$\mathcal{DIESECT}$: A $\mathcal{DI}$stributed $\mathcal{E}$nvironment for $\mathcal{S}$imulating $\mathcal{E}$-commerce $\mathcal{C}$ontrac$\mathcal{T}$s

Damian Wood, Özgür Kafalı, and Kostas Stathis

Department of Computer Science
Royal Holloway, University of London
Egham, TW20 0EX, UK
{damian.wood.2010,ozgur.kafali,kostas.stathis}@cs.rhul.ac.uk

Abstract. We study the development of a distributed, agent-based, simulation environment where autonomous agents execute e-commerce contracts. We present a multi-agent architecture in which contracts are represented as a set of commitments that an agent must be capable of monitoring and reason with in order to be able to verify that the contract is not violated during interaction. We employ the JADE agent platform to build the multi-agent simulation infrastructure, and the Reactive Event Calculus to provide agent reasoning for monitoring and verification of contracts. We then experimentally evaluate the performance of our system by analysing the time and memory requirements as the number of agents increases, and by looking whether the behaviours of agents have any significant effect on the system's overall performance.

Key words: Agent Technology for E-Commerce, Contracts and Commitments, Distributed Simulation

1 Introduction

Contracts are normally construed as agreements describing the terms of how two or more parties should act in exchanges between or among them. When a customer wants to buy a book from an online store, the terms of the contract describe how the payment should be done as well as the deadline for the delivery of the book. Unless the customer pays, the contract is not binding for the store. After customer payment the contract is fulfilled - if the store delivers the book on time, or violated - if the store delivers late.

In open environments where autonomous agents collaborate to do business together, contracts describe how agents should act to fulfil their duties. The fulfillment of contracts depends on how agents behave and communicate in the environment. Previous work has considered contract execution either in a centralised manner where a central authority manages contract monitoring for all agents, or without taking into account the effect of agent autonomy for contract outcomes. There has been a plethora of work in the literature for formal modeling of electronic contracts: preparation, negotiation, monitoring [1, 2, 3, 4, 5]. Among others, commitments are a widely accepted formalisation of representing contracts in agent-based environments [6]. A commitment binds two agents (e.g. a customer and a store) with a property (e.g. deliver), in which one agent is committed to the other (e.g. by paying) for bringing about the property. For the above example a commitment is represented as:

$$C^{c}_{store,customer}(pay, deliver)$$

That is, the store is committed to deliver the book after payment is completed. The commitment is in conditional state (denoted with the superscript c) until the customer makes the payment. As a result, to effectively monitor and verify contracts amounts to effectively monitoring and verifying commitments.

There has been considerable effort to provide several logic-based tools for monitoring and verification of commitments [7, 8, 9]. While these tools allow significant results to be achieved in terms of contract execution, e.g., detect violations, diagnose the cause of exceptions, predict future problems, they normally assume the existence of a trace representing the agents' interactions; they are used offline. However, a realistic system should take into account different agent behaviours as well as environmental effects online, when considering contract execution.

Our contribution in this work is two-fold: (i) we integrate agent autonomy with contract execution in order to provide a simulation environment for electronic contracts, (ii) we provide a practical implementation based on the widely-used JADE agent development framework and evaluate our system's performance via experiments. We build upon previous work with commitments to provide an agent-based distributed environment, which we call $\mathcal{DIESECT}$, for the simulation of contract executions in e-commerce. We use the JADE agent platform to build agents, and the Reactive Event Calculus to provide contract monitoring and verification. Our approach combines the strengths of object-oriented programming to provide the infrastructure and network operations for distribution and communication of agents, and logic programming to provide a declarative and efficient way to deal with agent reasoning for contracts. We describe the general architecture and the components for $\mathcal{DIESECT}$. Each agent has a partial view of the environment concerning its own contracts. We use commitments to represent agent contracts. Our contribution is the integration of agent autonomy with contract execution accompanied by a practical implementation.

