ATLANTIS PRESS

Advances in Intelligent Systems Research, volume 134

2nd International Conference on Control, Automation, and Artificial Intelligence (CAAI 2017)

Dynamic Generation of Commitment Protocol Based on Capability Matching in Open Systems

Jing Wang, Wei Liu*, Shuang Li and Kun Wu

Wuhan Institute of Technology, Wuhan, China

*Corresponding author

***Abstract*—Agent collaboration is a fundamental part of multi-agent systems when an agent cannot accomplish a task by itself. Such collaborations are usually regulated by commitment protocols, which are typically defined at design-time. However, in many situations a protocol may not exist or be predefined at design-time which may not fit the needs of the agents when environment changes. In order to deal with such situations, agents should be able to generate protocols at run-time. In this paper, we combine commitment with capability and apply capability matching algorithm to the dynamic generation of commitment protocols. In the first phase, capability matching algorithm is used to generate a graph of capability-to-goal, which denotes the capabilities needed to achieve a goal. The second phase, commitment protocols are generated through the capability-to-goal graph. The application of our approach will be demonstrated using a case study.**

Keywords-multi-agent system; collaboration; commitment protocol; capability; goal; capability matching

I. INTRODUCTION

Agent collaboration is a fundamental part of multi-agent systems when an agent cannot accomplish a task on its own. Such collaborations are usually regulated by commitment protocols.

Typically, protocols are defined at design-time and embedded into agents' implementation[1, 2, 3]. However, there are a number of reasons why defining protocols at design-time is, sometimes, too limited. For instance, in open systems, where agents can enter the system at run-time, it is necessary to construct new run-time protocols. In other words, predefined protocols at design-time it not sufficient for three major reasons: a) Change in agents; b) Change in preconditions; c) Change in goals.

For these reasons we suppose that, in open systems or some closed(but highly dynamic) systems, there is a need for agents to be able to derive new protocols at run-time, rather than relying on design-time protocols that are embedded into the agents' implementation.

There are three main lines of work that are closely related to this problem. One line of work studies the evolution of protocols according to changing requirement. That is, designers modify existing protocols in a systematic way so that the agents can receive the new protocol and continue working[4]. That approach assumes that there is already an existing protocol and a dedicated designer who modifies the protocol. In our case, we do not assume that there are any existing protocols available to the agents.However, we assume that the agents need to generate the protocol at run-time. The second line of work does not assume an existing protocol but a centralized planning agent that knows all preferences, goals and services of all the agents, that is, HTN planning[5,6]. As a result, a central agent can generate the right protocol to satisfy all agent. But in our case, we do not assume that a central agent would know all the details of all the agents. Actually, capabilities and goals would be enough. The third line of work assumes that agents can generate protocols on their own, using their local knowledge, that is, Goal Based algorithm [7]. While in our case, we do not assume that each agent has enough local knowledge to accomplish the collaboration process. In this paper, we propose a method that will dynamically generate protocols at run-time after the perception of capabilities and goals.

The rest of this paper is organized as follows. Section 2 describes the background of commitment and capability. Section 3 introduces our algorithms for generating commitment protocols based on agents' goals and capabilities using capability matching algorithm. Section 4 demonstrates our approach in a case study on the plugin which we have developed on Protégé(an ontology editor and framework for building intelligent systems). Section 5 summaries our work and future directions.

II. BACKGROUND: COMMITMENT AND CAPABILITY

In this section, we have discriminated the conceptions of ability and capability,commitment and capability commitment, which may be easily confused. Other relative definitions about agent and commitment can be found in[7].

Definition 1 (*Capability*) *Capability* describes how a process is to be executed within what current contexts are and what the contexts will be. *Capability* has the form of *Cap*(*A*, *InCts*, *P*, *OutCts*), in which *A*(Agent) is specified as who has the capability; *InCts*(InConstraints) describes what current contexts should be; *P*(Plan) describes how to execute the actions of agent; *OutCts*(OutConstraints) describes what the contexts will be.

