Model Based on Bayesian Networks for Monitoring Events in a Supply Chain

Erica Fernández[1], Enrique Salomone[1], and Omar Chiotti[1,2]

[1] INGAR – CONICET, Avellaneda 3657, Santa Fe, Argentina, 3000
[2] CIDISI- UTN FRSF, Lavaisse 610, Santa Fe, Argentina, 3000
{ericafernandez,salomone,chiotti}@santafe-conicet.gov.ar

Abstract. The execution of supply process orders in a supply chain is conditioned by different types of disruptive events that must be detected and solved in real time. This requires the ability to proactively monitor, analyze and notify disruptive events. In this work we present a model that captures this functionality and was used as the foundation to design a software agent. A reactive-deliberative hybrid architecture provides the ability to proactively detect, analyze and notify disruptive events that take place in a supply chain. For the deliberative performance of the agent, a cause-effect relation model based on a Bayesian network with decision nodes is proposed.

Keywords: supply chain, event management, agent, Bayesian Networks.

1 Introduction

Current planning and execution systems are rather limited in their ability to automatically respond to changes caused by disruptive events in the supply chain [1]. This is a shortage of these systems because the execution of supply process orders is conditioned by different types of disruptive events. These unexpected events (cancellation of an order, failure in a process, change in a process capacity, etc.) must be detected and solved in real time. This requires ability to proactively monitor, analyze and notify disruptive events. In this scenario, Supply Chain Event Management Systems (SCEM) [1], [2] have been proposed. SCEM is defined as a business process in which significant events are timely recognized, reactive actions are suddenly executed, flows of information and material are adjusted, and the key employees are notified. In other words, SCEM can be seen as a complex control problem. SCEM systems emphasize the need of managing the exceptions by means of short term logistics decisions, avoiding frequent re-planning processes.

Montgomery [3] defines 5 functions that a SCEM system should perform. These are: *Monitoring* (to provide data in real time about processes, events and current states of the orders and parameters); *Notification* (to alert the occurrence of exceptions to take decisions proactively); *Simulation* (to evaluate the effect of actions to be taken); *Control* (if an exception takes place, to evaluate the changes in the processes proactively); *Measurement* (to evaluate the supply chain performance).

In this work, based on a SCEM model defined as a network of resources and supply process orders, we propose a model for monitoring, analysis and notification of

B. Vallespir and T. Alix (Eds.): APMS 2009, IFIP AICT 338, pp. 358–365, 2010.

events. Based on this model a software agent has been designed. A reactive-deliberative hybrid architecture provides the ability to proactively detect, analyze and notify disruptive events that take place in a supply chain. For the deliberative performance of the agent, a cause-effect relation model based on a Bayesian network with decision nodes is proposed.

This paper is organized in the following way. Section 2 discusses related works. Section 3 presents a SCEM model. Section 4 presents a model for monitoring, analysis and notification of events. Section 5 presents an example and section 6 presents conclusions and future work.

2 Related Works

In order to provide a SCEM solution, several contributions have been proposed, among them: Tai-Lang [4] proposed a method to analyze and manage the impact of an exception during order execution; Chin-Hung [5] developed a model based on cause-effect relationship to represent the disruptive exception caused by unexpected events in a supply chain. Liu [6] presented a methodology that uses Petri nets to formulate supply chain event rules and analyze the cause-effect relationships among events. Maheshwari [7] described the events that throw exceptions through business processes. In Zimmermann [8], the orders to be monitored are initialized by different triggers: queries from customers, alerts from suppliers and critical profiles. The critical profiles are the orders with high probability of being affected by disruptive events. This SCEM solution is based on a multi-agents architecture to proactively monitoring orders, integrating and interpreting several data gathered from the supply chain members to evaluate and distribute the results. Kurbel [9] proposed a mobile agent-based SCEM system to collect and analyze data provided by a supply chain monitoring system. Here, the monitored resources are considered to be important to anticipate unexpected events. To detect exceptions, his approach is not only based on target-state comparison, but it includes statistical analysis as well.

