On the Cost of Agent-awareness for Negotiation Services

Andrea Giovannucci[1] and Juan A. Rodríguez-Aguilar[1]

Artificial Intelligence Research Institute, IIIA
Spanish Council for Scientific Research, CSIC
08193 Bellaterra, Barcelona, Spain
{andrea,jar}@iiia.csic.es

Abstract. Significant advances in the development of agent technology have spurred the development of agent-oriented information systems (AOIS). Nonetheless, accounts on the benefits and shortcomings of state-of-the-art agent technology when employed for the deployment of AOIS for electronic commerce are scant. The purpose of this work is to report on a case study that attempts at shedding some light on this matter.

1 Introduction

While a significant number of agent-based applications for electronic commerce has been presented to the agent community during the last years, little attention has been devoted to analysing the practical benefits and shortcomings of agent technology when applied to such domain. To the best of our knowledge little effort has been devoted to study the applicability of state-of-the-art agent technology to develop actual-world e-commerce applications. In particular, we believe that it is necessary to assess the computational cost added by agent technology in this type of applications so that we can diagnose the improvements required by state-of-the-art agent technology.

For this purpose we report on a case study that intends to shed some light on this matter. We depart from *iBundler* (fully described in [1]), an agent-aware negotiation service for combinatorial negotiations designed to be employed as: (1) an open agent platform within the Agentcities.RDT[1] (http://www.agentcities.org/EURTD) project that could be discovered, communicate, and offer services to any FIPA compliant agent (http://www.fipa.org); (2) an agent façade to *Quotes*[2], a commercial negotiation tool, to allow for the participation of third-party business agents in actual-world procurement events. In both cases, our aim has been to study the computational cost of agent awareness for the *iBundler* negotiation service so that its users are aware of the type of negotiation scenarios that *iBundler* can acceptably handle when buying and providing agents are involved. This exercise has also included the determination of those general or domain-dependent measures that can help reduce the cost of the service.

At this aim, we have measured the performance in time and memory of *iBundler* through a wide range of artificially generated negotiation scenarios. For each scenario we sampled at several stages both the time and memory that *iBundler* employed to handle it. We have interestingly observed that the management of ontologies is a rather

[1] The Agentcities.RDT project's objectives were to create an on-line, distributed test-bed to explore and validate the potential of agent technology for future dynamic service environments.

delicate issue that actually causes a significant overload. Furthermore, we have also observed that the design of highly expressive, compact bidding languages can definitely help cut down the computational cost for any agent-aware negotiation service considering combinatorial scenarios.

The paper is organised as follows. First, section 2 briefly reviews the literature concerning scalability and applicability of agent technology. Section 3 succinctly introduces *iBundler*. Section 4 deals with the description of the evaluation scenarios arranged to evaluate *iBundler*. In section 5 we present and thoroughly discuss the test results. Finally, section 6 discusses some conclusions deriving from the results' analysis.

2 Related work

The applicability analysis of agent technology in the literature primarily focuses on scalability issues as robustness, system performance with large populations of agents and ontology engineering. Brazier et al. [3] address the problem of scalability in naming services and location services. Besides, they analyse the concept of scalability in multi agent systems (MAS) and discuss scalability for many existing multi-agent frameworks. Deters [4] studies the problems derived from large number of agents running in a MAS, agent resource consumption, the exchange of great number of messages, identifying agent hosting and message routing as bottle-necks. Furthermore, he performs some scalability experiments. An important result in [4] is that the main deficiencies of JESS (http://herzberg.ca.sandia.gov/jess/) derive from serialisation processes. Kahn investigates how timing of sequential agent registration and lookup varies as the total number of registered agents increases in COABS [5]. The works in [6] and [7] analyse robustness and fault tolerance, whereas [8] exemplifies ad-hoc, domain-dependent agent technology scaling techniques. On the other hand, the literature on ontology scalability focuses on three major issues: the size of ontology contents, the complexity of ontology construction and knowledge re-usability ([9], [10]). In particular, Jarrar states that experience shows that ”unscalable solutions emerging from academic research often fails at the industrial level” [9].

Thus, we believe that it is an urging necessity to report on practical deployments of actual-world agent-based applications in order to: (1) progressively derive best methodological practices; and (2) assess the improvements required by state-of-the-art agent technologies to be adopted at industry level. Particularly since much of the research effort on agent technology does not consider the application of widely employed agent frameworks and programming tools to real-world problems.

We consider *iBundler* as representative of the main trends on the state-of-the-art agent programming tools and platforms. Firstly, because it is based on the FIPA specification standard, that is surely the most widely adopted by the agent community[2]. Secondly, the considerations emerging from the experiments derived in this paper are

[2] OGM (www.ogm.org) is another standardisation effort based on CORBA IDL interface. This solution is efficient for agent migration and client-server applications, but less suitable than FIPA-compliant platforms for peer-to-peer applications. For an interesting comparison refer to [11].

related to the FIPA nature of the agent platform, not to a particular JADE implementation. Thus, the results in section 5 are not limited to the JADE framework, being valid for all the FIPA-compliant agent frameworks.

