View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Abertay Research Portal

Simulation of Complex Environments: The Fuzzy Cognitive Agent

D Borrie, S. Isnandar, C.S. Özveren
School of Computing and Creative Technologies
University of Abertay Dundee
Dundee, Scotland, DD1 1HG, UK
c.s.ozveren@abertay.ac.uk

Abstract

The world is becoming increasingly competitive by the action of liberalised national and global markets. In parallel these markets have become increasingly complex making it difficult for participants to optimise their trading actions. In response, many differing computer simulation techniques have been investigated to develop either a deeper understanding of these evolving markets or to create effective system support tools. In this paper we report our efforts to develop a novel simulation platform using Fuzzy Cognitive Agents (FCA). Our approach encapsulates Fuzzy Cognitive Maps (FCM) generated on the Matlab Simulink platform within commercially available agent software. We will firstly present our implementation of Matlab Simulink FCMs and then show how such FCMs can be integrated within a conceptual FCA architecture. Finally we report on our efforts to realise an FCA by the integration of a Matlab Simulink based FCM with the Jack Intelligent Agent Toolkit.

1. Introduction

The growth of liberalised trading markets and the expectation of the potential benefits that they will bring continues to grow worldwide. Consequently to better understand the market domain and to gain competitive advantage, considerable interest has developed in the application of computer based simulation techniques, and in particular in advanced modelling tools such as neural, fuzzy, and hybrid systems. However, the application of these techniques has proven to be difficult as a direct construct of the unpredictability of the market variables and the complexity of their interaction.

The domain of political science exhibits similar complexity and in this field FCMs have emerged as a powerful simulation technique due to their inherent abstraction, and structural and relational flexibility [1]. Such characteristics are ideally placed to deal with large, imprecise, complex, multivariate systems and environments.

In previous papers we have demonstrated the potential of FCM to represent aspects of competitive electricity markets [2]. These FCMs were generated on within the Matlab Simulink platform, which offers considerable advantages over previously described FCMs. The foremost of these advantages is the combination of mathematical robustness and graphical capability provided by the Matlab Simulink platform. This permits the encapsulation of a wide range of complex concept interdependencies greatly increasing the overall representational accuracy. However we suggested that despite these advances the application of FCMs to complex and dynamic real world systems would remain limited as consequence of their visual nature. Essentially, increasing domain complexity degrades the comprehensibility of the visual FCM representation whilst management and updating become increasingly unmanageable.

To overcome this we have previously proposed a conceptual architecture for a FCA. The basic premise of the approach is the disintegration of the domain FCM into smaller manageable task specific FCMs and their further encapsulation into the shell of a commercial AI-Agent. Multiple encapsulated FCMs can then freely interact in agent space retaining the overall domain inference. This paper develops the conceptual architecture already proposed by the integration of Matlab Simulink supported FCM within the commercial Jack AI Development framework. We consider this to be a significant step in the development of realisable FCAs that in turn will ultimately allow the application of FCMs in environments previously considered too complex or expansive.

2. FCMs in The MatLab Simulink Framework

FCMs have remained largely underdeveloped in real world applications despite the obvious potential they offer. Their principal limitation is the simplistic representation of concept relationships by a single crisp variable, erroneously presupposing that all cause effect relationships are linearly independent. Many researchers have identified this deficiency citing the need to encode temporal effects, conditional relationships and accumulativity, for proper reasoning and inference to occur. [3] [4] [5] [6]. Paradoxically the proposed

Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06)
0-7695-2528-8/06 $20.00 © 2006 IEEE
IEEE COMPUTER SOCIETY

methodologies tend to be overtly mathematical and negate the simple visual ethos of the original approach. Our approach resolves this paradox by recognising the obvious commonalities between FCMs and the diagrammatic structures used within control systems [7].

By recognition of this fact, control system analysis software can logically be applied to the FCM domain. Matlab Simulink is such a tool. Indeed, the functionality and mathematical support of Simulink allows the encapsulation of complex relationships at a high level of mathematical abstraction. The FCM designer is then able to work with simple pictorial representations such as those in figure 1, in this case representing a simple FCM on a design screen representing the operation of the green energy market within the UK.

