A Multi-Agent Framework for Distributed Business Intelligence Systems

Alexander Loebbert
Bond University
aloebber@bond.edu.au

Gavin Finnie
Bond University
gfinnie@bond.edu.au

Abstract

This paper describes the design and development of a Multi Agent Based Business Intelligence System (MABBI). MABBI uses agent technology to address issues in the field of Business Intelligence (BI) including integration into business processes, reduced latencies and decision automation. Specifically the research discussed in this paper addresses the relative value of local as opposed to global business intelligence and describes an approach to implementing a multi-agent solution to local intelligence provision. To illustrate the MABBI concept a prototype is described (pMABBI) that applies the technology in the retail industry to automate item-level pricing decisions. The implementation of the prototype is explained and some results in a simulated environment are discussed.

1. Introduction

Organisations are generating increasingly larger volumes of data, driven by regulatory requirements, business needs or sometimes just because the technology is already in place. Using this data in business decisions can be difficult as the massive amount of data may make it hard to handle and process to gain actionable information. Often it is unknown what insights are hidden in those databases. Recent research in the Business Intelligence area is aimed at real-time use of BI, integration into business processes, automation and providing methods to adjust to the environment and the adaptiveness of the system [3-5, 14]. Software systems, including BI, have to be designed to be flexible to cope with today’s demands and frequently changing environments.

Most BI systems take a centralized global view of providing decision support information. A large data warehouse serves to consolidate corporate data and provide analysis of this data as needed. The current research addresses the question how to provide local decision making capability using BI tools as opposed to the global model. Specifically in cases where local conditions may vary from the global perspective, what models are available to support this type of decision support? The models should provide a standardised environment and facilitate consolidation for centralised use if necessary. The Multi-Agent approach provides a potential architecture and development methodology suitable for this type of distributed decision making.

To demonstrate the Multi Agent Based Business Intelligence System (MABBI) concept we have developed a prototype system, pMABBI, in the retail-pricing context. Pricing, in particular in the grocery industry, is a challenging task, with low margins and high competitive pressure. “Think Global, Act Local”, combining the advantages of a centralised organisation with local customer focus [18, 22] is a strategy retailers may try to follow which further increases the complexity of pricing decisions.

The paper follows a design science research (DSR) approach as described for example by Hevner et al. [11] and adopts the research process suggested by Peffers et al. [17]. DSR seeks solutions to “important and relevant business problems” [11] and MABBI aims to do this. The sections of this paper match Peffers process as follows. Section 1 and 2 describe the problem and motivation behind the research (Activity 1). Design objectives and design of the artifact are established in Section 3.1 and 3.2 (Activity 2 & 3). Demonstration and Evaluation of the artifact are presented in 3.3 and 3.4 (Activity 4 & 5). Activity 6, communication, is the paper itself.

The remainder of this paper is organised as follows: We first provide an overview of the technologies that built the foundation for the system and give a short description of current retail pricing issues. Then the MABBI concept and its components are presented in detail followed by the description of the prototype system. The simulation process is described and some simulation results discussed. Finally we summarise the key points of the research and give directions for future research.

2. Background Technologies

2.1. Decision Support Systems / Business Intelligence
Designing and implementing BI systems to gain actionable information or knowledge can be challenging for organisations. Reasons include (1) the increasing amount of data, data sources and their integration (2) knowledge of relations and influence factors of the objects to be analysed at design-time (3) integration into business processes and (4) faster reaction times. In this context terms like Real Time BI, operational BI, localised BI, embedded BI and similar have emerged [2, 13, 14]. Despite the different terminology the general goal is higher level of integration of BI and decreasing the delay between the occurrence of an event and the time the processed information is accessible for the organisation.

BI techniques and concepts have developed and improved over recent years to better address business needs. Tools to support the development and configuration of systems as well as best practice guidelines are available and in use. BI still remains challenging for businesses. For example Barone et al. [4] point out that data is readily available but “meaningful and productive” use of data can be difficult and require “significant efforts” of IT staff.

