Dynamic Decision Support Systems in a Multi-Agent Environment

Tofig Hasanov[1], Ozeki Motoyuki[2], Oka Natsuki[3]
Kyoto Institute of Technology, Kyoto, Japan
[1]*Gtofig@yahoo.com*, [2]*ozeki@kit.ac.jp*, [3]*nat@kit.ac.jp*

***Abstract*— Multi-agent systems are widely used in modeling and control of modern industrial systems in such areas as transportation, supply chain management, simulations, and fault detection. These systems are often used for tasks where the environment is complex and constantly changing and thus requires a high degree of flexibility. In this paper, we propose a method to increase the flexibility and security of multi-agent systems by using dynamic decision support systems for intelligent agents. We demonstrate that this method significantly improves the flexibility and security of multi-agent systems by simplifying run-time modifications to the system.**

Keywords*— multi-agent systems, decision-support systems, logic programming, artificial intelligence, intelligent agents

I. INTRODUCTION

As business environment gets more and more complex (not to be confused with complicated) usual solutions to decision making become inapplicable: as the time between changes in an environment (disruptive events) becomes smaller, the usual decision systems cannot process huge amount of information fast enough to generate response in time. One possible way to solve such complex tasks is to use complex multi-agent systems. However, modern multi-agent systems have several limitations, which limit their application in industry: Agents depend on hardware platform's processing power. This prevents agents from being placed on mobile platforms with low processing power. Another issue to consider is that in order to cooperate, agents have to exchange knowledge directly, which might result in nonmonotonic data conflicts, especially in a rapidly changing business environment.

As a solution to these problems, we discuss the possible application of Decision Support System (DSS) to the multi-agent environment to support individual agents' decision making. We will show that using logic-based knowledge-driven DSS can be particularly helpful in the context of multi-agent systems (MAS), where the environment and agents change rapidly. A DSS is usually a predesigned system with a well-defined set of rules, which remain intact throughout its usage. There is ongoing research about self-evolving DSS [1], but the focus of the research mainly concerns the representation of output to the user rather than the decision-making process itself. The drawback of having a predefined set of rules is the inability to accommodate run-time changes to the environment such as the addition of new classes of agents and objects or the adoption of new policies. It is unclear how to design a DSS without prior knowledge about possible future changes to the system. Finally, there should be a mechanism for handling nonmonotonic data, which often arise in multi-agent systems. We can summarize the limitations of a typical DSS in MAS as follows:

- Inability to accommodate run-time changes
- Inability to handle nonmonotonic data

We will address these issues by introducing a dynamically created DSS (DCDSS): a mechanism for creating and modifying the support system at run time. It will be shown that the proposed approach simplifies the construction of flexible, open MAS, where agents can be added or removed during run time.

II. WHY USE DYNAMICALLY CREATED DSS IN MAS?

The motivation behind this research is to increase the flexibility of MAS and to decrease specification requirements for agents, allowing more open architecture and less restricted agent behavior. We plan to achieve this by inserting DSS as an intermediate layer between an agent and the system in the decision-making process.

First, let us discuss the usefulness of DSS in MAS. The question that immediately arises is why intelligent agents would need DSS to support their decision-making. The answer is, for the same reasons humans need it: DSS allows agents to separate computationally expensive calculations from their main body, which is essential in cases the MAS environment's resources are limited (e.g., mobile platforms). This way, the agent can be placed on a platform with low processing power and bandwidth. The DSS engine in such a case could be placed on a separate platform. Another reason is that agents might be written in any programming language as long as communication protocol requirements are met. Separated DSS would still allow for logic-based reasoning, integrating the power of logic programming with other languages. In addition, this approach would be useful in the case of several separate decision advice sources (e.g., rational reasoning, emotional reasoning). In this case, DSS would serve as one source of decision making. In this setup, intelligent agents act as users of their personal (or common) DSS. Finally, using well defined logic program updates [7], we can combine knowledge of several agents with default environment rules and produce "cooperative" decision advice, without disclosing other agents' knowledge if necessary. This is very important in open MAS, where each agent might be designed and programmed by a different person or company, and each might run on a different physical platform. In this case, DSS would serve as a protective layer, separating sensitive environment data from agents.

