Pan-supplier Stock Control in a Virtual Warehouse

Emad El-Deen El-Akehal
Dept. of Computer Science
University of Bath, Bath, UK
emad@bookdepository.co.uk

Julian Padget
Dept. of Computer Science
University of Bath, Bath, UK
jap@cs.bath.ac.uk

ABSTRACT

We describe the commercial application of agents to the handling of catalogue and stock-control for the selling of books on the internet. The primary characteristic of the target market is (very) low volumes over a (very) large number of items, thus agility and extremely low overheads are the essential factors for a viable business model. Being a new company (established 2004), without legacy software and with the freedom to make new choices, it was decided that the agent abstraction offered both short-term software engineering and longer-term business advantages. This expectation has been borne out in practice, in that it has been possible to construct an e-trading platform, using a 4-person team over a period of a few months, and that is now part of a live business operation handling just over 12,000 transactions daily. In this paper we explain how agents helped focus attention on the responsibilities of key software functions, how different functions should interact with one another and how to identify and propagate key performance indicator information through the system to detect unexpected behaviour. Agent technology has many *potential* benefits for dynamic fast-moving businesses where software requirements change quickly and business needs grow rapidly, all within a dynamic environment that has entirely different rules across the axes of geography, market, customer and competitor. Using autonomous agents allowed The Book Depository to build quickly a complex network of P2P relationships with a large number of suppliers and publishers of very different sizes who each utilize a variety of different trading and data interchange standards.

Categories and Subject Descriptors

C.4 [**Performance of systems**]: Reliability, availability, and serviceability; D.2.1 [**Requirements/Specifications**]: Methodologies (Agent-oriented); D.2.11 [**Software Architectures**]; I.2.11 [**Distributed Artificial Intelligence**]: Multi-agent systems

General Terms

Software engineering, performance, fault-tolerance, distributed systems

Keywords

Supply-chains, multi-agent systems, AgentScape

Cite as: Pan-supplier Stock Control in a Virtual Warehouse, Emad El-Deen El-Akehal and Julian Padget, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)- Industry and Applications Track*, Berger, Burg, Nishiyama (eds.), May, 12-16., 2008,Estoril,Portugal,pp.11-18.

1. INTRODUCTION

The Book Depository[1] is a book (re-)selling company that operates around the world both in its own right and as one of the top alternative sellers through the Amazon market place. It functions primarily as an intermediary between buyers and many large and small publishers, book sellers and book distributors as part of its mission of "making all books available to all". The subject of this paper is how the concept of agents helped in the design and how actual agents help run the virtual warehouse that the company uses to deliver the (inventory data) catalogue synthesized from a large number of suppliers.

Much of the book business focusses on selling large numbers of a few titles, ignoring — for logistical as well as business reasons — the huge variety of offerings with niche interest. The Book Depository targets this market with the "long tail" of books that will never be best-sellers but for which there is nevertheless demand. The idea is to put together lots of these small markets and on a global scale. The first consequence of this approach is a large catalogue — but at the same time, a close-to-nil inventory, despite the current 1.7 million offerings. The key to the business model is, instead of holding a huge stock, an e-trading platform that works out the best way (that is, lowest cost and fastest delivery) of sourcing the books once the order is placed[2]. The system described here helped significantly in increasing the profitability of the company, putting it in the position to win Online Business of the Year and Retailer of the Year awards[3]. Alongside this trading activity, there is a new development that is part of the overall vision of delivery of small print-run books, namely the digitizing of books that have gone out of print and out of copyright and then re-publishing — under the Dodo Press brand — via on-demand printing. So, the business objective can be summarized as: to deliver rapidly, anywhere in the world, books from small-scale publishers and now even books that were no longer available. The technical challenge is how to handle the delivery quickly and cheaply without resorting to holding small stocks of a vast number of items. It is that part of the operation on which we focus in this paper.

The (virtual) stock control problem is characterized by (i) providing (and monitoring) a continuously available system (currently ships about 12,000 orders/day) (ii) handling the variety of data formats used by the different service providers (iii) handling regular catalogue updates. Particular notable influences on the design have been institution modelling [12, 15, 17], fault-tolerant architecture patterns developed for Erlang [1], and proposals for self-managing

[1] `http://www.bookdepository.co.uk`
[2] The system won the Nielsen BookNet Supply Chain Initiative of the Year at The Bookseller's Retail Awards in 2006.
[3] Startups Awards 2007 is organized by `http://www.startupsawards.co.uk`

architectures from Van Roy [16]. The system itself is built on the AgentScape agent platform [8, 13].

We report here on our experience of building and deploying this live system, starting with an overview of the architecture and the components in section 2, then moving in to look at the detail of the feedback mechanism that is employed to provide a degree of self-management (section 3). We discuss our choice of AgentScape in section 4 and conclude with some reflections on the experience so far and a discussion of future developments (section 5).

2. DISTRIBUTED STOCK CONTROL ARCHITECTURE

The overall system design has been influenced by experience from FishMarket [14] and from Carrel [21] (organ and tissue exchange), while the design process itself has been supported by the Prometheus design tool [20, 18]. Consequently, the system is conceived of in terms of roles, scenes and protocols, although future developments will take advantage of developments in norm-driven specification [3] and the use of multiple institutions [4]. At present, the system could be regarded as closed, in that all the agents populating the trading platform have been developed in-house, however, once again, in looking to the future the design has kept the principle of open systems in mind throughout.