3 *iBundler* An Agent-aware Negotiation Service

Consider the problem faced by a buying agent when negotiating with providing agents. In a negotiation event involving multiple, highly customisable goods, buying agents need to express relations and constraints between attributes of different items. Moreover, it is common practice to buy different quantities of the very same product from different providing agents, either for safety reasons or because offer aggregation is needed to cope with high-volume demands. This introduces the need to express business constraints on providing agents and the contracts they may have assigned. Not forgetting the provider side, providing agents may also wish to impose constraints or conditions over their offers. These may be only valid if certain configurable attributes (e.g. quantity, delivery days) fall within some intervals, or assembly and packing constraints need to be considered. Once a buying agent collects all offers, he is faced with the burden of determining the winning offers. It would be desirable to relieve buying agents from solving such a problem. *iBundler* is an agent-aware decision support service that makes headway in this direction by acting as a combinatorial negotiation solver (solving the winner determination problem) for both multi-item, multi-unit negotiations and auctions. Thus, the service can be employed by both buying agents and auctioneers in combinatorial negotiations and combinatorial reverse auctions[12] respectively. To the best of our knowledge, *iBundler* represents the first agent-aware service for multi-item negotiations, since agent services have mostly focused on infrastructure issues related to negotiation protocols and ontologies.

The *iBundler* service has been implemented as an agency composed of agents that cooperatively interact to offer a negotiation support service. A fundamental aspect of *iBundler* is that it was not only intended as a stand-alone agent-aware service. *iBundler* was also designed to become the agent façade of the commercial sourcing tool *Quotes* [2] with the aim of providing a higher level of automation to external parties. In this manner, the negotiations run through *Quotes* allow for the participation of both human and software buyers and providers. However, while human buyers and providers negotiate via web-based interfaces, buying and providing agents owned by third parties can also negotiate through the service whenever they incorporate protocols and the ontology required by *iBundler*. In this work we do not address security issues, such as buyers and providers trusting a central server. It could be considered as a next step in the deployment of an actual-world negotiation service.

Figure 1 depicts the components of the *iBundler* agency (along with the fundamental connections of buying and providing agents with the service):
[Logger agent]. It manages the access to the *iBundler* agency from outside.
[Manager agent]. Agent devoted to providing the solution of the problem of choosing the set of bids that best matches a user's requirements. There exists a single Manager agent per user (buyer or auctioneer), created by the Logger agent, offering the following services: brokering service to forward buyersŕequirements (RFQs) to selected providers

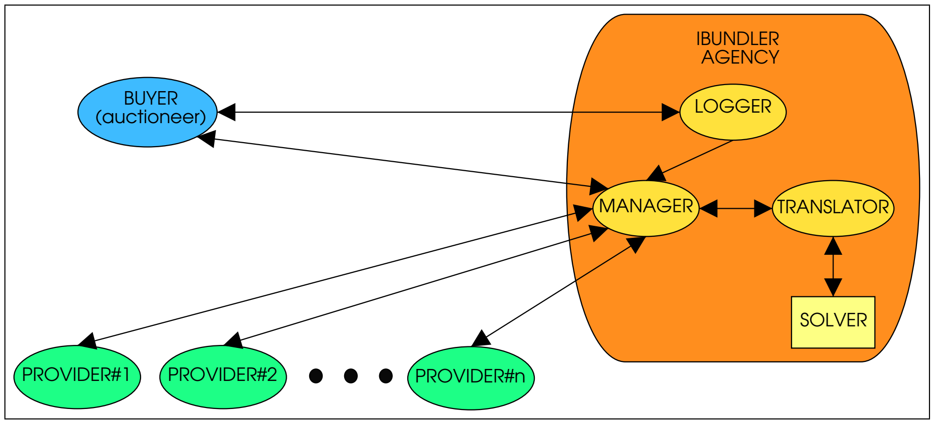

Fig. 1. Architecture of the *iBundler* Agency

capable of fulfilling them; collection of bids; winner determination in a combinatorial negotiation/auction; and award of contracts on behalf of buyers. Furthermore, the manager agent is also responsible for: bundling each RFQ and its bids into a negotiation problem in FIPA-compliant format to be conveyed to the Translator agent; and to extract the solution to the negotiation problem handled back by the Translator agent.

[Translator agent]. It creates a representation of the negotiation problem in a format understandable by the Solver departing from the FIPA-compliant description received from the Manager. It also translates the solution returned by the Solver into an object of the ontology employed by user agents.

[Solver component]. The *iBundler* component itself extended with the offering of a language for expressing offers, constraints, and requirements. The specification is parsed into a Mixed Integer Programming (MIP) formulation and solved using available MIP solvers (a version using ILOG CPLEX; and another version using using a Java MIP modeller that integrates the GNU (www.gnu.org) Programming Kit GLPK. The Solver component is complete in the meaning that if an optimal solution exists, it will find it. If the problem has a set of Pareto-optimal, equivalent solutions, the solver component will return only one solution, which one depending on the underlying branch-and-bound algorithm ([13]).

Our design manages to separate concerns among the three members of the agency. On the one hand, the Manager is strictly devoted to coordination. It represents the façade of the service. Besides, since every negotiation requested by a buyer makes the agency create an instance of the *Manager*, the service can cope with asynchronous and multiple accesses to the service. The Translator agent is in charge of relieving both Managers and Solver from the burden of translating FIPA-compliant specifications into the language required by Solver. Notice that the fact of having only one Translator agent represents a bottle-neck in the overall process when many buyers access the service concurrently. Such limitation could be overcome by creating multiple instances of Translator Agents and Solvers on different machines. Anyway in this work we focused on the service performances in managing big size negotiation scenarios, not on multiple concurrent accesses to the service. We leave such issue as a possible future development.

