opportunity to study phenomenon in their own natural setting where complex links and underlying meanings can be explored [20,21]. It is also appropriate where existing knowledge is limited because it generates in-depth contextual information which may result in a superior level of understanding [22].

4.2. The supply processes

Two production processes of very different regional industries (dairy industry and moulding industry) and a distribution process (marine freight transport) were selected. They are: the *cheese production process*, from dairy industry; the *process of freight transport by sea*, from marine freight transport services; and the *casting production process*, from moulding industry.

4.3. Data collection

Data was collected through visits to a regional dairy industry, a regional moulding industry and a shipping agency. The planned orders and resources were obtained from the production and distribution schedules in the data base of the planning system, and the execution values were obtained from the production and distribution reports in the data base of the execution system, in which disruptive events are registered.

The causal relationship function that relates a disruptive event with changes of planned resource attributes and expected environmental variables are not explicitly defined, then it should be obtained through interviews with the responsible of either production or logistics areas of each enterprise. They reported the following causal relationships for each of the selected supply processes:

In the cheese production process the milk acidity can affect the cheese quality. High acidity can produce sandy cheese, bitter cheese or increase the curdling rate causing surface cracks. Low acidity can produce insipid cheese.

In the process of freight transport by sea the navigation conditions of the ship can be unfavourable due to the weather conditions (storms, winds, etc.). These unfavourable weather conditions are more frequent in the winter season and can produce a delay in the ship arrival to the port. The delay can be increased if in the arrival port or in the intermediate ports there are unfavourable weather conditions or the port is congested. This prevents to carry out unload operations. A ship can carry multiple orders for freight from different ports. Therefore, the ship with an order that requires carrying freight from one port to another may need to go through several intermediate ports to meet the requirements of other orders.

In the casting production process the first stage consists in making the mould for the casting. A mixture of sand, resin and catalyst is prepared. After 2 h, a sample for verifying the traction resistance that the mould will have is taken. If this does not agree with the specification, parameters of the mixture must be changed allowing the mixture to be used only after 2 h. Other inconvenient that could happen after this stage is the break of the mould, implying the restart of the process of production and a delay that depends on the mould to be made. After the casting is unmoulded, it must be inspected visually that, in case it does not pass the tests, the process of production has to restart causing a delay that depends on the mould to be made. When the moulding stage ends different subprocesses can be made over the casting with the objective of improving the physical properties. Finally, a special inspection is done, which in case of not being passed all subprocesses must be restarted, causing a delay that depends on the casting to be produced.

4.4. Case study cheese production process

This case study is described to show the structure of the monitoring model associated with the *cheese_production: SupplyProcess* (cheese production is an object of the class *SupplyProcess*) to predict a disruptive event. The Monitoring System that integrates the SCEM system (Section 2) generates the monitoring model by instantiating the reference model (Section 3.1). In this task the milestones set associated with this supply process is defined. The milestones defined and their observed variables, observation policy and estimated variables are the following:

```
[ Initial Milestone] process_start:StateMilestone
  Variable with ObservedState:acidity; OneTimeObservationPolicy
  Variable with EstimatedState:texture, time_of_curdle, surface_cracks, taste, quality
[ Intermediate Milestone I] curdle_finish:StateMilestone
  Variable with ObservedState:time_of_ curdle; OneTimeObservationPolicy
  Variable with EstimatedState:surface cracks, texture, taste, quality
[ Intermediate Milestone II] time1_after_process_start:TimeMilestone
  Variable with ObservedState:texture; OneTimeObservationPolicy
  Variable with EstimatedState:taste, quality
[ Intermediate Milestone III] time2_after_process_start:TimeMilestone
  Variable with ObservedState:taste; OneTimeObservationPolicy
  Variable with EstimatedState:quality
[ Final Milestone] process_end:StateMilestone
  Variable with ObservedState:quality; OneTimeObservationPolicy
```

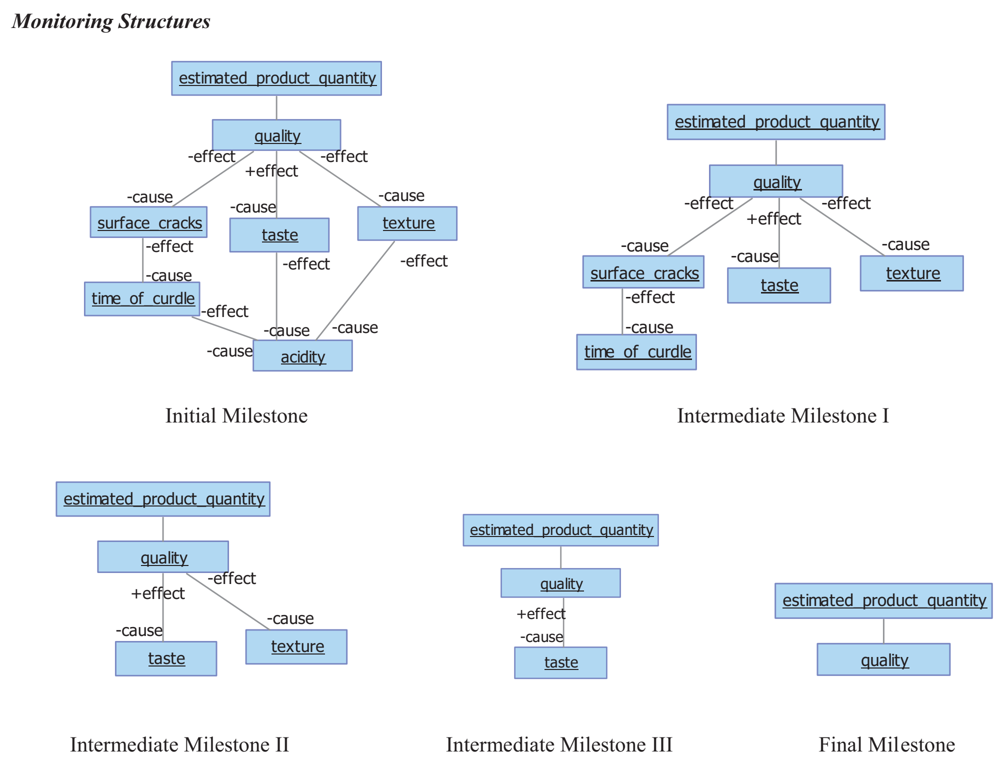


Fig. 5. Milestones set associated with the *cheese_production:SupplyProcess.*

The monitoring structure (instance of the reference model) associated with each milestone will be modified as the process evolves from the initial milestone to the final milestone (Fig. 5).

Each variable of the monitoring structure associated with a milestone is an attribute of a resource or an environment variable. In this example, *acidity:AttributeVariable* and *time_of_curdle:Attribute-Variable* are attributes of the *milk:Resource* and; *texture:Attribute-Variable*, *surface_cracks:AttributeVariable*, *taste:AttributeVariable* and *quality:AttributeVariable* are attributes of the *cheese:Resource*. The target variable is *estimated_product_quantity:TargetVariable.*

Using a transformation tool that implements the transformation rules defined in Section 3.4, the Monitoring System automatically generates from the reference model instance the monitoring model of the *cheese_production:SupplyProcess* based on a Bayesian Network, which is an instance of the Bayesian Network model (Section 3.2). As sample, in Fig. 6 a part of the target model generated from the source model by running the rules 1 and 2, both model specified as XMI file, are shown.

For processing the Bayesian Network model we have selected HUGIN [23], but this tool does not process XMI input file, so the instance of the Bayesian Network model representing the monitoring model of the *cheese_production: SupplyProcess* is transformed from XMI specification to a text file using a transformation engine we developed to this aim. The monitoring model based on a Bayesian Network generated by HUGIN is graphically represented in Fig. 7. It is composed of the following discrete nodes, whose state values are represented in braces: *acidity:DiscreteNode* {normal, low, high}, *time_of_curdle: Discrete-Node* {normal, low, high}, *surface_cracks:DiscreteNode* {no, yes}, *texture:DiscreteNode* {good, bad}, *taste:DiscreteNode* {good, insipid, bitter} and *quality:DiscreteNode* {good, bad}. The function node is *estimated_product_quantity:FunctionNode.*

To illustrate the case, the monitoring process of an order that requires producing 1000 kg of soft cheese (quantity = 1000) is described. To perform the monitoring process, the Monitoring System through the transformation engines starts generating the Bayesian Network model and then, whenever it obtains evidence executes the model by using HUGIN. The total time of the process depends on the kind of cheese to be produced. The total process time for soft cheese is 240 h. The milestones set contains: *process_start:StateMilestone*, *curdle_finish:StateMilestone, 48hs_after_process_start:TimeMilestone, 120hs_after_process_start:TimeMilestone* and *process_end:StateMilestone.*

