Eng Int Syst (2012) 1: 65–75
© 2012 CRL Publishing Ltd

Engineering
Intelligent Systems

Agent assisted paper collection for recycling

Nikolaos Bezirgiannis[1] and Ilias Sakellariou[2]

[1]Department of Inf. and Comp. Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands.
E-mail:n.bezirgiannis@students.uu.nl
[2]Department of Applied Informatics, University of Macedonia, 156 Egnatia Str. 54124 Thessaloniki, Greece.
E-mail: iliass@uom.gr

Recycling has been gaining ground, thanks to the recent progress made in the related technology. However, a limiting factor to its wide adoption, is the lack of modern tools for managing the collection of recyclable resources. In this paper, we present EcoTruck, a management system for the collection of recyclable paper products. EcoTruck is modelled as a multi-agent system and its implementation employs Erlang, a distribution-oriented declarative language. The system aims to automate communication and cooperation of parties involved in the collection process, as well as optimise vehicle routing. The latter have the effect of minimising vehicle travel distances and subsequently lowering transportation costs. By speeding up the overall recycling process, the system could increase the service throughput, eventually introducing recycling methods to a larger audience.

Keywords: Agent Systems, Contract-Net, Functional Logic Programming, Erlang

1. INTRODUCTION

The term *Recycling* refers to the reinsertion of used materials to the production cycle. It is an important component of the well-known 3R hierarchy “Reduce, Reuse, Recycle”. The initial phase of the recycling process is the collection of the recycled materials from consumers. In a sense, this is a kind of a transportation problem, i.e. transporting items from geographically distributed locations to a central location (material recovery facilities) in a highly dynamic manner. Due to the problem’s dynamic nature, we consider it to be an excellent area of application for multi-agent technology [1]. Although the latter has been around for many years, still presents a number of excellent opportunities for its application in new domains. In the light of new technological advances in the areas of telecommunications, portable computing devices and development platforms, such applications can really “go” mainstream.

Paper recycling has gain significant attention due to a) the large quantities of paper used in everyday tasks (offices, packaging), and b) possible reductions in energy consumption and landfill waste. Currently, municipal authorities employ a rather outdated procedure for the collection of recyclable large packaging cartons and other paper waste. This procedure involves telephone communication and manual truck assignment for pick up services. The drawbacks of such a process are rather obvious: delays in the collection process, unoptimised use of resources, and can potentially lead to a low acceptance of recycling practise and thus to failure of the whole process.

EcoTruck [2] aims to address the above issues, replacing the current "manual" collection process by a multi-agent system. This offers a number of advantages, such as distributed coordination of collection services, speed, robustness, absence of a central point of control and scalability. The system proposed is implemented entirely in Erlang[3], a concurrency-oriented declarative language. While Erlang is not widely adopted by the agent community, it is quite heavily used in industry to develop mission critical server products and soft real-time systems. The reasons for adopting Erlang as the platform of choice were its strength in fault-tolerance and distributed programming, aspects critical to the EcoTruck application.

Thus, the aim of this paper is twofold: firstly, the paper supports that the application of multi-agent technology can improve the recyclable paper collection process; secondly, we argue that the development of such multi-agent systems can be easily done in Erlang and prove the latter by presenting the implementation of our system in the language. The present article extends previous work on the EcoTruck system [2] in two ways. The first extension concerns an implementation of a more intelligent decomposition of failed task delegations, in order to lead to an increased performance of the overall system. Under the new approach agents decompose a unsatisfied paper recycling request into two smaller ones of appropriate size taking into consideration information from truck agents instead of simply dividing the original request in two requests of equal size. The second concerns an experimental evaluation of both the original system and the improvement mentioned above in a cloud computing environment, in order to assess its overall performance. These extensions are reported in sections 6 and 7 of the present article.

The rest of the article is organised as follows. Section 2 presents the related work on the agent based paper recycling and agent development. The problem addressed is introduced in more detail in section 3. Section 4 presents the system's architecture and cooperation protocol, i.e. the Contract-net protocol. Section 5 describes the implementation of the system in Erlang while section 6 presents experimental results in a cloud computing platform. The section that follows (section 7) presents an improvement of the initial system. Finally, section 8 concludes the paper and presents potential extensions of the current system.

2. RELATED WORK

The MAS approach has been applied to recycling/environmental processes in various contexts. For instance, the EU funded project E-MULT [4] aimed at developing a dynamic network of SMEs based on multi-agent solutions for the recycling of end-of-life vehicles, a process that is reported to be particularly complex. In [5] a simulation of an animal waste management system between producers and potential consumers (farms) is presented that allows to test different management scenarios.

In [6] authors employ multi-agent simulation coupled with a GIS to assist decision making in solid waste collection in urban areas. Although the approach concerned truck routing, it was in a completely different context than the one described in the present paper: the former aimed at validating alternative scenarios by multi-agent modelling in the waste collection process, while the EcoTruck system aims at dynamically creating truck paths based on real time user requests.

The problem of efficiently collecting recyclable material from customers is in essence a dynamic transportation problem, although simpler in the sense that the destination location is fixed. A number of approaches in the literature have attacked the same problem, such as [7], where an extended contract net protocol (ECNP) is used to solve a transportation logistics problem. The ECNP protocol introduces new steps in the original contract net (temporal grant/reject and definitive grant/reject) and allows to dynamically decompose large demands (contracts). In [8] authors propose a new protocol the "provisional agreement protocol", where agents under the protocol are committed to bids sent only when they are (provisionally) granted the award. Thus, agents are allowed to participate simultaneously in multiple task announcements. The PAP protocol has been applied together with open distributed planning to the simulation of a real logistics data of a European company. Since the problem EcoTruck is dealing with is less complex than the dynamic transportation problem, EcoTruck follows a simpler approach: it allows agents to place multiple bids and allows multiple task announcements for an unserviced request if the bidder has already committed to another "contract" by the time it receives the award, as described in section 4.1.