We provide two sets of experiments to evaluate the performance of our system. The first set is designed to test the system by increasing the number of agents. The second set focuses on the agent behaviour. We take a fixed number of agents and change the agents' behaviours to see whether it has an effect on the system's performance. We record the time it takes to complete simulation and the peak memory usage for the system, and comment on the results.

The rest of the paper is organised as follows. Section 2 reviews relevant background on JADE, commitments and the Reactive Event Calculus. Section 3 introduces a delivery protocol from e-commerce as our running example. Section 4 describes the distributed multi-agent architecture for simulating contract execution. Section 5 shows the performance results for our system. Section 6 reviews related work and concludes the paper.

2 Background

We describe next the background for the infrastructure and main components of our system, by explaining their main characteristics.

2.1 JADE Agent Platform

The JADE[1] agent platform is a distributed Java-based middleware for developing multi-agent systems [10]. We have chosen JADE to develop our agents since it is the most widely used agent platform that provides reliable agent communication and documentation support. JADE consists of a runtime environment, a library of classes which to develop agents, and a set of graphical tools to allow agent administration and monitoring. JADE agents use the FIPA[2] specification to communicate with each other via messages. The platform provides a set of behaviours to describe agent tasks, which the developers can extend to implement their own agent behaviours. It also provides a yellow page service for publish & subscribe type services, allows mobile agents to be developed for J2ME, and has graphical tools for debugging agents during run-time execution. JADE allows agents to be distributed over a network via containers, possibly located in a separate physical machine and holding agents connected to a main container where JADE is initiated from.

2.2 Commitments

A contract describes how the participants should act in a business dealing. We represent contracts with commitments between two agents: the debtor agent commits to the creditor agent about a specific property [6]. Definition 1 defines a commitment formally. Below, X and Y denote agents, Ant and $Cons$ are propositions (either atomic propositions or conjunctions of them).

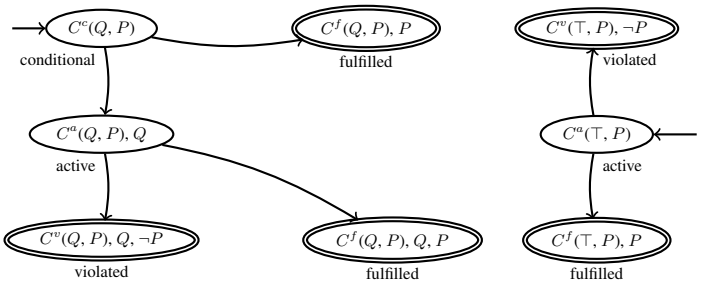
Definition 1 A commitment $C^S_{X,Y}(Ant, Cons)$ denotes the commitment between the agents X and Y, where S is the state of the commitment. Four commitment states are meaningful for our work: conditional, active, fulfilled and violated. The above is a conditional commitment; if the antecedent Ant is satisfied (i.e., becomes true), then the debtor X becomes committed to the creditor Y for satisfying the consequent $Cons$, and the commitment becomes active. If Ant is already $True$ (denoted $\top$), then this is an active base-level commitment; X is committed to Y for satisfying $Cons$ unconditionally. ∎

We follow the idea and notation of [11] to represent commitments (i.e., every commitment is conditional). A base-level commitment is simply a commitment with its condition being true. Commitments are live objects; we always consider a commitment with its state. The commitment states are demonstrated in Fig. 1. Further details on the use of commitments in multi-agent systems can be found in [6, 3, 11].

2.3 $\mathcal{R}$eactive $\mathcal{E}$vent $\mathcal{C}$alculus

The Reactive Event Calculus ($\mathcal{REC}$) [12, 8] is a logic programming tool that extends the Event Calculus [13] for run-time monitoring, including commitments. When used for monitoring the states of commitments the $\mathcal{REC}$ engine takes as input the following:

[1] `http://jade.tilab.com/`
[2] `http://www.fipa.org/`



(a) Conditional (b) Base-level

Fig. 1. Commitment states.

