Predicting Exceptions in Agent-Based Supply-Chains ⋆

Albert Özkohen and Pınar Yolum

Department of Computer Engineering
Boğaziçi University
TR-34342, Bebek, Istanbul, Turkey
aozkohen@elitsoft.com.tr, pinar.yolum@boun.edu.tr

Abstract. Exceptions occur frequently in supply-chains. Consequently, detecting exceptions timely is of great practical value. To enable a timely detection of exceptions, this paper develops an agent-based model for supply-chains. The participants are modeled as autonomous agents that can reason about their interactions and communicate with other agents. The communications of the agents create and manipulate commitments. Violation of commitments leads to exceptions. We provide two methods for detecting such violations. First one uses an AND/OR tree to break down a commitment into smaller commitments and to decide on violation when sub parts of a commitment are violated. The second one uses an ontology to predict if a commitment is going to be violated. Using both approaches jointly, we can detect exceptions in a timely manner.

1 Introduction

A supply-chain is composed of producers that are responsible for producing and delivering a service and consumers that receive the produced service [1]. Entities that are involved in supply-chains are autonomous parties and are thus operated independently. To be successful in a supply-chain, entities need to be cooperative but this does not mean that they will not have individual motives or unpredictable operations based on context. Fox *et al.* [2] state that properties such as dynamism, reasoning, readiness to cooperation, interaction and adaptability show us the necessity to use an agent-based structure while modeling supply-chains. Given these properties of supply-chains, the most natural way to model the entities in supply-chains are as autonomous agents that perceive, reason, act and communicate on their own [2, 3]. This is because entities have to be intelligent and to possess enough initiative in order to reason about their situation and about the appropriate action plans.

The elements of a supply-chain can vary from one design to another. Generally, we might say that suppliers, customers, production units, storage units, logistic support units such as shipping companies, technical support units, management departments, all can be different agents belonging to a multiagent system. Each agent will have its own set of goals and appropriate action plans to achieve the goal according to the current context.

⋆ This research was supported by Bogazici University Research Fund under grant BAP05A104. We thank the anonymous reviewers for their helpful comments.

As in traditional supply-chains, agents follow a protocol to regulate their activities. When protocols rules are violated, exceptions occur. The exceptions have various consequences such as productivity drops, increases in various costs or decrease in customer satisfaction [4, 5]. Because of these consequences, it is crucial to detect and handle exceptions appropriately. The exception handling process is one which requires complex reasoning activities under defined and extendable rule sets. In current practice, exceptions are managed using brain power of working professionals who use the supply-chain management tools. But, this is costly and inefficient because human-based operations are more error prone and more costly compared to automated approaches. For these reasons, there is an ongoing research on automating exception handling in supply-chain management systems [6, 4, 7, 8]. These approaches mostly assume that first exceptions occur and then the approaches recover from these exceptions. What we are proposing here, on the other hand, is to *predict* that the exceptions are going to happen before they actually take place. Depending on how early the exceptions can be predicted, the corrective actions can be taken more appropriately. For example, if a supplier knows that a delivery will not reach its destination several hours before the delivery is due, she might schedule an alternative delivery. However, if she learns that the delivery has not reached its destination at the scheduled delivery time, it may be too late to schedule an alternative delivery.

The exception prediction heavily involves receiving data from outer world (through communication or sensing), which will help the system to infer that an exception is about to happen. To allow exception prediction, we develop an approach based on participants' commitments to each other [9, 10]. Each commitment between participants is broken down into smaller commitments and are placed in an AND/OR graph. Violation of one commitment can lead to violation of other commitments depending on its position in the AND/OR graph. Detecting the first violation is enough to predict that the other commitments will be violated. As a further step, we use an ontology and a rule set to derive facts about the external world. If any one derived facts conflicts with an existing commitment, we conclude that the commitment cannot be fulfilled as promised and predict that an exception is going to take place.