Hofmann [10] proposed an agent based system that allows customers track and trace their orders. In order to know the status of an order, the requirement must be entered by the user. The system is proactive regarding notification of unexpected events. Kärkkäinen [11] presented an agents based system to manage the information flow related to a tangible object. It offers information about the state of an order but it does not consider disruptive events management. These last two approaches are limited to functions of Tracking and Tracing while SCEM is not.

A common feature of these contributions is that the proposed SCEM models are focused on orders and not on resources, when in fact, the resources are the ones affected by disruptive events in a direct way and the orders are the ones affected by the exceptions triggered by such events.

3 The SCEM Model

We have defined a SCEM Model (Figure 1) as a network of *resources* linked among them by *supply process orders*. The *schedule* (provided by the planning system) defines the execution timetable of a set of orders and determines the *resources* (with their

parameters values) that are linked to *supply processes*. This representation shows the origin and propagation of an exception, allowing the monitoring actions focus on resources where the unexpected events can occur. The monitoring of resources during the plan execution can help to anticipate the occurrence of exceptions and proactively take decisions [9]. Resources are control points with a set of parameters that must be monitored to detect the occurrence of an event. This event must be analyzed using a *cause-effect relation network* to determine if an exception occurs. Different from other approaches described in Section 2 (Related Works), in which the control actions or the monitoring actions are centered in the orders, in our approach, the monitoring actions focus on resources because they are directly affected by the disruptive events. This allows us to generate a model to monitor and analyze exceptions with a higher predictive quality.

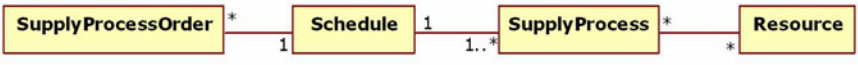

Fig. 1. SCEM Model

4 Model for Monitoring, Analysis and Notification of Events

Based on the defined SCEM model, we have developed a reference model (Figure 2) satisfying the following requirements: ability of reasoning out uncertain scenarios and with partial observations; ability of preventing potential exceptions caused by unexpected events, and ability to predict future states (potential unexpected events) based on past and present observations.

The *Monitor* (Figure 2) is the main component of the model, which is responsible of the information required by monitoring, analysis and notification functions. Each *SupplyProcess* has assigned a *Monitor*. Each *Resource* linked to a *SupplyProcess* (Figure 1) is a control point with a set of *Parameters* that must be monitored to detect the occurrence of an event. This event must be analyzed using a *cause-effect relation network* to determine if an exception occurs. A network defines a causal relation among parameters. Each parameter can take a *Value* in a discrete or continuous domain. The network topology represents temporal causality. Causality refers to the impact or influence among the parameters. Temporal refers to the ability to proactively detect events and evaluate their impact. A *Parameter* has two disjunctive *State* values: *Observed*, the parameter is observed and its value is captured; and *Inferred*, the parameter value is calculated from the value of precedent parameters. Although a causal network has a lot of parameters, only one (the root) is used to evaluate if an exception occurs. Initially, all parameters of the network have the state value inferred, which is calculated with a priori information. During the plan execution, an *ActivationCondition* defines the parameter that has to be observed to incorporate an evidence. Based on this evidence, the *CausalModelAnalizer* uses the causal network to infer an upgrade value of the root parameter. Based on this inferred value and using the *PlannedValue*, the *Monitor* evaluates if an exception occurs. When it predicts an exception, the *Notifier* reports the *DisruptiveEvent*. It can be: change in the requirements of an order (quantity of material, deadline), and/or change in the parameters of a resource (transition time between states, available capacity).