Definition 2 (*Capability Commitment*) *Capability commitment* is similar to commitment but it uses capabilities to accomplish goals rather than abilities. *Capability Commitment* has the same form of C(*Debtor*, *Creditor*, *Antecedent*, *Consequent*).

Definition 3 (*Commitment Protocol*) A *commitment protocol p* is a set of commitments.

Discrimination 1 (*Ability* and *Capability*) *Ability* A_x (*d*, *r*) denotes the *ability* of agent *x* to bring about the proposition *r*, if the precondition *d* holds , which are eventually executed by service and incentive(*beliefs*), while c*apability Cap*(*A, InCts, P, OutCts*) denotes that agent *A* has the capability in the context of *InCts* to achieve *OutCts* through P(*plan*).

Discrimination 2 (*Commitment* and *Capability Commitment) Commitment* and c*apability* commitment are similar in the form *C*(*Debtor,Creditor, Antecedent, Consequent*) but there exists some differences. Comparing with previous representation of agent commitment, antecedent and consequent in capability commitment are not propositional variables but context descriptions.The contractual relationship between debtor and creditor is in accordance with the collaboration of their capabilities.

III. Generation of Commitment Protocol

In this section we develop a two phase framework to dynamically generate capability commitment protocols. In the first phase, capability algorithm is used to match capabilities with goals, which will find out all capabilities needed to accomplish a goal. The second phase, commitment will be generated according to the capabilities needed, which, sometimes are more than one.

A. Matching Capability with Goal

We present our CapabilityMatching algorithm in Table I. The algorithm matches capabilities with goals to generate the commitments. Basically, the goal of our approach is to achieve outConstraints in the current contexts of inConstraints based on all capabilities that exist in the system. First, the central agent starts to search capabilities which satisfy the inConstraints given above. Then, outConstraints of each selected capability will be found out and added to the current situation, with some of outConstraints may matched the final outConstraints. Next, based on the outConstraints newly added and inConstraints given before, central agent continues to search capabilities that satisfy the current situation for several loops until all outConstraints have been matched.

TABLE I APABILITY MATCHING ALGORITHM

Algorithm 1 CapabilityMatching(InCts, *G*)
Input: InCts(InContraints), which denotes the current contexts
Input:G<OutCts>, a set of OutConstraints, which denotes the terminal contexts expected to achieve
Require:Collection<Cap>, a set of capabilities in the system
Require:CList<Cap>, a set of capabilities needed to finish a goal
Require: InCtsList<InCts>, a set of InContraints, which denotes the dynamic current contexts

1: **while** G<OutCts> $\not\subset$ InCtsList **then**
2: **for all** Cap_i **in** Collection<Cap> **do**
3: **if** Cap_i. getInCts(inCts) $\subset$ InCts and $Cap_i \notin$ CList **then**
4: CList. add(Cap_i)
5: **end if**
6: **end for**
7: **for all** Cap_j **in** CList<Cap> **do**
8: InCtsList. add(Cap_j. getOutCts(outCts)) //the type of InCts and OutCts are same
9: **end for**
10: **if** G<OutCts> $\subset$ InCtsList **then**
11: **return** //algorithm is over
12: **end if**
13:**end while**

B. Generation of Commitments

After matching capabilities with goal, it comes to the generation of commitments. According to the form of Commitment(*Debtor, Creditor, Antecedent, Consequent*), with antecedent and consequent (inConstraints and outConstraints) given, if debtor and creditor can be determined, a commitment will be able to generated. Since the capabilities that are needed to achieve a goal have been selected, it's not complex to find out debtor and creditor, for each capability may be contained in one or more debtors or creditors.

IV. Experiment

A. Scenario

To elaborate the generation of commitments more explicitly, we will illustrate our approach in a case study. The home owner has bought a new house and wants to decorate his study room. What he needs are a bookcase and a computer desk. At the same time, he wants to ask a decorator to get the wall and floor decorated. He has paid for the bookcase, computer desk, materials as well as wages of the decorator. However, the decorator requires materials such as wooden floor and wall paint before he can finish the decoration while all the goods are on the process of transportation.