2.1 The Bigger Picture

It would take more space than there is available — and probably be rather tedious — if we were to explore the design of the entire system, so we have chosen to focus on a key component: how the (inventory data) catalogue is kept up to date and discuss that in detail. However, to give the reader some context, this section outlines the components of the e-trading platform and relates them through a high-level architectural diagram (see Figure 1)

The motivation driving the development of the framework is the belief that the traditional supply chain in the book industry is relatively inefficient as a result of the huge changes that have occurred with the push towards e-commerce. The use of the internet and the emergence of new technologies have contributed significantly to changes in human behaviour, whether they are customer, seller or provider.

In a traditional supply chain, there are barriers between the parties in the chain, in that certain kinds of relationships are not considered or perhaps even not permitted. For example, it is usually not the case that a publisher sells directly to individuals, yet in reality, there are many small publishers who are forced to act as retailers because wholesalers and distributors are not interested in carrying all publishers' stock. Our aim has been to re-engineer the traditional book trade supply chain and recreate it as a set of independent *services* as distinct from independent *roles*.

A useful observation arising from the roll-out of electronic trading methods is that most active trading parties are taking on more than one role inside the industry. Consequently, the framework is designed around the principle of breaking down the supply chain into a set of services (functions) regardless of the role of their users, in order to allow small providers to focus on their area of expertise and not have to concern themselves with issues such as payments processing, handling shipping, etc..

We have identified four core services that cover all activities in the trade supply chain (see Figure 1). These are:

1. Catalogue Information Services (CIS)
2. Trade Information Services (TIS)
3. ORdering Services (ORS)
4. PAyments Services (PAS)

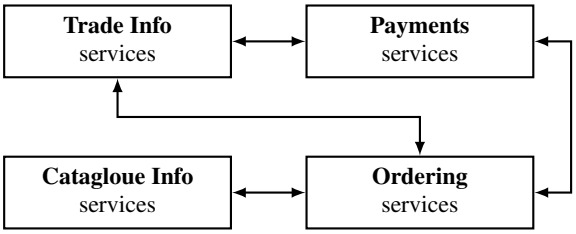

Figure 1: Generic Supply Chain Architecture

- **Name:** publish/update of central catalogue
- **Participants:** SPA, AA, CDBA
- **Description:**
  1. SPA downloads delta file or full-feed file from supplier
  2. SPA obtains CDBA identity from AA, if not already known
  3. SPA sends revisions to CDBA
- **Exceptional flows:**
  1. No response from supplier FTP server
  2. No response from AA
  3. No response from CDBA
  4. File fails soft consistency checks

  The SPA reports these condition to the AA

Figure 2: Use Case 1

- **Name:** SPA information request
- **Participants:** SPA1, SPA2, AA
- **Description:**
  1. SPA1 requests catalogue information from SPA2
  2. SPA2 sends catalogue information to SPA1
  3. SPA1 sends feedback on transaction to AA
  4. SPA2 sends feedback on transaction to AA
- **Exceptional flows:**
  1. No response from SPA2
  2. No response from SPA1

  The SPA reports these conditions to the AA

Figure 3: Use Case 2

The main focus of this paper is Trade Information Services (TIS)

2.2 Trade Information Services

To provide the reader with an overview of what we wanted to achieve, we begin with a short statement of the (functional) requirements for this component. In this context, it should be understood that the concepts of service provider agent (SPA), administrator agent (AA) and central database agent (CDBA) have already been identified, thus with focus on the SPA, we state:

R1 The SPA must be able to publish/update its stock-level information in the central database

R2 The SPA must provide stock-level information about any number of its catalogue items on request from another agent

R3 The SPA must report any errors in transaction with other agents

R4 The SPA must report any suspected data transmission failure

R5 The AA must log and may take corrective action in response to an error reported by a SPA

These requirements are then elucidated in the summary use cases (using Cockburn's undressed style) in figures 2 and 3.

The virtual warehouse acts as an aggregator, bringing together

the BookDepository's own physical warehouse and those of some 35 other suppliers, so that the front-end business logic interfaces to a single repository using a standard query and response language. The virtual warehouse is populated by agents playing four different roles:

1. Administrator agents (AA)
2. Service provider/client agents (SPA)
3. Helper agents (HA)
4. Central database agent (CDBA)

We discuss these in more detail in the following subsections.