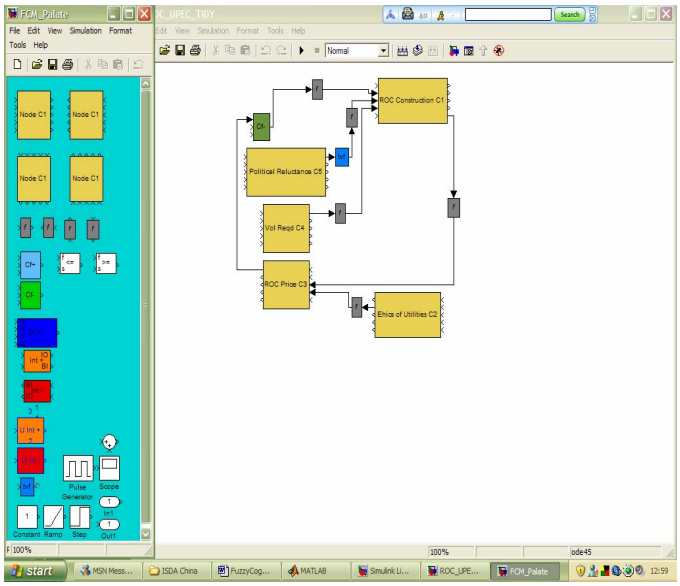

Figure 1. Green Energy Market FCM and Design Palate

3. The Fuzzy Cognitive Agent

FCMs have generally been proposed to describe behaviour within structurally invariant and open systems. Unfortunately competitive markets do not exhibit such a stable characteristic. Indeed, they are typically characterised by permanent structural flux and asynchronous, incomplete data flow.

These characteristics are difficult to accommodate within a traditional highly interconnected FCMs that rely on free interaction between concepts to allow equilibrium or a limit cycle to be reached. To address this, the disintegration of the domain FCM into smaller task specific FCMs within a loosely connected hierarchal structure has been proposed [8]. However within this approach the interconnections between task specific FCMs remain both 'hard wired' and multidirectional rendering it inflexible to ongoing structural and relational change in the domain space.

Our approach is that of the FCA harnesses the structural and relational flexibility that AI-Agents can offer [9] [10]. The basic premise of the FCA considers AI-Agents as comprising of two essential parts; a communication shell facilitating interaction between agents and their environment, and an inference engine that determines their actions. Conceptually the FCA is the substitution the 'IF' 'THEN' rule base that currently defines AI-Agent inference engines by a task specific FCM, effectively encapsulating the task specific FCM within the AI-Agent communication shell. Multiple FCAs each with a task specific FCM than freely interact within the agent space with the degree of flexibility required to simulate real world applications. We term this type of AI-Agent a *Fuzzy Cognitive Agent* and groups of such agents a *Fuzzy Cognitive Agency*. The elegance of the approach is evident from the generic market FCA agency presented in Figure 2, which illustrates the structural simplicity that is possible.

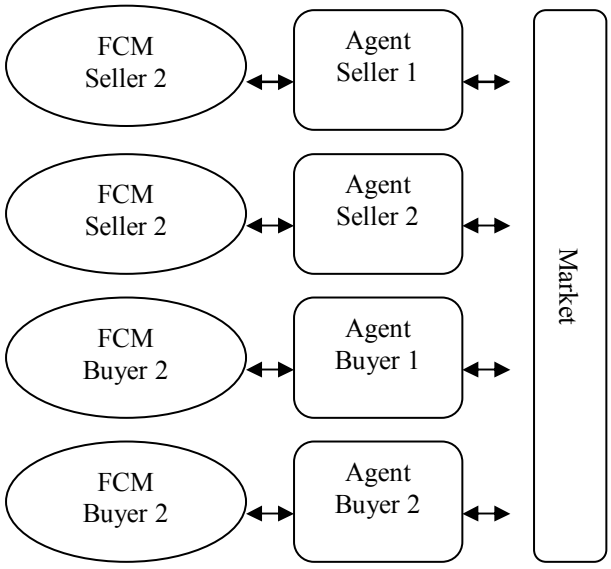

Figure 2. The FCA Agency

4. Designing Fuzzy Cognitive Agent : Choosing The Appropriate Methodology

In previous chapter, the underlying concept of fuzzy cognitive agent, which basically an agent which exploit capabilities of its internal and external fuzzy cognitive maps to solve an inherently complex system, has been presented. However, the methodology to practically build a software system based on fuzzy cognitive agent needs to be addressed and investigated. Details steps are needed to specify, design and build FCA-oriented systems.