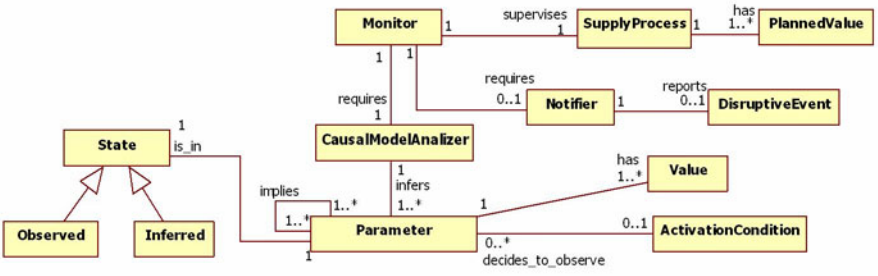

Fig. 2. Reference Model for monitoring, analysis and notification of events

5 An Application Example

As has been said above, the main functions of the model are: *monitoring*, *analysis* and *notification*. Based on this model, we have designed a hybrid agent responsible of the three functions. It has two components: a *reactive* component for the *monitoring* and *notification* functions; and a *deliberative* component for the analysis function, which allows the agent to make complex reasoning and to make plans and to take decisions. This last agent ability has been implemented by means of a Bayesian Network model. Bayesian networks [12] are a method to represent uncertain knowledge, which allows reasoning based on probability theory. Each node of a Bayesian network is composed by 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.

An example of a cheese production plant is described to show the behavior of the agent. The agent, based on the SCEM model (Figure 1), receives a schedule where an order requires producing a quantity of a cheese type. It links the SupplyProcessOrder: *cheese_production_process* with Resource:*milk*, Resource:*cheese_type* and Resource: *cheese_production_plant*. It performs this function through the *reactive* component.

Milk acidity is a parameter that 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. Normal acidity produces cheese with required quality. The agent, based on the reference model (Figure 2), defines the monitoring structure by means of a Bayesian Network with a priori probabilities (Figure 3). These probabilities are obtained from statistical data of previous results. This monitoring structure is composed by the following *Parameter*s, whose possible *Value*s are represented in braces: *acidity* {normal, low, high}, which is a parameter of resource:*milk; time_of_curdle* {normal, low, high}, *fresh_cheese_texture_quality* {good, bad}, *fresh_cheese _texture* {no_granulated, granulated}, *surface_cracks* {no, yes}, *fresh_cheese_taste* {good, insipid, bitter} and *fresh_cheese_taste_quality* {good, bad}, which are parameters of the resource:*cheese_production_plant*; and *cheese_type_quality* {good, bad}, which is a parameter of resource:*cheese_type.* The *ActivationCondition* are: *test_ time_of_curdle*, *test_fresh_cheese _texture* and *test_ fresh_cheese _taste.*

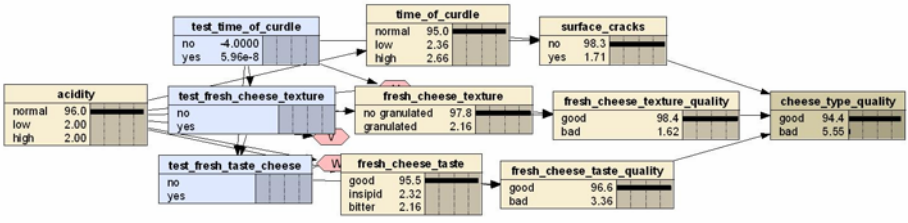

Fig. 3. Monitoring structure based on Bayesian Network with a priori probabilities

The agent notifies changes in the requirement of the order if the probability of the product will be outside the specification is greater than a threshold value; it performs this function through the *reactive component*. In this example *threshold = 97.0.* The total process time depends on the type of cheese to be produced. In this example *cheese_type* = soft cheese is considered, and the total process time is 240 hours.

Thus, the monitoring structure of the agent (Figure 3) initially includes the observed parameter *acidity*. The *acidity* is monitored at the beginning of the process and for each of the three possible values; the agent will define different plans of action which arise from *deliberative component*:

1. If *acidity == high*, the agent inserts this evidence in the net (Table 1, row 1) assigning *acidity:(high, 100)* and propagate it. As a result, the agent obtains the expected value of the parameter, *cheese_type_quality:(bad, 85.1)*. Since, *P(bad) = 85.1 < threshold*, it decides to analyze the inferred parameters *surface_cracks:(yes, 50.7), fresh_cheese_texture_quality:(bad, 45)* and *fresh_cheese_ taste_quality:(bad, 45).* The three parameters explain the value of the parameter *cheese_type_quality*:

1.1 *surface_cracks:(yes, 50.7)* indicates the probability that the cheese has surface cracks caused by a low time of curdle. Based on this information, the agent decides to monitor this parameter. This is done by assigning to the *Activation-Condition test_ time_of_curdle == YES* (Table 1, row 2). This test is made 2 hours after the process has been started. The test has two possible results {*normal, low*} (Table 1, row 1):

1.1.1 If *time_of_curdle == low,* the agent inserts this evidence in the net (Table 1, row 2) assigning *time_of_curdle:(low, 100)* and propagates it. The result is *cheese_type_quality:(bad, 98.5).* Since *P(bad) = 98.5 > threshold*, which implies high probability that product will be outside the specification, so the agent decides to notify the value of the parameter, *cheese_type_quality*, and to FINISH the monitoring process. This allows the agent to predict the result 238 hours before the process ends.