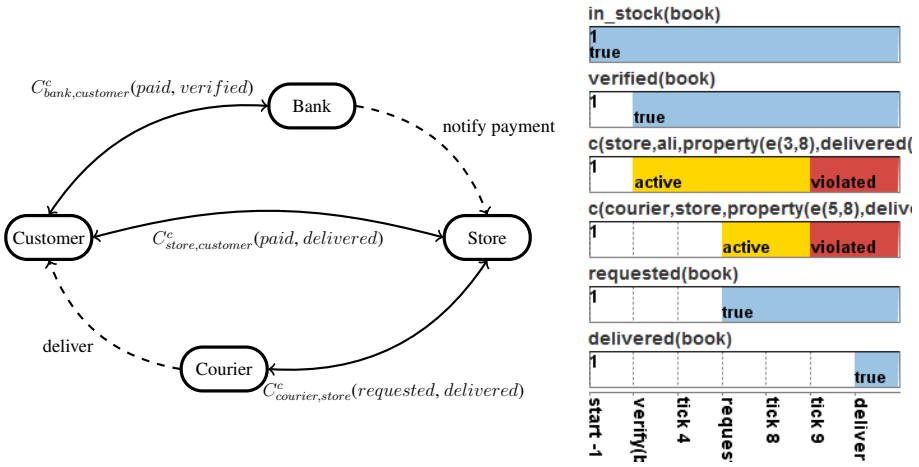
– a *commitment theory* that contains the rules on how commitments are manipulated, e.g., a commitment is fulfilled when its consequent is satisfied within its deadline. This rule-base is shared amongst all agents. Listing 1 shows part of the implementation for the commitment theory.
– a *protocol description* that contains rules describing the consequences of the agents' actions as well as domain facts, e.g., customer payment makes the commitment for delivery active. This is agent and domain dependent rule-base; each agent has a separate protocol description that relates to its own view. For example, a courier does not know the rules between the customer and the store.
– an *event trace* that contains the actions performed throughout time, e.g., the customer has paid at time 4 , the courier has delivered at time 7. Like protocol descriptions, event traces are also agent-dependent. That is, each agent is aware of only the events that are related to it, but does not see the events that might take place among other agents.

Once the $\mathcal{REC}$ engine is run with above input, it produces an outcome that demonstrates the fluents the agent is aware of through time (e.g., states of commitments). A detailed explanation of how $\mathcal{REC}$ manipulates commitment states can be found in [12].

```
% create as conditional
initiates(E, status(C, conditional), T):- ccreate(E, C, T).

% conditional to active
terminates(E, status(C, conditional), T):- detach(E, C, T).

initiates(E, status(C, active), T):- detach(E, C, T).

detach(E, c(X, Y, property(e(T1, T2), Q), P), T):-
   conditional(c(X, Y, property(e(T1, T2), Q), P), T),
   T >= T1, T =< T2, initiates(E, Q, T).
```

Listing 1. Commitment theory in $\mathcal{REC}$.

Commitment tracking with $\mathcal{REC}$ is extended in [2] to integrate exception handling behaviour for agents using an *exception theory* in addition to the above input.



(a) Delivery protocol. (b) $\mathcal{REC}$ output.

Fig. 2. E-commerce protocol.

3 Running Example

In the sequel we use a delivery protocol [14] from e-commerce to demonstrate our simulation environment. Figure 2(a) shows the delivery protocol with four parties. In a desired execution, first the customer sends the payment to the bank regarding its purchase of a book from the store (*pay*). Then, the bank verifies the payment of the customer (*verify*), and informs the store about the payment (*notify payment*). Upon receiving the payment, the store requests the delivery of the book from the courier (*request*). Finally, the courier delivers the book to the customer (*deliver*).

```
% payment
initiates(exec(pay(Customer, Bank, Item)), paid(Item), _).

% verification of payment
initiates(exec(verify(Bank, Customer, Item)), verified(Item), _).

% commitment for payment
create(exec(pay(Customer, Bank, Item)), Bank,
   c(Bank, Customer, property(e(Ts, Te), verified(Item))), Ts):-
   Te is Ts + 3.
```

Listing 2. Domain dependent $\mathcal{REC}$ rules for the customer.