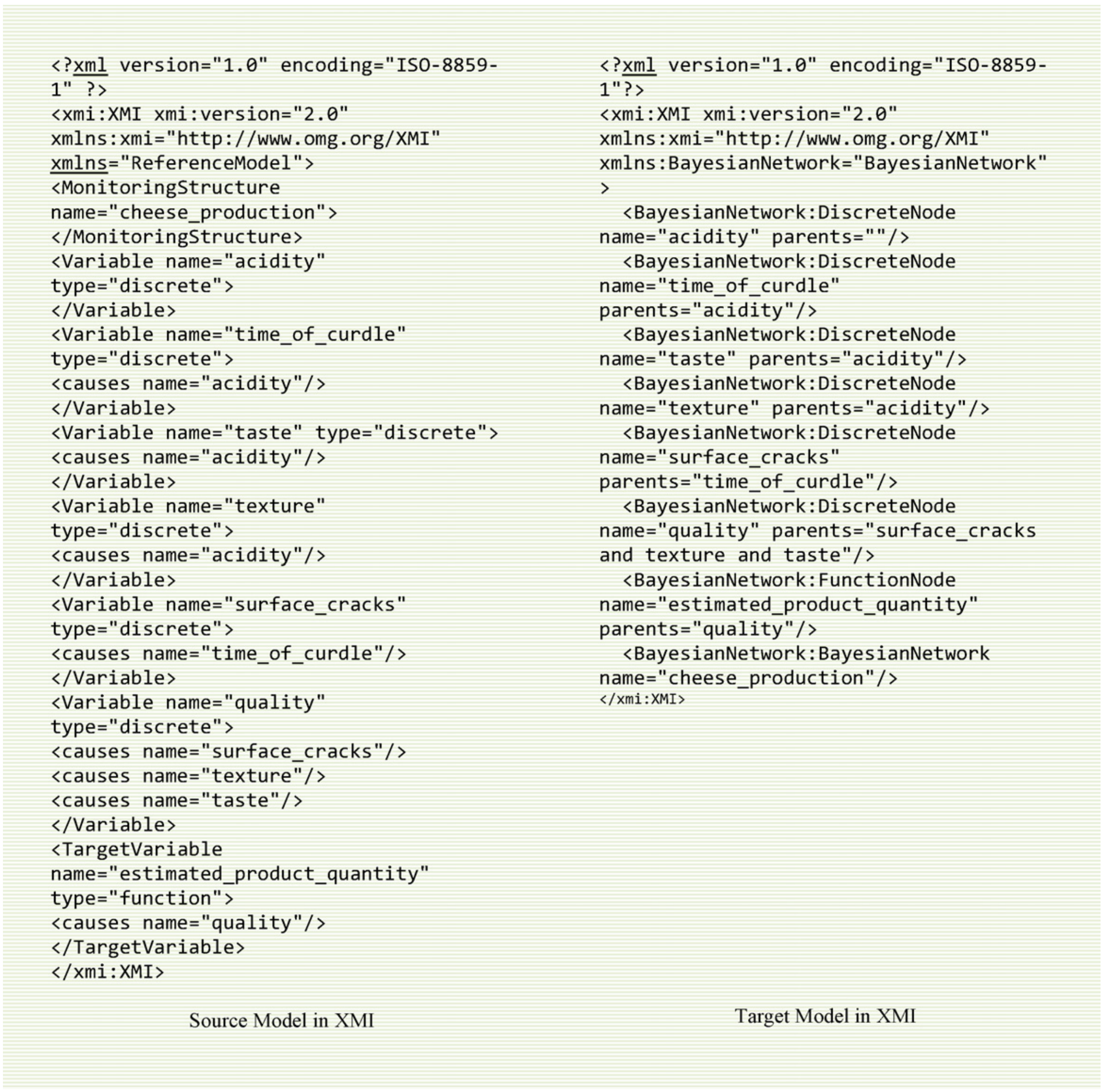
The decision criterion used by the *Monitor* establishes that the product is outside the specification when its probability is greater than a *threshold.* In this example, the *threshold* = *97.0.* If the *threshold* is exceeded, the *Monitor* uses the target variable to predict the occurrence of a disruptive event that is associated with the last milestone. The target variable is *estimated_product_quantity:FunctionNode* and the parent node, which contains the estimated value, is *quality:DiscreteNode*. The function of the target variable is:

```
if (Probability(quality(bad)) ≥ threshold)
then { estimated_product_quantity = 0}
else { estimated_product_quantity = 1000}
```

The *acidity:AttributeVariable* of *milk:Resource* is monitored when the order execution starts, i.e. at the *process_start:StateMilestone.* For each of the possible three values; the *Monitor* explores a different branch in the Bayesian Network. The activity diagram in Fig. 8 represents the decision process carried out by the *Monitor.* While it performs this process it decides the milestones to be evaluated.

Following, as illustrative example, the scenario when acidity is high is described.

process_start:StateMilestone: Fig. 9a shows the Bayesian Network defining the monitoring structure associated with this milestone where a priori probabilities associated with each node are deployed. This Bayesian Network has been obtained by transforming the monitoring structure (instance of the reference model) associated with the initial milestone (Fig. 5). The current



```
<?xml version="1.0" encoding="ISO-8859-
1" ?>
<xmi:XMI xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns="ReferenceModel">
<MonitoringStructure
name="cheese_production">
</MonitoringStructure>
<Variable name="acidity"
type="discrete">
</Variable>
<Variable name="time_of_curdle"
type="discrete">
<causes name="acidity"/>
</Variable>
<Variable name="taste" type="discrete">
<causes name="acidity"/>
</Variable>
<Variable name="texture"
type="discrete">
<causes name="acidity"/>
</Variable>
<Variable name="surface_cracks"
type="discrete">
<causes name="time_of_curdle"/>
</Variable>
<Variable name="quality"
type="discrete">
<causes name="surface_cracks"/>
<causes name="texture"/>
<causes name="taste"/>
</Variable>
<TargetVariable
name="estimated_product_quantity"
type="function">
<causes name="quality"/>
</TargetVariable>
</xmi:XMI>
```

Source Model in XMI

```
<?xml version="1.0" encoding="ISO-8859-
1"?>
<xmi:XMI xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:BayesianNetwork="BayesianNetwork"
>
  <BayesianNetwork:DiscreteNode
name="acidity" parents=""/>
  <BayesianNetwork:DiscreteNode
name="time_of_curdle"
parents="acidity"/>
  <BayesianNetwork:DiscreteNode
name="taste" parents="acidity"/>
  <BayesianNetwork:DiscreteNode
name="texture" parents="acidity"/>
  <BayesianNetwork:DiscreteNode
name="surface_cracks"
parents="time_of_curdle"/>
  <BayesianNetwork:DiscreteNode
name="quality" parents="surface_cracks
and texture and taste"/>
  <BayesianNetwork:FunctionNode
name="estimated_product_quantity"
parents="quality"/>
  <BayesianNetwork:BayesianNetwork
name="cheese_production"/>
</xmi:XMI>
```

Target Model in XMI

Fig. 6. Target model generated from the source model.

statistical data of this supply process indicates that its failure probability is 7.83. When the *Monitor* gets the value of the *Variable* with *StateObserved*, inserts it in the Bayesian Network assigning *acidity:(high, 100)* to the *acidity:Discrete*

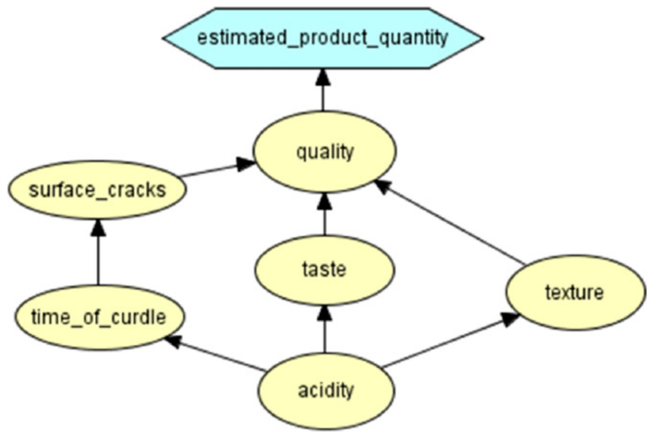


Fig. 7. Monitoring model based on Bayesian Network associated with the *cheese_production:SupplyProcess.*

Node, and performs the inference process by propagating this evidence along the network. Fig. 9b shows posterior probabilities associated with each node. As a result, the *estimated Value* is *quality:(bad, 95.29)*. Because of *Probability(bad) = 95.29 < threshold*, the *Monitor* concludes that there is not enough evidence to predict that a disruptive event will occur so it continues with the monitoring process and defines the next milestone (Fig. 8).

curdle_finish:StateMilestone: The *Monitor* gets the value of the *Variable* with *StateObserved* and if *time_of_curdle* == *low*, it inserts the evidence in the Bayesian Network assigning *time_of_curdle(low, 100)* to the *time_of_curdle:DiscreteNode*, and performs the inference process. Due to the low time of curdle, the cheese will present surface cracks and the *estimated Value* is *quality:(bad,100)*. Because of *Probability(bad) = 100 > threshold*, the Monitor predicts that a disruptive event will occur. The *target variable estimated_product_quantity = 0* indicates that the cheese quantity in specification that will be produced is 0. The disruptive event is reported and the monitoring process ends (Fig. 8). This allows the Monitor to predict the result 238 h before the supply process ends.