Finally, the Erlang language has been used to implement software agents with quite promising results. In [9] authors proposed the agent platform eXAT and support that it offers a number of benefits compared to the well-known JADE [10] platform, such as robustness and fault tolerance, more natural integration of the necessary components to implement such systems (FSM support and production rules components). Additionally, in [11] authors argue for the suitability of Erlang language compared to JAVA based platforms.

3. RECYCLABLE PAPER COLLECTION

While recycling technology has made significant progress, the collection of the recyclable materials still relies on old-fashion practises. Currently, municipal offices responsible for the paper collection and management operations act as mediators between companies (i.e. large companies, shopping centres, supermarkets) and dedicated truck vehicles for transporting paper from the former to the materials recovery facilities (MRFs). Typically, offices employ a rather outdated non-automated procedure that relies on telephone communication: a) the company contacts the office and places a request, b) the municipal office checks the current truck schedules and finds an appropriate truck for the task c) the truck is immediately informed of the new task and adds it to its current schedule. Of course the procedure described is an ideal "manual" operation. Usually, municipal offices collect requests and construct the next-day trucks' schedule, with an obvious impact of such delay both to the urban environment and acceptance of recycling practices by customers.

The "ideal" manual procedure presents quite a few disadvantages, mainly due to the centralised approach followed, that imposes a serious bottleneck to the system. It can be easily seen that:

- Efficient resource allocation (trucks to collection tasks) cannot be easily achieved, since the municipal office is not aware of the current status or location of the trucks.
- Communication is slow, resulting to a low number of handled requests.
- Trucks do not coordinate their actions and thus resources are underexploited in most of the cases.
- Finally, customers have very little information on the progress of their request and this has an impact on the adoption of recycling by a wider community of professionals.

The agent based approach proposed in this article attempts to resolve most of the above problems. Imposing a distributed cooperation model between interested parties alleviates the need for a central coordination office, allows trucks to form their daily schedule dynamically based on their current state (e.g. load, distance from client and other commitments), and increases the number of requests serviced. Finally, since the system's status is available, customers have access to information regarding the progress of their request.

The case of serving recycling requests fulfils three out of four criteria for employing agent based systems stated in [12], namely the existence of open, highly dynamic environment, geographic distribution of data and control and agents being a natural metaphor of the system.

- The environment is highly dynamic and open as the number of customers, their location and the size of their recycling requests varies from day to day and is definitely not known at design time.
- Geographic distribution of data is present in the system since truck agents hold data regarding their current capacity, location and plan and customer agents maintain data regarding their requests and location. Therefore adopting an agent oriented approach removes the necessity of collecting all the above in a single point and dealing with the task in a centralized manner.
- Decisions regarding system's operation can be based on local data (state) of each agent, thus favouring a solution that supports autonomy, a central characteristic of agent based systems.
- The organizational characteristics of the overall system, i.e. independent customers that require a single truck to service their request, leads to a natural modelling of the problem as a set of interacting active agents, that form organizational links depending on the current state of the world. The latter also allows to deal gracefully with abnormal situations, as for instance a malfunction of a truck that has already committed to a recycling request.

Finally,"the interplay between system components can be naturally viewed in terms of social interactions" [13] since customer and truck agents form organizational links using a high level message exchange using the well known Contract Net protocol.

4. ECOTRUCK AGENTS

A natural modelling of the problem is to map each interested party in the process to an agent. Thus, EcoTruck is conceived as a multi-agent system that consists of two types of agents: *Customer Agents*, each representing a company in the system and *Truck Agents*, each representing a dedicated collection vehicle, that is responsible to manage the latter's everyday schedule.

The *Customer agent* has the goal to satisfy "its" user's recycling request by allocating the latter to an available truck. Thus, a customer agent receives input, i.e. the paper amount the company wishes to recycle, and initiates a cooperation protocol to find the best possible truck to handle the request. The best candidate truck is considered to be the one that can service the request in the *shortest attainable amount of time*. Furthermore, the customer agent is responsible for monitoring the entire process, provide a user friendly display of the progress of the user request, as well as take action in the light of possible failures.

A *Truck agent* is modelled as a software agent mounted on truck vehicles that operates as an assistant to the driver. The agent has the goal of collecting as much recyclable material as possible, thus tries to maximise the number of Customer agent requests it can service. Each incoming request is evaluated based on the current available capacity of the truck, its geographical position and the tasks (paper collections) it has already committed to, i.e. its plan. Once a new collection task is assigned to the agent through the cooperation protocol discussed later, it is added to the truck's plan. Each Truck agent maintains its own plan, that is a queue of collection tasks it intends to execute. Additionally, the truck agent proactively adds a "paper unloading" task to the plan upon detecting that its current capacity reaches a lower limit. Finally, by processing real-time routing information, the agent "guides" the driver inside the city, much like a GPS navigation system, reducing total travel distances and response time.

The operation of the system is *cooperative*, in the sense that all involved parties work together in order to fulfil the overall goal of increasing the amount of recyclable material collected during a working day. Consequently, the interaction protocol that was selected for EcoTruck system was the Contract-Net protocol, as discussed in the subsection that follows.

4.1 Agent Cooperation

The Contract-Net protocol (CNP) [14, 15], is probably the most well-known and mostly implemented task sharing protocol in distributed problem solving. It offers an elegant yet simple way of task sharing within a group of agents and it was considered to be suitable for the case of the EcoTruck system.

In the latter, Customer agents play the role of *managers* while Truck agents act as *contractors* according to the protocol's terminology. The overall process is the following:

1. A Customer agent initiates a CNP protocol by announcing the paper collection task to truck agents. This "call for proposals" (CFP) contains information regarding the geographical location of company and the paper quantity for collection. The latter plays the role of the *eligibility criterion* in the protocol, i.e. truck agents must make sure that they can indeed service the request.