Figure 2 depicts the interaction protocol involved in the interplay of buyers and provides with *iBundler*. It is expressed in AUML (Agent Unified Modelling Language)[14] following the FIPA interaction protocol library specification compiled in [15]. Observe that the specification in figure 2 involves four roles, namely: buyer, manager, translator, and provider. Whereas multiple agents can act as providers, the remaining roles can be uniquely adopted by a single agent each. Notice too that the *iBundler* interaction protocol is composed of several interleaved interaction protocols:

[IP-RFQ] Held between a buyer and the manager agent created by the Logger agent after registration. The buyer delivers an RFQ to his manager agent requesting to obtain the optimal set of offers from the available providers. In case it is not possible to obtain a solution to the problem, the received response is an empty bid set.

[IP-CFP] Prior to delivering the optimal set of offers, the manager interacts with the available providers to request their offers under the rules of this CFP interaction protocol. If no offers are received the manager refuses to deliver the optimal set of offers in the context of the IP-RFQ interaction protocol. Otherwise, the manager agrees on providing the service and proceeds ahead by starting out an instance of the IP-Request-Solution interaction protocol. The protocol winds up with the notification of contract awards to selected providers according to the buyer's decision. In the case in which no optimal solution could be found, the buyer is sent an empty bid set and the IP-CFP protocol is ended communicating a Reject-Proposal to each provider involved. Notice that the manager mediates between buyer and providers.

[IP-Request Solution] This interaction protocol held between the manager and the translator agent within the *iBundler* agency aims at calculating the optimal set of offers considering the offers submitted by providers, along with the buyer's requirements and constraints. The result delivered by the translator is further conveyed by the manager to the buyer in the context of the interleaved IP-RFQ interaction protocol.

[IP-AWARD] At the end of the IP-RFQ interaction protocol the buyer obtains the optimal set of offers. He may request also to receive all offers. Thereafter, if the buyer received a non-empty optimal set of offers ($k > 0$ in figure 2), the buyer initiates the IP-AWARD interaction protocol in order to request the manager to award contracts to selected providers. Observe that the contract award distribution is autonomously composed by the buyer, and thus the buyer may decide to either ignore or alter the optimal set.

iBundler's ontology is founded on the following core concepts: *RFQ*, *ProviderResponse*, *Problem*, and *Solution*. As an example, figure 3 depicts -as shown by the Ontoviz Protégé plug-in (http://protege.stanford.edu)- the *Problem* ontological concept. The RFQ concept is employed by buying agents to express their requests for bids (via request in IP-RFQ). An *RFQ* is composed of a sequence of *Request* concepts, one per requested item along with the buyer's business rules expressed as constraints. On the provider side, providers express their offers in terms of the *ProviderResponse* concept (via a propose in IP-CFP), which in turn is composed of several elements: a list of *Bid* concepts (each Bid allows to express a bid per either a single requested item or a bundle of items) along with; constraints on the production/servicing capabilities of the bidding