Otherwise (Fig. 8), if *time_of_curdle* == *normal*, the *Monitor* gets the value of the *Variable* with *StateObserved*, it inserts the evidence in the Bayesian Network assigning *time_of_curdle(normal, 100)* to

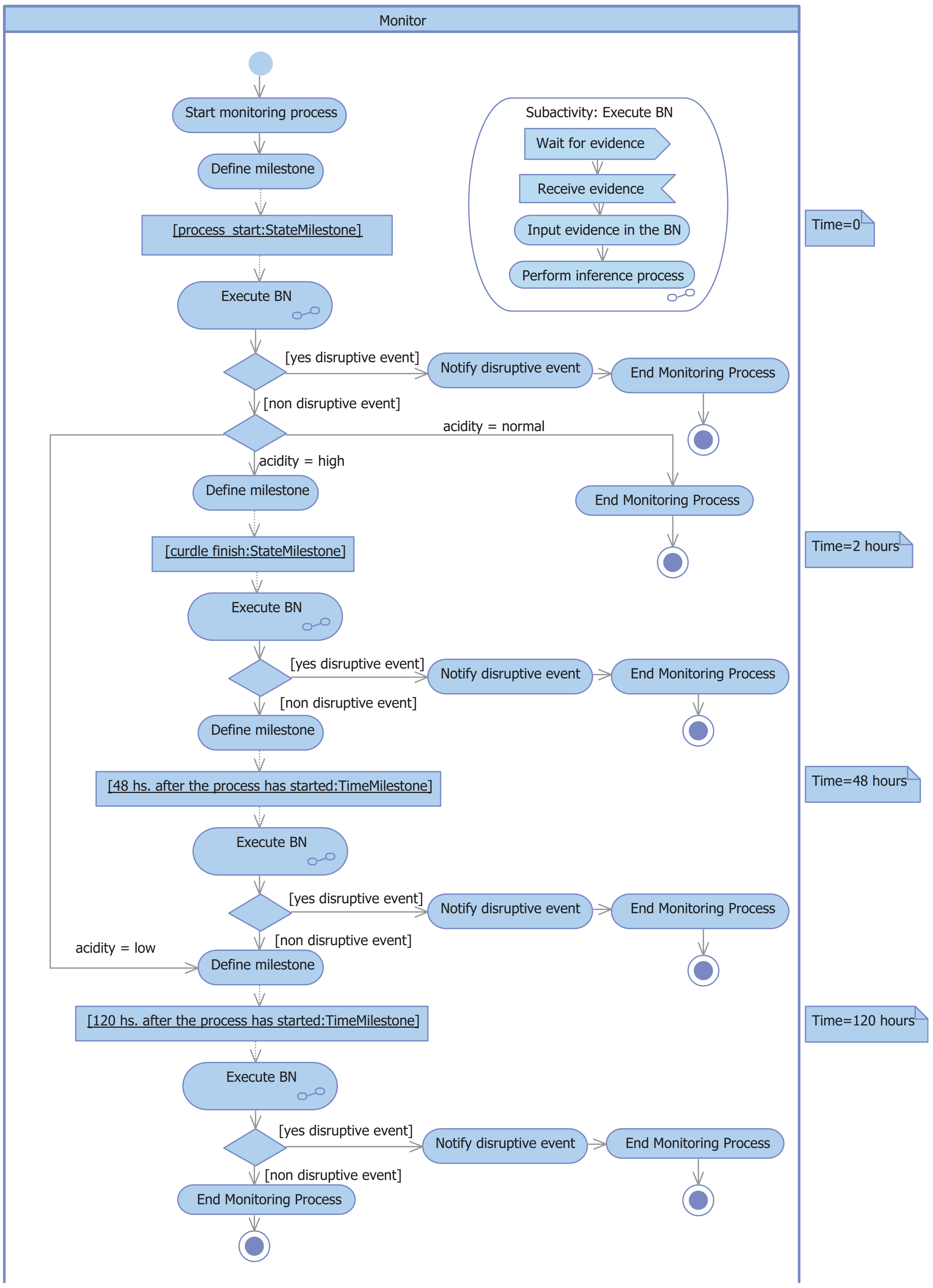


Fig. 8. Activity diagram of the decision process for monitoring the *cheese_production:SupplyProcess.*

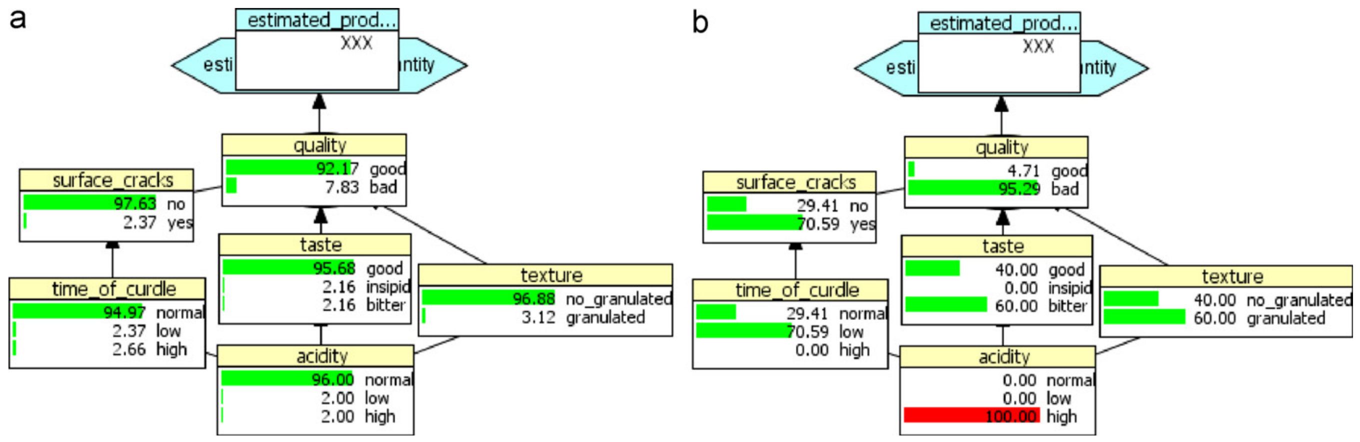


Fig. 9. (a) BN with a priori probabilities. (b) BN with evidence *acidity* == *high.*

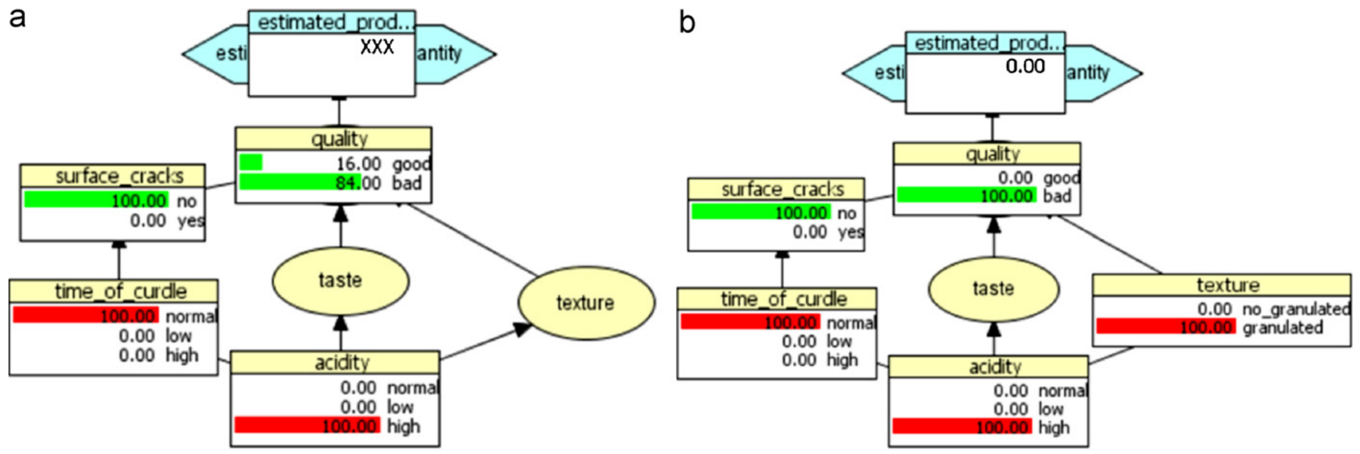


Fig. 10. (a) BN with evidence *time_of_curdle* == *normal*. (b) BN with evidence *texture* == *granulated.*

the *time_of_curdle:DiscreteNode* (Fig. 10a), and performs the inference process. As a result, the *estimated Value* is *quality:(bad,84)*. Because of *Probability(bad)* = *84* < *threshold*, the *Monitor* concludes that there is not enough evidence to predict that a disruptive event will occur so it defines the next milestone (Fig. 8).