The decision process itself has become more complex. This means in respect to the analysis and data mining techniques employed in BI that one method is rarely sufficient to achieve a satisfying result. Systems that combine different mining methods are referred to as Hybrid Intelligent Systems [25]. They argue that the design of such systems is complex as they consist of a large number of different components. Agent and in particular multi agent systems present an IS design paradigm that has the required functionality and characteristics to deal with such hybrid systems.

2.2. Multi Agent Systems

Merriam-Webster’s dictionary defines the word agent as “one who is authorized to act for or in the place of another” [12]. Similar to Object Oriented Programming (OOP), Agent based software development is built around the metaphor of “copying” real world objects into digital replicas. Defining agents accurately, in an IT context, is difficult because of the variety of functions and applications an agent can have or execute. Wooldridge’s [13] definition of agents is one of the most cited: “An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives”.

Wooldridge notes that although the agent approach to systems development has some similarity to the object-oriented approach, agents do not expose methods (or other ‘internals’) to the rest of the application or system. In an agent application it is only possible to pass a message to an agent and ‘hope’ it gets back to you with the result you want. Agents are to some extent autonomous and it is not necessarily in the respective self-interest to pursue a goal (execute a function) that another agent desires.

The agent metaphor is not just a way to design software, but can be used as a type of software architecture. Agents have been used to support data mining. Cao et al. [14] give a general introduction to the topic of agent / data mining interaction. Zhang et al. [15] describe the opportunities for the integration of data mining and agents in more detail. One particular field of application of agents in a data mining context is distributed data mining which is explored by Satoh [16]. Information retrieval and delivery is a common application of agents, for example [17]. Landqvist and Pessi [18] also cover the field of information retrieval dealing with personalised information for the user.

Perko & Bobek [19] suggest the integration of agents, BI and knowledge management.

2.3. Pricing

The roots of pricing theory go back to economic supply and demand theory, developed in the 19th century, when products were commodities and purely fulfilled customers’ basic needs. Today however these economic theories might only apply in small markets, e.g. where professional buyers trade commodities. In most other markets price, supply and demand are much more complex to determine as consumers do not just buy because they need, but because they “want to” [20]. Simon et al. [21] are the same opinion and use the term “classic pricing theory” and stress the microeconomic objectives of these theories and their limited applicability. The price eventually links supply and demand. Pricing for managers remains a problem. Often pricing is not addressed appropriately, which results in a “cost plus something” approach or other rule of thumb strategies. This “strategy” is surprising, as price is one of three profit drivers, together with volume and costs, in the retail industry. Retailers are more focused on reducing costs and increasing volume than improving pricing, which has a much greater impact on profits than the other two drivers [21]. A study by Marn and Rosiello [22] investigated the impact of price on company profits and showed that a mere increase of 1% in price resulted in an average of 11% in additional profit.

Schwind [23] classifies pricing in two main categories, static and dynamic. Static refers to pricing that does not change over time (or price changes in the long run) and the buyer has no influence on the pricing process. Dynamic on the other hand refers to approaches which include the buyer directly.
(interactive pricing) or indirectly (dynamic price posting). Indirectly in this context usually means that the seller analyses sales/demand data to adjust prices accordingly. Elmahraby & Keskinocak [24] view dynamic price posting as current retail pricing practice. The trend towards a more dynamic and customer focused pricing, is driven by advances in IT, in particular scanner based checkouts. The advent of web based shops (with easy to change prices) brought the dynamic pricing approach back into marketing focus [23, 24]. Levy et al. [25] describe the process of changing prices in the retail environment (bricks and mortar stores) in all its complexity and challenges for managers and stores personnel. Today Electronic Shelf labels (ESL) eliminate cost and time associated with printing and replacing of paper labels. The availability of demand data and advances in data mining allow retailers to use technology enhanced pricing strategies to align pricing with customers willingness to pay [26].

3. Multi Agent Based Business Intelligence (MABBI)

3.3. The MABBI Concept

Research into BI has asked for systems that are adaptive, flexible, embedded and similar [e.g. 3, 13] and that decrease the delay between the occurrence of an event and the point in time the processed information is accessible for the organization. Some authors propose approaches that go even further and suggest that the software system should be empowered to make decisions rather than just support the decision maker [2, 5].