As described in Table 2, the inConstraints of this case are BkPaid, CmpPaid, MtrPaid, DcrPaid and outConstraints are BkProvided, CmpProvided, DcrFinished while the capabilities needed to fulfill the goal are C_1, C_2, C_3, C_4. The meaning of each proposition can be seen in Table 3 and the capabilities of each agent in Table 4.

TABLE II CONTEXT OF THE CAPABILITY MATCHING SCENARIO

InCts	BkPaid, CmpPaid, MtrPaid, DcrPaid
Caps	C_1(BkMer, BkPaid, Transport, BkProvided) C_2(CmpMer, CmpPaid, Transport, CmpProvided) C_3(MtrMer, MtrPaid, Transport, MtrProvided) C_4(Decorator, DcrPaid^MtrProvided, Decoration, DcrFinished)
OutCts	BkProvided, CmpProvided, DcrFinished

TABLE III MEANINGS OF THE RUNNING EXAMPLE

BkPaid	The customer has paid for the bookcase
CmpPaid	The customer has paid for the computer desk
MtrPaid	The materials needed to decorate the room have been paid
DcrPaid	The customer has paid the decoration cost to the decorator
BkProvided	The bookcase has been provided
CmpProvided	The computer desk has been provided
MtrProvided	The materials have been provided
DcrFinished	The decoration has been finished

TABLE IV AGENTS OF THE RUNNING EXAMPLE

BkMer	The bookcase merchant can provide bookcase
CmpMer	The computer desk merchant can provide computer desk
MtrMer	The material merchant can provide materials
Decorator	The decorator can decorate the room

B. Graph of Capability Matching

We generate the graph of capability matching on the plugin which we have developed on Protégé. As is depicted below, in the first loop, central agent begins to search the collection of

capabilities and find that C_1, C_2, C_3 meet the inConstraints of BkPaid, CmpPaid, MtrPaid. Then C_1, C_2, C_3 will be added to the collection which contains all capabilities that are needed to achieve the goal. In the second loop, the outConstraints of C_1, C_2, C_3, namely BkProvided, CmpProvided, MtrProvided will be added to current situation and when MtrProvided and DcrPaid are meet, C_4 will be found and added to the collection. Noting that the outConstraints of C_1, C_2, C_4 have meet the final OutConstraints BkProvided, CmpProvided and DcrFinished , the capability matching is over. The process of capability matching is generated by the plugin which we have developed on Protégé and shown in Figure. 1.

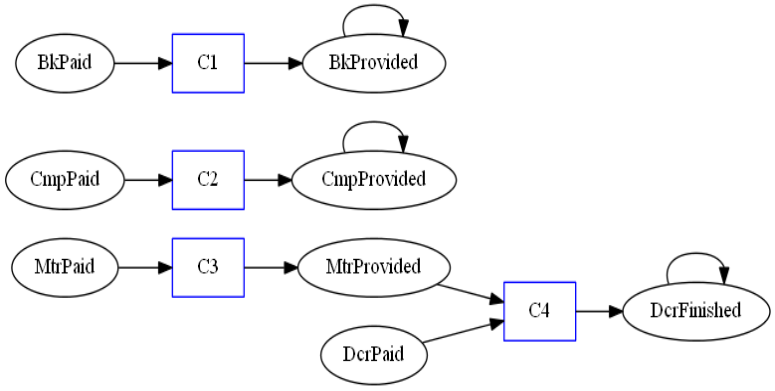

FIGURE I RUNNING EXAMPLE OF CAPABILITY MATCHING

C Generation of Commitments

Since C_1, C_2, C_3, C_4 belongs to BkMer, CmpMer, MtrMer, Decorator respectively, the commitments will be generated by the central agent. In order to compare the typical and extended commitments, they are generated in two forms:

a) Typical commitment protocol.