IEEE COMPUTER SOCIETY

A practical and usable methodology in addition to high-levels steps such as 'specify the system' or even 'identify the system's goals' is needed to provide detailed guidelines explaining how these steps are carried out. As it is sometime difficult to give hard-rules in a general-purpose methodology, these guidelines could be expressed as a collection of heuristic and examples. As the process is followed, design artefacts are produced which often specified in some formal notation, such object-oriented notation UML (Unified Modelling Language) [11].

There are many existing methodologies for designing software. In particular, object-oriented analysis and design have extensively studied and developed. Unfortunately, it is not quite appropriate to use object-oriented technique to build FCA system. Although agents and objects share similarities, the differences are significant. It is possible to use object-oriented analysis and design techniques to design agent systems. However, the fit is not natural and the resulting design is less likely to make good use of agents.

For instance, one important aspect of agents is that they are *proactive*, that they pursue their own agenda over time, as realised in terms of *goals*, which is not generally a part of object-oriented methodologies.

A large number of agent-oriented methodologies have been proposed in recent years [12, 13, 14, 15, 16]. From these proposals, some are described in details, offer tool support and do appear to be ready for use. In particular, the MaSE [17] and TROPOS [13] methodologies are both complete, have been developed over a period of time, and both provide detailed descriptions.

The GAIA methodology [14] has been developed over a number of years. For a generality purpose, however, this method does not provide a detailed agent design process. For some application which is tightly connected to a certain and specific problem design, this method does not offer sufficient support. The TROPOS methodology [13] covers early requirements to detailed design. Its detailed design is oriented very specifically towards JACK as implementation platform.

The MaSE methodology [17] is one of the few methodologies that appears to have significant tool support. However, MaSE limits itself with its view of agents as merely a convenient abstraction, which may or may not possess intelligence. Therefore MaSE intentionally does not support the construction of plan-based agents that are able to provide a flexible mix of reactive and proactive behaviour. Rather, MaSE aims to be general and treats agents as a simple software processes that interact with each other to meet an overall system goal.

In this paper, the agent design methodology called Promotheus [18] is used rather than other methods based on consideration above. The Promotheus methodology defines a detailed process for specifying, designing, implementing and testing/debugging agent-oriented software systems. In addition to detailed processes, it defines a range of artefacts that are produced along the way. Some of these artefacts are kept, and some are only used as stepping stones, in form of graphical or structured texts (form).

The Promotheus methodology consists of three phases :

1. The *system specification phase*, focuses on the following :
    a. Identifying the **system goals**.
    b. Developing use case **scenarios** illustrating the system's operation.
    c. Identifying the basic **functionalities** of the system.
    d. Specifying the interface between the system and its environment in terms of **actions** and **percepts.**
2. The *architectural design phase*, focuses on :
    e. Deciding what **agent types** will be implemented and developing the **agent descriptors**.
    f. Capturing the system's overall (static) structure using the **system overview diagram**.
    g. Describing the dynamic behaviour of the system using **interaction diagrams** and **interaction protocols.**
3. The *detailed design*, focuses on :
    h. The refinement of agents in term of capabilities, giving the **agent overview diagram** and **capability descriptors**.
    i. The development of **process specifications**.

Some other areas where Promotheus differs significantly from object-oriented methodologies include :

- The provision of a process for determining the types of agents in the system.
- Treating messages as components in their own right, not just as labels. This allows a message (or an event) to be handled by multiple plans, which is crucial to achieving flexibility and robustness.
- Distinguishing percepts and actions from messages, and looking at explicitly at percept processing. Agents are situated in an environment, and it is important to define the interface between agents and their environment.
- Distinguishing passive components (data, beliefs) from active components (agents, capabilities, plans). With object-oriented modelling, everything is modelled as (passive) objects.
- One view of agents (the intentional stance) ascribes *mental attitudes*, such as beliefs, and desires to agents. Existing non-agent methodologies do not ascribe mental attitudes to software components. Also some agent-oriented methodologies (such as MaSE) do not subscribe to this view, and consequently, do not address mental attitudes. Others, including

IEEE COMPUTER SOCIETY

Promotheus, do capture mental attitudes during the analysis and design processes.

Although there are clear differences between Promotheus and object-oriented methodologies, there are also commonalities. As current object-oriented methodologies are not sufficient for engineering agent-oriented software, they are all relevant. Agents are software, and many aspects of Promotheus methodology have been based on object-oriented methods and notations. For example, use case scenarios are adapted from standard practice (Jacobson et al 1992), interaction diagrams are UML sequence diagrams, AUML is used directly, and the Rational Unified Process (Krutchen 1998) and Promotheus share a similar approach to applying an iterative process over clearly delineated phase.