Another important issue is whether DSS will be able to adapt to changes in environment and produce valuable advice for agents. In a fast-changing, unpredictable environment, we expect agents to adapt and make reasonable decisions. The usual notion of DSS suggests that the DSS engine is designed and implemented in advance. Such a design would include inference rules, updates, and maintenance procedures. This approach might be sufficient for many applications for which a set of rules does not change with time. However, in a dynamic multi-agent environment, where not only the agents but also the environment itself changes over time and evolves according to user needs, constructing a DSS that will be able to accommodate all future changes is a complicated task. Moreover, as time passes, new classes of agents might be added to the system at run time. A good example of such a multi-agent system would be a smart-house system, where various devices in the house act as independent agents, with possibly independent goals. As time passes, the user might connect additional agents to the system or remove old ones; or, new users might come to the house. In such cases, a dynamically created DSS might be useful because it would provide a mechanism for old agents to evolve their decision-making process to account for new changes. The main idea is that DSS is being created and updated at run time rather than beforehand. In the subsequent sections, we will discuss the implementation of such a system.

III. MAS ARCHITECTURE

For the class of applications where our approach might be useful, we consider such a central system to exist in the form of a central server. Below, we present the abstract diagram of a multi-agent system with DSS. This diagram depicts MAS components and their interconnections.

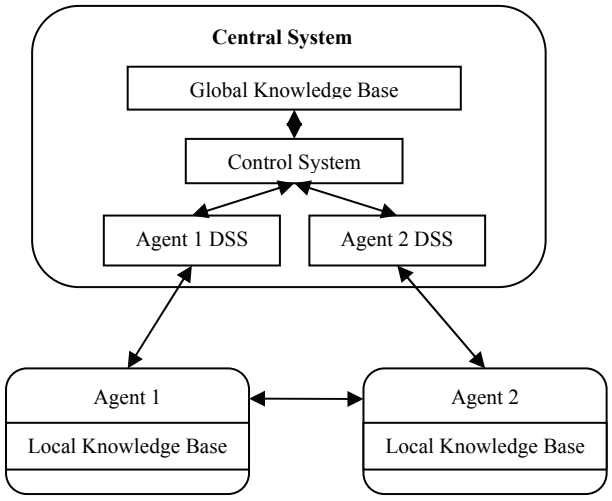

Figure 1. Sample diagram of MAS with two agents.

As seen in Fig. 1, although agents can be located on any remote platform, their DSS are stored inside the main server. An agent can connect to his DSS at any time and can request decision advice or update his DSS by providing new knowledge. During cooperation, an agent might give access to his DSS to other agents, allowing them to update its DSS by providing additional facts or rules. Note that in this way, an agent can receive help without having direct access to other agents' knowledge. This might be useful in cases where agents need to cooperate without disclosing important information to each other, as in the case of various companies working on the same project.

IV. CREATING AND UPDATING DSS

Whenever a new agent is added to the system, the control system creates a separate DSS for him with certain predefined basic rules, such as the physical rules of the environment, security measures, and user preferences. The agent then can send his own knowledge and rules to the DSS at any time. The DSS will automatically update with the new knowledge, resolving any logical conflicts. Whenever the agent needs to perform some logical reasoning, he can send a query to his DSS along with any additional facts and rules. In the cooperation scenario, other agents can participate in this process by providing partial access to their local knowledge base, as seen in Fig. 2. Depending on the system, it is also possible for the cooperating agent to send his knowledge directly to another agent's DSS without disclosing it to that agent. The DSS then produces decision advice. This is done by taking the agent's knowledge base as a set of facts and rules, updating them with environmental facts and rules (chosen by the Control System for that particular agent and goal) using logic update [7], and then compiling the resultant logic code in an external inference engine, such as Prolog. The result of the execution is then sent to the agent. Obviously, the knowledge base should be formatted according to the requirements of the selected programming language.