2.3 Administrator agents

Here we describe the function and purpose of a single administrative agent, but in practice, there will be many of these. The purpose of the administrative agent is to oversee the functioning of the warehouse, to which end it has several responsibilities:

1. **Controlling registration and de-registration** of supplier agents to/from the warehouse and quarantining of norm-violating agents
2. Establishing the status of an agent for the rest of the community. We have chosen to implement three policies: (i) **public**: this means the arrival of the agent is advertised to all the agents already present and it shall be contactable by any other agent (ii) **whitelist**: the incoming agent specifies a list of acquaintances who shall be notified of its arrival and who shall be permitted to contact it (iii) **private**: arrival is not announced and agent is effectively uncontactable by others present.
   The system was built to form a generic platform for trading services, where trading partners might adopt different polices, as an example a wholesaler might limit its trade to retailers who maintain a minimum of several thousands of purchases per day, while a small publisher would sell as little as one copy to any customer. In real-world trading there is dynamic pricing mechanism that is normally realized by applying different discounts to each client, so a service provider could keep its stock level and pricing information private and supply them on demand after application of the discount policy.
3. **Receiving feedback** from suppliers about transactions with other agents. This aspect is important to the good running and for the self-management of the warehouse and is discussed detail in section 3.
4. **Management of helper agents:** helpers are created lazily by the administrator and in the case where a supplier reports that a particular helper appears not to be responding it will shut the existing copy down and start another (see discussion of feedback and self-management in section 3).

2.4 Service Provider agents

There is typically one service provider agent (SPA) per supplier, but in some cases there may be more, such as when there is more than one account, as a means to capture the different rules under which each account is operated. For example, one particular book distributor in this system is represented by two SPAs with different business logic for each — reflecting different contracts — and each with access to different (physical) warehouses, affecting product availability and geographical spread for delivery. Service provider agents have five tasks:

1. **Communicate with other suppliers' agents** This is for the purpose of one SPA supplying accurate information about its stock level to another SPA. For instance, the SPA of a distributor may request an update on a range of products from the SPA of a publisher before publishing its catalogue. The system provides for setting up communication channels, so a newly joined SPA can find other SPAs without prior knowledge of their existence, then it engages in data exchange transactions. This is an essential element of open market structure where the trading parties have no prior knowledge of each other, however they do have clear knowledge of their needs and the tools, or the logic to evaluate and decide on engagement with other parties.
2. **Translate queries** The system has a standard query language, but each supplier typically has their own specific query language. Each SPA representing a supplier has its own data feed format and while some of them use well-known international standards such as ONIX[4], others have their own formats, such as tab-delimited, fixed length, or comma-separated-value files. We have defined a simplified format for data exchange between all agents called Trading Information Services Standard Public Format (TIS-SPF). TIS-SPF captures the minimum required information that a trading party needs to know, namely the book identifier ISBN[5], suggested retail price, applicable discount percentage, and available quantity for sale. Other (standard) message formats such as ONIX, might contain much superfluous information. Further information, such as delivery period and currency are available on a per supplier basis and do not need to be included in the data file. When the SPA requests an update from another SPA it would normally receive the update in the TIS-SPF format, and in some cases it might need to translate it to a specific format before any internal processing. This can be done by the SPA itself or it can use a translation Helper Agent (HA).
3. **Translate responses:** The converse of the previous, namely to convert supplier-specific format data into TIS-SPF. For example, when a SPA needs to submit and update to another agent, it must use the TIS-SPF format, and indeed TIS-SPF is the only format accepted by the CDBA for update/publish actions. Thus, an SPA must either be able to translate its specific format to TIS-SPF or use a translation HA to translate and then carry out the submission to the central catalogue. The system we describe here is able to translate from at least 16 supplier-specific formats into TIS-SPF, and *vice versa*.
4. **Update the supplier catalogue** This takes two forms: (i) small changes — *via* a delta file of edits to the current catalogue — which may take place hourly or daily, depending on the supplier and (ii) major changes — *via* a full-feed file that *replaces* the current catalogue — which take place weekly. Clearly timeliness — that is having an up-to-date catalogue is very important — but so also is consistency, and thus an element of soft-checking is employed to protect system integrity. For example, it may be that the supplier has not deposited the update file in the agreed place by the agreed time, so the fetch will result in a null file. Thus it is considered better to retain the old file than to remove all the current entries. In this circumstance the supplier agent is responsible for implementing a retry policy and after a number of failed attempts will advise the supplier's human system administrator of the situation. Likewise, if there is a substantial difference in size or number of entries between the old and new files, this condition is detected and communicated to the supplier's human system administrator.

[4]ONIX for Books; Developed and maintained by EDItEUR jointly with Book Industry Communication (UK), the Book Industry Study Group (US); `http://www.editeur.org/onix.html`