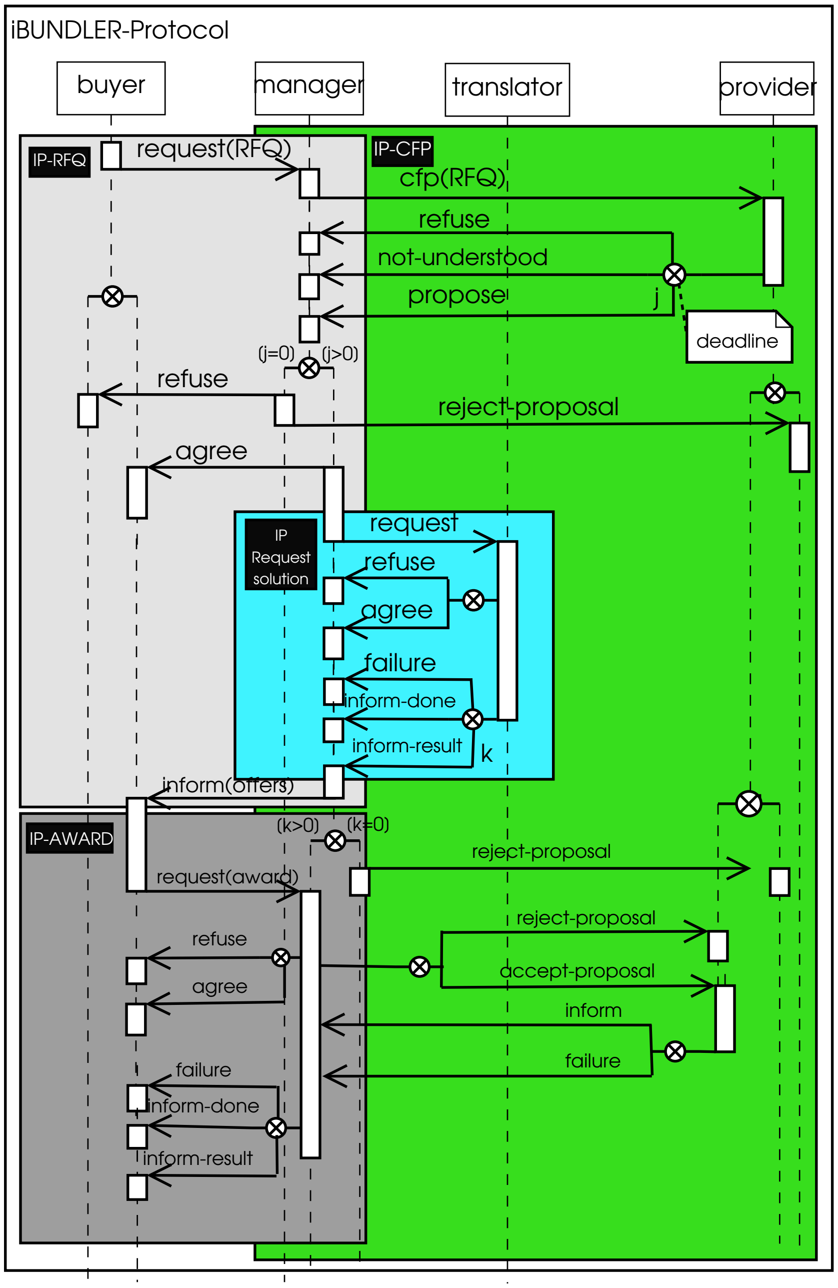

Fig. 2. *iBundler* Interaction Protocol

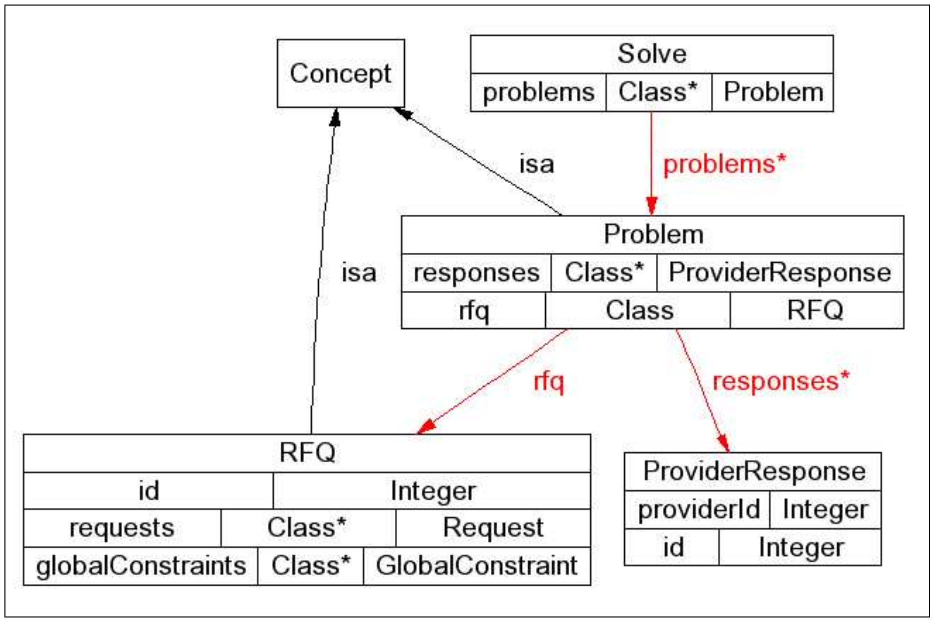
Concept
Solve
problems | Class* | Problem
isa
problems*
Problem
responses | Class* | ProviderResponse
rfq | Class | RFQ
isa
rfq
responses*
RFQ
id | Integer
requests | Class* | Request
globalConstraints | Class* | GlobalConstraint
ProviderResponse
providerId | Integer
id | Integer

Fig. 3. Problem concept

provider (*Capacity* concept); and constraints on bundles of bids formulated with the *BidConstraint* concept.

Once the manager agent collects all offers submitted by providers, he wraps up the *RFQ* concept as received from the buyer along with the offers as *ProviderResponse* concepts to compose the negotiation problem to be solved by the Solver component (via request in IP-Request-Solution). Finally, the solution produced by the Solver component is transformed by the translator agent into a *Solution* concept, that is handed over to the manager (via inform-result in IP-Request-Solution). The *Solution* concept contains the specification of the optimal set of offers calculated by Solver. Thus *Solution* contains a list of *SolutionPerProvider* concepts, each one containing the bids selected in the optimal bid set per provider, as a list of *BidSolution* concepts, along with the provider's agent identifier, as an *AID* concept. Each *BidSolution* in turn is composed of a list of *BidItemFixed* concepts containing the number of units selected per bid along with the bid's total cost.

4 Evaluation Scenario

In this section we detail the way we conducted our evaluation. Firstly, we describe how to generate artificial negotiation scenarios for testing purposes. Next, we detail the different stages considered through our evaluation process.

4.1 Artificial Negotiation Scenarios

In order to evaluate the agent service performance, the times needed by *iBundler* to receive an RFQ from a *Buyer* agent and to collect the different bids from providers is considered of no interest. Because they depend on some uncontrolled variables (e.g. the time needed by providers to send their bids and the network delay). Thus, our evaluation starts from the moment at which all the required data (RFQ and bids) are available to the *Manager* agent. We tried to simulate such an ideal situation generating multiple datasets in separate files, each one standing for a different input negotiation problem composed of FIPA messages, each one containing both an RFQ and the bids received as a response to this. In this way we can use the file stream as if it was the incoming message stream, and perform all the subsequent message manipulation as if the message had been received from a socket.

Another important consideration has to do with the way we sampled time and memory. We established checkpoints through the process carried out by *iBundler* when solving a negotiation problem. Such checkpoints partition the process into several stages. We observed time and memory at the beginning and at the end of these stages.

In order to automate the testing it was necessary to develop a generator of artificial negotiation scenarios involving multiple units of multiple items. The generator is fed with mean and variance values for the following parameters: *number of providers* participating in the negotiation; *number of bids per provider* (number of bids each provider sends to the *Manager* agent); *number of RFQ items* (number of items to be negotiated by the *Buyer* agent); *number of items per bid* (number of items within each bid sent by a provider); *number of units per item per bid*; and *bid cost per item*. In this first experimental scenario we did not generate neither inter-item nor intra-item constraints.