Below is an abstract diagram of the described decision making process (Fig. 2).

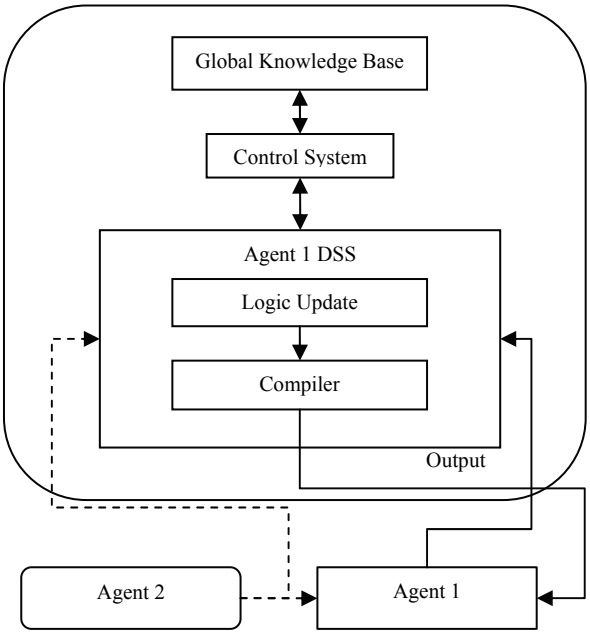

Figure 2. Sample diagram of decision making process.

This approach ensures that the DSS is updated each time an agent uses it. For example, if, at some point, a new rule is added to the environment by a user, it will be stored in the global knowledge base, and then sent to the DSS the next time the agent uses it. The resultant code will be a result of the combination of global knowledge and the agent's local knowledge, combined by the logic update engine, which will be discussed in the next section.

V. Cooperation and Conflicts

In dynamic multi-agent systems where agents act independently and have private knowledge and goals, it is usually inevitable that their knowledge will be nonmonotonic in nature. Such nonmonotonicity can result from environment changes not yet observed by some agents, or from differences in two agents' knowledge. It is thus crucial that the system be able to handle the differences between two knowledge bases and resolve them before compilation. There are several possible approaches to solving nonmonotonicity. One of them is to use probabilistic logic [10]. In our research, we use a slightly modified version of an approach called "Dynamic Logic Programming" [7]. In this approach, we perform an update of a logic program P1 by another program P2, denoted by P1⊕P2, which results in a new program P3, where all nonmonotonic data is resolved. Details of this method are beyond the scope of this paper. For details, see [7]. In our experiments, each source of knowledge is given a priority value, with the general knowledge base having maximum priority. The agent can rank other sources based on his experience. Thus, the agent can trust some other agent more or less by increasing or reducing his knowledge priority, respectively. These priority values are then used in the logic update engine to resolve conflicting data coming from different sources.

At any time, two agents might decide to exchange experience by providing a partial or full knowledge base to the other agent or directly to the other agent's DSS. Agents then use this update mechanism to generate a new version of the knowledge base, resolving nonmonotonic data. A similar process occurs when an agent learns a new fact or a rule about the environment that contradicts his previous knowledge.

VI. Conclusions and Future Work

In this paper, we demonstrated a concept of dynamically created self-evolving DSS for intelligent agents in MAS. We showed that DSS can dynamically change itself by combining knowledge from various sources and eliminating conflicts. This method might be useful in cases where new types of rules are learned or created dynamically by the user and agents, allowing for highly adaptable and flexible MAS. This also allows for more independent behavior in multi-agent systems, as agents do not use the same decision system and can have very few common properties, allowing for interaction of multiple different agent types without disclosing an agent's structure to the system or other agents. This is achieved by restricting the communication protocol rather than the structure of the agent himself. This can be particularly useful in situations where agents are designed by different companies that do not want to disclose the technical characteristics of their agents to others (e.g., different companies working on the same project, but unwilling to share patented technologies). It can also find uses in situations where it is not possible to predict the number and types of agents. For example, in a smart house, a system governing multiple agents (TV, computers, refrigerators, robots, etc.) should be designed to be able to accommodate any new agent added by the user into the system in the future.