2. Truck agents evaluate the CFP and decide on their eligibility for the task. If they can collect the paper load reported in the CFP, they reply with a *bid* that contains the *estimated time of servicing* (ETS) the request; the latter is the sole criterion on which Customer agents decide on the agent to assign the task to. Naturally, Trucks can also refuse a request, if they cannot handle it, are currently full or for any other reason, by responding with an appropriate message.

3. The Customer agent receives the bids, processes them, decides on the best truck to assign the task, and broadcasts the corresponding messages (accept-proposal/refuse-proposal) to all interested truck agents.

4. The *winner* truck adds the task to its current plan. Upon arriving at the designated location, it will collect the paper, mark the job as finished and inform the corresponding Customer agent.

Obviously, there is always the risk that the Customer agent receives no bids [14]. This can occur when the paper quantity in the CFP exceeds either:

a. the current capacity of any truck in the system,

b. the maximum capacity of even the largest truck.

Although different in nature, both cases are treated uniformly: the Customer Agent decides to decompose the original request into two smaller ones of equal size and handle each new request separately, by initiating a different CNET protocol on each one. While the decision is obvious in the case that the CFP quantity exceeds the maximum capacity of all trucks, it requires some explanation in case (a) above. Since for trucks to regain their capacity they need to unload their cargo in Material Recovery Facility (MRF), a rather time consuming process, it was considered better in terms of service time reduction to allow the decomposition of the task, instead of the Customer agent waiting for some truck with the appropriate capacity to appear in the community. However, such a decomposition leads inevitably to an increased number of messages exchanged between the agents, with an impact to the overall efficiency of the system.

Another issue, that is also present in the original CNP specification, regards whether to allow contractors to participate in multiple call for proposals [16, 7]. In the EcoTruck system such cases would naturally occur, since multiple companies operate in the system's environment that might place simultaneous requests. Therefore, it was decided that when a "winner" truck is awarded a contract (accept-proposal message) then it checks it against its current schedule, since the latter can be different than the one the agent had during the bidding phase. If the task can be added in the schedule with no differences in the ETS that appeared in the initial bid, then it is simply added in the list of tasks the truck has to service. However, any differences in servicing time lead to a *failure* message being send to the customer agent. Obviously, the latter has to re-initiate a CNP protocol on the user request once more. Although, this approach is not very sophisticated like the one described in [16], and certainly leaves plenty of room for improvements, it was considered to be adequate for the specific case.

5. IMPLEMENTING ECOTRUCK

Erlang [17, 18], is a concurrent declarative language aiming at the development of large distributed systems with soft real-time constraints [3]. The language offers dynamic typing, a single assignment variable binding scheme and automatic garbage collection so as to facilitate program development. Support for concurrency is achieved through process based programming and asynchronous message passing communication. Erlang follows the now popular "execution through virtual machine" model and a distributed system is composed of a number of different Erlang nodes running in one or more computers, each node being a unique instance of a virtual machine.

A useful extension to the standard Erlang language is the Open Telecom Platform (OTP)[19], that is a set of libraries and tools to facilitate the design and development of large distributed systems (and not only of telecommunication applications as its name suggests). In essence OTP presents a middleware that bundles a list of the most common design patterns that can occur in a distributed setting, called *OTP behaviours*. The latter have been used in the development of the system.

A most interesting and extremely useful aspect in programming agent systems in Erlang (and any other distributed application for that matter) is the *let-it-crash* philosophy. According to the latter fault tolerance is not achieved by defensive programming techniques (e.g. guards), but by imposing a hierarchy of *supervisor* and *worker* processes. The former monitors the operation of the latter and can take action (e.g. restart) in case a failure occurs. This *supervision tree* allows the development of fault tolerant applications, a crucial aspect to the development of any real world agent based system.

EcoTruck employs *Server behaviour* processes, for asynchronous client-server communication, *FSM behaviour* processes, for interactions that require state transitions, and *Supervisor behaviour* processes, that monitor other OTP processes, so as to restart them in case they crash. These processes are nicely packaged into three distinct applications, using another OTP behaviour, called *Application behaviour*. Thus, our agents in Erlang consist of a number of processes, each being an instance of a specific OTP behaviour.

This approach greatly facilitated the design and implementation of the EcoTruck system, since it provided all the tools necessary to built the agents as well as the necessary features such as fault tolerance and real-time characteristics. In the sections that follow, a presentation of the implementation of each agent participating in the system is provided.

5.1 Directory Facilitator

Since the EcoTruck implementation did not rely on any agent development platform, the necessary directory services for the agent community were provided by a system -specific *Directory Facilitator (DF)*. This entity stores the roles and addresses of every active agent.

Before engaging in any interaction, EcoTruck agents (trucks and customers) subscribe to the system by passing relevant information to the DF (role and communication details). The DF maintains this information in an Erlang Term Storage (ETS) table, i.e. a fast in-memory data-store, part of the standard Erlang distribution. The DF itself is implemented as a server OTP behaviour. The server monitors all subscribed agent processes and if any of them become unresponsive, due to network failure or abnormal termination, will automatically unsubscribe them from the system, thus maintaining a list of only active agents at each point of execution. It should be noted that the server process itself is monitored by a supervisor behaviour, providing the fault tolerant characteristics, necessary for this component of the application.

5.2 Customer Agent Structure

A set of Erlang processes constitute the customer agent:

- the *customer Server*,
- the *customer Supervisor*,
- one or more *Manager processes*, and
- the *Customer GUI* process.

The *customer Server* is an OTP Server instance that is the master process of the agent. Its role is to store agent's preferences and control the user interface. The server process is monitored by a *customer Supervisor* process to provide all necessary fault-tolerant features described above. When the user places a request through a web interface that is handled by the *customer GUI process*, the server spawns a new *Manager process* and passes to it the relevant information, i.e. the paper quantity to recycle.