[5]The International Standard Book Number

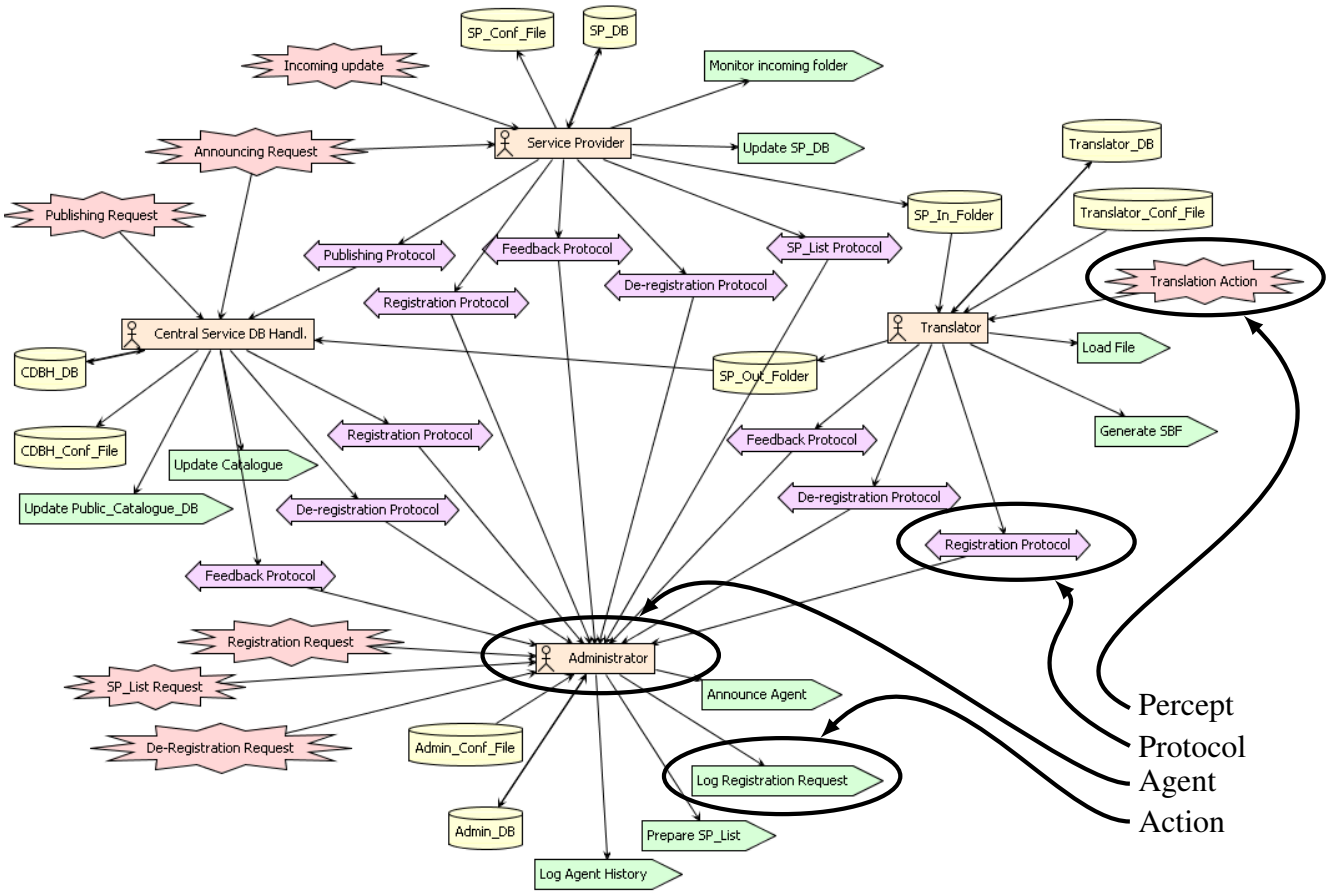

Figure 4: Trader information sub-system overview (designed using Prometheus)

2.5 Helper agents

Helper agents (HAs) are simple service providers and are created (lazily) on demand when supplier agents request the administrator agent for a particular helper function and persist thereafter. There are three kinds of service they provide:

- **Document translation:** For the most part helper agents translate documents between formats, such as from TIS-SPF to the format that a particular supplier uses and *vice versa*. Thus, a service provider agent requests a helper agent to translate a file containing a document in a particular format. This is agreed (using a pre-determined protocol) and the helper agent generates a new file containing the translated material and informs the supplier that the task has been completed. This describes the normal course of events: section 3 discusses what happens under other circumstances. Because the formats in use are all known *a priori*, there is no need for any sophisticated matchmaking/brokerage function: a supplier agent simply asks the administrator for the identity of a helper agent that can translate to or from a specific named format. Thus, as new formats are added to the system, so they are named and registered with the administrator. Nevertheless, maintenance of this approach and continued extension could become tiresome so that a (semi-)automatic approach like that described by Szomszor [19] may be worth investigating in the future.
- **FTP requests:** A second service function of helpers is to carry out FTP requests for suppliers, typically for the purpose of obtaining update files from supplier sites.
- **DB helper:** One critical function for overall system performance is to limit the number of concurrent connections to the DB server. The DB connector agent holds a pool of DB connections to be used by active agents, eliminating the need for establishing new connection on a per agent basis. If all the connections are in use, either a new one will be created, if the limit has not been reached, or the request will be queued and responded to when a connection is freed.
  Another aspect of the DB Helper is to facilitate the process of storing temporary or permanent data for each agent so the agent implementation can be independent of the type of database server and the database structure, and ignorant of how to connect to the database server. Instead, it is just able to send its queries to the DB Helper, that will in turn deliver these queries in the proper format to the database server, and return the result set to the agent. As mentioned earlier, each agent might need to pre-process its catalogue data before executing a public update, so it might ask the CDBA to allocate temporary space on the server, where it can upload, filter carry other functions as necessary.