The main contribution of this paper is to develop an approach for predicting exceptions in supply-chains. The approach models supply-chains as multiagent systems in order to enable exception handling. It further defines communications between agents as commitments and detects an exception if a violation of a commitment is perceived. Commitments are placed within an AND/OR tree in order to decide on exceptional cases with a better accuracy. An ontology and domain for the rules existing between agents are defined to reason about the domain and predict if any commitments will be violated. The ontology is useful to predict the occurrence of exceptional cases even before any violation of commitment is detected.

The rest of this paper is organized as follows: Section 2 models exceptional situations using commitments, which exist as nodes within AND/OR trees. Section 3 develops an ontology to represent knowledge about exception scenarios. Section 4 compares our method with other research done in the area. Section 5 summarizes our main contributions and points at directions for further research. Finally, Section 6 gives the listing of the ontology for our sample case study.

2 Modeling Exceptions

Different kinds of exceptions can take place in supply-chain systems. Huhns *et al.* [4] group different exception categories such as missed deadlines, errors in product definitions, late payments, and so on. Most exceptions can be viewed as breach of commitments between some of the parties in the supply chain. If a supplier promises to deliver the material on a particular date but misses that deadline, we end up with an exception. Or, if the customer does not pay her debt as he has promised, then again we have an exception. Based on this intuition, we propose to model exceptions as violations of *commitments*.

2.1 Commitments

Commitments are obligations from one party to another to carry out a particular task [11]. Exceptions occur when one of the agents does not comply by not carrying out its commitment.

Definition 1 A commitment, denoted by $C(debtor, creditor, condition, timeout)$ shows that the debtor has to satisfy the condition in favor of the creditor before timeout. ▮

When condition becomes true, the commitment C is discharged [11, 9] that is the conditions is made to hold. A conditional commitment adds into the picture a pre-condition in order to create the commitment itself.

Definition 2 A conditional commitment, denoted by $CC(debtor, creditor, pre-condition, condition, pre-condition-timeout, condition-timeout)$ shows that a debtor will be committed to satisfy a condition until a condition-timeout on behalf of a creditor, if the pre-condition becomes true before pre-condition-timeout. ▮

Commitments exist in all aspects of communications between agents (humans or software) [12]. Commitments are created as a result of agent activities (interaction or communication with outer world). For instance, on a supply-chain, if a supplier offers a product for $50\$$ then the supplier is committing itself to deliver the goods if the customer pays. Commitments can be discharged within time, may be revised, canceled or delegated. In order to work out exceptional situations, it is sufficient to differentiate between fulfilled and violated commitments. When commitments are fulfilled successfully there will be no exceptions. So one needs to detect a violated commitment in order to decide that there exists an exception. In the above example, if the customer pays to the supplier and the supplier does not deliver the good until the timeout, then the commitment is violated hence we have an exceptional situation. We conclude that commitments as stated above play a crucial role in understanding and modeling exceptions.

2.2 Commitment AND/OR Trees

In business life many tasks can be decomposed into smaller tasks. That is, for finishing successfully a task we may have to finish a number of other tasks also successfully.

We model this fact by decomposing the commitments into other commitments. For example, if a driver violates his commitment to show up to work on time, most likely the company's commitment of delivering goods at a certain time will be violated, too, unless there is a backup driver.

To capture such dependencies, we propose two commitment variants, conjunctive and disjunctive commitments. A conjunctive commitment is a commitment, which can be fulfilled by fulfilling two separate commitments. These commitments can themselves be conjunctive.

Definition 3 $C(x, y, d, t)$ is a conjunctive commitment if $\exists a, b$ such that $(a \not\equiv b) \wedge (d \equiv a \wedge b)$. $C(x, y, d, t) \equiv C(p, q, a, t_1) \wedge C(r, s, b, t_2)$ such that $(t_1 \leq t) \wedge (t_2 \leq t)$ ▮

The two commitments that make up the conjuncts may not be between the same individuals as the conjunctive commitment. Hence x, p and r may be different agents. Similarly, y, q and s could be different too: Section 2.4 explains this point in more depth.