Due to limitations in the paper size, we could not include our experimental results in this paper. The described system is currently in the initial state of development, with many features still waiting to be implemented and tested. Further research is needed to optimize handling of non-monotonic data and resolve conflicts between agents. In addition, the execution efficiency of the system must yet be tested on larger projects to determine if it is practically useful. The proposed method seems to have good potential considering the above-discussed advantages.

References

[1] T.P. Liang and C. Jones, "Design of a Self-evolving Decision Support System", Journal of Management Information Systems, vol. 4(1), 1987, pp.59-82.

[2] F. Schober and J. Gebauer, "How Much to Spend on Flexibility? Determining the Value of Information System Flexibility", In Proceedings of the 15th Americas Conference on Information Systems, San Francisco, 2009.

[3] R.H. Bonczek, C.W. Holsapple and A.B. Whinston, "The Evolving Roles Of Models In Decision Support Systems", Decision Sciences, vol. 11, 1980, pp.337–356.

[4] B.G. Espinasse, G. Picolet and E. Chouraqui, "Negotiation Support Systems: A Multi-Criteria and Multi-Agent Approach", European Journal of Operational Research, 103, 1997, pp.389-409.

[5] J. Tweedale, N. Ichalkaranje, C. Sioutis, B. Jarvis, A. Consoli and G. Phillips-Wren, "Innovations in Multi-Agent Systems", Journal of Network and Computer Applications, 30(3), 2007, pp.1089–1115.

[6] B. van Linder, "Modal Logics for Rational Agents", PhD thesis, Department of Computing Science, University of Utrecht , 1996.

[7] J.J. Alferes, J.A. Leite, L.M. Pereira, H. Przymusinska and T.C. Przymusinski, "Dynamic Logic Programming", in Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98), eds. A. Cohn, L. Schubert and S. Shapiro, Morgan Kaufmann Publishers, 1998, pp.98–111.

[8] J.A. Leite and L. Soares, "Enhancing a Multi-Agent System with Evolving Logic Programs", In Pre-Procs. of the 7th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA VII), eds. K. Inoue, K. Satoh and F. Toni, 2006, pp.207–22.

[9] G. Brewka, I. Niemelä and M. Truszczynski, "Nonmonotonic Reasoning", in Handbook of Knowledge Representation, eds. V. Lifschitz, B. Porter and F. van Harmelen, Elsevier, 2007, pp.239-284.

[10] M. Gelfond and N.J. Rushton, "Nonmonotonicity of Probabilistic Reasoning", Computer Science Department, Texas Tech University, 2010.

[11] J. Chomicki, J. Lobo and S. Naqvi, "Conflict Resolution Using Logic Programming", IEEE Trans. Knowl. Data Eng, vol. 15(1), 2003, pp.244–249.

[12] U. Nilsson and J. Maluszynski, "Logic Programming and Prolog", 2nd ed., John Wiley, 1995.

[13] W. Vasconcelos, "A Prolog Simulation Platform for Logic-Based Multi-Agent Systems", School of Artificial Intelligence, Division of Informatics, The University of Edinburgh, 2001

[14] P. Dell'Acqua, U. Nilsson and L.M. Pereira, "A Logic-Based Asynchronous Multi-Agent System", Computational Logic in Multi-Agent Systems (CLIMA02), Electronic Notes in Theoretical Computer Science (ENTCS), vol. 70(5), 2002, pp.72-88.