There are three commitments among the parties that regulate their interactions:

– $C^c_{store,customer}$ $(paid, delivered)$: if the customer pays, the store commits to deliver.
– $C^c_{bank,customer}$ $(paid, verified)$: the customer uses a bank for payment.
– $C^c_{courier,store}$ $(requested, delivered)$: the store delivers via a courier.

Listing 2 shows the $\mathcal{REC}$ rules that describe the interaction between the customer and the bank. Fig. 2(b) shows the output of $\mathcal{REC}$ for a sample trace of the protocol. The

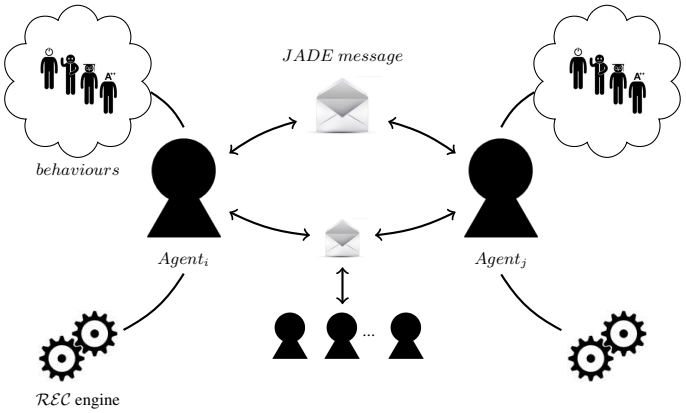


Fig. 3. Distributed architecture for contract execution and monitoring.

horizontal axis shows the events that happened throughout time, and the vertical axis demonstrates how fluents (i.e., predicates and commitments) evolve due to the events happened.

4 Multi-agent Simulation Architecture

$\mathcal{DIESECT}$ is a distributed agent-based architecture to simulate e-commerce contracts. Previous work has paid little attention on environments where agents can track down their contracts while they autonomously act within the environment and interact with each other. That is, there are either distributed agent platforms that do not deal with contract execution, or systems that support offline contract monitoring, i.e., given a trace of agent actions that accounts to a predefined scenario designed prior to execution. Among others, the most similar work to ours is that of Faci *et al.* 's [4]. Their focus is on normative multi-agent systems where contracts are described by a set of norms, while we deal with commitment-based protocols. In addition, they provide centralised entities for monitoring of contracts such as observer, manager, contract store. In contrast, execution in our system is fully distributed such that each agent monitors and verifies its own contracts using its partial knowledge of the environment.

Our proposal enables the simulation of distributed contract execution and monitoring for e-commerce protocols by following two directions: (i) we develop a fully distributed multi-agent system using JADE and provide agents with distinguished behaviours (e.g., strategies) that lead to different contract executions, (ii) we enable agents to reason on their contacts throughout execution using logic programming. Fig. 3 depicts the proposed multi-agent architecture. Agents are developed using JADE libraries and combined with logic programming capabilities. The underlying JADE infrastructure enables distributed execution and inter-agent communication (e.g., social aspect) while the powerful temporal reasoning capability allows the agent to perform reasoning on its commitments through time (e.g., individual aspect). Different agent behaviours can be associated with the roles in the protocol, leading to different contract outcomes,

as if the agents have a personality that affect how the agent acts during the protocol (e.g., a courier that always delivers on time) [15].

Agents in JADE communicate via messages. Messages correspond to executed actions among agents. For example, when the customer pays for a book, this corresponds to a message from the customer agent to the bank agent with the content payment. Actions have consequences in terms of changes in the environment: (i) they cause fluents to be initiated or terminated, e.g., the payment action causes the fluent *paid* to be initiated, (ii) they also caused related commitments to change states, e.g., the payment action causes the commitment of store to become active since its antecedent is satisfied. These are handled by the $\mathcal{REC}$ reasoner component of an agent. At certain points in time, the agent may run its $\mathcal{REC}$ engine to check the states of its commitments. In order to do so, the agent creates a trace of events using its message history. Each agent has a separate $\mathcal{REC}$ engine that it can run at any time throughout the execution. Thus, an agent operates only with partial information that it has gained through messages in the environment.

```
<simulation>
 <agents>
  <customer name="bob" eagerness="0.3" lateness="0.0">
   <wanteditems><product name="ipad"/></wanteditems>
  </customer>

  <store name="ebay" eagerness="0.0" lateness="0.0" bank="hsbc" courier="ups">
   <inventory>
    <product name="ipad" deliveryCost="5" price="450"/>
    <product name="iphone" deliveryCost="5" price="350"/>
   </inventory>
  </store>

  <bank name="hsbc" eagerness="0.0" lateness="0.0"/>

  <courier name="ups" eagerness="0.0" lateness="0.2"/>
 </agents>
</simulation>
```