A disjunctive commitment is a commitment, which can be fulfilled in one of several ways.

Definition 4 $C(x, y, d, t)$ is a disjunctive commitment if $\exists a, b$ such that $(a \not\equiv b) \wedge (d \equiv a \vee b)$. $C(x, y, d, t) \equiv C(p, q, a, t_1) \vee C(r, s, b, t_2)$ such that $(t_1 \leq t) \wedge (t_2 \leq t)$ ▮

An AND/OR tree is n-ary tree where children of a node are connected with each other with an AND relationship or an OR relationship. If the children of a node are connected with an AND relation, then all the propositions in the children have to be true for the parent proposition to be true. If the children are connected with an OR relations, then at least one proposition in one of the nodes need to be true [13].

We apply the same idea using commitments. A commitment AND/OR tree is a tree where each node corresponds to a commitment, which models a party's obligation to carry out a task. In addition, all non-leaf nodes contain either a conjunctive or a disjunctive commitment. All leaf nodes are commitments that cannot be decomposed further. These are important to capture exceptions because the only way they can be violated is when there is a timeout and the proposition is still false. When this is the case, the system easily detects the violation and hence the exceptions. Following this detection, the commitment AND/OR tree is traversed from bottom to up in order to check if the violated commitment will have any other consequences. This is used to predict other exceptions. In other words, if a leaf commitment that has an AND relation with its siblings is violated, then the commitment that lies in the parent node will certainly be violated: There is no need to wait for other sibling commitments to be fulfilled or violated. However, if the leaf commitment has an OR relation with its siblings, then as soon as a leaf commitment is violated, its siblings need to be checked.

The main reason to use AND/OR trees for modeling exceptions is to let us search and find the real non-complying commitment in order to execute the proper corrective action. The system begins with an initial rule configuration. During execution, the system creates a parallel structure for each occurrence of an exception and tries to search within the AND/OR tree for the node(s) where the violation occurs. In system definition phase, commitments have to be entered and labeled accordingly to form the AND/OR tree. This decomposition activity can continue until all debtors and creditors of the commitments are within the boundaries of the multiagent system. If sensing the termination of commitments (successfully or with a violation) cannot be done, then we cannot define commitments for these debtors or creditors. We conclude in this way because we need to know if a leaf commitment is discharged successfully or not, in order to understand that if it is an exception or not and in order to reason appropriately with its curing action plan.

2.3 Monitoring Exceptions

To keep track of the commitments that are violated or fulfilled, we envision a monitoring agent (MA) in the multiagent system. The monitoring agent is responsible to record all commitments persistently with their timeout values. When a commitment is discharged by the creditor, the MA is notified and the associated commitment drops form the list of commitments that are waiting to be discharged. Then, MA checks when a commitment is giving timeout. This means the responsible (debtor) agent has not fulfilled its obligation, otherwise MA would have received a fulfillment message. In this case MA sends a message to the creditor that the debtor did not succeed.

2.4 Example

Let us consider the following example depicted in Figure 1, which tries to model the delivery of some goods between the receiver, which is Customer and the sender, which is Supplier using alternative methods:

Our simplified MAS contains the following agents:

Supplier: It has a primary goal of arranging that goods reach to the customer on time.
Customer: It waits to receive goods that will be delivered.
Shipping-Company-1: Its primary goal is to carry load on time.
Shipping-Company-2: Its primary goal is the same as Shipping-Company-1.

1. Assume the following commitment $C_{deliver}$ = $C(Supplier, Customer, Deliver-good, ArrivalTime)$ is being created with a timeout value as ArrivalTime. If the commitment is not discharged after ArrivalTime, it will mean that it has not fulfilled its preposition (Deliver-good) and this will result in an exception.