48hs_after_process_start:TimeMilestone: the *Monitor* gets the value of the *Variable* with *StateObserved*. If *texture* == *granulated*, the *Monitor* inserts the evidence in the Bayesian Network assigning *texture* (*granulated*, *100*) to the *texture:DiscreteNode* (Fig. 10b), and performs the inference process. The texture of the cheese results granulated. Therefore, it does not satisfy the quality specification, being the *estimated Value quality(bad,100)*. Because *Probability(bad)* = *100* > *threshold*, the *Monitor* predicts that a disruptive event will occur. The *target variable estimated_product_quantity* = 0 shows that the cheese quantity in specification that will be produced is 0. The disruptive event is reported and the monitoring process ends (Fig. 8). This allows the *Monitor* to predict the result 192 h before the supply process ends.

Otherwise (Fig. 8), if *texture* == *no_granulated*, the *Monitor* inserts the evidence in the Bayesian Network assigning *texture (no_granulated, 100)* to the *texture:DiscreteNode* (Fig. 11a), and performs the inference process. As a result, the *estimated Value* is

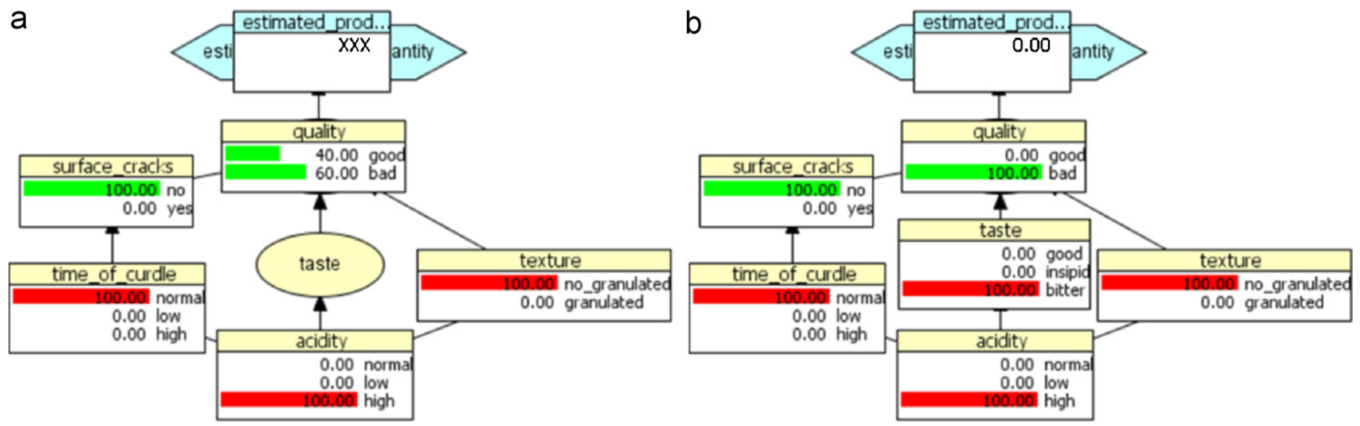


Fig. 11. (a) BN with evidence *texture* == *no_granulated*. (b) BN with evidence *taste* == *bitter.*

quality(bad, 60). Because *Probability(bad)* = 60 < *threshold*, the *Monitor* concludes that there is not enough evidence to predict that a disruptive event will occur so it defines the next milestone (Fig. 8).**120hs_after_process_start:TimeMilestone**: the *Monitor* gets the value of the *Variable* with *StateObserved*. If *taste* == *bitter* the *Monitor* inserts the evidence in the Bayesian Network assigning *taste(bitter, 100)* to the *taste:DiscreteNode* (Fig. 11b), and performs the inference process. The taste of the cheese is bitter. Therefore, it does not satisfy the quality specification, being the *estimated Value quality(bad,100)*. Because *Probability(bad)* = *100* > *threshold*, the *Monitor* predicts that a disruptive event will occur. The target variable *estimated_product_quantity* = *0* shows that the cheese quantity in specification that will be produced is 0. The disruptive event is reported and the monitoring process ends (Fig. 8). This allows the *Monitor* to predict the result 120 h before the supply process ends.

Otherwise (Fig. 8), *taste* == *good*, the *Monitor* inserts the evidence in the Bayesian Network assigning *taste(good, 100)* to the *taste:DiscreteNode* and performs the inference process. As a result, the *estimated Value* is *quality:(good,100)*. The *target variable estimated_product_quantity* = *1000* indicates that the cheese quantity in specification that will be produced is 1000. With this result, the *Monitor* has the certainty that the product will fulfil the specification, so it decides to end the monitoring process 120 h before the supply process ends (Fig. 8).

4.5. Case study marine freight transport process

The monitoring model associated with the *freight_transport_by_sea:SupplyProcess* to predict a disruptive event is briefly described. The milestones defined and their observed variables, observation policy and estimated variables are the following:

```
[ Initial Milestone] port_departs:StateMilestone
  Variable with ObservedState:delay_in_departure;
OneTimeObservationPolicy
  Variable with EstimatedState:season and the variables
of all following milestones.
  Variable with ObservedState: season; OneTimeObserva-
tionPolicy
  Variable with EstimatedState: the variables of all
following milestones.
[ Intermediate Milestone] arrival_to_intermediate_
position:StateMilestone
  Variable with ObservedState:navigation_condition;
RepetitiveObservationPolicy
  Variable with EstimatedState:delay_in_transit,
delay_in_position and the variables of all following
milestones.
  Variable with ObservedState: delay_in_position;
OneTimeObservationPolicy
  Variable with EstimatedState: the variables of all
following milestones.
[ Intermediate Milestone] arrival_to_intermediate_
port:StateMilestone
  Variable with ObservedState:navigation_condition;
RepetitiveObservationPolicy
  Variable with EstimatedState:delay_in_transit,
weather_condition_at_port, delay_in_ intermediate_
port, congestion, and the variables of all following
milestones.
  Variable with ObservedState:weather_condition_at_
port; RepetitiveObservationPolicy
  Variable with EstimatedState: congestion, delay_in_
intermediate_port and the variables of all following
milestones.
  Variable with ObservedState:congestion; OneTimeOb-
servationPolicy
  Variable with EstimatedState:delay_in_intermedia-
te_port and the variables of all following milestones.
```

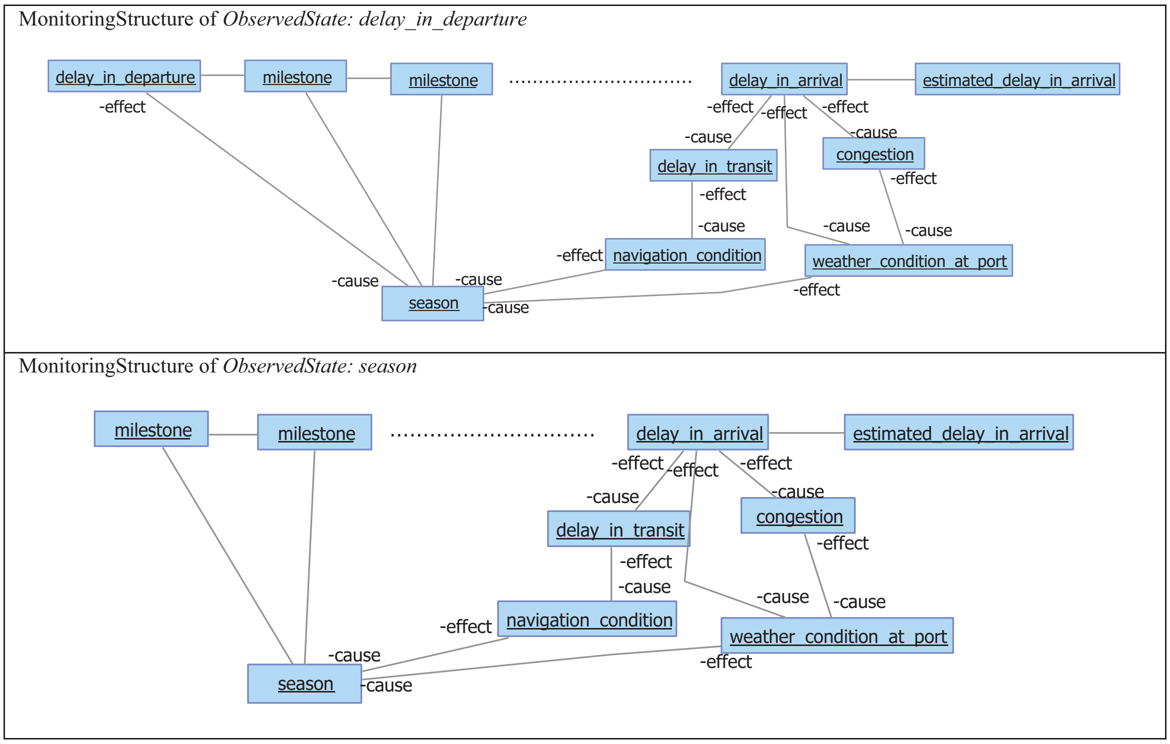


Fig. 12. Initial Milestone associated with the *freight_transport_by_sea:SupplyProcess.*

Variable with ObservedState:delay_in_intermediate_port; OneTimeObservationPolicy

Variable with EstimatedState: the variables of all following milestones.