One aspect important to consider is what parts of a system should be treated as agents and designed using an agent-oriented methodology (such as Promotheus), and also how the link between an agent-oriented sub-system and non-agent software can be designed and implemented.

Not all software components are best viewed, modelled and designed as agents. Sometimes, a certain system is best to model entirely as a multi agent system. However, this is not always a case, as some sub-systems may not benefit from being viewed as a collection of agents.

How parts of a system can be viewed as agents (and which parts should not?) is agents should only be used where they are more natural and offer a benefit. The following questions can be used to help identify components that should be treated as agents (if the answers are yes) :

a. Is it autonomous ?
b. Does it have goals ?
c. Viewed as an object, is it active (in the sense of having internal threads that run concurrently with the rest of the system) ?
d. Does it do multiple things at once ? If so, does it need to reason about interaction between the different activities ?
e. Does it need to change the way it is doing things on the basis of changes in its environment ?

If the answers to these questions are mostly yes, then the components can be think as agents and needs to be designed accordingly using appropriate tool such as Promotheus.

5. Implementation of Fuzzy Cognitive Agent

As discussed earlier, a fuzzy cognitive agent (FCA) basically is a software agent which has a knowledge model and inference capabilities based mainly on the FCM theory and its extension.

To implement a fuzzy cognitive agent system into a real working software prototype, a tool is needed to help to do such process. One of the most leading commercial software available is JACK version 5 Intelligent Agent Development Toolkit, produced by Agent Oriented Software Pty Ltd (AOS), Melbourne, Australia. JACK can be seen as extended naturally from object-oriented programming paradigm, it is support fully an agent-oriented programming paradigm, with an ease of object oriented methods which has been around for decades and is a mature and stable technology in software engineering.

The mainstream thought of agent oriented programming is a concept of Belief-Desire-Intention (BDI) architecture, which is fully complied and supported by JACK 5, make it possible to implement Fuzzy Cognitive Agent design using this toolkit. JACK 5 is entirely developed using Java programming language, make it a complete Java-enabled framework for fuzzy cognitive agent development. According to AOS, this framework supplies a high performance, light-weight implementation of the-Belief-Desire-Intention (BDI) architecture, and can be easily extended to support different agent models or specific application requirements.

As the final product of JACK 5 software package are collection of Java classes, the other non-agent software will simply see these classes as ordinary Java objects, enables them to be used as software components as part of a larger environment. JACK agents are not bound to any specific agent communications language, and therefore any high level protocols such as KQML [3] may be used. However, JACK has been geared towards industrial object-oriented middleware (such as CORBA) and message passing infrastructures (e.g Parallel Virtual Machines in simulated environments). Furthermore JACK also provides a native lightweight communication infrastructure.

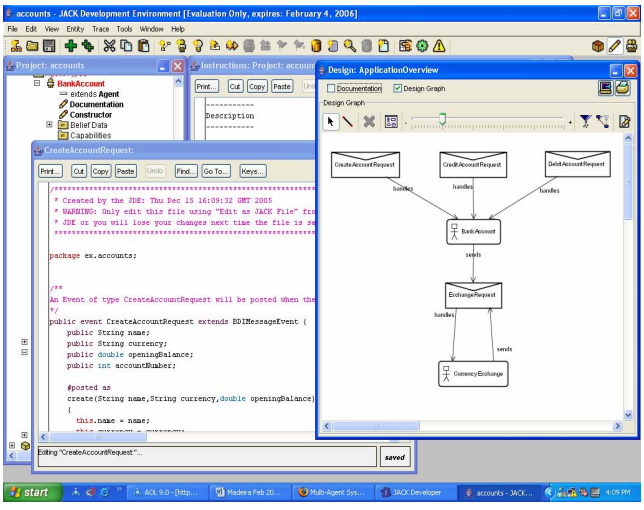

Figure 3. Screenshot of JACK Integrated Development Environment (IDE)