2. Then MA (see Section 2.3) records $C_{deliver}$ just after the creation.

3. MA tracks any information from Customer notifying if the commitment is fulfilled.

4. If no appropriate reply comes from Customer, and timeout (ArrivalTime) is reached, then MA understands that Supplier is not complying and sends a message to Receiver notifying this problem.

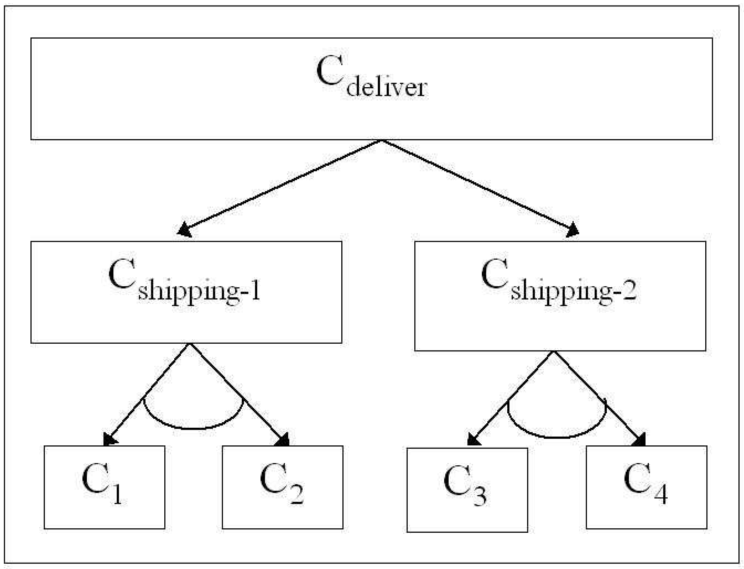

Fig. 1. A commitment AND/OR tree

5. For better understanding and examining the non-complying situation one may further decompose the commitment $C_{deliver}$ (or the just happened exception) to linked commitments such as
$C_{deliver} = C_{shipping-1} \vee C_{shipping-2}$

6. The delivering commitment can be further decomposed as follows:
$C_{shipping-1} = C_1 \wedge C_2$ where
$C_1 = C(Supplier, ShippingCompany1, GiveLoad, t_1)$ and
$C_2 = C(ShippingCompany1, Customer, CarryLoad, ArrivalTime)$ and
$t_1 < ArrivalTime$
Alternatively we can state that C_2 is not responsible to the Customer , but to the Supplier. Then we have the following equality
$C_2 = C(ShippingCompany1, Supplier, CarryLoad, ArrivalTime)$

7. We can also divide (from the main OR branching in Figure 1) the driver commitment as follows:
$C_{shipping-2} = C_3 \wedge C_4$ where
$C_3 = C(Supplier, Driver, GiveLoad, t_2)$ and
$C_4 = C(Driver, Customer, CarryLoad, ArrivalTime)$
$t_2 < ArrivalTime$
Alternatively we can state that C_4 is not responsible to the Customer , but to the Supplier. Then we have the following equality

$C_4 = C(ShippingCompany2, Supplier, CarryLoad, ArrivalTime)$

8. If MA detects that C_1 is non-complying, then the system comes to a result that $C_{shipping-1}$ will be false because one of the children resulting false in an AND branch, which makes the parent false too. So an alternative commitment, namely $C_{shipping-2}$ can be tried before waiting the result of C_2. That is an alternative path instead of a non-complying branch can be selected in order to fulfill the top-level commitment $C_{deliver}$.
9. Figure 1 shows an example decomposition as an AND/OR graph. The branchings that are accompanied by an arc are the AND branches and the rest are OR branches.

10. The decomposition yields the AND/OR commitment tree where the leaves can no longer be decomposed . Hence, violation of commitments in the leaves signal exception.

2.5 AND/OR Tree Usage

Our system is composed of agents, which collect and process information. As it is stated above, commitments reside in the nodes of the AND/OR tree and each node is decomposed in other nodes (i.e. other commitments) where these other (child) commitments cause the parent commitment to be fulfilled successfully. As our structure is a tree, it has leaf nodes. Leaf nodes hold these commitments, which are not able to be further decomposed.