Research into data driven agent or agent/data mining interaction is somewhat disconnected from BI research. MABBI aims to address the need for a more flexible BI that can be embedded and deliver localized decision-making capabilities in a distributed environment. This happens by using the agent metaphor in three ways: 1) as a means of a flexible architecture to represent the structure of the organisation, 2) use data mining to make the agents intelligent in a sense that they optimise their behaviour and 3) as a software development metaphor to encapsulate different functions and objects.

The core component of the MABBI approach is a Decision Unit (DU), an agent that has the necessary function and methods to make a decision in its respective domain. Figure 1 shows the design of a Decision Unit and its components.

3.1.1. Database / Data Warehouse (DB/DW). - Every decision unit has its own data storage or access (via Web Service, TCP/IP, Named Pipes, etc.) to a dedicated database or database server. (A relational database might be the most common form of data storage but others are possible). This data storage is used to store 1) agent meta data like the Agent ID and other parameters, 2) data about the problem and problem domain (e.g. Pricing and Sales data), 3) feedback data (learning or experience data). The DB/DW component is the agent equivalent to the Agent’s Belief database / set.

3.1.2. Data Mining / Knowledge Discovery (DM/KD). - The DM/KD module in a DU has the role of analysing the problem data and delivering a result that can be used in the decision execution module. Methods used might vary depending on the problem domain and environment; however Time Series Analysis, Artificial Neural Networks and k-Nearest Neighbour are methods that are frequently used in BI. Complex systems might interact with specialized software such as Mathlab.

3.1.3. Decision Execution (DE). - In an approach which may differ from traditional BI/DSS systems, a DU in MABBI can implement the result of a decision into the business process at the local level. This means that as soon as new knowledge becomes available the organisation can benefit and act upon it. It is not necessary that a user transfer the results from the BI system to the operational system, which can be a time consuming and error prone process.

3.1.4 Learning (Feedback). - To improve the outcome of the Decision Unit for future decisions a learning component tracks the “real world” after a decision was implemented and adjusts future decisions based on the feedback if necessary. The learning module is closely related to the DM/KD component to utilize different methods and models. To give a DU the means to do this, previous outcomes can be logged in the DB and used as supplementary input. This feature is similar to the concept of Adaptive BI suggested by [14].
3.1.5. Communication. - Flexibility is a key aspect of the system that requires broad communication facilities. In Figure 1 the communication layer around the DU is easily recognisable. It indicates that all modules in a DU can communicate (e.g. have access to data sources / operational systems). If required, DU can communicate with each other. These communication capabilities can be realised in various ways. The prototype for example uses a blackboard approach [21].

3.1.6. Configuration Engine (CE – not included in Figure 1). - The Configuration Engine’s task is to monitor systems, for example a DB and apply changes to the structure of the agent system. The Configuration Engine, which in turn is also an agent, is a helper in the system where DUs cannot take required actions. For example the CE executes the creation of a DU, as a DU cannot create itself. Once a DU is created it accesses suitable DBs to acquire data and information that is necessary to adjust to the environment and structure (eg the organisational hierarchy). After receiving the start up parameters the DU acts on its own and performs tasks according to the design objectives. The internal or local DB allows storing data that is of relevance for the particular DU and the current context. In contrast to traditional distributed data mining [16] the models generated in the DM/KD module are not shared and aggregated to a system wide model.

3.2. pMABBI Prototype

To demonstrate the MABBI concept and the localised decision making capabilities, we have implemented a prototype system in the context of item pricing in retail chains. This problem domain was chosen as frequent updates are required of a large number of items in a geographically dispersed organisation. Current practices generally collect store level data that is transferred to the corporate headquarters (HQ) for analysis. HQ processes the data for all stores and distributes it back to the stores. Working with data that is in some way aggregated or pruned may lead to non optimal results [22, 24]. This means that item level transactional data is collected and stored but the investment in barcode systems and advanced checkout systems are not leveraged to their fullest potential.

The pMABBI prototype addresses these problems with its modular decentralised structure. This means that each store has its own Decision Unit, which can adjust to local market condition by analysing local POS data, making predictions of future demand and adjusting prices based on local store demand characteristics. Despite the fact that autonomy is one of the key attributes of agents, business software has to conform to the general strategic focus of the organisation. This means that the autonomy of an agent might have to be limited in some cases, and non-optimal (non-data driven) decisions have to be implemented. For this reason a DU contains a decision execution module (DE) that ensures that only allowed decisions are implemented. What is allowed can refer to various criteria, ranging from corporate strategy, marketing or legal / regulatory requirements.