The customer's *Manager* process is responsible to handle the request placed by the user, by delegating the task of its collection to a suitable truck agent. Thus, a manager process implements essentially the role of the manager in the CNP protocol. Upon its initialisation the process acquires the list of truck agents from the DF and initiates a CNP interaction as described in section 4.1. It is implemented as an OTP Finite State Machine (FSM) behaviour, for the reason that CNP requires successive steps of interaction. These communication steps are essentially the transition states of the FSM. At the end of the contract assignment process, the Manager will link itself with the "winner" truck and in specific with its Contractor process, described later in this section. This link can be perceived as a "bidirectional monitor"; if one of the linked processes exits abnormally, the other process becomes aware of it. The code is depicted in figure 1. As observed the declarative nature of the functional model of Erlang allows for an elegant implementation of message passing in this case. The strong module system of Erlang is also shown in the figure: for example the `foreach` function belongs to the `lists` module of the platform. Readers familiar with the Prolog logic programming language will notice that variable names start with a capital letter, a feature that comes from the first Erlang implementation in the former.

The established link ensures that if a winner Contractor crashes, the Manager will notice the failure and re-start a new CNP interaction, on its pending paper recycling request. When the requested task is completed, the Manager reaches its final state to gracefully terminate its execution. The process spawn tree is depicted in figure 2. As shown in the figure, the Manager process communicates with multiple Contractor processes, whereas each Contractor process communicates only with one Manager.

A Manager process terminates gracefully also in the case of a request decomposition: it spawns two new Manager processes, passing to each one a request divided in two, and terminates its execution. The corresponding code is shown in figure 3. Compared to the customer Server which lives as long as the application is running, the Manager's lifetime span covers only the period of time from request announcement (call for proposals) to collection. The described behaviours together with a GUI are packaged into an OTP application, forming the customer agent.

5.3 Truck Agent Structure

The truck agent is similarly composed of a number of supervised processes (figure 2):

- the *truck Supervisor* process,
- the *truck Server* process,
- one or more *Contractor* processes, and
- the *Driver* process.

The *truck Server* is responsible for maintaining the plan (list of collection tasks) of the agent, and is monitored by a *truck Supervisor* process as in the case of customer agents. For every incoming call-for-proposals (CFP) request, the Server will spawn a new *Contractor process* to handle the request. The latter is implemented as an OTP FSM behaviour and is responsible to handle all communication regarding the agent's participation in the CNP. If the truck agent is eligible, the Contractor replies with a bid that contains the estimated time of servicing. In the case that it is the "winning" bidder, it will instruct the truck Server to schedule the task for execution by appending it to the plan. Upon completing the task, the Contractor process will notify the customer Manager process that initiated the CNP via an "inform" message (figure 4) and gracefully terminate.

A conflict that commonly arises in a multi-CFP setting is when two or more Contractors attempt to place a recycling task in the exact same position in the truck's plan. In EcoTruck, such cases are resolved, by having the truck Server to accept one of the tasks that its Contractors propose and reject the rest. The Contractors who failed to include their task in the plan, will signal this failure by appropriate message exchange to the associated Manager, who will in turn restart a CNP interaction.

```
send_acceptance(BestProposer, Proposers) ->
   gen_fsm:send_event(BestProposer, accept_proposal),
   link(BestProposer),
   lists:foreach(fun (T) ->
       gen_fsm:send_event(T, reject_proposal) end,
               lists:delete(BestProposer,Proposers)).
```

Figure 1 Erlang Code for accepting a proposal.