Following our previous example, let us suppose that software agents exist in the shipping companies, but they do not exist in the trucks or roads. So our system will not be able to sense about what happened on the roads. It can only know if the driver has arrived to the destination or not. So boundaries of that multiagent system is the shipping companies and the leaf commitment is the arrival of the driver to the destination point.

2.6 AND/OR Tree Manipulation

A supply chain is hardly ever static: New participants enter, some tasks become obsolete, or new operations are added. To keep up with the dynamic structure of the supply-chain, the AND/OR tree has to be updated frequently. This means that new nodes will need to be added or removed as desired.

For adding a node (i.e. a commitment) to the structure we need to specify its prospective parent node in the tree and the operator with the siblings (AND/OR). The rest is a regular tree insertion algorithm. For instance let us assume that the definition of making a delivery is changed and a new commitment C_0 such as paying in advance comes into the picture

$C_0 = C(Customer, ShippingCompany1, Pays, t_0)$ where $t_0 < ArrivalTime$
$C_{shipping-1} = C_0 \wedge C_1 \wedge C_2$
The new node C_0 has to be inserted as a sibling for C_1 and C_2.

For deleting a node, the specification of its correct place in the tree is enough. The system has to check if there were any siblings for the deleted node. If not, that is if it

was the only child of the parent then the parent becomes leaf node. For instance suppose that the agreement with Shipping Company 2 is cancelled and no shipment will be done with them anymore. In this case we have to correct the structure of the AND/OR tree, so we have to delete $C_{shipping-2}$. This results in deleting that node and all of its children from the tree structure. The system in our example becomes then as follows:
$C_{deliver}$ = $C_{shipping-1}$ = $C_0 \wedge C_1 \wedge C_2$

A leaf node contains a leaf commitment where further decomposition cannot be done within the given boundaries of the domain. All subsequent actions and plans for handling the exceptions will be initiated from the commitments existing on the leaf nodes.

3 Ontology-Based Exception Prediction

An ontology is a conceptualization of a domain. It is a vocabulary of terms, concepts, rules of inferences and their inter-relationships [3]. To reason about different situations, we developed a supply-chain ontology using OWL, a well-known ontology language [14]. This ontology captures relations between different supply-chain entities, such as drivers, delivery, and so on. It also allows relations to be specified. On top of that, we specify rules, such as business rules, in Semantic Web Rule Language [15]. This enables us to represent a large set of knowledge systematically and effectively. The underlying idea is that if we can get information that can be related to working of any one entity in the supply-chain from the outside world, we can derive knowledge as to whether any one commitment is about to be violated. An illustrative example is that if the system hears that there is a tremendous traffic jam on the highway, it may derive that the truck that will follow that highway is going to be late. This reasoning is done much earlier than the expected arrival time of the truck.

3.1 Reasoning Using The Ontology

Let us give an example reasoning, which is related with Figure 1: We can assume that continuous truck-status information may be given by a truck agent (assuming trucks have agents detecting about the situation or status and broadcast this information while our MA receives it using a predefined format) to our multiagent system. This information can contain a problem about the truck where the driver was carrying our load.
Given the following facts:

– Delivery are done by drivers.
– Drivers drive trucks.
– Trucks can be either in working mode or stopped.

and the following rules:

– If a truck is stopped, its driver cannot drive it.
– A delivery is canceled when drivers cannot drive their trucks.