IEEE COMPUTER SOCIETY

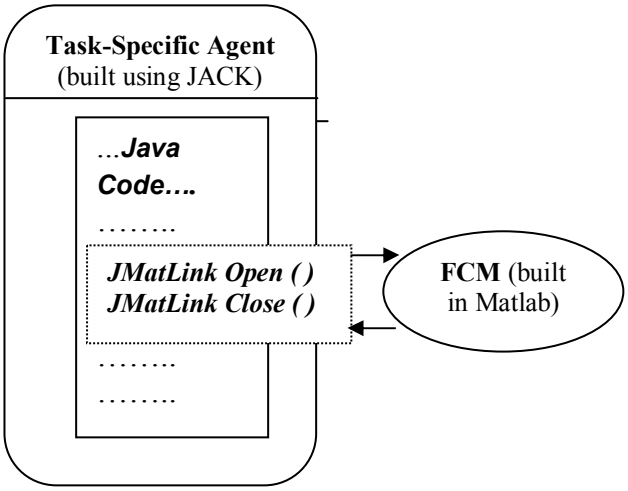

Figure 4. Generic Structure of FCM Calling Out Procedure Using JMatLink

As AOS puts it, JACK's relationship to Java is analogous to the relationship between the C++ and C languages. C was developed as a procedural language and subsequently C++ was developed to provide programmers with object-oriented extension to the existing language. Similarly, JACK has been developed to provide agent-oriented extensions to the Java programming language. JACK source code is first compiled into regular Java code before being executed.

Further as AOS describes, from an engineering perspective, JACK consists of architecture-independent facilities, plus a set of plug-in components that address requirements of specific agent architectures. An example of such a plug-in is the default BDI reasoning model supplied with JACK. From a programming perspective, JACK Intelligent Agent is an agent-oriented development environment built on top of and integrated with the Java programming language. It includes all components of the Java development environment as well as offering specific extensions to implement agent behaviour.

As discussed in the earlier chapter, FCM is implemented using Simulink from MatLab, which acts as a specialised adaptive sub-routine to perform the FCM algorithm. Figure 9 and 11 depicts this relation. The intelligent agent, on the other hand, is designed and developed using JACK with the end product of a collection of Java Code. Therefore a way to link this Java code to Simulink/MatLab needs to be discovered. There are numerous methods available to link Java to MatLab, one of the latest is using JMatLink statements.

JMatLink is an open source plug-ins to current Java technology, enables developer to link Java smoothly with MatLab. JMatLink provides an effective way to pass any variables from Java to MatLab, initiate a MatLab instance, and get the result back to Java. Using this way an effective coding can be achieved without losing focus from agent-oriented programming.

Figures 11 and 12 describes this basic architecture of relationship between those set of technologies : Java (JACK 5), JMatLink and Simulink/MatLab.

The Java codes in the main program basically triggered the MatLab engine to be fired, sending a test command and get the random number generated by MatLab and pass it back to Java code. Using similar principles Simulink can be activated trough Java by accessing the m script file which automate Simulink process.

This m script file will automate the execution of any FCM module which has been drawn using Simulink. By using this decoupling and modular approach then it is easier to update FCM module without having involved to scrutinise and modify the main codes line.

The output of FCM module then passed back to main Java code which contains the agent decision procedure to provide meaningful decision to the user. As the main role of agent to communicate and negotiate the most optimum interest then this technique will provide a modular approach to implement a fuzzy cognitive agent system.