The generator starts by randomly creating a set of winning combinatorial offers. After that, it generates the rest of bids for the negotiation scenario employing normal distributions based on the values set for the parameters above. Thus, in some sense, the negotiation scenario can be regarded as a set of winning combinatorial bids surrounded by noisy bids (far less competitive bids). Notice that the generator directly outputs the RFQ and bids composing an artificial negotiation scenario in FIPA format. In this manner, both RFQ and bids can be directly fed into *iBundler* as buyers' and providers' agent messages.

We have analysed the performance of *iBundler* through a large variety of negotiation scenarios artificially generated by differently setting the parameters above. The data representing each negotiation scenario are saved onto a file, named by a string of type A.B.C.D, where A stands for the number of providers, B stands for the number of bids per provider, C stands for the number of RFQ items, and D stands for the number of items per bid. For instance, *250.20.100.20* represents the name of a dataset generated for 250 providers, 20 bids per provider, 100 RFQ items, and 20 items per bid.

The artificial negotiation scenarios we have generated and tested result from all the possible combinations of the following values:
Number of providers: 25, 50, 75, 100
Number of bids per provider: 5, 10, 15, 20
Number of RFQ items: 5, 10, 15, 20
Number of items per bid: 5, 10, 25, 50

4.2 Evaluation Stages

In order to introduce the evaluation stages that we considered, it is necessary to firstly understand how JADE manipulates messages and ontological objects. In particular we summarise the process of sending and receiving messages (for a complete description refer to the JADE documentation). Figure 4 graphically summarises the activities involved in sending and receiving messages. In the figure, the squared boxes represent data, whereas the rounded boxes represent processes.

JADE agents receive messages as serialised objects in string format. JADE decodes the string into a Java class, the *ACLMessage* JADE class (which represents a FIPA ACL Message). One of these class fields is the *content* field, which usually contains either the action to be performed or the result of a performed action. Next, JADE extracts the content of the message. The content is once more a string, on which JADE needs to perform an ontology check to decode it. As a result, a Java object representing the ontological object is built upon the *content* field, guaranteeing that the ontological structure is not violated.

As to the dual case, i.e. when a JADE agent sends a message, the process works the other way around. JADE encodes the ontological object representing the communication content into a string, that sets the *content* field of the *ACLMessage* class. During this process JADE verifies that the message content matches perfectly with an ontology object. Once the *content* field is set, the agent sends the message: the ACLMessage class is decoded into a string that is sent through a socket.

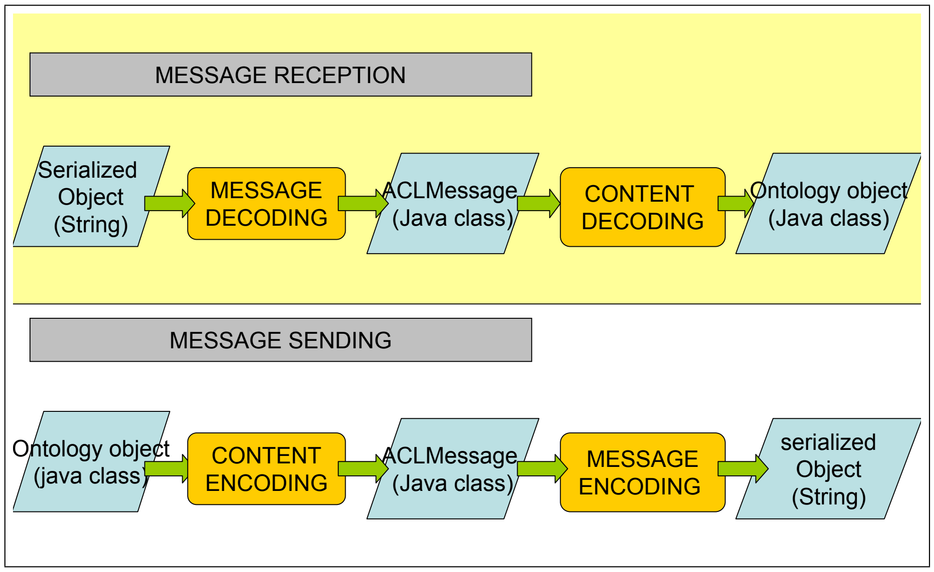

Fig. 4. Message life cycle in JADE

Considering the process above, we sampled both the time and memory use through the following stages of the *iBundler*'s solving process:
Δ**t1**: JADE decodes all the FIPA messages contained in the data set file containing the input negotiation problem, converting them into instances of the *ACLMessage* Java class. Δ**t2**: the *Manager* agent composes the problem by creating an instance of the *Problem* Java ontology class and setting its fields after merging the RFQ and the collected bids. Δ**t3**: the ACLMessage to be sent to the *Translator Agent* is filled with the Java class representing the *Problem* ontology class. At this stage an ontology check occurs.
Δ**t4**: the above-mentioned ACLMessage is now encoded by the *Manager* agent, and subsequently sent to the *Translator* agent through a socket. Once received, the *Translator* agent decodes it into an *ACLMessage* class.
Δ**t5**: the *Translator* agent extracts from the received message the *Problem* ontology class containing the *RFQ* and all the collected *Bids*. Another ontology check occurs.
Δ**t6**: this stage is devoted to the transformation of the *Problem* ontology class into a matrix-based format to be processed by the *Solver* component.
Δ**t7**: at this stage the *Solver* component solves the MIP problem using ILOG CPLEX.
Δ**t8**: the output generated by *Solver* in matrix-based format is decoded by the *Translator* agent into the *Solution* ontology class.
Δ**t9**: the *Translator* agent fills the response message with the *Solution* ontology class, encodes the corresponding *ACLMessage* class, and sends it. Then, the *Manager* agent decodes the message upon reception.
Δ**t10**: the *Manager* agent extracts the *Solution* concept from the received *ACLMessage* with a last ontology check.
Δ**t11**: the solution is decomposed into different parts, one per provider owning an awarded bid.
Δ**t12**: the solution containing the set of winning offers is sent from the *Manager* agent to the *Buyer* agent. Note that this object is small with respect to the original problem since it only contains the winning bids.