[Final Milestone] arrival_to_port:StateMilestone

Variable with ObservedState:navigation_condition; RepetitiveObservationPolicy

Variable with EstimatedState:delay_in_transit, weather_condition_at_port, congestion, delay_in_arrival

Variable with ObservedState:weather_condition_at_port; RepetitiveObservationPolicy

Variable with EstimatedState:congestion, delay_in_arrival

Variable with ObservedState:congestion; OneTimeObservationPolicy

Variable with EstimatedState:delay_in_arrival

Variable with ObservedState:delay_in_arrival; OneTimeObservationPolicy

The initial milestone has two variables with *ObservedState:delay_in_departure* and *ObservedState: season*, their monitoring structure (instances of the reference model) are shown in Fig. 12.

The amount and relation of precedence of the intermediate milestones, between the departure port (initial milestone) and the arrival port (final milestone), depends on the distance to be travelled by the ship and its schedule. The intermediate milestones can be: *arrival_to_intermediate_position:StateMilestone* or *arrival_to_intermediate_port:StateMilestone*.

In this example, congestion:AttributeVariable is an attribute of the port:Resource; delay_in_departure: AttributeVariable, delay_in_transit:AttributeVariable, delay_in_position:AttributeVariable, delay_in_ arrival:AttributeVariable and delay_in_intermediate_port:AttributeVariable are attributes of the ship:Resource and; navigation_condition:EnvironmentVariable, season:EnvironmentVariable and weather_condition_at_port:EnvironmentVariable are environment variables. The target variable is estimated_delay_in_arrival:TargetVariable.

The monitoring model of the *freight_transport_by_sea:SupplyProcess* automatically generated from the reference model instance by the Monitoring System (Fig. 13) is composed of both discrete and continuous nodes. The discrete nodes are the following: *season:DiscreteNode* {winter, non_winter}, *navigation_ condition:DiscreteNode* {favourable, neutral, unfavourable} and *weather_condition_at_ port: DiscreteNode* {favourable, unfavourable}, *delay_in_departure:DiscreteNode* {[0–0.5),[0.5–2),[2–6),[6–24),[24–48),[48–inf)} and has a Gamma distribution. The continuous nodes are the following: *delay_in_transit: ContinuousNode*, *delay_in_position:ContinuousNode*, *delay_in_intermediate_port: ContinuousNode* and *delay_in_arrival: ContinuousNode*. The function node is *estimated_delay_in_ arrival: FunctionNode*.

An example to illustrate the case is an order that requires transporting freight from Point Lisas (Trinidad and Tobago), unload freight in Santos (Brazil) and arrival in Zarate (Argentina). According to the schedule, the ship is in transit 213 h from Point Lisas to Santos (Brazil) and 125 h from Santos (Brazil) to Zarate (Argentina). The total navigation time of the ship will be of 338 h. The milestones set contains: *departs_of_the_port:StateMilestone* (Fig. 12), *arrival_to_intermediate_position_1:State Milestone, arrival_to_intermediate_position_2:StateMilestone*, *arrival_to_intermediate_port:State Milestone* and *arrival_to_intermediate_position_3:StateMilestone* and *arrival_to_port:StateMilestone*.

The decision criterion used by the *Monitor* establishes that the ship is delayed when its probability is greater than a *threshold*. In this example, the *threshold = 24 hours*. If the *threshold* is exceeded, the *Monitor* uses the target variable to predict the occurrence of a disruptive event which is associated with the last milestone. The target variable is *estimated_delay_in_arrival:FunctionNode* and the parent node, which contains the estimated value, is *delay_in_arrival:ContinuousNode*. The function of the target variable is:

```
if (Probability(delay_in_arrival) ≥ threshold)
then { estimated_delay_in_arrival = delay_in_arrival}
else { estimated_delay_in_arrival = 0}
```

In this example, the milestones have several observed variables. Each one is observed according to the relation of precedence and, when it is observed, the impact of this variable over the current and next milestones is analysed to assess the likely occurrence of a disruptive event. If no disruptive event it is predicted, the *Monitor* continues with the monitoring process.

Following, as example, the initial milestone of a scenario is described.

depart_of_the_PointLisas_port:StateMilestone: Fig. 14 shows the Bayesian Network defining the monitoring structure associated with this milestone where a priori probabilities associated with each node are deployed. This Bayesian Network has been obtained by transforming the monitoring structure associated with the initial milestone (Fig. 12). The delay in the arrival to Zarate port (x) has a normal distribution which is represented as $x \sim Normal(\mu = 6.66, \sigma^2 = 200.64)$. Then, a priori probability that the ship is delayed more of 24 h is 0.11 ($P\{x \geq 24\} = 0.11$).

When the *Monitor* gets the value of the *Variable* with *ObservedState* and if *depart_of_the_PointLisas_ port = 0–0.5*, it inserts the evidence in the Bayesian Network assigning *depart_of_the_ PointLisas_ port:(0–0.5, 100)* to the *depart_of_the_PointLisas_port: DiscreteNode*, and performs the inference process by propagating this evidence along the network. The *Monitor* concludes that there is not enough evidence to predict that a disruptive event will occur.

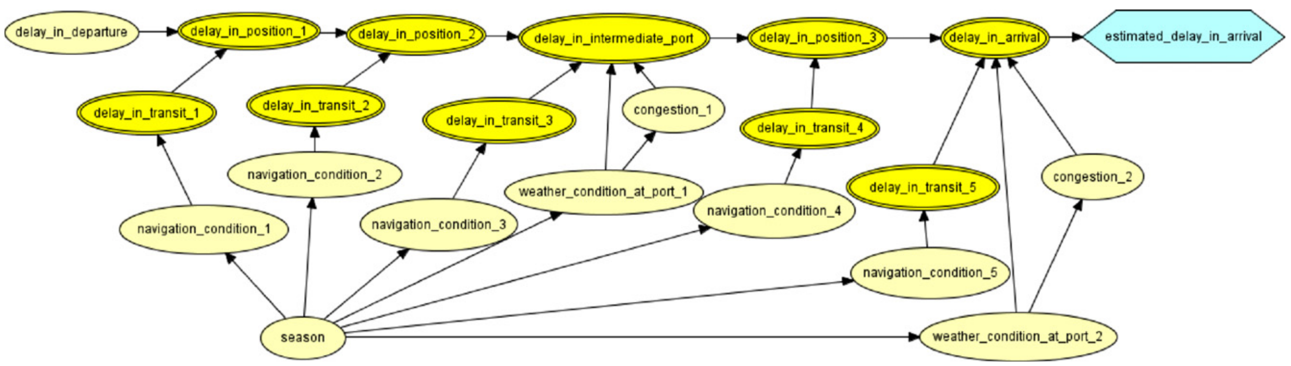


Fig. 13. Monitoring model based on Bayesian Network associated with the *freight_transport_by_sea:SupplyProcess*.