1.2 If *time_of_curdle == normal,* the agent inserts this evidence in the net (Table 1, row 3) assigning *time_of_curdle:(normal, 100)* and propagates it. The result is *cheese_type_quality:(bad, 69.7).* Since *P(bad) = 69.7 < threshold*, the agent decides to analyze the parameters *surface_cracks:(yes, 0.0), fresh_cheese_texture_quality: (bad, 45.0)* and *fresh_cheese_taste_quality: (bad, 45.0).* The first parameter indicates that there is no risk that the product will have surface cracks. The two last parameters explain the value of the parameter *cheese_type_quality*.

1.3 *fresh_cheese_texture_quality:(bad, 45.0),* indicates the probability that the cheese results granulated and, therefore, not satisfying the quality specification. Based on this information, the agent decides to monitor this parameter. This is done by assigning to the *ActivationCondition test_ fresh_cheese _texture == YES* (Table 1, row 4). This test is made 48 hours after the process has been started. The test has two possible results {*no_granulated, granulated*} (Table 1, row 3).

1.3.1 If *fresh_cheese _texture == granulated,* the agent inserts this evidence in the net (Table 1, row 4) assigning *fresh_cheese_texture:(granulated, 100)* and propagates it. The agent is certain that the product will be outside the specification, so it decides to notify the value of the parameter, *cheese_type_quality,* to the resource:*cheese_type*, and to FINISH the monitoring process. This allows the agent to predict the result 192 hours before the process has been finished.

1.3.2 If *fresh_cheese _texture == no_granulated,* the agent inserts this evidence in the net (Table 1, row 5) assigning *fresh_cheese_texture: (no_granulated, 100)* and propagates it. Since *P(bad) = 45.0 < threshold*, the agent decides to analyze the parameters *surface_cracks:(yes, 0.0), fresh_cheese_texture_quality:(bad,0.0)* and *fresh_cheese_taste_quality:(bad, 45.0).* The first two parameters indicate that there is not risk that the product will have surface cracks or texture problems. The last parameter explains the value of the *cheese_type_quality* parameter:

1.4 *fresh_cheese_taste_quality:(bad, 45.0),* indicates the probability that the cheese has taste problems and, therefore, not satisfying the quality specification. Based on this information, the agent decides to monitor this parameter. This is done by assigning to the *ActivationCondition test_ fresh_cheese _taste ==YES* (Table 1, row 6). This test is made 120 hours after the process has been started. The test has two possible results {*good, bitter*} (Table 1, row 5).

1.4.1 If *fresh_cheese _taste == bitter,* the agent inserts this evidence in the net (Table 1, row 6) assigning *fresh_cheese_taste:(bitter, 100)* and propagates it. The result is *cheese_type_quality:(bad, 100)*; the agent is certain that the product will be outside the specification, so it decides to notify the value of the parameter, *cheese_type_quality*, to the resource:*cheese_type*, and to FINISH the monitoring process. This allows the agent to predict the result 120 hours before the process ends.

1.4.2 If *fresh_cheese_taste == good,* the agent inserts this evidence in the net (Table 1, row 7) assigning *fresh_cheese_taste:(good, 100)* and propagates it. The result is *cheese_type_quality:(good, 100)*; the agent is certain that the product will not have taste problem, so it decides to FINISH the monitoring process.