5 Evaluation

In this section we give a quantitative account of the tests we run. Firstly, in section 5.1, we analyse time performance, and secondly, in section 5.2 the memory use for all the evaluation stages described above. In order to run our tests we employed the following technology: a PC with a Pentium IV processor, 3.1 Ghz, 1 Gbyte RAM running a Linux Debian (kernel v.2.6) operating system (http://www.debian.org); Java SDK 1.4.2.04 (http://java.sun.com); JADE v2.6; and ILOG CPLEX 9.0 (http://www.ilog.com).

5.1 Time performance

Next we show the variation in time performance per stage by varying the different degrees of freedom available to create an artificial negotiation scenario. In particular, we consider the following types of negotiation scenarios:

100.20.100.X: the number of items contained in a single bid varies (where X takes on the 5,10,25, and 50 values).

100.X.100.50: the number of bids each provider sends varies (where X takes on the 5,10,15, and 20 values).

X.20.100.50: the number of providers varies (where X takes on the 25,50,75, and 100 values).

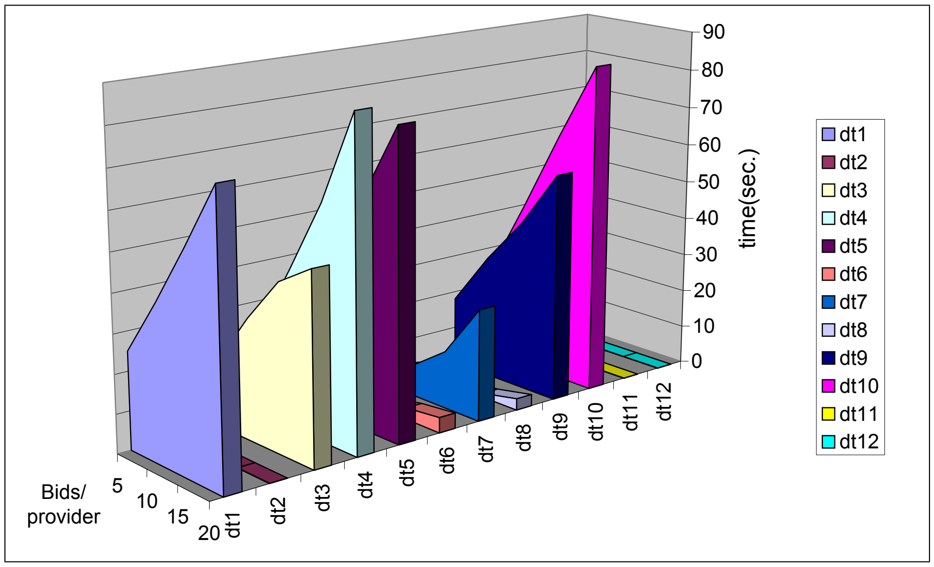

Fig. 5. Time measures when varying the number of bids per provider.

Figure 5 depicts the time spent in each of the described stages, considering different number of bids per provider. We experimented similar trends varying the number of items and the number of providers[3]. These results suggest that the variables' sensitivity is similar in all cases, i.e. varying the *number of items per bid*, the *number of providers* or the *number of bids per provider* leads to similar trends. Therefore, the stages that are more time-consuming are quite the same in every possible configuration: for instance, stage Δt10 is always the most time consuming, no matter the parameter being varied. Moreover, we can observe similar trends for the rest of stages (from Δt1 to Δt10). Hence, it seems that the time distribution along the different stages can be regarded as independent from the parameter setting.

Figure 6 illustrates the average percentage, over all the performed trials, of the total time that each stage consumes. We observe that: (1) The **Δt1**, **Δt3**, **Δt4**, **Δt5**, **Δt9**,

[3] The way the times vary when increasing those parameters is not linear. Nonetheless we did not deeply study this aspect, because the main issue for us was to assess the difference of these times with respect to the solver component time by itself.

Δt10 stages are the most time-consuming (92% of the total time). Since these stages involve ontology checking and message encoding and decoding, we can conclude that these activities are a bottle-neck. (2) The solver time (Δt7) is almost a negligible part of the total time. (3) Manipulating classes (stages Δt2, Δt6, Δt8 and Δt11) and solving the combinatorial problem (Δt7) is not as time-consuming as encoding and decoding messages and ontology objects.