Listing 3. A simulation profile.

A simulation can then be run in one of three modes:

- *Manual* mode: where the user increments the simulation clock and selects what actions the agents should perform at each timestep. Note that with this mode, we can simulate what has already been proposed by existing systems, e.g., test specific execution traces offline that would lead to different contract outcomes.
- *Simulation* mode: where agents schedule their actions to be executed at a specific timestep and perform them autonomously according to a customised simulation profile. A sample profile is given in Listing 3. Note that this mode can be used to test different agent behaviours online, and see how contract execution is affected.
- *Silent* mode: where the user again initiates the simulation by selecting a profile and the system carries it out automatically. In this mode, the interface is not shown but rather text-based statistics are logged after the simulation is finished. We use this mode to evaluate performance.

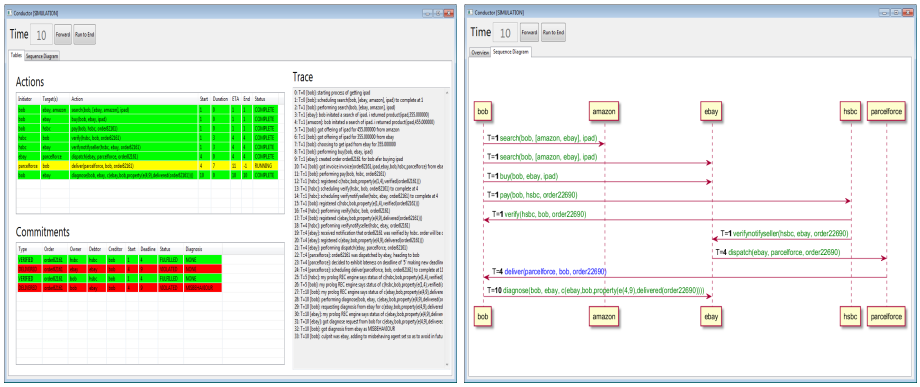


Fig. 4. $\mathcal{DIESECT}$ simulation panel.

The manual and simulation modes provide a graphical demonstration of how the protocol is executed. Fig. 4 demonstrates this simulation panel after the execution is started. The current simulation time is displayed at the top left. The user can press the "Forward" button to advance simulation time. In simulation mode, the user can press the "Run to End" button to make the simulation cycle through each timestep until the end of it.

The "Actions" panel shows running actions in yellow and completed in green, while the "Commitments" panel shows active commitments in yellow, fulfilled commitments in green and violated commitments in red. The status column of the "Commitments" panel shows the status of the commitment that the agent's $\mathcal{REC}$ engine has determined[3].

The sequence diagram shows the ordering of agents' actions through time. Completed actions are represented in green text, while running actions are represented with blue. If operating in manual mode, the user may either click a blue action text on the sequence diagram to trigger its completion, or double click the action in the top table. Note that the underlying JADE environment also allows us to utilise the *sniffer* agent which helps debug and diagnose certain exceptions regarding the messaging of agents, see Fig. 5.

5 Experimental Evaluation

We carry out two sets of experiments to test the performance of our agents: (i) with increasing number of agents, (ii) with different agent behaviours. We run simulations in silent mode on an Intel Core2 Quad 2.40 GHz computer with 4 GB of memory running Windows 7 64-bit OS. We repeat each experiment five times and record the average statistics for (i) the time it takes for the agents to complete the simulation, and (ii) the peak memory usage of the system.