2.6 Protocol observance

As can be seen from the system overview diagram (figure 4) several protocols are employed in different places to describe the inter-

```
1  PLANS
2    TRANSLATION PLAN
3      SUB PLANS
4        CONNECT PLAN
5          PREREQUISITES
6            COMMUNICATION
7            - Agent: name + handle
8            REQUIRED
9            - Feed back report
10           OPTIONAL
11       END CONNECT PLAN
12       TRANSLATION REQUEST PLAN
13         PREREQUISITES
14         CONNECT PLAN: Success
15           File types: delimited OR
16                       fixed-length OR
17                       xml
18         PROTOCOL
19           REQUEST
20           - Performative
21           - Formats
22           - Paths
23           REPLY
24           - Performative
25         REQUIRED
26           - Feed back report
27             - Success reply
28             - Failure reply
29         OPTIONAL
30       END TRANSLATION REQUEST PLAN
31     END SUB PLANS
32   END TRANSLATION PLAN
33 END PLANS
```

Figure 5: A plan for translation

action between two agents. It would be both clumsy and a source of strong coupling, if these protocols were hard-coded into each agent, as well as making it tedious to update the protocols both during development and later during deployment. In consequence, the service provider agents instead have a task menu that lists each of the goals they might have and in order to fulfill that goal, a SPA asks the administrator agent to send it a plan to achieve the goal. It then simply follows the sequence of actions laid out in the plan, sending messages and waiting for answers. Thus, the SPAs are relatively simple generic agents that when instantiated only know how to request a plan from the administrator, how to interpret that plan and how to provide feedback on the interactions arising from the plan.

Raw XML is not so pleasant for the human reader, so we present a sanitized version of a translation plan in Figure 5. This is an example of the kind of document that an Administrator may communicate to a Service Provider so that it may request a helper agent to carry out a translation for it. On line 7 the AA has identified a helper agent that is capable of providing the service that the SPA requested (the handle is an AgentScape-specific identifier). The document then proceeds to specify (i) the prerequisites (line 15) for the translation process, namely that the input file must satisfy one of the specified types and (ii) the protocol (in this case it is just one message exchange) and the format and content of the messages sent (lines 20–22) and received (line 24)

3. SELF-MANAGEMENT AND FEEDBACK

From the foregoing, it is clear that the administrator agent has a critical role to play and the current design is centralized, in that all agents are responsible to the administrator agent (AA). Thus, a reasonable question at this point is what happens if the administrator agent fails? This centralized approach is an interim solution, for the purpose of enabling operator control over all the agents in the system, however as discussed in section 5, we are planning for an early transition to a fully self-managed and open implementation. Clearly, having a central point of dependency might lead to deadlock if the AA is not available for any reason, so that none of the other agents would be able to communicate or interact with any other. As a short term measure agents currently notify the (human) system administrator by email if they cannot communicate with the AA.

In a commercial setting, software systems are frequently required to run continuously, in line with the global market. This immediately raises several major practical questions:

1. How to monitor the system for anomalous behaviour
2. How to apply corrective action when faults occur, and
3. How to update a running system

The approach taken to this is inspired by Van Roy [16] who addresses the delicate problems that arise from interacting feedback loops. For the sake of keeping this article self-contained, we summarize the basics here and then explain how we have applied this in the case of the virtual warehouse.

Van Roy puts forward a simple pattern that can be applied to the monitoring of distributed systems so that multiple interacting control loops work together rather than conflict with one another. He argues that systems should converge (rapidly) to the desired behaviour and not be perturbed by changes in the system's environment [16], but that in practice it may well collapse, oscillate or become chaotic. The difficulty is that programs typically only behave well close to or possibly even only at their equilibrium point — move the system away from that point by even the slightest amount and disaster may ensue. Similar arguments also motivate the concept of autonomic computing [9], and indeed this is just one scheme whereby a particular pattern of self-management may be implemented. Van Roy proposes that by understanding the relationship between a system and its sub-systems, it is possible to predict system behaviour and thus design a system with the desired behaviour. Thus, a feedback loop consists of three interacting components:

1. A component that **monitors** the state of a (sub-)system
2. A component that **calculates** a corrective action
3. A component that **applies** the corrective action to the (sub-)system

Van Roy goes on to explain how it can be extended to multiple interacting feedback loops, wherein the system parameters depend on each other, but for now that is more than we need. Our scenario has two instances of a feedback loop running in parallel, where the two parties in a transaction provide feedback on each other, as we will now go on to explain.

We have chosen to build monitoring into our agents rather than add a separate collection of monitoring agents. The reason for integration rather than separation of this function lies in our plans for future work in which we aim to distribute control of the system between its members, wherein the agents might learn from their experiences with other agents and adapt their behavior accordingly. For example, when a SPA_a completes a satisfactory data exchange with SPA_b, then SPA_a might choose to add SPA_b to its whitelist, having the feedback information available to and assessed by the agent itself, rather than a third-party.