We can apply the rules on the facts and reason about which drivers cannot drive their trucks. If we can figure out that a driver has a commitment to deliver but cannot drive his truck, we conclude that the driver will not fulfill his commitment. Recall from example given in Section 2.4 that if a driver cannot fulfill its commitment (leaf node), the delivery will not succeed (parent node). Hence, we cascade the effects of our findings to other commitments and predict the violation before it happens.

```
del101 is a delivery
del102 is a delivery
del103 is a delivery
del104 is a delivery
John is a driver
George is a driver
fiat50NC is a truck
bmc is a truck
delivery del101 is done by John
delivery del102 is done by george
delivery del103 is done by John
delivery del104 is done by george
John drives fiat50NC
George drives bmc
fiat50NC's workmode is stopped
bmc's work mode is working
```

Fig. 2. Assignment of individuals to classes

The inference of the above ontology example can be implemented using various rule-based reasoning systems to the instances of the created ontology. We have tried various reasoner systems available and have decided on KAON2 [16]. We have seen that KAON2 is performing well by combining ontology definition, instance creation, rule definition and querying the ontology with all the instances and rules. A subset of the OWL and SWRL specification required to run the following example is given in Section 6. The developed system first creates an ontology with the properties and actual instances as shown in Figure 2. The example has four deliveries, namely $del01, del02, del03$ and $del04$. It has two drivers $John$ and $George$. It has two trucks $fiat50NC$ and bmc. But this information is not enough for defining the instances. It has to specify the drivers, which do the deliveries. For instance it says that $John\ delivers\ del01$. Also it specifies, which trucks are driven by which drivers. Then it mentions the working mode of the trucks, it reads the rules that should be applied on the instances and applies these rules. Finally, the system can be queried for various cases. In the example in Figure 3, when the system is queried for cancelled deliveries, the system responds with the URIs of the canceled deliveries. The ontology creation, reasoning and querying activities are performed through a Java program that we developed using SDK 1.5.0.

We are currently working on enhancing our program in order to support various types of queries. Additionally, we are creating a user interface, which lets the user of the system to define number of drivers, deliveries, working modes, and other properties for configuring the system as well as interfaces for generating various types of queries.

```
The list of deliveries and their drivers
----------------------------------------------------
Delivery: http://boun.edu.tr/driver#del03 is cancelled by
http://boun.edu.tr/driver#fiat50NC .
Delivery: http://boun.edu.tr/driver#del01 is cancelled by
http://boun.edu.tr/driver#fiat50NC .
------------------------------------------
```

Fig. 3. Responding to a query

3.2 Hybrid Systems and Benefits

The ontology works simultaneously with the AND/OR tree as follows. The system receives information from the environment and other agents. Using its inference mechanism, the system tries to infer new facts from the set of information it receives. If the newly derived facts, contradict the proposition of any of the commitments in the system, then the agent can figure out that, the commitment will not be fulfiled. Note that this may happen before the commitment timeout has been reached. It may also be the case that information from the outside world will not help the agents derive facts that violate any of the existing commitments. In this case, the agent will wait for the MA to signal timeouts from commitments to decide that one of its commitments has been violated. Hence, the system can be thought of operating both methods in a hybrid way: Some exceptions are caught by the rules defined in our ontology even before any violation takes place and some exceptions are caught when the timeout takes place. If the exceptions are caught by the ontology rules, then the involved participants have a chance to take corrective actions before the timeout.

4 Discussion

Several approaches have been followed in dealing with supply-chain automation and exception handling supply-chains.

Huhns *et al.* make use of linguistic models for modeling the coordination in supply-chain management [4]. They develop interaction diagrams and they identify the conversations by giving a use case model. From there, they obtain Dooley Graphs. Finally they build the state machines in order to generate agent skeletons. Exception handling is only a step that is taken care in this process. They define the groups of exceptions but no mechanism is proposed for handling or predicting exceptions.

Dellarocas and Klein propose a knowledge base approach instead of the linguistic approach [7]. This work seems targeting in a stronger fashion for dealing and handling exceptions. The taxonomy is an n-ary tree without any relationships between the child nodes of a parent exception. Moreover the relationship between the exceptions is not clearly defined.