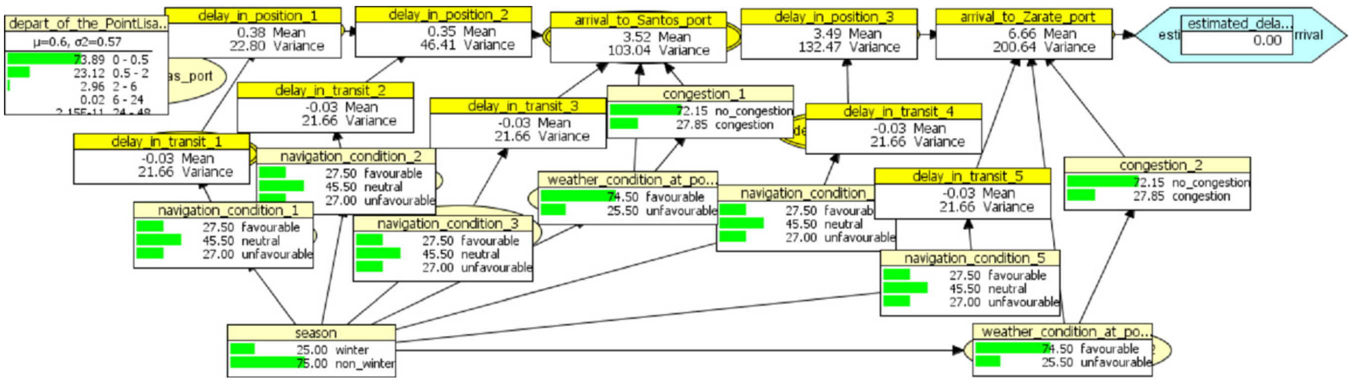


Fig. 14. BN with a priori probabilities.

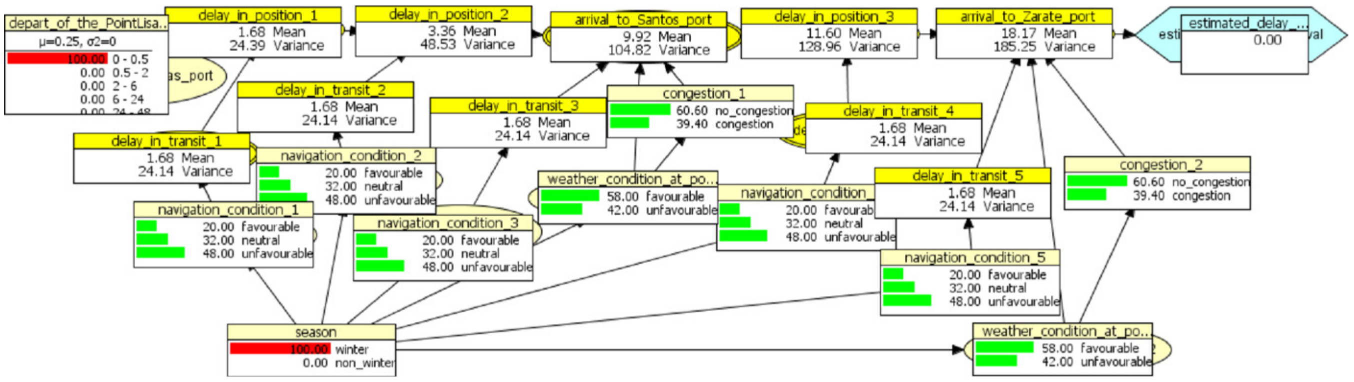


Fig. 15. BN with evidence *delay_in_departure = [0–0.5)* and *season == winter*.

Then, it continues the monitoring process getting the value of the *Variable* with *ObservedState*, if *season == winter*, it inserts the evidence in the Bayesian Network assigning *season:(winter, 100)* to the *season:DiscreteNode*, and performs the inference process. Fig. 15 shows posterior probabilities associated with each node. As a result, the *estimated Value* is *Normal*$(\mu = 18.17, \sigma^2 = 185.25)$. Because *Probability*($\mu$ = *18.17*) < *threshold*, the *Monitor* concludes that there is not enough evidence to predict that a disruptive event will occur. The *target variable* shows that a delay to arrival port is not predicted. Then, it continues the monitoring process and defines the next milestone.

4.6. Case study castings production plant

The monitoring model associated with the *casting_production:-SupplyProcess* to predict a disruptive event is briefly described. The milestones defined and their observed variables, observation policy and estimated variables are the following:

[Initial Milestone] process_start:StateMilestone

Variable with StateObserved:delay_in_moulding_previous; OneTimeObservationPolicy

Variables with StateEstimated: delay_in_moulding, test_of_mixture, breaking_the_mould, visual_inspection, special_inspection, delay_in_casting

[Intermediate Milestone] unmould_end:StateMilestone

Variable with StateObserved:test_of_mixture; OneTimeObservationPolicy

Variables with StateEstimated: delay_in_moulding, breaking_the_mould, visual_inspection, special_inspection, delay_in_casting

Variable with StateObserved:breaking_the_mould; OneTimeObservationPolicy

Variables with StateEstimated: delay_in_moulding, visual_inspection, special_inspection, delay_in_casting

Variable with StateObserved:visual_inspection; OneTimeObservationPolicy

Variables with StateEstimated: delay_in_moulding, special_inspection, delay_in_casting

[Intermediate Milestone] sub_processes_end:StateMilestone

Variable with StateObserved:special_inspectionOneTimeObservationPolicy

Variables with StateEstimated: delay_in_casting

[Final Milestone] process_end:StateMilestone

Variable with StateObserved:delay_in_casting; OneTimeObservationPolicy

The monitoring structure (instance of the reference model) is shown in Fig. 16. In this example, *test_of_mixture:AttributeVariable*, *breaking_the_mould:AttributeVariable*, *delay_in_moulding:Attribute Variable*, *delay_in_moulding_previous:AttributeVariable* are attributes of the *mould:Resource*; *visual_inspection:AttributeVariable*, *special_inspection:AttributeVariable*, *delay_in_casting:Attribute Variable* are attributes of the *casting:Resource*. The target variable is *estimated_delay_in_casting: TargetVariable*.

The monitoring model (Fig. 17) is composed of both discrete and continuous nodes. The discrete nodes are: *test_of_mixture:-DiscreteNode* {good, bad}, *breaking_the_mould: DiscreteNode* {no, yes}, *visual_ inspection:DiscreteNode* {good, bad} and *special_inspection:Discrete Node* {good, bad}. The continuous nodes are: *delay_in_moulding:ContinuousNode*, *delay_in_moulding_previous:-ContinuousNode* and *delay_in_casting:ContinuousNode*. The function node is *estimated_delay_in_casting:FunctionNode*.

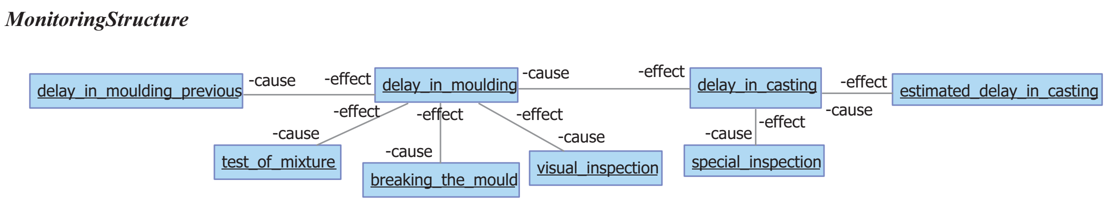


Fig. 16. Monitoring structure associated with the *casting_production:SupplyProcess.*

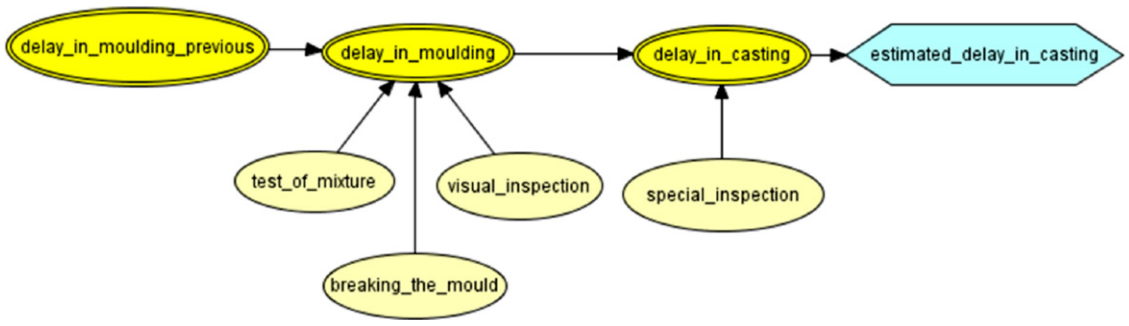


Fig. 17. Monitoring model based on Bayesian Network associated with the *casting_production:SupplyProcess.*

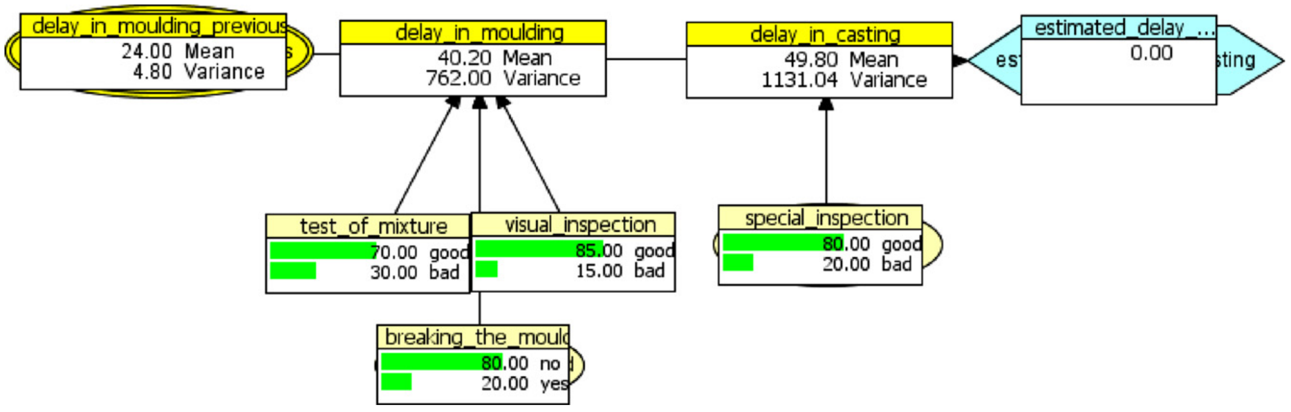


Fig. 18. BN with a priori probabilities.