Figure 2 shows how the system relates to the hierarchy of the underlying organisation. There is one HQ with a number of stores. All components can communicate in some way, e.g. XML Web Services or ‘black board’ posting through a DB system.

The architecture of the test bed is illustrated in Figure 3. It is an all Microsoft .Net implementation, using C# code for the GUI, SQL Server Suite as the Database and Data Mining platform and Axum as the agent language. Axum is a prototype programming language developed by Microsoft [7, 8]. The language follows the actor model and is conceptually influenced by languages like Scala and Erlang. Syntactically however Axum is very similar to C#, integrated in the .Net Framework and implemented on top of the Concurrency and Coordination Runtime (CCR). OOP constructs, like Classes, Interfaces and Structs do not exist in Axum. Instead the language uses Agents, Domains and Channels as building blocks [7]. One aim of the developers of Axum was to provide a language that forces software developers to follow parallel and
concurrent design instead of providing libraries that just enables developers to do so [8].

Axum defines four major components that are different to OOP, namely Agent, Channel and Ports, Domain and Schema. The OOP concepts like Class, Interface, Property etc. do not exist in Axum [7].

- In Axum an agent is the organization unit that is the ‘implementing end of a channel’. It is the structure that contains the program code to perform actions on the data. Agents can have Read/Write, Read or No Access rights on domain state.

- A channel is the way agents communicate. In OOP programming a channel is roughly comparable to an interface of a class. Note that a channel is instantiated rather than the agent directly. Each channel defines one or more ports. A port is an input (into the agent) or output (return value from the agent) of a specific data type.

- A domain is an isolation unit that can contain agent definitions and host objects for agents.

- Schemas in Axum are used to define sets of data that can be transferred between agents. The syntax is similar to a channel. However a schema is not implemented by an agent but it can be used by an agent. Schemas provide basic rule support; fields can be defined as required or not empty.

Axum is a very interesting approach to agent oriented development. It integrates with tools that developers already use and the syntax is very similar to mainstream languages. This means that developers can focus on the new agent metaphor without having to get used to new tools. In general, it will be challenging to use libraries in agent systems, as the functions offered have to be atomic in nature.

Microsoft already announced that they will not continue to pursue a production release of Axum. However, the fact that a ‘big player’ like Microsoft has shown interest in the field of agent oriented software development might give the agent community some momentum.

3.3. Simulation and Evaluation

Testing and evaluation of multi-agent systems is challenging. Experiences and best practices are not yet established or not yet agreed on and systems are inherently complex [9, 20]. The approach chosen for the evaluation of the pMABBI system is an experimental/simulation approach [10, 23], which is listed as a suitable evaluation option by Hevner et al. [11] and consistent with agent literature [19]. The advantage of simulations in general, compared to other methods, is the possibility of analysing the system in its context. It further allows defining test cases and scenarios, which might be difficult to find in real data. In this retail situation, output of the simulation system is synthetic demand data (Market Basket Data) i.e. what products the customers want at specific prices. This data in turn is the input of the pMABBI system and the control system (the centralised system).

To test the application we used a software-in-the-loop simulation approach. This means that the simulated Customer is part of the pMABBI application. Simulation of customer demand is challenging for two reasons. Firstly, marketing models are generally qualitative and subsequently not transferable or only partially transferable to actual simulation code and secondly there is no general model that describes the buying and decision making of a customer. This generally results in simplified models.