Kalakota *et al.* port the supply-chain architecture form static into dynamic architecture [17]. Instead of collecting historical data from customers, retailers, distribution centers, production outlets and so on, and to prepare the knowledge-base according to this gathered information for giving decisions (static infrastructure), they propose to store all information dynamically in local agents and they intend to give quicker decisions in much smaller time intervals, i.e. more frequently (dynamic infrastructure). Their assumption on internal operations is based on each individual agent's definition about the degree of self-determination by using statistical methods. They also propose an example supply-chain ontology but they do not develop methods for the automation of exception handling.

Frey *et al.* successfully show that multiagent infrastructure is suitable for globally flexible and locally autonomous business needs where competition and cooperation have to coexist without disrupting each other. They make use of already existing multiagent projects, which are developed or under development. Each multiagent system is responsible for a different group of tasks, such as negotiation, process planning and scheduling, production planning and controlling. Their work mainly defines the interactions and integration between those sub-systems. Despite all scheduling, tracking, tracing, reliability building efforts, exceptions are not taken into consideration, which empowers the need for special care for exceptions in supply-chain management.

5 Conclusion and Further Research

We have developed an automated tool to predict exceptions in supply-chains. The example illustrated in this paper is simple yet important for showing why it is important to predict exceptions before they occur. We are currently working on more complex examples in order to cover additional aspects for our proposed system. Our model uses a hybrid method, which either uses an ontology to detect proactively if an exception will occurr (or conditions show that an exception is about to occur when the relevant timeout comes), or a structured AND/OR tree holding commitments in its nodes detects timeouts.

In this work, we worked with deterministic rules. It will be interesting to enhance this approach by probabilistic rules. For instance if the main road is blocked, the driver may surprise the customer by using a tiny alternative road for reaching its destination although this may have low probability. Hence the sensing from the ontology can further be modeled using a probabilistic approach.

If the system becomes bigger and more complex, it may become difficult and unnecessary to load all commitment data into a single MA. Communication and storage capacity constraints can impose problems. A distributed version of MA can be designed for the performance and privacy reasons. We will investigate these questions in our future work.

References

1. Swaminathan, J.M., Tayur, S.R.: Models for supply chains in e-business. Management Science **49** (2003) 1387–1406
2. Fox, M.S., Barbuceanu, M., Teigen, R.: Agent-oriented supply-chain management. International Journal of Flexible Manufacturing Systems **12** (2000) 165–188
3. Singh, M.P., Huhns, M.N.: Service Oriented Computing—Semantics, Processes, Agents. Wiley (2005)
4. Huhns, M.N., Stephens, L.M., Ivezic, N.: Automating supply-chain management. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), ACM Press (2002) 1017–1024
5. Becker, T.J.: Putting a price on supply chain problems: Study links supply chain glitches with falling stock price. Georigia Tech Research News (2000) Available at: $http://www.gtresearchnews.gatech.edu/newsrelease/$.
6. Mallya, A.U., Singh, M.P.: Modeling exceptions via commitment protocols. In: Proceedings of the 4th International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), ACM Press (2005) 122–129
7. Dellarocas, C., Klein, M.: A knowledge-based approach for designing robust business processes. In van der Aalst et al., W., ed.: Business Process Management, LNCS 1806. (2000) 60–65
8. Dellarocas, C., Klein, M., Rodriguez-Aguilar, J.A.: An exception handling architecture for open electronic marketplaces of contract net software agents. In: Proceedings of the ACM Conference on Electronic Commerce. (2000) 225–232
9. Yolum, P., Singh, M.P.: Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. Annals of Mathematics and Artifi cial Intelligence **42** (2004) 227–253
10. Fornara, N., Colombetti, M.: Operational specifi cation of a commitment-based agent communication language. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), ACM Press (2002) 535–542
11. Singh, M.P.: An ontology for commitments in multiagent systems: Toward a unifi cation of normative concepts. Artifi cial Intelligence and Law **7** (1999) 97–113
12. Castelfranchi, C.: Modelling social action for AI agents. Artifi cial Intelligence **103** (1998) 157—182
13. Nilsson, N.J.: Principles of Artifi cial Intelligence. Springer Verlag (1980)
14. OWL: Web ontology language specifi cation (2004) Available at: $http://www.w3.org/TR/owl-features/$.
15. SWRL: A semantic web rule language combining OWL and RuleML (2004) Available at: $http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/$.
16. Motik, B.: (Kaon2 infrastructure library for managing owl-dl and swrl ontologies) Available at: $http://kaon2.semanticweb.org/$.
17. Kalakota, R., Stallaert, J., Whinston, A.B.: Implementing real-time supply chain optimization (1996) Available at: $http://cism.mccombs.utexas.edu/jan/sc\_imp.html$.