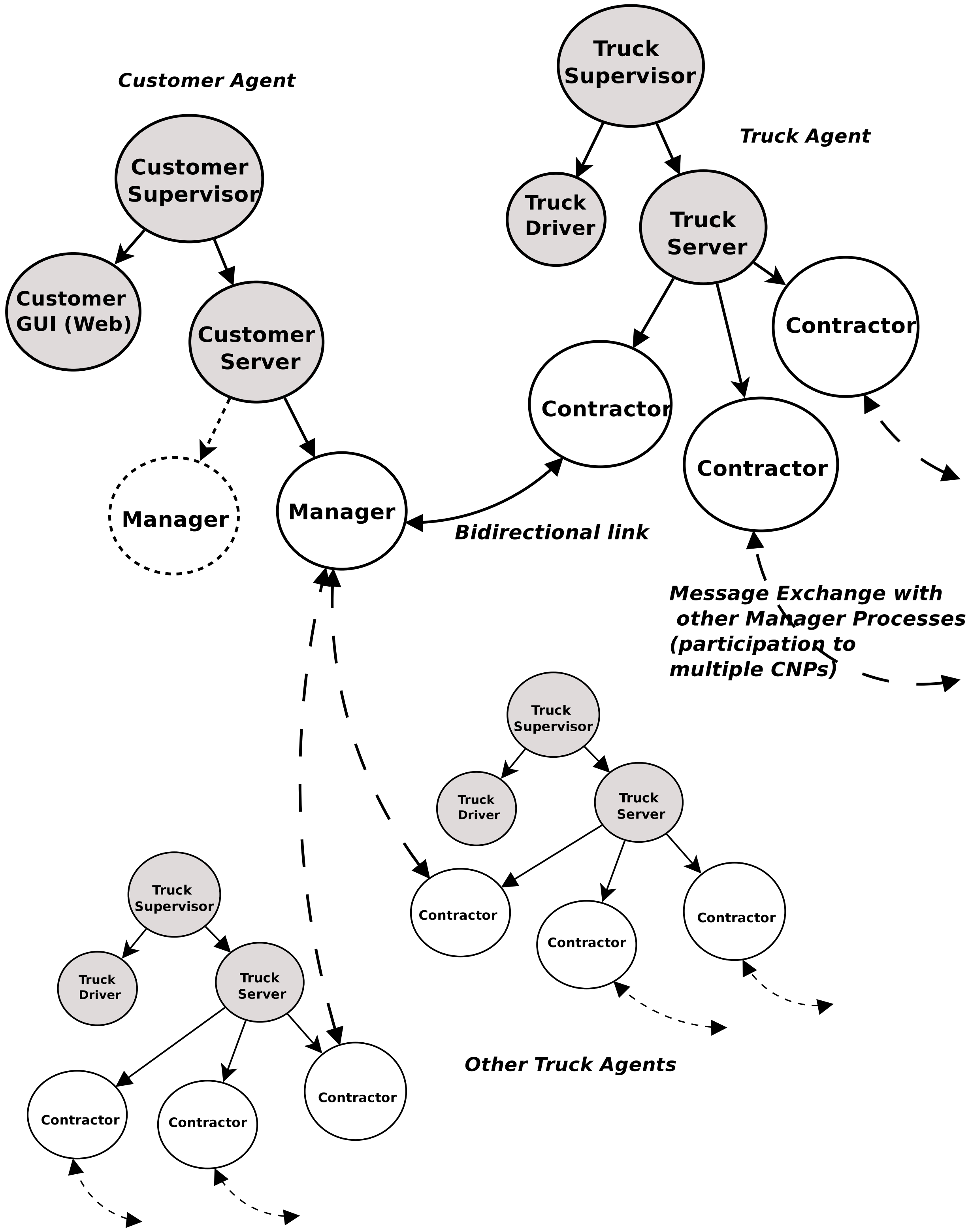

Figure 2 EcoTruck Agent processes hierarchy and interaction.

```
break_cfp(_Contract = {Position,Value}) ->
   case Value > 1 of
    true ->
      start({Position, Value/2}), % start 2 new managers
      start({Position, Value/2});
    false ->
      start({Position, Value}) % small quantity just reset
   end,
   {stop, normal, []}.
```

Figure 3 Erlang code for CFP decomposition.

Finally, the *truck driver process* is an extra Server Erlang process, which simulates the motion of the truck vehicle inside the city roads. This was considered necessary in order to be able to have testing and simulation results on the implemented platform. In the final production system the process will be replaced by some kind of global positioning system (GPS) module, mounted on the device of the truck. These four behaviours are bundled in a truck application. The process spawn trees of EcoTruck agents are shown in figure 2.

Since Erlang follows the virtual machine model, the applications developed can execute both in desktop and mobile environments. This is particularly interesting in the case of the truck application, since it allows easily executing it on a mobile device mounted on truck vehicles. Figure 4 presents the message exchange under the Contract Net Protocol of the EcoTruck agents processes.

Finally, it should be noted that the Truck application employs the *Google Maps Directions Service* to provide the truck driver with routing information as well as live traffic data where applicable. Additionally, both Customer and Truck applications have a GMAPS based web monitoring tool. The service responses are parsed with the *xmerl* Erlang parsing library. Each application relies on Google Maps Javascript API to display to the user a live view of the system's state. The Google Maps JSON encoded content along with any HTML files are displayed to the user by Misultin, a lightweight Erlang web server. The reason for not having, instead, a native GUI, is that a web interface can be more portable, thus the EcoTruck software will run on any hardware an Erlang VM exists for. Figure 5 shows the GUI of EcoTruck.

6. EXPERIMENTAL RESULTS

While EcoTruck is being developed and tested solely on a single machine, for the simulation and benchmarking part of the program a series of virtual private servers (VPS) are deployed to better match a real-world scenario and provide a more accurate evaluation of the overall system. In this setting each server corresponds to a separate EcoTruck agent, uniquely identified by an IP address. The EcoTruck software was priorly being configured and installed on each server, essentially forming a distributed Multi-Agent System.

The servers/agents of the system were hosted in a "cloud"; the term mainly refers to an infrastructure where computing power, storage, and networking are provided to the user on-demand. Cloud-hosting companies follow a pay-by-use pricing scheme and usually package their services with extra managing and supervising tools. A cloud setup, compared to a traditional dedicated-server setting, has the advantage of being flexible and in the meantime inexpensive. Within our system, agent nodes were being instantly created or destroyed, to accommodate on-the-fly changes to the experiment configuration. Thus, we were able to run experiments with a varying number of agents. Finally, a RESTful API, offered by the cloud company, made possible to snapshot and clone EcoTruck agents for easily populating the agent community.

New software modules were specifically developed for simulation and benchmarking purposes and were bundled together with the EcoTruck software. A customer simulator process resides in each server and provides input to the customer agent, i.e. creating new recycling requests at random times and with varying quantity, much like a real user of the system would do. The results presented here are based on a recycling request generation pattern of one new request in time intervals varying from 1 to 10 minutes and with the paper recycling quantity varying between 10 to 100 units.

Additionally, on each agent a *tracer process* was running constantly with the task of logging messages and function calls related to the EcoTruck application. The logs were collected and combined by *Inviso*, a distributed tracing system included in the official Erlang distribution. Afterwards, each "merged" log was processed and analysed by custom-made Erlang functions, yielding the presenting results.

Other libraries and features of the language were used to assist experimentation. Hot-code loading made possible to change on-the-fly running code – in this case switching on and off EcoTruck parameters – without even restarting the Erlang VM or the server. Based on the rich *rpc* library, code was written to systematically run, stop and control, within Erlang itself, the EcoTruck agents of the running system. Thus, Erlang, apart from providing a set of features to implement agent systems, it also proved to have all the necessary facilities to allow easy testing and benchmarking of the system, in a close to real-world setting.

The described experimental setting above allowed to obtain some initial results regarding the system. These results are summarised in the table 1. Columns "Customers" and "Trucks" provide the size of the agent community, whereas the "Total Msgs" column presents the total messages exchanged between agents. The column "Total Request" is the number of paper collection tasks generated in each experiment run.