[3] The complete implementation with sample simulation profiles for $\mathcal{DIESECT}$ can be downloaded from `http://dice.cs.rhul.ac.uk/article.php?id=7`.

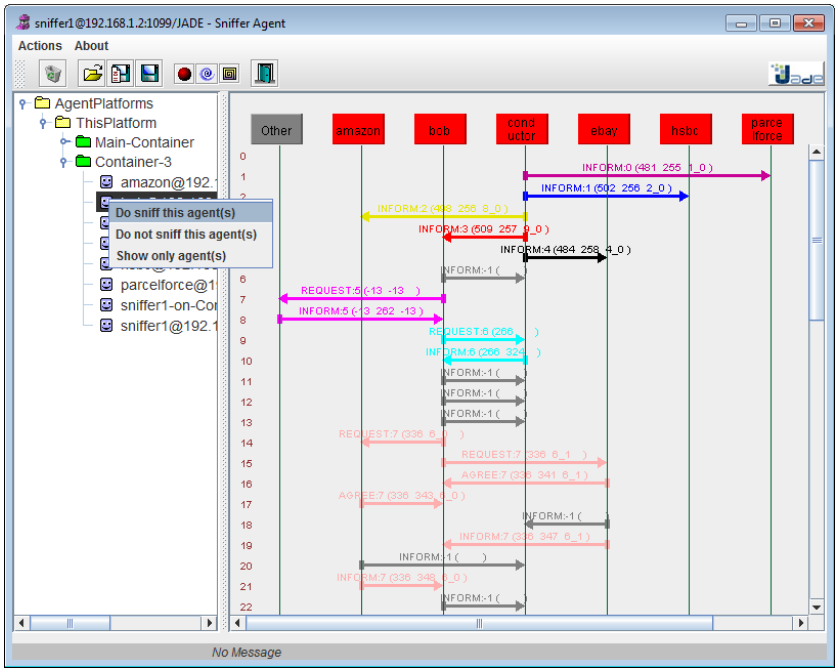


Fig. 5. JADE sniffer agent used in $\mathcal{DIESECT}$.

5.1 Increasing Agents

For the first set of experiments, we gradually increase the number of agents to test how it affects the system's overall performance. Fig. 6 shows the performance results for 10 to 140 customer agents with the addition of two store agents, one bank agent and one courier agent. We record the time it takes in seconds to complete simulation, and the peak memory usage in megabytes. It can be seen that there is a linear increase for memory usage, and the time requirements stay within reasonable values for a considerable number of agents executing an e-commerce protocol. Note that these results are compatible with the performance of $\mathcal{REC}$ discussed in [16].

5.2 Different Agent Behaviours

For the second set of experiments, we take 30 customer agents (again with the addition of other agents as above), and change the simulation profile by assigning different behaviours to the agents. Fig. 7 shows the performance results for changing behaviours of each agent type. It can be seen that there is no significant difference in performance, and the results are compatible with the previous one with 30 customer agents.

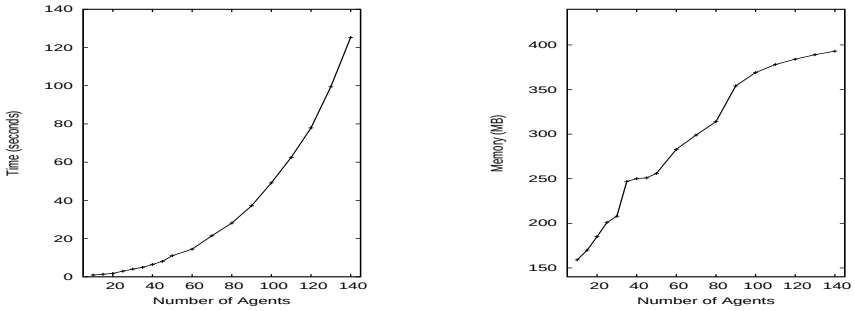


Fig. 6. Performance of $\mathcal{DIESECT}$ for increasing number of agents. The figure on the left shows time it takes to complete simulation while the figure on the right shows peak memory usage.