6 Appendix

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
]>
<rdf:RDF
    xml:base="http://boun.edu.tr/driver"
    xmlns:a="http://boun.edu.tr/driver#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/
                      22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#">
<owl:Ontology rdf:about=""/>
<owl:ObjectProperty rdf:ID="cancelled">
    <rdfs:domain rdf:resource="#delivery"/>
    <rdfs:range rdf:resource="#truck"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="doneBy">
    <rdfs:domain rdf:resource="#delivery"/>
    <rdfs:range rdf:resource="#driver"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="drives">
    <rdfs:domain rdf:resource="#driver"/>
    <rdfs:range rdf:resource="#truck"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="workMode">
    <rdfs:domain rdf:resource="#truck"/>
</owl:ObjectProperty>
<a:driver rdf:ID="George">
    <a:drives rdf:resource="#bmc"/>
</a:driver>
<a:driver rdf:ID="John">
    <a:drives rdf:resource="#fiat50NC"/>
</a:driver>
<a:truck rdf:ID="bmc">
    <a:workMode rdf:resource="#working"/>
</a:truck>
<a:delivery rdf:ID="del01">
    <a:doneBy rdf:resource="#John"/>
</a:delivery>
<a:delivery rdf:ID="del02">
    <a:doneBy rdf:resource="#George"/>
</a:delivery>
<a:delivery rdf:ID="del03">
```

```
    <a:doneBy rdf:resource="#John"/>
</a:delivery>
<a:delivery rdf:ID="del04">
    <a:doneBy rdf:resource="#George"/>
</a:delivery>
<a:truck rdf:ID="fiat50NC">
    <a:workMode rdf:resource="#stopped"/>
</a:truck>
<swrl:Variable rdf:ID="V"/>
<swrl:Variable rdf:ID="T"/>
<swrl:Variable rdf:ID="D"/>
<swrl:Imp>
    <swrl:head rdf:parseType="Collection">
        <swrl:IndividualPropertyAtom>
            <swrl:propertyPredicate
          rdf:resource="#cancelled"/>
            <swrl:argument1 rdf:resource="#V"/>
            <swrl:argument2 rdf:resource="#T"/>
        </swrl:IndividualPropertyAtom>
    </swrl:head>
    <swrl:body rdf:parseType="Collection">
        <swrl:IndividualPropertyAtom>
            <swrl:propertyPredicate
          rdf:resource="#doneBy"/>
            <swrl:argument1 rdf:resource="#V"/>
            <swrl:argument2 rdf:resource="#D"/>
        </swrl:IndividualPropertyAtom>
        <swrl:IndividualPropertyAtom>
            <swrl:propertyPredicate
          rdf:resource="#drives"/>
            <swrl:argument1 rdf:resource="#D"/>
            <swrl:argument2 rdf:resource="#T"/>
        </swrl:IndividualPropertyAtom>
        <swrl:IndividualPropertyAtom>
            <swrl:propertyPredicate
          rdf:resource="#workMode"/>
            <swrl:argument1 rdf:resource="#T"/>
            <swrl:argument2 rdf:resource="#stopped"/>
        </swrl:IndividualPropertyAtom>
    </swrl:body>
</swrl:Imp>
<owl:Class rdf:ID="delivery"/>
<owl:Class rdf:ID="driver"/>
<owl:Class rdf:ID="truck"/>
</rdf:RDF>
```