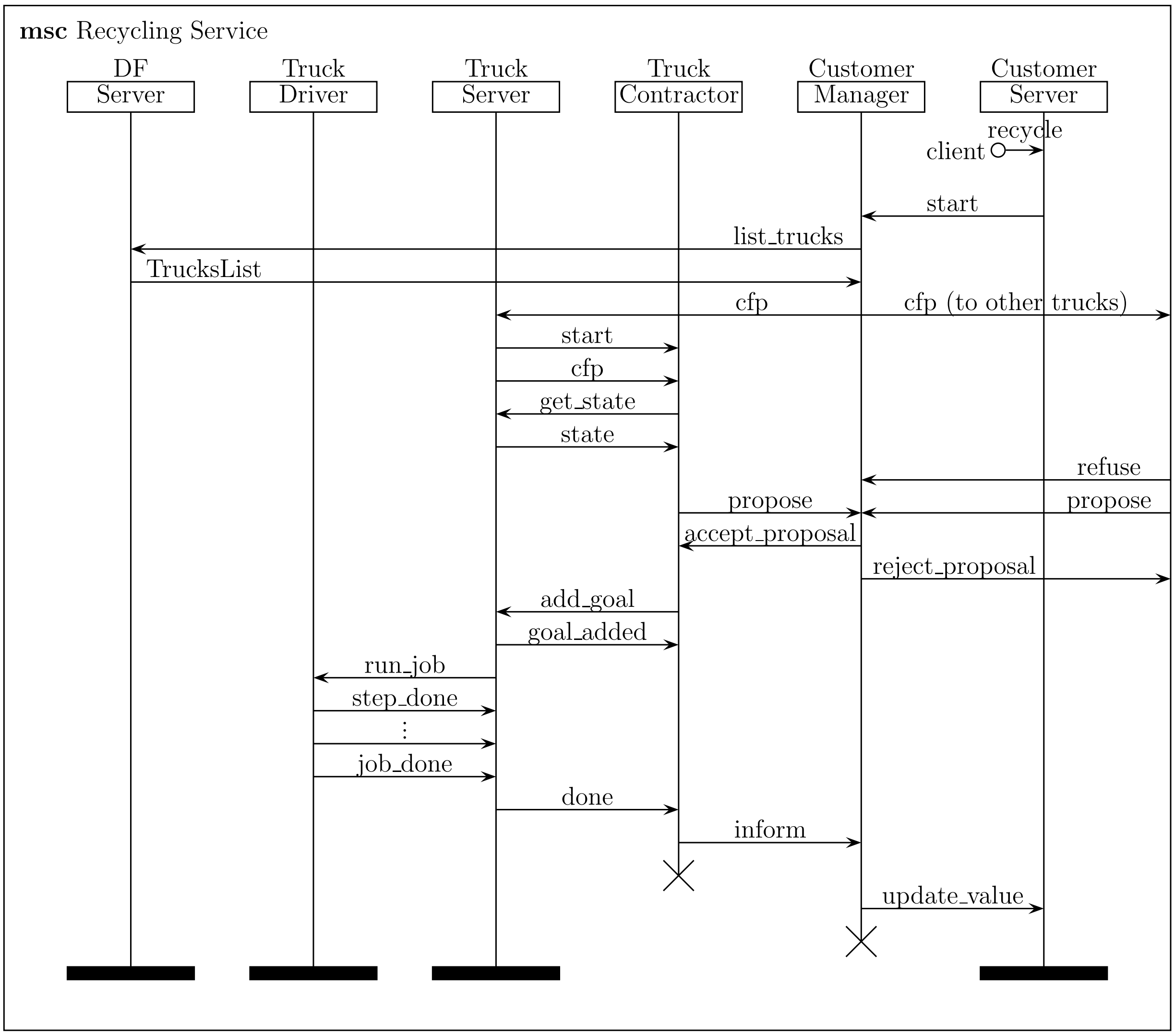

Figure 4 Message exchange between EcoTruck Processes

Finally, the "CFP/Request" column depicts the average number of call-for-proposal rounds that were necessary in order to successfully assign the task to a truck agent.

As seen from the results, the system behaves as expected; an increase in the number of customers leads to an increased number of messages and requests sent, while the CFP/Request ratio stays close to 1, meaning that requests are in most of the cases fulfilled in the 1st round. The exception manifested in the case of having 100 customers -10 trucks in the experiment, where there is a high number of call-for-proposals per request. This is expected since the increased number of customer agents participating leads to many failed CFPs that have to be decomposed in order to be successfully delegated to a truck. However, when the number of participating trucks increases this ratio drops.

Table 2 presents results regarding the total paper quantity collected and the average waiting time of customer agents. Again the results depicted are as expected, with the average waiting time dropping when the number of trucks increases, and raising when the customer agents participating in the system is increased. Thus, this initial set of experiments demonstrates that the ideas, cooperation protocol and design of the system achieve its original goals.

Table 1 Results from initial run of Ecotruck with each truck having a capacity of 1000 units.

Customers	Trucks	Total Msgs	Total Requests	CFP/ Request
100	20	49238	543	1.69
100	10	159512	558	13.78
50	10	8501	287	1
20	10	3305	107	1
10	10	1922	62	1
10	5	832	52	1
10	3	476	49	1

7. AN IMPROVED CFP DECOMPOSITION

Initial versions of the EcoTruck system relied on a somewhat naive implementation of the CFP decomposition; recycling requests that were simply too large to be handled by trucks in the agent community, were broken down into two smaller requests of equal size, in hope that these new collection tasks