3.3.1. Customer - Demand is simulated as a population of customers that shop in the respective stores. There are various theories and models that describe customer and buying behaviour, however they are often qualitative and difficult to transfer into quantitative models that can be simulated. To focus on IS issues a simple decision model was implemented. The model integrates three generic and an income/budget attribute that reflect the buying characteristics of a customer and influences the buying decision. This approach is based on [6, 12, 15]. The implemented customer workflow is illustrated in Figure 4.
Figure 4. Customer Buying Process

The customer decides, based on the value of a product, whether or not she will buy the product. At each shopping round this value is calculated for every customer/product combination and evaluated as to whether the value is enough to trigger a purchase. The value function is adopted from [12].

\[ V_{ij} = U_{ij} \times B_i \times (1 - P_j) \]

Product Value Function [12]

with

- \( V_{ij} \) Value for money of product \( j \) for consumer \( i \)
- \( U_{ij} \) Utility for consumer \( i \) for product \( j \)
- \( P_j \) Price for Product \( j \)
- \( B_i \) Budget of consumer \( i \)

3.3.2. Assumptions. - The following major assumptions were made in setting up the simulation. Although not necessarily realistic, they provide a simplified environment to assess the design.

- Change in Quantity sold has no impact on Cost (production)
- There is an endless supply of product
- No out of stock situations arise
- Price \( \geq \) cost i.e. no loss leader situations
- Cost \( \geq 0.25 \) (the minimum prices chosen for this simulation)
- Products are independent - there are no substitutes available

- All variables (such as customer and product characteristics) are discrete in the set \( \{0.25, 0.5, 0.75, 1\} \)
- Customers are perfectly rational

3.3.3. Timing. - The simulation distinguishes between two types of stores, called Agent and Head Quarter stores (HQ Stores). Agent stores implement the pMABBI system and time is recorded in week slices (3 slices a day, 21 a week). HQ Stores implement a traditional centralised approach for comparison with the pMABBI stores. The HQ Stores are timed on a weekly schedule.

3.3.4. Pricing - Every product comes with an assigned price (i.e. recommended retail price). To allow the system to collect sales data at various price points, the first week of the simulation is used for “Price Testing” [6]. During this price testing period the price of a product is not calculated but a random price is selected from the discrete set (see assumptions) and assigned to the product as an initial price.

The pMABBI system uses a simple approach to determine the price for individual products. As noted in the literature, there are a number of challenges involved in pricing and there are a multitude of qualitative and quantitative approaches documented. The pricing algorithm used here should be conceptually correct to illustrate the function of the system. This algorithm does not claim to be either new or optimal. Store profit is assumed to be the single objective. The overall directive of the system is to improve (i.e. increase) profits. Profit is calculated for Product \( R \) at price \( i \) in time period \( t \). Cost \( C \) of a product is constant during the simulation and can be seen as Total Cost (Fixed + Variable).

\[ R_{it} = (C_tP_{it})D_{it} \]

Revenue

With

- \( R_{it} \) Revenue for Product \( i \) at time \( t \)
- \( C_t \) Cost for Product \( i \)
- \( P_{it} \) Price for Product \( i \) at time \( t \)
- \( D_{it} \) Demand for Product \( i \) at time \( t \)

To predict future demand, respective price/sales data of a product has to be analysed in form of a time series. The mining model used in SSAS is the Microsoft Artificial Neural Network algorithm [e.g. 1]. ANNs were chosen to better reflect the relationships between unit sales and varying prices over time.

Inputs to the demand forecast are:
Output of the model is unit demand at $t+1$.

### 4. Simulation Results

The goal of the pMABBI prototype is to show that the MABBI architecture can deliver decision-making capabilities to individual retail stores and utilise local knowledge to implement appropriate changes to business processes.

For every simulation run the test bed software creates a new set of SQL and SSAS databases. During the initiation stage, parameters like duration, store count and product count are read from the GUI and the required data is inserted into the respective DBs. This completes the simulation setup process. The user starts the simulation timer object that controls the simulation. It will wait till other simulation objects report that they completed the initialisation process and are in a ‘runnable’ state. The timer then sets the simulation status to running and increments the simulation clock until the simulation is complete.

The simulation results presented here are averaged over three runs. In each run 3 pairs of stores sell 50 different products each and have a customer base of 50 shoppers. Each of the pair consists of a HQ Store (old centralised approach) and an Agent Store (pMABBI approach) that have the same start up parameter. This means that the characteristics of the customer and the products are equal, thus a direct comparison is possible. In other words, the results present 9 comparisons between de-centralised (MABBI) and a centralised (tradition) approach.