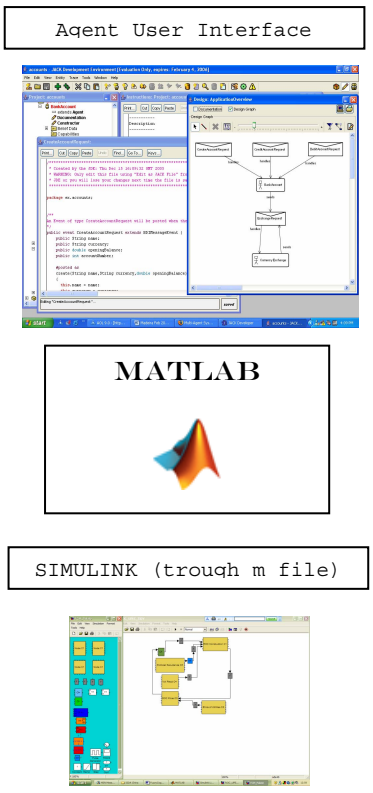

Figure 5. Implementation of FCA

IEEE COMPUTER SOCIETY

6. Conclusion

The simulation of complex environments remains one of the most challenging issues within the field of artificial intelligence today. FCMs have previously been proposed as a alternative methodology but their application has remained limited due to a combination of their representational simplicity and their inability to manage the dynamics and scale of real world environments. In this paper we have proposed the FCA whose basis is the encapsulation task specific FCMs within the communication shell of an AI-Agent. Such an approach to have value must be realisable within the currently available commercially available software platforms. We have demonstrated that FCMs can be generated on Matlab Simulink platform, and indeed such an implementation can capture complex relationships at high levels of abstraction greatly improving inference performance. Further we have shown that the encapsulation of these FCMs is achievable within the JACK version 5 Intelligent Agent Development Toolkit. We therefore consider FCAs to be a realisable technology and in further papers we will present a fully developed FCA and investigate the interaction and performance of such agents within an agency.

References

[1] Jennings N.R. Sycara K. Wooldridge M. A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems Journal, Volume 1, Issue 1, 7-38, 1998.

[2] Borrie D. Őzveren C.S. Behavioural Simulation using Augmented Fuzzy Cognitive Maps For Decision and Policy Making in Competitive Electricity Markets, 40th International Universities Power Engineering Conference, Cork, Ireland, 603-607, 2005.

[3] Koulouriotis D.E. Diakoulakis I.E Emeris D.M. Anamorphosis of Fuzzy Cognitive Maps for Operation in Ambiguous and Multi-Stimulus Real World Environments, Proceedings of the 10th Annual IEEE International Fuzzy Systems Conference, Melbourne, Australia, 1156-1159, 2001.

[4] Miao Y. Liu Z.Q. Siew C.K. Miao C.Y. Dynamical Cognitive Network – An Extension of Fuzzy Cognitive Map. IEEE Transactions on Fuzzy Systems, Volume 9, No. 5, 760-770, 2001.

[5] Haigwara M. Extended fuzzy cognitive Maps. The proceedings of the IEEE International Conference on Fuzzy Systems FUZZ-IEEE, San Diego, USA, 795-801, 1992.

[6] Zhuge H. Luo X. Knowledge Map Model, Grid and Cooperative Computing GCC-2004, Third International Conference Proceedings Lecture Notes in Computer Science, Wuhan, China, 381-388, 2004.

[7] Styblinski M.A. Meyer B.D. Signal Flow Graphs vs. Cognitive Maps in Application to Qualitative Circuit Analysis. International Journal of Man Machine Studies, Vol. 35, 175-186, 1991.

[8] Lee K.C. Lee W.J. Kwon O. Han J.H. Yu P.I. Strategic Planning Simulation based on Fuzzy Cognitive Map Knowledge and Differential Game, Simulation Vol. 71-5, 316-327, 1998.

[9] Kurbel K.Loutchko I. Towards multi-agent electronic marketplaces: what is there and what is missing? Knowledge-Engineering-Review 18(1), 33-46, 2003

[10] Klaue S. Kurbel K. Loutchko I. Automated Negotiation on Agent Based E-Marketplaces: An Overview. 14th Bled electronic Commerce Conference, Bled, Slovenia, 2001.

[11] Booch G, Rumbaugh J and Jacobson I, *The Unified Modelling Language User Guide*, Object Technologies Series. Addison-Wesley, 1999.

[12] Brazier FMT, et al. DESIRE : Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information System* **6**(1), pp 67-94, 1997.

[13] Bresciani P, et al. TROPOS : An agent-oriented software development methodology. Technical Report DIT-02-0015, University of Trento, Department of Information and Communication Technology, Trento, Italy, 2002.

[14] Wooldridge M, et al. The GAIA methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems* **3**(3), pp 285-312, 2000.

[15] Dam KH and Winikoff M. Comparing agent-oriented methodologies. In *Proceeding of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems* (ed. Giorgini P, Henderson-Sellers B and Winikoff M), pp 52-59, Melbourne, Australia.

[16] Sherory O and Sturm A. Evaluation of modelling techniques for agent-based systems. *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, Canada, pp 624-631, ACM Press, 2001.

[17] DeLoach SA, Wood MF, and Sparkman CH. Multiagent System Engineering [MaSE]. *International Journal of Software Engineering and Knowledge Engineering*. **11**(3), pp 231-258, 2001.

[18] Winikoff M, Padgham L and Harland J. Simplying the development of Intelligent Agents. In *AI 2001:Advances in Artificial Intelligence, 14th Australian Joint Conference on Artificial Intelligence*, pp 557-568 Adelaide, December 2001.