2. If *acidity == low*, the agent inserts this evidence in the net (Table 1, row 8) assigning *acidity:(low, 100)* and propagates it. As a result, the agent obtains information of the expected value of the parameter, *cheese_type_quality: (bad, 51.0).* Since *P(bad) = 51.0 < threshold,* it decides to analyze the parameters *surface_cracks: (yes, 0.0), fresh_cheese_texture_quality:(bad, 0.0)* and *fresh_cheese_taste_quality: (bad, 51.0).* The last explains the value of the *cheese_type_quality* parameter:

2.1 *fresh_cheese_taste_quality:(bad, 51.0),* indicates the probability that the cheese has taste problems and, therefore, not satisfying the quality specification. Based on this information, the agent decides to monitor this parameter. This is done by assigning to the *ActivationCondition test_ fresh_cheese_taste == YES*

(Table 1, row 9). This test is made 120 hours after the process has been started. The test has two possible results {*good, insipid*} (Table 1, row 8).

2.1.1 If *fresh_cheese_taste == insipid,* the agent inserts this evidence in the net (Table 1, row 9) assigning *fresh_cheese_taste:(insipid, 100)* and propagates it. The result is *cheese_type_quality:(bad, 100),* the agent is certain that the product will be outside the specification, so it decides to notify the value of the parameter, *cheese_type_quality,* to the resource:*cheese_type*, and to FINISH the monitoring process. This allows the agent to predict the result 120 hours before the process ends.

2.1.2 If *fresh_cheese_taste == good,* the agent inserts this evidence in the net (Table 1, row 10) assigning *fresh_cheese_taste:(good, 100)* and propagates it. The result is *cheese_type_quality:(good, 100),* the agent is certain that the product will not have taste problems, so it decides to FINISH the monitoring process.

3. If *acidity == normal,* the agent inserts this evidence in the net (Table 1, row 11) assigning *acidity:(normal, 100)* and propagates it. As a result, it obtains the the expected value of the parameter, *cheese_type_quality (good, 97.1). Since P(good) = 97.1 > threshold,* the agent decides to FINISH the monitoring process.

Table 1. Bayesian Network inference process

Row	A	TC	testTC	SC	FCTa	testFCTa	FCTaQ	FCTe	testFCTe	FCTeQ	CTQ
1	normal 0 low 0 high **100**	normal 30 low 70 high 0	no yes	no 49.2 yes 50.7	good 40 insipid 0 bitter 60	no yes	good 55 bad 45	no_granulated 40 granulated 60	no yes	good 55 bad 45	good 14.9 bad 85.1
2	normal 0 low 0 high **100**	normal 0 low **100** high 0	no **yes**	no 5 yes 95	good 40 insipid 0 bitter 60	no yes	good 55 bad 45	no_granulated 40 granulated 60	no yes	good 55 bad 45	good 1.51 bad 98.5
3	normal 0 low 0 high **100**	normal **100** low 0 high 0	no **yes**	no 100 yes 0	good 40 insipid 0 bitter 60	no yes	good 55 bad 45	no_granulated 40 granulated 60	no yes	good 55 bad 45	good 30.2 bad 69.7
4	normal 0 low 0 high **100**	normal **100** low **yes**	no yes	no **100** yes 0	good 40 insipid 0 bitter 60	no yes	good 55 bad 45	no_granulated 0 granulated **100**	no **yes**	good 0 bad 100	good 0 bad 100
5	normal 0 low 0 high **100**	normal **100** low high	no **yes**	no **100** yes 0	good 40 insipid 0 bitter 60	no yes	good 55 bad 45	no_granulated **100** granulated 0	no **yes**	good 100 bad 0	good 55 bad 45
6	normal 0 low 0 high **100**	normal **100** low high	no **yes**	no **100** yes 0	good 0 insipid 0 bitter **100**	no **yes**	good 0 bad 100	no_granulated **100** granulated 0	no **yes**	good 100 bad 0	good 0 bad 100
7	normal 0 low 0 high **100**	normal **100** low high	no **yes**	no **100** yes 0	good **100** insipid 0 bitter 0	no **yes**	good 100 bad 0	no_granulated **100** granulated 0	no **yes**	good 100 bad 0	good 100 bad 0
8	normal 0 **low 100** high 0	normal 15 low 0 high 85	no yes	no 100 yes 0	good 32 insipid 68 bitter 0	no yes	good 49 bad 51	no_granulated 100 granulated 0	no yes	good 100 bad 0	good 49 bad 51
9	normal 0 **low 100** high 0	normal 15 low 0 high 85	no yes	no 100 yes 0	good 0 insipid **100** bitter 0	no **yes**	good 0 bad 100	no_granulated 100 granulated 0	no yes	good 100 bad 0	good 0 bad 100
10	normal 0 **low 100** high 0	normal 15 low 0 high 85	no yes	no 100 yes 0	good **100** insipid 0 bitter 0	no **yes**	good 100 bad 0	no_granulated 100 granulated 0	no yes	good 100 bad 0	good 100 bad 0
11	normal **100** low 0 high 0	normal 98 low 1 high 1	no yes	no 99.3 yes 0.72	good 98 insipid 1 bitter 1	no yes	good 98.5 bad 1.5	no_granulated 99 granulated 1	no yes	good 99.3 bad 0.75	good 97.1 bad 2.95