Com_1(BkMer, CmpMer, BkPaid^CmpPaid, BkProvided ^CmpProvided);

Com_2(CmpMer, MtrMer,CmpPaid^MtrPaid, CmpProvided ^MtrProvided);

Com_3(Decorator, MtrMer, MtrProvided^DcrPaid, DcrFinished).

b) Extended commitment protocol.

Com_1(CmpMer^MtrMer, BkMer, BkPaid^CmpPaid^MtrPaid, BkProvided^CmpProvided^MtrProvided);

Com_2(Decorator,MtrMer,MtrProvided^DcrPaid,DcrFinished).

In comparison, it can be found that extended commitments will be more concise and can reflect the combination relationship of commitments while classical commitments are less concise and cannot reflect the combination relationship only by their forms. To illustrate our approach, the case is designed rather simple. But when it comes to a more complex situation, the concision of extended commitments will be more obvious.

After generation of commitment protocols, they can be regarded as a large grain capability and stored in commitment library. When the antecedent triggers, it will be invoked to achieve a goal.

V. Conclusion and Future Work

This paper has made some contributions. Firstly, we develop the capability matching approach to generate commitment protocols dynamically at run-time. Secondly, we combine capability with commitment and illustrate the difference between ability and capability. Thirdly, in order to reflect the combination relationship and simplify the generation of commitments, we extend the definition of classical commitment and make a comparison of the classical and extended one.

There are also some problems remain to be solved in our work. On one hand, since the commitment protocols generated sometimes are more than one, which means that there may exist more than one plan to achieve the same goal. Therefore, it requires an applicable and effective mechanism to rank and select the proper protocols. On the other hand, there does not exist any repository of commitment based protocols we could use to store the generated protocols.

Two directions might be considered in our future work. One direction is to find an effective way to rank and select the most suitable protocol, considering that current ranking mechanisms are basically in the fixed benefit calculation which, sometimes, cannot reflect the dynamic change of concerns about protocol executions. Other direction is to develop a repository of commitment based protocols, so that the generated protocols can be stored in the repository and when antecedent triggers it can be invoked as a large grain capability to achieve a certain goal.

Acknowledgment

The authors would like to thank B. Nuseibeh, Y. J. Yu, A. Bennaceur in Open University. Project supported by the National Natural Science Foundation of China under Grant (No. 61502355, and No.61272115), the Doctor foundation for Science Study Program of Wuhan institute of technology (No.K201475).

References

[1] Desai, N. Mallya, A. U., Chopra, A. K. &Singh, M. P. 2005). Interaction protocols as design abstractions for business processes. IEEE Transactions on Software Engineering, 31(12), 1015–1027.

[2] Winikoff, M. (2006). Implementing flexible and robust agent interactions using distributed commitment machines. Multiagent and Grid Systems, 2(4), 365–381.

[3] Yolum, P. (2007). Design time analysis ofmultiagent protocols. Data and Knowledge Engineering, 63(1), 137–154.

[4] Gerard, S. N., &Singh, M. P. (2013). Evolving protocols and agents inmultiagent systems. In Proceedings of the twelfth international conference on autonomous agents and multiagent systems (AAMAS), (pp. 997–1004)

[5] Telang, R. P., Meneguzzi, F. , & Singh, M. P (2013). Hierarchical planning about goals and commitments. In Proceedings of the twelfth international conference on autonomous agents and multiagent systems, AAMAS, (pp. 877–884).

[6] Meneguzzi, Telang, Singh. A first-order formalization of commitments and goals for planning[C]. Proceedings of the twenty-seventh AAAI conference on artificial intelligence. 2013:255.

[7] Akin Gunay, Michael Winikoff, Pinar Yolum. (2014). Dynamically generated commitment protocols in open systems. Auton Agent Multi-Agent Syst. DOI 10. 1007.

[8] Singh, M. P. (1999): An ontology for commitments in multiagent systems. AI and Law 7, 97‑113.