An example to illustrate the case is an order that requires producing a casting of 1500 kg. The total time of the process depends on the kind of casting. In this example, the total process time is 250 h. The milestones set contains: *process_start:StateMilestone, unmould_end:StateMilestone*, *sub_processes_end: StateMilestone* and *process_end:StateMilestone.*

In the example, *threshold* = *72 hours*. The target variable is *estimated_delay_in_casting:FunctionNode* and the parent node, which contains the estimated value, is *delay_in_casting:ContinuousNode*. The function of the target variable is:

if (Probability(delay_in_casting) $\geq$ threshold)
then { estimated_delay_in_casting = delay_in_casting}
else { estimated_delay_in_casting = 0}

Following, as example the initial milestone of a scenario is described.

process_start:StateMilestone: Fig. 18 shows the Bayesian Network defining the monitoring structure associated with this milestone where a priori probabilities associated with each node are deployed. The delay in casting (x) has a normal distribution which is represented as $x \sim Normal(\mu = 49.80, \sigma^2 = 1131.04)$. Then, the a priori probability that the casting is delayed more of 72 h is 0.25 ($P\{x \geq 72\} = 0.25$).

The *Monitor* gets the value of the *Variable* with *ObservedState delay_in_moulding_previous*, which implies the accumulated delay in the end time of the casting, it inserts the evidence in the Bayesian Network assigning $Normal(\mu = 0, \sigma^2 = 0)$ to the *delay_in_moulding_previous:ContinuousNode* (Fig. 19), and performs the inference process. As a result, *estimated Value* is $Normal(\mu = 25.80, \sigma^2 = 1126.24)$. Because *Probability* (μ = *25.80*) < *threshold*, the *Monitor* concludes that there is not enough evidence to predict that a disruptive event will occur. The *target variable* shows that a delay in the end time of the casting is not predicted. Then, the *Monitor* continues the monitoring process and defines the next milestone.

4.7. Results analysis

The results show that for each supply process whose execution time is of long duration, the proposal allows the monitoring system generating a monitoring model with different milestones. Based on this model the monitoring system could predict the occurrence of disruptive events in the last milestone, early enough to make decisions that allow absorbing in the best way the possible impact of the disruption by using the buffers of the schedule.

In the example of the case study cheese production process, for a processing time of 240 h, a disruptive event can be predicted

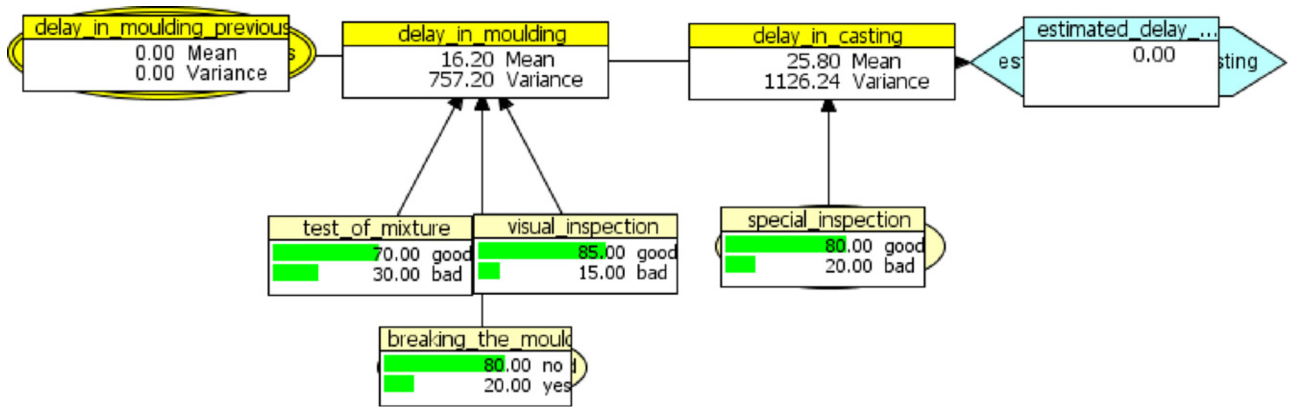


Fig. 19. BN with evidence *delay_in_moulding_previous* = *Normal*$(\mu = 0, \sigma^2 = 0)$.

between 120 and 240 h prior to completion. While, in the case study marine freight transport process, for an example whose processing time is of 338 h, a disruptive event could be predicted from its start in the depart port until the arrival at the destination port. Finally, in the case study castings production plant, for an example whose processing time is of 250 h, a disruptive event can be predicted from the 24 h of the process start until 30 h prior to completion.

In the three examples, based on historic data, the monitoring model of the supply processes predicted 87% of the disruptive events (error type I was 13%) and did not predict disruptive event 89% of time (error type II was 11%). These results depend on the supply process type and of each particular implementation, so best data have to be collected to improve the reliability of the prediction functions.

5. Related works

In order to provide a SCEM solution, several contributions have been proposed, among them: Tai-Lang Yao [24] proposed an Order Fulfillment Process Controller (OFP-Controller) to prevent the occurrence of an exception. When the schedule is generated (through a collaborative and centralised process), it is provided to the order monitoring system to monitor and diagnose the material flow through supply chain execution. The monitoring strategy is focused on the order: through monitoring of two key performance indicators it captures changes in the order attributes. The performance indicators are: the cycle time and rate of order fulfilment. In addition, the occurrence of an exception is used to revise the plan associated with similar next orders. Chin-Hung [25] developed a generic model to represent the relationship and operations within the extended supply chain. Based on this model, time-dependent fault trees were developed to show the failure modes that resulted in the disruptive events. The monitoring strategy is focused on the order: the model variables are attributes of the order and the material flow.

In the work of [26], a method of fault detection is applied to supply chain fault propagation analysis over inventory data. Disruptive events are detected through significant changes over inventory data. From the observation of these significant changes, fault propagation generic patterns are discovered in order to anticipate negative effects. The monitoring strategy is focused on the material resource and the prediction function of a disruptive event is based on data.

Bansal et al. [27] proposed a framework for disruption management in the supply chain. In this work a disruptive event is captured through monitoring of key performance indicators (stock inventory, throughput) when these exceed a threshold. This model is applied to a supply chain of a refinery and changes in the material resource are detected through monitoring of the crude oil inventory indicator. The monitoring strategy is focused on the material resource and the prediction function of a disruptive event is deterministic. The work of [28], proposed a conceptual modelling approach for supply chain event management. Disruptive events are detected through ratios, which are associated with the material resource or order attributes. In this work the prediction function of a disruptive event is deterministic (based on rules). Liu et al. [29] proposed coloured time Petri Nets as formalism for supply chain event management. Events are classified in the following types: task status related events, events produced by a task and external events. Based on the interactions between partners in the supply chain, events are identified. Next, the rules that relate these events to one another are defined. Each rule is associated with the corresponding Petri Net pattern in order to generate the monitoring structure. In this work, the monitoring strategy is focused on the resource and the environment. The prediction function of a disruptive event is deterministic.

A common feature of these works is that they do not consider the structure of the supply process the order represents to define the monitoring structure. This limits the control points that can be used and the set of variables that can be observed in each of them, thereby limiting the potential of the predictive model. This is the most significant difference with our proposal.

In the work of [30], the monitoring of resources is considered to be important to anticipate disruptive events. These are captured through monitoring of resource attributes which are compared with their target value. In addition, to target-state comparisons, statistical test is used to assess the significance of the deviations. Despite its usefulness, this model cannot infer changes in the planned values of the attributes of an order. Kim and Choi [31] developed an active data acquisition language for proactive exception handling. Despite this work is limited to manufacturing, it considers the monitoring of resource attributes significant. The proposal is applied to predict the tool breakage through monitoring of the following attributes: the axis displacement, tool bending load, tool compression load.These last two works do not discuss how the predicted behaviour of resources can affect the attributes of a supply chain order. This is a particular difference of them with our proposal.