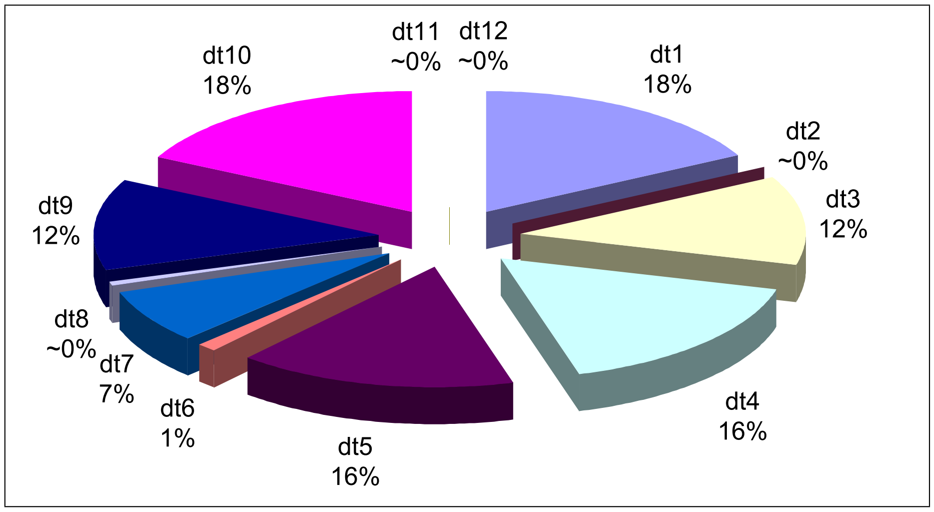

Fig. 6. Average times spent at the different evaluation stages.

Figure 8 depict the accumulated time spent on all stages for a collection of negotiation scenarios, which we refer to as the *total time*. More precisely, figure 8 depicts configurations whose total time lies between 30 and 50 seconds. It is conceivable to regard them as the edge values, although it is a very arbitrary matter. Some observations follow from analysing the figures above:

1. The agent-awareness of *iBundler* is costly. We observe that the percentage of total time employed to solve the winner determination problem is small with respect to agent related tasks.
2. Using the solver component we can easily solve problems of more than 2000 bids in less than one minute, whereas the agent service can handle in reasonable time less than 750 bids.
3. Therefore, small, and medium-size negotiation scenarios can be soundly tackled with *iBundler*. Nonetheless, time performance significantly impoverishes when handling large-size negotiation scenarios.

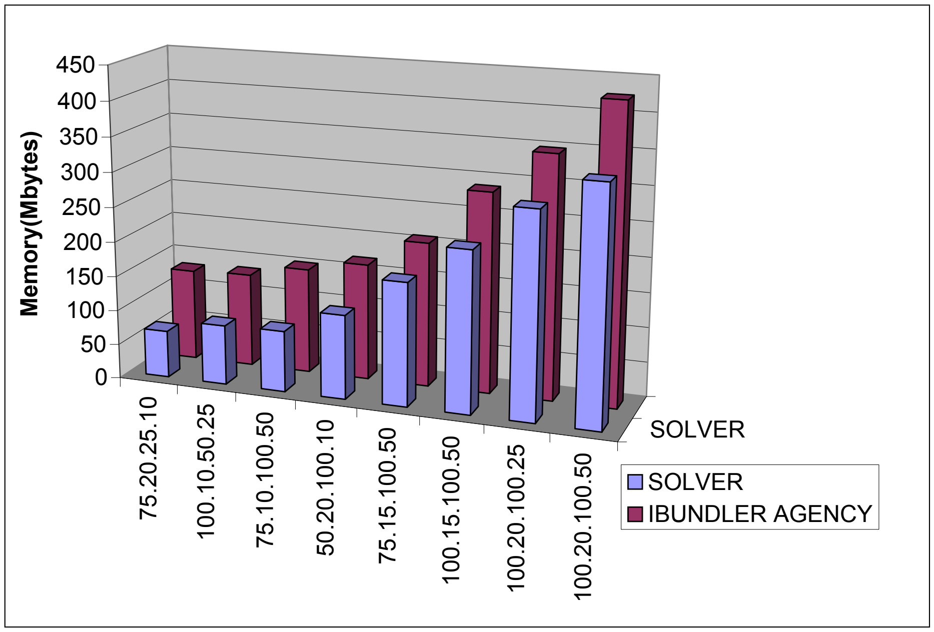

Fig. 7. Memory consumption.