Agent behaviour	Time (s)	Memory (mb)
Customer eagerness 100%	5.62	218
Customer eagerness 50%	4.63	207
Bank lateness 100%	3.87	208
Bank lateness 50%	4.02	211
Store lateness 100%	3.41	217
Store lateness 50%	4.30	213
Courier lateness 100%	3.98	208
Courier lateness 50%	4.27	210

Fig. 7. Performance of $\mathcal{DIESECT}$ for different agent behaviours. The time and memory values are recorded for full and half eagerness / lateness.

6 Discussion

In this paper, we have presented $\mathcal{DIESECT}$ to provide a distributed simulation environment for contract execution and monitoring. Contracts have been discussed extensively in the literature in the context of business workflows [17, 1, 18], modeling, execution and exceptions [19, 7, 2], and ontologies [20]. However, most of these work have either approached contract monitoring in a centralised manner ignoring the distributed aspect of open systems where the contents of a contract should be kept private among its participants, and thus be managed individually by each agent, or they have failed to the take into account the relation between agent autonomy and contract outcomes in their systems. Here, we present a simulation environment where the autonomous behaviour of an agent may lead to different contract outcomes during execution.

Commitments are proven to be effective in modeling multi-agent interactions [6, 3]. In central monitoring systems, tracking the states of individual commitments is an effective way to detect protocol exceptions [8], since all the interactions of agents are observable. However, this is not a valid assumption for realistic e-commerce scenarios. In our system, each agent has forms a partial view of its environment via interacting with other agents through JADE messages.

Normative multi-agent systems are an alternative to commitment-based protocols, where artificial institutions and organisations are modeled via norms rather than commitments [21, 22, 4]. Similar to commitments, norms represent obligations for agents to follow, but they also possess additional properties like power, which is needed to represent the hierarchical behaviour in organisations, e.g., whether an agent possessing a certain role can enforce a norm. In this paper, we do not consider power or the hierarchy among agents when managing commitments.

We have shown via two sets of experiments that (i) our systems performs well under increasing number of agents (with a linear increase in memory usage and reasonable simulation times), and (ii) the changing of agent behaviours does not have a significant effect on the system's performance. We plan to extend $\mathcal{DIESECT}$ with the following possible extensions:

– We aim for a generic contract execution and monitoring environment where protocols can be described by defining commitment templates and the associated agent roles. We are currently working on this direction so that new e-commerce protocols can be created and tested in our platform.
– We use $\mathcal{REC}$ as the current reasoning mechanism for agents to detect and diagnose commitment violations. Another interesting direction for contract execution is to predict that a commitment might be violated in the future. One powerful tool for such prediction is model checking [23, 9]. Model checking is based on temporal logic, and creates possible future worlds given an initial world model and a set of transition rules. We plan to integrate the model checking capability besides $\mathcal{REC}$ into the agents' reasoning. By doing so, we could report on the soundness of the system, i.e., whether commitments reach their final states.
– We plan to extend our performance evaluation by distinguishing between the time spent on JADE side and the time spent for executing $\mathcal{REC}$. This will provide insight on how to improve the system's overall performance, e.g., agents might not execute $\mathcal{REC}$ at each timestep. We also plan to run experiments with larger agent populations and report the communication costs among agents.

References

1. Krishna, P.R., Karlapalem, K., Chiu, D.K.W.: An erec framework for e-contract modeling, enactment and monitoring. Data Knowl. Eng. **51**(1) (October 2004) 31–58
2. Kafalı, Ö., Torroni, P.: Exception diagnosis in multiagent contract executions. Annals of Mathematics and Artificial Intelligence **64**(1) (2012) 73–107
3. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: Applying event calculus planning using commitments. In: Proceedings of the 1st International Conference on Autonomous Agents and Multiagent Systems (AAMAS). (2002) 527–534
4. Faci, N., Modgil, S., Oren, N., Meneguzzi, F., Miles, S., Luck, M.: Towards a monitoring framework for agent-based contract systems. In: Proceedings of the 12th international workshop on Cooperative Information Agents XII. CIA '08, Berlin, Heidelberg, Springer-Verlag (2008) 292–305
5. McGinnis, J., Stathis, K., Toni, F.: A formal model of agent-oriented virtual organisations and their formation. Multiagent and Grid Systems **7**(6) (2011) 291–310