Thus, each agent collects information about its interactions with

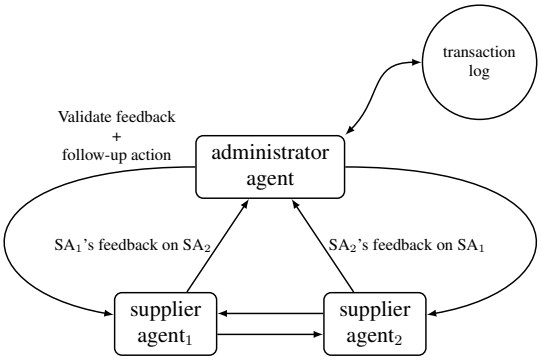

Figure 6: Agent transaction feedback model

other agents — simply as part of executing the protocol — and passes that information to the administrator. We note in passing that it is clear that the administrator could become overloaded by this activity if there were significantly more supplier agents: our long term solution to this is to replicate and federate the warehouse (see section 5).

The overall purpose of the collection of information about transactions is to enable the system to perform a degree of self-management. This in turn currently takes two forms: (i) collecting evidence to indicate whether a given agent is functioning — akin to the heartbeat function of autonomic systems [9] (ii) collecting performance information to inform a ranking of the supplier agents — this aspect is currently unfinished and forms part of our future work plan.

3.1 Recognition of conditions

We implement Van Roy's model by filling in the functions identified above (see figure 6):

- **Monitoring:** carried out by each SPA in the transaction — refer back to the exceptional flows in use case 2 — so that each SPA sends a feedback report to the AA about its transaction.
- **Calculation:** carried out by the AA. The AA receives the feedback reports, stores them in the transaction log in the event that a post-mortem is required, and proceeds to determine which action to take. At present this might be: (i) *sandboxing* the agent so that no other agents can communicate with it, since it appears to be faulty, or (ii) *killing* the agent; typically this applies to internal agents, such as helper agents, that have stopped functioning for some reason, and (iii) *creating* a new (helper) agent to replace the faulty one, or (iv) *announcing* a moratorium on central database updates, if for example, the CDBA appears not to be responding.
- **Actuation:** also carried out by the AA, being one of the actions identified above.

4. THE CHOICE OF AGENTSCAPE

The system has been implemented using the AgentScape [8, 13] platform, which provides a framework for heterogeneous, mobile agents. This section serves to give a brief overview of AgentScape and to outline the advantages that led to its adoption.

4.1 AgentScape

The AgentScape middleware creates a distributed environment that supports multiple, mobile agents for agent-based applications. This middleware provides the functions for agents to perform their tasks, communicate and migrate. In addition, the middleware implements security mechanisms, including in particular agent sandboxing to prevent malicious code from accessing the host machine, and vice versa, to protect an agent from access by the host.

AgentScape is designed to run across multiple hosts, grouped into *locations*. Each AgentScape location consists of one or more hosts running the AgentScape middleware, typically within a single administrative domain. Agents are active entities, residing in a location, that can communicate with other agents and may access services provided by AgentScape. Agents can migrate from location to another, however, it is up to the middleware to determine which host within the location actually runs the agent.

The middleware processes running within AgentScape provide services to agents. For example, *agent servers* provide a run-time environment for agents, a *Message Center* enables agents to communicate with other agents, and a *Web service gateway* enables agents to communicate with web services using the SOAP/XML protocol. Figure 7 (supplied from [13]) shows an overview of AgentScape.

4.2 Perspective on AgentScape

In AgentScape, any number of hosts can form a single location. The application is not aware of the actual number of hosts, so adding hosts does not require changes to the application. This *host transparency* offers a flexible way of distributing an application. Although we are not taking significant advantage of this facility at present, the direct support for distribution is an important factor for the near future.

The middleware manages the distribution of agents among the actual hosts. Each agent is assigned a unique *handle*, that can be used to address agents without having to know or care on which host and location the agent actually resides. This *agent location transparency* eases the task of developing agent applications. Communication between agents is done via sending and receiving messages, which can contain arbitrary data.

Each agent runs in an isolated environment, and can only communicate with other agents by sending messages, addressed to the receiver's *handle*. The messages are delivered to the correct receiver by AgentScape, regardless of the host on which it is running.

AgentScape also supports migration of agents, which is useful for allowing agents to have local access to certain resources thereby reducing communication overhead.

That is not to say that the experience of using AgentScape has all been positive, although we are well aware this is software under development. Various shortcomings are a current lack of documentation, difficulties in killing threads, the complicated nature of testing and debugging in a non-deterministic environment (not restricted to AgentScape of course) and the lack of an automated clean-up on system shut-down. At a different level, there is the agent programming model: AgentScape is purposely an agent operating system that permits the user to write agents in whichever language they wish using whichever internal model they will, while providing the platform functions through an API. We are aware of a need for higher-level support for full-agency *within* the trading platform agents and are currently evaluating various possibilities, including Jason[2].

5. REFLECTION AND DISCUSSION

A number of reflections on the rationale and process are scattered through the article. In this section we aim to draw these together and look back on the development experience so far. Then, we discuss briefly our plans for future developments of the trading platform under the headings of load-balancing, ranking, service identification and self-management.