The duration of each simulation run is 8 virtual weeks. To add dynamic variation to the simulation, customers can change during the simulation. Change is introduced randomly. If a customer changes, one or more of the 3 character attributes can change by ± 0.25. The respective DUs concerned with pricing have to pick up this change in demand and react by adjusting the price to maximize profit.

Table 1 shows the overall sales of all stores over 3 simulation runs. The stores that implement pMABBI sold significantly fewer items (34.58% of the sales of the comparison group). However the pMABBI stores realised a much higher average price that was 82.3% over the average price of the comparison group.

Table 2 summarises the average values. In Figure 5 these numbers are broken down to stores (1 – 6) and grouped by pMABBI (y) or comparison group (n).

### Results (total)

<table>
<thead>
<tr>
<th></th>
<th>pMABBI Stores</th>
<th>comparison stores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Sales</td>
<td>617,504</td>
<td>1,785,407</td>
</tr>
<tr>
<td>Total Cost</td>
<td>204,916.25</td>
<td>609,400.50</td>
</tr>
<tr>
<td>Total Revenue</td>
<td>383,668.25</td>
<td>732,277.25</td>
</tr>
<tr>
<td>Profit</td>
<td>178,752.00</td>
<td>122,876.75</td>
</tr>
<tr>
<td>Avg Price</td>
<td>0.62</td>
<td>0.34</td>
</tr>
</tbody>
</table>

### Results (average)

<table>
<thead>
<tr>
<th></th>
<th>pMABBI Stores</th>
<th>comparison stores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Sales</td>
<td>68,612</td>
<td>198,379</td>
</tr>
<tr>
<td>Total Cost</td>
<td>22,768.47</td>
<td>67,711.17</td>
</tr>
<tr>
<td>Total Revenue</td>
<td>42,629.81</td>
<td>81,364.14</td>
</tr>
<tr>
<td>Profit</td>
<td>19,861.33</td>
<td>13,652.97</td>
</tr>
</tbody>
</table>

The results of the simulations presented here suggest that the MABBI architecture applied in form of the pMABBI systems does have advantages over the centralised / traditional system design. The results are understandably limited by the synthetic nature of the simulation – the primary value of the simulation has been to show that the implementation does in fact work and that the architecture is a viable form of solution. It might be beneficial to extend the simulation system towards a more realistic artificial economy that is parameterised with real data from retailers. This might help to evaluate in more detail other, in particular qualitative benefits of MABBI, for example during deployment.
Figure 5. Sales / Profit

9. Conclusion / Future Research

This research proposed a new architecture, called MABBI that combines Business Intelligence with Multi Agent Technology to deliver localised decision-making capabilities throughout an organisation. Traditional business intelligence solutions use centralised data warehouses and distribute the results to local level if needed. Most applications will tend to use a global view of the data for analysis. It is however possible to consider environments in which local knowledge is more valuable than a global assessment. In pricing for example, issues such as local tastes and socio-economic level will influence demand for specific products which may differ from demand in other locations. The research issue considered was whether an agent-based architecture could deliver a solution to such local decision making in a business intelligence context i.e. learning from and analysing the local data.

A prototype, pMABBI, was implemented in the context of retail pricing. Simple simulation results indicate that the distributed nature of MABBI and the embedded DU could improve store profitability compared to a traditional centralised system. The prototype was tested using a simulation system and results suggest that the pMABBI, using local transactional data, outperforms the comparison stores. Although the simulation results are much to be expected, their real role was to provide a test environment to show that the architecture was in fact a viable solution.

The pMABBI system focuses exclusively on the pricing problem, however the MABBI architecture can be applied to various other problem domains. In particular domains with high decision frequencies and where local characteristics differ. Fleet management or patient monitoring are examples of such environments. Either cars/trucks or patients are represented by DUs and make decisions that are most suitable for the respective local environment.

An interesting part of this work was the evaluation of Axum as a potential agent environment. Axum and in particular the integration with mainstream development tools, is an interesting and promising approach to allow the use of agent oriented software development and system design on a broader scale. Although Microsoft is not continuing with Axum as a product, a number of the concepts will undoubtedly become further developed.

Future research in the MABBI domain will focus on how to transform the concept into a library that can be more easily used, to apply the concept in more areas to further evaluate and benchmark MABBI.

10. References