A: acidity; **TC:** time_of_curdle; **testTC:** test_time_of_curdle; **SC:** surface_cracks; **FCTa:** fresh_cheese_taste; **testFCTa:** test_fresh_cheese_taste; **FCTaQ:** fresh_cheese_taste_quality; **FCTe:** fresh_cheese_texture; **testFCTe:** test_fresh_cheese_texture; **FCTeQ:** fresh_cheese_texture_quality; **CTQ:** cheese_type_quality

6 Conclusions and Future Work

In this work we have proposed an approach to proactively monitor, analyze and notify disruptive events that take place in a supply chain. The new SCEM Model proposed, based on a network of *resources* linked among them by *supply process orders*, allowed to focus the monitoring actions on resources where unexpected events can occur. Thus, the generated model for monitoring, analysis and notification of events that has two advantages: 1) ability to dynamically change the *network of analysis*. That is to say, after an unexpected event is detected, the monitoring strategy can be extended including

other parameters and increasing its monitoring frequency; 2) the model can be used to monitor any process provided the monitoring structure is composed of causal relations.

Based on this last model, we proposed a software agent with hybrid architecture. This architecture allows the agent to perform the model functions, defining a Bayesian network with decision nodes representing temporal causality. The agent proposed, has the ability of anticipating, based on evidence, changes in the values of the parameters of the resource or in the requirements of an order.

As future works, our objective is to add the agent ability to learn from the new experiences. This will allow the agent updating its knowledge base. For the particular causal model implemented in this work, it implies updating the probabilities associated to the Bayesian network.

References

1. Masing, N.: SC Event Management as Strategic Perspectiva – Market Study: SCEM Software Performance in the European Market. Master Thesis. Universitié du Québec en Outaouasis (2003)
2. Zimmermann, R.: Agent-based Supply Network Event Management. In: Walliser, M., Brantschen, S., Calisti, M., Hempfling, T. (eds.). Whitestein Series in Software Agent Techonologies (2006)
3. Montgomery, N., Waheed, R.: Event Management Enables Companies To Take Control of Extended Supply Chains. AMR Research (2001)
4. Tai-Lang, Y.: 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)
5. Chin-Hung, C.: 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)
6. Liu, E., Akhil, K., Van Der Aalst, W.: A formal modeling approach for Supply Chain Event Management. Decision Support Systems 43, 761–778 (2007)
7. Maheshwari, P., Sharon, S.: Events-Based Exception Handling in Supply Chain Management using Web Services. In: Proceedings of the Advanced International Conference on Internet and Web Applications and Services (2006)
8. Zimmermann, R.: Agent-based Supply Network Event Management. Verlag (2006)
9. Kurbel, K., Schreber, D.: Agent-Based Diagnostics in Supply Networks. Issues in Information Systems VIII(2) (2007)
10. Hoffmann, O., Deschner, D., Reinheimer, S., Bodendorf, F.: Agent- Supported Information Retrieval in the Logistic Chain. In: Proceeding of the 32nd Hawaii International Conference on System Sciences, Maui (1999)
11. Kärkkäinen, M., Främling, K., Ala-Risku, T.: Integrating Material and Information Flows Using a Distributed Peer-to-Peer Information System. In: Collaborative Systems for Production Management, Boston, pp. 305–319 (2003)
12. Jensen, F.: An Introduction to Bayesian Networks. Springer, New York (1996)