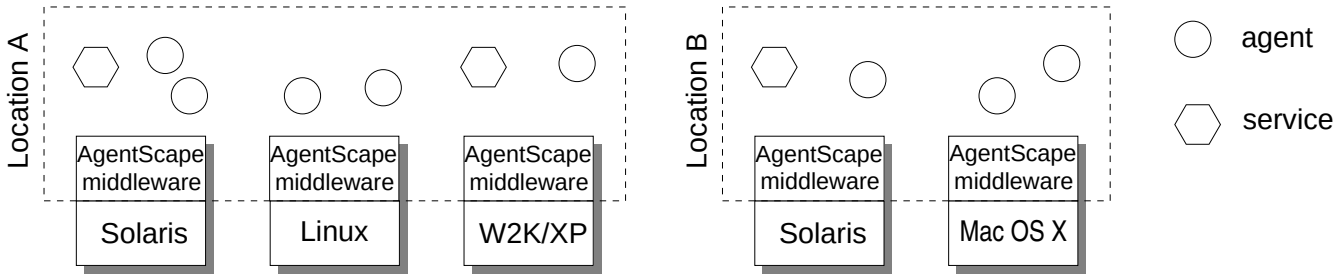

Figure 7: AgentScape middleware environment

Deployment of and cut-over to the agent implementation is a similar story to that of most software systems. The agent-based implementation was run for two weeks in parallel with the existing conventional system, starting with a few of the large suppliers, during which time data integrity checks were made regularly, gradually adding more and smaller suppliers until the full complement were engaged. There were numerous instances of manual intervention during this period, while the integrity checks were a mix of automatic and manual verification, covering issues such as the number of records, values after and before calculations, performance (transaction/time) and error reports. Some problems were at the functional level, such as when the Administrator Agent shut down for three days and no new agents could register, others were at the supporting level, such as memory leaks. One characteristic of using agents that has been particularly beneficial, is the ease with which we are able update the running system. In that sense, the current deployment is never the final system, but just a step on the way.

Although it might seem quite obvious, we are very positive about the utilization of agents and agent-oriented software engineering in building business applications, even if like any development process, it has not been without its problems. From the company's point of view, the vindication of the technology is how it has helped the company to carve out a niche in the book supply industry and this largely rests on the effectiveness of the (software-supported) supply-chain, parts of which we have outlined here. Because development was also a learning experience with new technology, the initial costs were inevitably higher, but the payback has come with lower overheads in deployment, update and extension. For example, we can test and incorporate a new supplier agent into the running system in under half an hour.

While organizations invest in building workflow management systems and B2B integration to be able to handle multiple transactions in support of their business processes, multi-agent systems are rarely part of this process. Building supply chains is probably one of the more common applications of agents, as highlighted in [10], and so the degree of novelty here is possibly not very high. On the other hand, although there are doubtless many such systems deployed, detailed descriptions are not that easily found. We have chosen to invest the time in exploring the potential advantages of using multi-agent systems to facilitate the huge number of transactions that are part of a live business system, which in itself was a risky strategy. Looking back on this, we believe the current system is evidence for the benefits of agent oriented architectures in general and the current state of development holds plenty of promise for the near future.

Consequently, we believe the main contributions of this article are the exposition of the architecture that has been deployed, and more importantly, the message that both the ideas and the tools being developed in the software agents community can help deliver sophisticated software systems. In particular, we emphasize how the abstractions of agents and institutions, the principles of self-management and the agent-oriented SE tools have actually helped both in the conceptualization and (rapid) realization of the solution, as well as enabling its short- and longer-term evolution.

5.1 Load-balancing and mobility

An unsurprising artifact of the need to have a functional system in a short timescale is that we have not taken advantage of Agent scape's built-in support for distribution and mobility. As currently deployed, the system runs on a single platform and so although the transaction load has not reached current throughput limits, we are aware of the need to replicate various components of the system — especially the administrator agent, as mentioned in section 3 — and distribute them across several physical platforms for the sake of both resilience and load-balancing. We will be addressing these issues in the near future.

5.2 Ranking

One of the most important issues within trading relations is the level of trust between the trading parties. That is a given in our existing closed system, but as we move towards an open system, it is necessary to establish degrees of security and agents should be able to assess each other before getting involved with risky transactions. We will use ranking functions operating on an individual (private) level and a system (global) level. The private ranking is maintained by each agent about the other agents with which that it has engaged in transactions. The global ranking will be maintained by the administrator agent and will function as an indicator of the degree of stability and reliability of an agent, as well as warning the administrator when it may be necessary to consider quarantining an agent whose ranking falls below some threshold [7].

5.3 Adapters

Translation from supplier-specific format to TIS-SPF and *vice versa* is currently handled on a case-by-case basis and while it is manageable at the current scale, this is clearly a problem that could get worse. It is however also a problem that has been encountered before, in particular in the bio-informatics community [6], where the solution has typically been to write scripts for each mapping required and then automatically insert them in the workflow where necessary. In fact, there are two problems here: one is authoring and maintaining an increasing number of mapping scripts, the other is finding the right one when it is needed. Szomszor [19] describes a sophisticated and extensible approach using OWL reasoning to synthesize and discover adapters, that may be applicable to our needs.