6. Singh, M.P.: An ontology for commitments in multiagent systems: Toward a unification of normative concepts. Artificial Intelligence and Law **7** (1999) 97–113
7. Kafalı, Ö., Yolum, P.: A distributed treatment of exceptions in multiagent contracts. In: Proceedings of the 9th International Workshop on Declarative Agent Languages and Technologies (DALT). (2011)
8. Chesani, F., Mello, P., Montali, M., Torroni, P.: Commitment tracking via the reactive event calculus. In: Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI). (2009) 91–96
9. El Menshawy, M., Bentahar, J., Qu, H., Dssouli, R.: On the verification of social commitments and time. In: Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). (2011) 483–490
10. Bellifemine, F., Poggi, A., Rimassa, G., Turci, P.: An object-oriented framework to realize agent systems. In: WOA Workshop: From Objects to Agents. (2000) 52–57
11. Chopra, A.K., Singh, M.P.: Multiagent commitment alignment. In: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). (2009) 937–944
12. Chesani, F., Mello, P., Montali, M., Torroni, P.: Monitoring time-aware social commitments with reactive event calculus. In: 20th European Meeting on Cybernetics and Systems Research, 7th International Symposium "From Agent Theory to Agent Implementation" (AT2AI-7). (2010) 447–452
13. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing **4**(1) (1986) 67–95
14. Malone, T.W., Crowston, K., Herman, G., eds.: Organizing Business Knowledge: The MIT Process Handbook. MIT Press, Cambridge, MA (2003)
15. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: Declarative agent control. In Leite, J.A., Torroni, P., eds.: Computational Logic in Multi-Agent Systems, 5th International Workshop, CLIMA V, Revised Selected and Invited Papers. Volume 3487 of Lecture Notes in Computer Science., Springer (2005) 96–110
16. Bragaglia, S., Chesani, F., Mello, P., Montali, M., Torroni, P.: Logic programs, norms and action. Springer-Verlag, Berlin, Heidelberg (2012) 123–146
17. Grefen, P., Aberer, K., Ludwig, H., Hoffner, Y.: Crossflow: Cross-organizational workflow management in dynamic virtual enterprises. International Journal of Computer Systems Science & Engineering **15** (2000) 277–290
18. Urovi, V., Stathis, K.: Playing with agent coordination patterns in MAGE. In: Coordination, Organizations, Institutions and Norms in Agent Systems V, COIN 2009 International Workshops. Revised Selected Papers. Volume 6069 of Lecture Notes in Computer Science., Springer (2010) 86–101
19. Molina-jimenez, C., Molina-jimenez, C., Shrivastava, S., Shrivastava, S., Solaiman, E., Solaiman, E., Warne, J., Warne, J.: Contract representation for run-time monitoring and enforcement. In: In Proc. IEEE Int. Conf. on E-Commerce (CEC, IEEE (2003) 103–110
20. Grosof, B.N., Poon, T.C.: Sweetdeal: Representing agent contracts with exceptions using xml rules, ontologies, and process descriptions, ACM Press (2003) 340–349
21. Sadri, F., Stathis, K., Toni, F.: Normative KGP agents. Computational & Mathematical Organization Theory **12**(2-3) (2006) 101–126
22. Fornara, N., Colombetti, M.: Specifying and enforcing norms in artificial institutions. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS). (2008) 1481–1484
23. Kafalı, Ö., Günay, A., Yolum, P.: $\mathcal{PROTOSS}$: A run time tool for detecting $\mathcal{PR}$ivacy vi$\mathcal{O}$la$\mathcal{T}$ions in $\mathcal{O}$nline $\mathcal{S}$ocial network$\mathcal{S}$. In: IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. (2012)