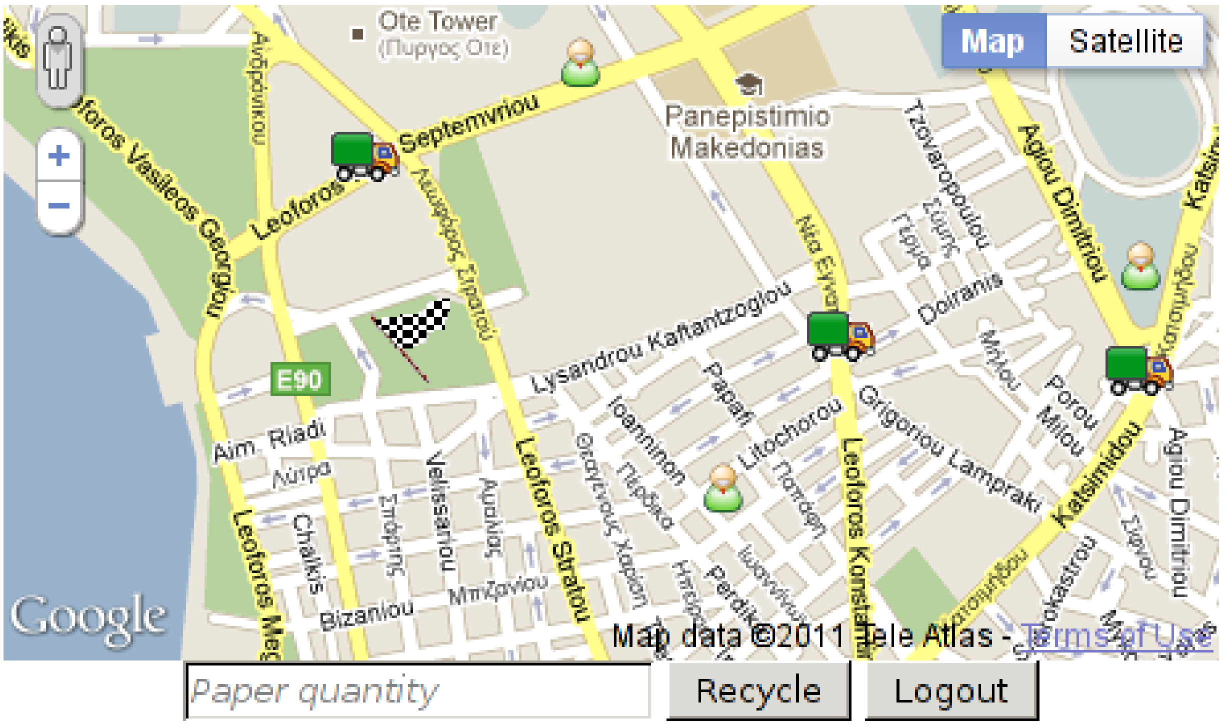

Figure 5 Web Interface of EcoTruck Application

Table 2 Results regarding total paper quantity collected and Average Customer Waiting Time. Again each truck having a capacity of 1000 units.

Customers	Trucks	Quantity	Customer Waiting
100	20	29386	42.49
100	10	17809	150.55
50	10	14407	35.22
20	10	5680	30.87
10	10	3082	29.48
10	5	2906	35.21
10	3	2728	63.67

could be delegated more easily to trucks in the system. In order to improve this rather simplistic approach, a more informed decomposition mechanism was introduced that aims at the reduction of interactions (CFPs) in the agent community.

In the new setting, each *refuse* message in the CNP protocol is annotated with the truck's current available capacity, i.e. each truck that refuses to perform the request annotates its reject message with the capacity it would be willing to collect based on its current plan. If at the end of the bidding phase there are no available trucks to handle the request, the customer Manager process extracts this information from the refuse messages and decomposes the initial CFP to a series of CFPs that better match the current available capacity of the truck agents. Thus, the next round of CFPs has increased chances of succeeding.

Table 3, summarizes some preliminary results on the new technique. It is particularly interesting to note the dramatic decrease in the CFP/request ratio when the resources are scarce. For instance when there are only three trucks available in the community a similar number of requests leads to a reduction in half of the necessary average rounds of CFPs in order to delegate the task.

The average waiting time to satisfy a collection request can be considered as a measure of the system's performance, since one of the goals of the system is to pick up the recyclable paper as quickly as possible. Table 4 shows that more informed decomposition leads to an increase in the performance with respect to the former.

Of course, this is an initial evaluation of the system and a more thorough experimental assessment is needed in order to safely conclude on the improvement of the decomposition technique.

8. CONCLUSIONS AND FUTURE WORK

EcoTruck is a multi-agent system for the management of recyclable paper collection process. We believe that the specific approach can greatly facilitate and optimise the process, thus allow its wider adoption by the parties involved.

As discussed, the system's implementation is based on concurrency and distribution mechanisms of the Erlang language. We strongly believe that robustness and fault-tolerance are important qualities that a multi-agent system should meet. Although, the Erlang language is not a MAS platform it appears to have the necessary features to facilitate simple and elegant implementations of multi-agent applications. It should also be noted that the language proved to have excellent tools and facilities in order to deploy and run the system in a cloud environment to perform simulation and testing.

The present EcoTruck system follows a simple and natural modelling of parties involved in the process, by mapping each participant to an agent. In the case of the EcoTruck system, such a modelling is favoured since each customer is indepen-

Table 3 Results comparing the naive and more informed approaches to task decomposition.

		Msgs		Requests		CFP/Request	
Customers	Trucks	simple	impr	simple	impr	simple	impr
10	10	2593	2244	59	58	1.86	1.67
10	5	8949	1260	67	54	12.67	2
10	3	20916	9453	65	60	52.92	25.97

Table 4 Results comparing the naive and more informed approaches to task decomposition in relation to the Average Distance and the Average Customer Waiting Time.

		Distance/Request		WaitingTime/Request	
Customers	Trucks	simple	impr	simple	impr
10	10	7667.58	6803.62	73.69	61.72
10	5	7954.19	7570.52	157.93	75.8
10	3	5361.69	6211.37	201.52	150.98

dent and thus can be naturally modelled as a single agent. In the case of trucks, the system could have adopted a flexible organisation of the former in teams, which would have led to a more complicated cooperation schema. However, such an organisation might not have demonstrated significant benefits since all members of the team would have had the same capabilities, and only one member (truck) is required for completing the task, and thus no team formation is necessary. In the case of large requests, that demand more trucks, decomposition is performed by customer agents, solving the overall task in a distributed manner. Since this is a design issue, and an extensive experimental evaluation is necessary, it is one of the future directions that work presented in this article would extend to.