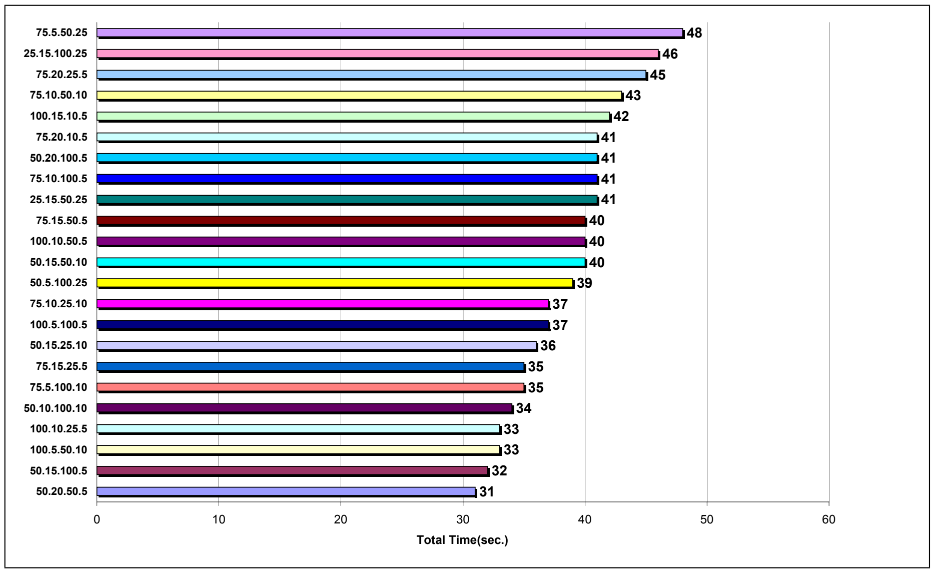

Fig. 8. Time performance for negotiation scenarios on the edge of acceptability

5.2 Memory Use

In this case we found similar results when comparing the *Solver* component with *iBundler*. The amount of memory required in the worst case is quite the same for both cases. The memory consumption in both cases is highly dependent on the ontology structure. It is not surprising that the memory peak is similar in both cases, as the information quantity to represent is actually the same. The biggest amount of information is used to represent all the bids. Both *Solver* and JADE have to load in memory the information representing a problem, namely an RFQ and the received bids (the former as a Java object and the latter as a file containing matrices). Figure 7 compares the memory use for the *iBundler* agency and *Solver*.

6 Conclusions

The tests we ran show that offering *iBundler* as an agent service implies a significant time overload, while the memory use is only slightly affected. The main cause of such an overload is related to the encoding and the decoding of ontological objects and messages. The message serialisations and deserializations, along with ontology checkings heavily overload the system as the dimensions of the negotiation scenario grow. We propose several actions to alleviate this effect. Firstly, we have observed that the main amount of information is gathered in representing bids. Their presence in objects and messages is the foremost cause of *iBundler*'s time overload. Thus, a suitable workaround is to use, at ontology design time, a more synthetic bidding language, in which bids can be expressed more concisely. For instance, introducing a preprocessing phase in which equal (and even similar) bids are grouped, in order to obtain a more compact representation. The resulting ontology would generate more tractable objects. Secondly, it would be also helpful to improve the performances of the JADE modules devoted to the ontology checking and serialisation processes. All in all *iBundler* can satisfactorily handle small and medium-size negotiation scenarios. Thus, although the automation of the negotiation process with agents helps in saving time in managing negotiations, the scalability in terms of time response of *iBundler* is limited.

As future work we propose a comparison of *iBundler* with other distributed solutions such as CORBA (http://www.corba.org) or JAVA RMI (http://java.sun.com). Nonetheless, we should notice that agent technology offers a higher level of abstraction, and thus we would lose the transparency and portability offered by the agent paradigm.

We conclude that, while agent technology adds a higher level of abstraction and eases inter-platform communication, state-of-the-art agent technologies require further improvements to tackle real-world domains.

Acknowledgments

This work has been funded by the Spanish Science and Education Ministry as part of the Web-i2 (TIC-2003-08763-C02-00) and IEA (TIN2006-15662-C02-01) projects, and by the Spanish Council for Scientific Research as part of the 20065 OI 099 project. Andrea Giovannucci enjoys the BEC.09.01.04/05164 CSIC scholarship.

References

1. Giovanucci, A., Rodríguez-Aguilar, J.A., Reyes-Moro, A., Noria, F.X., Cerquides, J.: Towards automated procurement via agent-aware negotiation support. In: Third International Joint Conference on Autonomous Agents and Multiagent Systems, New York (2004) 244–251
2. Reyes-Moro, A., Rodríguez-Aguilar, J.A., López-Sánchez, M., Cerquides, J., Gutiérrez-Magallanes, D.: Embedding decision support in e-sourcing tools: Quotes, a case study. Group Decision and Negotiation **12** (2003) 347–355
3. Brazier, F., van Steen, M., Wijngaards, N.: On MAS scalability. In: Proceedings of Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal (2001) 121–126
4. Deters, R.: Scalability & multi-agent systems. In: Proceedings of Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal (2001)
5. Kahn, M.L., Della Torre Cicalese, C.: COABS grid scalability experiments. Autonomous Agents and Multi-Agent Systems **7**(1-2) (2003) 171–178
6. Klein, M., Rodriguez-Aguilar, J.A., Dellarocas, C.: Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. Autonomous Agents and Multi-Agent Systems **7**(1-2) (2003) 179–189
7. Fedoruk, A., Deters, R.: Improving fault-tolerance by replicating agents. In: AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, ACM Press (2002) 737–744
8. Yoo, M.J.: An industrial application of agents for dynamic planning and scheduling. In: AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, ACM Press (2002) 264–271
9. Jarrar, M., Meersman, R.: Scalability and knowledge reusability in ontology modeling. In: Proceedings of the International conference on Infrastructure for e-Business, e-Education, e-Science, and e-Medicine. Volume SSGRR2002s., Rome, SSGRR education center (2002)
10. Wache, H., Serafini, L., García-Castro, R.: D2.1.1 survey of scalability techniques for reasoning with ontologies. Technical report, Knowledge Web (2004)
11. OMG, FIPA: OMG and FIPA standardisation for agent technology: competition or convergence? http://www.cordis.lu/infowin/acts/analysys/products/thematic/agents/ch2/ch2.htm (1999)
12. Sandholm, T., Suri, S., Gilpin, A., Levine, D.: Winner determination in combinatorial auction generalizations. In: First Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02), Bologna (2002) 69–76
13. Hillier, F.S., Liberman, G.J. In: Introduction to Operations Research. Mc Graw Hill (2001) 576–653
14. Odell, J., van Dyke Parunak, H., Bauer, B.: Extending UML for agents. In: Proceedings of the Agent-Oriented Information Systems Workshop, Austin, TX, 17th National Conference on Artificial Intelligence (2000) 3–17
15. FIPA: FIPA interaction protocol library specification. Technical Report DC00025F, Foundation for Intelligent Physical Agents (2003)