In the field of event management, Complex Event Processing (CEP) [32] has evolved into the paradigm of choice for the development of monitoring and reactive applications. CEP already plays an important role in many application areas like logistics, schedule and control processes, network monitoring, etc. CEP addresses two crucial prerequisites to built highly scalable and dynamic systems. CEP-systems support the detection of relationships among events that can be specified by defining models of causal relations. In these models events are detected in deterministic a way.

This last work presents a set of tools based on the approach for reactive monitoring of order, which marks the main difference with our proposal whose aim is to anticipate the occurrence of disruptive events monitoring variables that can predict them.

6. Conclusions and future work

Based on the abstract language provided by the reference model, the user could represent the monitoring process of a supply process without the need of known the specific language of the implementation technology. The monitoring model represented in terms of the reference model can be automatically transformed into different technological languages the final implementation platform interprets, increasing the development productivity and quality of applications for monitoring supply processes.

The specific model presented in this work allows generating a monitoring model in which the relationships among resource attributes, environment variables and order attributes, are defined by a Bayesian Network that defines a probabilistic model to capture and propagate changes based on the structure of the supply process the order represents. This extends the predictive potential of the monitoring model by adding control points and a set of variables that can be observed in each of them.

The reference model allows modelling a generic causal relationship function for any supply process. For a required particular prediction function based on Petri Net, Bayesian Network, etc., the corresponding specific reference model and its transformation rules have to be developed and implemented.

An empirical validation of the approach has been performed through three case studies. An illustrative scenario for the first case shows the ability of the monitoring system that implements the approach to anticipate a disruptive event. To perform this task, the monitoring component of the SCEM system uses the inference engine of HUGIN [23] to process a probabilistic model based on the structure of the supply process and statistical data, which is a remarkable characteristic of this approach.

The monitoring component of the SCEM system notifies to the control component of the SCEM system a disruptive event when it has enough evidence that it will occur (Fig. 1). This allows the control component to derive a mechanism for automatic repair of disrupted supply processes. This mechanism is able to detect whether a disruptive event generates a significant disruption in the schedule, and if so, it searches for a feasible solution using planned buffers [33]. This solution can reduce the impact and propagation of the schedule disruption. The earlier the disruptive event is predicted, the more time the control system has to perform its work and better solutions can be obtained.

As it has been highlighted, this approach can be applied when the supply process time is sufficient to justify the advance.

The monitor gets the value of the monitored variables (resource attribute and environment parameters) and analyses their impact on the order attributes (change in the amount specified in the order and/or in the time in which the order must be fulfilled) to predict a disruptive event. A similar approach can be used to monitor environment and resource variables to anticipate changes in the resource availability. This will be considered in future works.

References

[1] J.A. O'Brien, G.M. Marakas, Management Information Systems, McGraw-Hill Irwin, Boston, 2009.
[2] R.C.a.L.L. Mitsuo Gen, Advanced planning and scheduling models, in: Network Models and Optimization, Springer, London, 2008, pp. 297–417.
[3] J. Galbraith, Designing Complex Organizations, Addison-Wesley, Reading, MA, 1973.
[4] S. Pradhan, Implementing and Configuring SAP Event Management, Galileo Press Inc., Boston, 2010.
[5] H.L. Lee, V. Padmanabhan, S. Whang, The Bullwhip Effect in Supply Chains, Sloan Management Review 38 (3) (1997) 93–102.
[6] N. Radjou, L.M. Orlov, T. Nakashima, Adapting to supply network change, in: F. REsearch (Ed.), The TechStrategy Report, 2002.
[7] V. Landeghem, H. Vanmaele, Robust planning: a new paradigm for demand chain planning, Journal of Operations Management 20 (2002) 769–783.
[8] S. Sendall, W. Kozaczynski, Model transformation: the heart and soul of model-driven software development, IEEE/IET Electronic Library, VDE VERLAG Conference Proceedings 20 (5) (2003) 42–45.
[9] N. Masing, SC Event Management as Strategic Perspectiva – Market Study: SCEM Software Performance in the European Market, Master Thesis, Universitié du Québec en Outaouasis, 2003.
[10] R. Zimmermann, in: M. Walliser, S. Brantschen, M. Calisti, T. Hempfling (Eds.), Agent-based Supply Network Event Management. Whitestein Series in Software Agent Techonologies, 2006.
[11] N. Montgomery, R. Waheed, Event management enables companies to take control of extended supply chains, AMR Research (2001).
[12] Object Management Group, Model Driven Architecture (MDA), 2003.
[13] Object Management Group, XML Metadata Interchange (XMI), 2006, http://www.omg.org/spec/XMI/2.4/Beta2/.
[14] Object Management Group, I. UML 2.0 Superstructure Specification, 2010, http://www.omg.org/spec/UML/2.4/.
[15] F. Jensen, An Introduction to Bayesian Networks, Springer Verlag, New York, 1996
[16] ATL (ATLAS Transformation Language), http://www.eclipse.org/atl/.
[17] Object Management Group, Query/View/Transformation (QVT), 2011, http://www.omg.org/spec/QVT/1.1/.
[18] Object Management Group, Object Constraint Language (OCL), 2010, http://www.omg.org/spec/OCL/2.3/Beta2/.
[19] Object Management Group, Meta Object Facility, 2008, http://www.omg.org/spec/MOF/2.4/Beta2/.
[20] M.B. Miles, A.M. HUberman, Qualitative Data Analysis: An Expanded Sourcebook, 2nd ed., SAGE Publications, USA, 1994.
[21] R.K. Yin, Case Study Research: Design and Method, 4th ed., SAGE Publications, USA, 2009.
[22] A. Oke, M. Gopalakrishnan, Managing disruptions in supply chains: a case study of a retail supply chain, International Journal of Production Economics 118 (1) (2009) 168–174.
[23] Hugin Expert A/S, Hugin Researcher, 2010. www.hugin.com.
[24] Y. Tai-Lang, An Exception Handling Method of Order Fulfillment Process in the i-Hub Supported Extended Supply Chain, Master's Thesis, National Central University, Institute of Industrial Management, Taiwan, 2002.
[25] C. Chin-Hung, Assessing Dependability of Order Fulfillment in the i-Hub Supported Extended Supply Chain, Master's Thesis, National Central University, Institute of Industrial Management, Taiwan, 2002.
[26] R. Ramon Sarrate, F. Fatiha Nejjari, F.D. Mele, J. Quevedo, L. Puigjaner, Event-based approach for supply chain fault analysis, Computer Aided Chemical Engineering 20 (2005) 1261–1266.
[27] M. Bansal, A. Adhitya, R. Srinivasan, I.A. Karimi, An online decision support framework for managing abnormal supply chain events, Computer Aided Chemical Engineering 20 (2005) 985–990.
[28] A. Winkelmann, S. Fleischer, S. Herwig, J. Becker, A conceptual modeling approach for supply chain event management (SCEM), in: Proceedings of the 17th European Conference on Information Systems (ECIS 2009), Verona, Italy, 2009.
[29] R. Liu, A. Kumar, W. van der Aalst, A formal modeling approach for supply chain event management, Decision Support Systems 43 (3) (2007) 761–778.
[30] K. Kurbel, D. Schreber, Agent-based diagnostics in supply networks, Issues in Information Systems VIII (2) (2007).
[31] K. Kim, I. Choi, Active data acquisition for proactive exception handling in manufacturing, The International Journal of Advanced Manufacturing Technology 43 (3–4) (2009) 365–378.
[32] L. David, The Power of Events. An Introduction to Complex Event Processing in Distributed Enterprise System, Pearson Education, 2002.
[33] 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), Special Issue: On intelligent and Innovative Computing in Business Process Management (2011).

Erica Fernandez is a Ph.D student at INGAR, a research center of CONICET Argentina (National Council of Scientific and Technical Research). She is assistant professor of the UTN (National Technological University) since 2007.Her research mainly focuses on information technology and supply chain management.

Enrique Salomone has obtained his PhD in 1993. He is Professor of UNL (Litoral National University) since 2008 and member of the CONICET (National Council of Scientific and Technical Research) of Argentina since 1993. His area of interest is information technology as support for productive systems management.

Omar Chiotti has obtained his PhD in 1989. He is Professor of the UTN (National Technological University) since 1989, head of the CIDISI (R & D Center in Information System Engineering) and member of the CONICET (National Council of Scientific and Technical Research) of Argentina since 1991. His area of interest is information technology as support for productive systems management.