5.4 Self-Management

The current system cannot survive a failure of the administrator agent. Furthermore, the administrator is also a bottleneck, so placing a limit on the number of transactions that the system can handle. We believe that the correct solution to this problem is to move towards decentralized architecture using principles from peer-to-peer network. We intend to refactor our system in two phases to achieve a P2P structure: in the first phase we will use a hybrid architecture with a centralized directory storing the locations of the services of the system, and in the second phase we will move to a relaxed-ring structure[11] linking the helper and administrator agents to achieve self-healing properties[5].

6. ACKNOWLEDGMENTS

Thanks to Michel Oey (VUA) and Reinier Timmer (VUA) for assistance with using and describing AgentScape. Frances Brazier (VUA) provided feedback on an earlier version of this article. Emad El-Deen El-Akehal is partially supported by The Book Depository Ltd. and by an EPSRC CASE award through the UK's South-Western Regional Development Agency (SWRDA).

7. REFERENCES

[1] J. Armstrong. *Making reliable distributed systems in the presence of software errors*. PhD thesis, Royal Institute of Technology (KTH), Kista, Sweden, 2003.

[2] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-agent Systems in AgentSpeak Using Jason*. Wiley, 2007. ISBN-13 978-0470029008.

[3] O. Cliffe. *Specifying and Analysing Institutions in Multi-Agent Systems Using Answer Set Programming*. PhD thesis, Dept. Computer Science, University of Bath, June 2007.

[4] O. Cliffe, M. De Vos, and J. Padget. Specifying and reasoning about multiple institutions. In P. Noriega, J. Vázquez-Salceda, G. Boella, O. Boissier, V. Dignum, N. Fornara, and E. Matson, editors, *Proceedings of Coordination, Organizations, Institutions and Norms (COIN) at AAMAS06*, volume 4386 of *Lecture Notes in Artificial Intelligence*, 2006.

[5] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. Towards architecture-based self-healing systems. In *WOSS '02: Proceedings of the first workshop on Self-healing systems*, pages 21–26, New York, NY, USA, 2002. ACM.

[6] D. Hull, R. Stevens, P. Lord, C. Wroe, and C. Goble. Treating 'shimantic web' syndrome with ontologies. In *First Advanced Knowledge Technologies workshop on Semantic Web Services (AKT-SWS04)*, volume 122 of *KMi*. The Open University, Milton Keynes, UK, 2004. Workshop proceedings available from CEUR-WS.org. ISSN:1613-0073. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-122/.

[7] T. D. Huynh, N. R. Jennings, and N. R. Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, 2006. ISSN 1387-2532. Last viewed December 2007 at http://eprints.ecs.soton.ac.uk/9559/1/dong-ecai2004.pdf.

[8] IIDS. AgentScape Agent Middleware. http://www.agentscape.org.

[9] J. O. Kephart. Research challenges of autonomic computing. In G.-C. Roman, W. G. Griswold, and B. Nuseibeh, editors, *ICSE*, pages 15–22. ACM, 2005.

[10] M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*. AgentLink, 2005.

[11] B. Mejias and P. V. Roy. A relaxed-ring for self-organising and fault-tolerant peer-to-peer networks. *sccc*, 0:13–22, 2007.

[12] P. Noriega. *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autonoma de Barcelona, 1997.

[13] B. J. Overeinder and F. M. T. Brazier. Scalable middleware environment for agent-based Internet applications. In *Applied Parallel Computing*, volume 3732 of *Lecture Notes in Computer Science*, pages 675–679. Springer, Berlin, 2006.

[14] J.-A. Rodríguez, P. Noriega, C. Sierra, and J. Padget. FM96.5 A Java-based Electronic Auction House. In *Proceedings of 2nd Conference on Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM'97)*, pages 207–224, London, UK, Apr. 1997. ISBN 0-9525554-6-8.

[15] J. A. Rodriguez-Aguilar. *On the Design and Construction of Agent-mediated Institutions*. PhD thesis, Universitat Autonoma de Barcelona, 2001.

[16] P. V. Roy. Self management and the future of software design. *Electr. Notes Theor. Comput. Sci.*, 182:201–217, 2007.

[17] J. V. Salceda. *The role of Norms and Electronic Institutions in Multi-Agent Systems applied to complex domains*. PhD thesis, Technical University of Catalonia, 2003.

[18] C. Sierra, J. Thangarajah, L. Padgham, and M. Winikoff. Designing institutional multi-agent systems. In L. Padgham and F. Zambonelli, editors, *AOSE*, volume 4405 of *Lecture Notes in Computer Science*, pages 84–103. Springer, 2006.

[19] M. Szomszor. *Dynamic Discovery, Creation and Invocation of Type Adaptors for Web Service Workflow Harmonisation*. PhD thesis, University of Southampton, April 2007.

[20] J. Thangarajah, L. Padgham, and M. Winikoff. Prometheus design tool. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *AAMAS*, pages 127–128. ACM, 2005.

[21] J. Vázquez-Salceda, J. Padget, U. Cortés, A. López-Navidad, and F. Caballero. Formalizing an electronic institution for the distribution of human tissues. *Artificial Intelligence in Medicine*, 27(3):233–258, 2003. ISSN: 0933-3657, available via http://dx.doi.org/10.1016/S0933-3657(03)00005-8.