There are quite a few features that could be incorporated in the current system among which the most interesting ones include:

- *Dynamic re-planning and scheduling.* In the present system, each truck agent maintains its own plan that has a static ordering of jobs, to which new jobs are inserted in a FIFO manner. A more intelligent planning process would include more dynamic features, such as prioritisation of jobs based on a number of criteria such as proximity to the current position and estimated time of arrival, and could help minimise the total travel path of the truck.

- *Smarter Truck Positioning.* Based on past data, truck agents can identify geographic areas where most requests appear in, and could decide to place themselves closer to those areas and consequently increase system performance and success rate.

- *Better Decomposition of CFP's.* A more informed manner of splitting large requests could involve taking advantage of information attached in "refuse" messages of the CNP, and breaking down the request in simpler ones of appropriate size, based on the current availability of the trucks. Although an initial idea of the former was tested in this work, as experiments show there is room for more elaborate techniques and further experimentation. Such sophisticated techniques would potentially lead to and increased overall performance, since fewer agent interactions would occur.

There are also a few improvements that could be done on the implementation level. For instance, the Erlang Term Storage (ETS), can be replaced by a Mnesia database, that would allow to exploit the fault-tolerance and distribution advantages that come for free with the Mnesia system. The latter would allow to have multiple replicated instances of the DF database, and thus achieve robustness through redundancy.

Finally, deployment of the application in a real-world environment, would allow to fine-tune the system and examine possible patterns and procedures emerging in real-life situations. Since the system is based on a extensively tested, industrial strength platform (Erlang), we believe that the transition to a full-fledged real-world application can be accomplished with relative ease.

REFERENCES

1. Jennings, N, Sycara, K, and Wooldridge, M: A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, **1** 275–306, (1998).
2. Bezirgiannis, N, and Sakellariou, I: Ecotruck: An Agent System for Paper Recycling. In Lazaros Iliadis, Ilias Maglogiannis, and Harris Papadopoulos, editors, *Artificial Intelligence Applications and Innovations*, volume 364 of *IFIP Advances in Information and Communication Technology*, pages 303–312. Springer Boston, (2011).
3. Armstrong, J: *The Development of Erlang.* In Proceedings of the Second ACM SIGPLAN International Conference on Functional programming (ICFP '97), pages 196–203, New York, NY, USA, ACM (1997).
4. Kovacs, GL and Haidegger G: *Agent-based solutions to support car recycling.* Mechatronics, 2006 IEEE International Conference, pages 282 –287, July (2006).
5. Courdier, R, Guerrin, F, Andriamasinoro, FH, and Paillat, JM: Agent-based Simulation of Complex Systems: Application to collective management of animal wastes. *Journal of Artificial Societies and Social Simulation*, **5**(3) (2002).

6. Karadimas, NV, Rigopoulos, G, and Bardis, N: Coupling Multi-agent Simulation and GIS - an application in waste management. *WSEAS Transactions on Systems*, **5** 2367 âŁ“ 2371 (2006).
7. Fischer, K, MÃ¼ller, JP, and Pischel, M: Cooperative Transportation Scheduling: an application domain for DAI. *Journal of Applied Artificial Intelligence*, **10** 1–33 (1995).
8. Perugini, D, Lambert, D, Sterling, L, and Pearce, A: Provisional Agreement Protocol for Global Transportation Scheduling. *Applications of Agent Technology in Traffic and Transportation* pages 17–32, Calisti, M, Walliser, M, Brantschen, S, Herbstritt, M, KlÃ¼gl, F, Bazzan, A, and Ossowski, S, Whitestein Series in Software Agent Technologies and Autonomic Computing, BirkhÃ¤user Basel (2005).
9. Di Stefano, A and Santoro, C: *eXAT: An Experimental Tool for Programming Multi-Agent Systems in Erlang.* AI*IA/TABOO Joint Workshop on Objects and Agents (WOA 2003), VILLASIMIUS (2003).
10. Di Stefano, A and Santoro, C: *Designing Collaborative Agents with eXAT.* 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises 2004 (WET ICE 2004), pages 15 – 20, IEEE Computer Society, (2004).
11. Varela, C, Abalde, C, Castro, L, and Gulías, J: *On modelling agent systems with Erlang.* The 2004 ACM SIGPLAN workshop on Erlang, (ERLANG '04), pages 65–70, New York, NY, USA, ACM (2004).
12. Wooldridge, M: *An Introduction to MultiAgent Systems*. John Wiley & Sons (2002).
13. Jennings, NR: An Agent-based Approach for Building Complex Software Systems. *Communications of the ACM*, **44**(4) 35–41 (2001).
14. Smith, RG: The Contract Net Protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers,* , **C-29**(12) 1104 –1113 (1980).
15. Smith, RG, and Davis, R: Frameworks for Cooperation in Distributed Problem Solving. *IEEE Transactions on Systems, Man, and Cybernetics*, bf 11 61–70 (1981).
16. Aknine, S, Pinson, S, and Shakun, MF: An Extended Multi-agent Negotiation Protocol. *Autonomous Agents and Multi-Agent Systems*, **8** 5–45 (2004).
17. Armstrong, J: *Programming Erlang: Software for a Concurrent World.* Pragmatic Bookshelf (2007).
18. Armstrong, J: *A History of Erlang.* Third ACM SIGPLAN conference on History of programming languages (HOPL III), pages 6–1–6–26, New York, NY, USA, ACM (2007).
19. Torstendahl, S: Open Telecom Platform. *Ericsson Review*, **1**, (1997).