IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4319

Temporal Interaction Diagrams

T. Y. Chen
School of Information Technology, Swinburne University of Technology, Australia
Tel: +6 (139) 214-5505, Fax: +6 (139) 819-0823, tychen@it.swin.edu.au

Iyad Rahwan
Department of Information Systems, University of Melbourne, Australia
Tel: +6 (138) 344-9236, Fax: +6 (139) 349-4596, i.rahwan@pgrad.unimelb.edu.au

Yun Yang
School of Information Technology, Swinburne University of Technology, Australia
Tel: +6 (139) 214-5505, Fax: +6 (139) 819-0823, yun@it.swin.edu.au

ABSTRACT

An interaction diagram is a graphical view of computation processes and communication between different entities. It can be used for the design and testing of distributed systems. In particular, interaction diagrams offer significant advantages to the design of multi-agent systems, especially when they can be expressed in a linear form, known as fragmentation, facilitating automation of design and testing of such systems. Existing interaction diagram formalisms lack the capability of describing flexible temporal order constraints. They only support rigid temporal order, and hence have limited semantic expressiveness. In this paper, we propose an improved interaction diagram formalism in which more flexible temporal constraints can be expressed.

INTRODUCTION

Agent-based computing promises to allow software developers to naturally understand, model, and develop complex distributed systems [2][8]. The following definition is adapted from [7]: "an agent is a software system (or system component) that is *situated* in an environment, which it can *perceive*, and that is capable of *autonomous* action in this environment in order to meet its design objectives." Jennings and Wooldridge [3] proposed that an *intelligent* agent is capable of social behaviour; that is, capable of communicating with other agents in the system. In this context, there is a need for formalising such interaction in multi-agent systems. Such formalisation would be useful for designing and testing multi-agent systems, as well as visualising the computation flow of each agent and communication among agents.

Various attempts have been made to formalize interaction in multi-agent systems (e.g. [1][4][6]). Usually the system design stage involves a description of the steps taken in processing a particular task. In multi-agent systems, however, the implementation requires a clear picture of the separate computational threads of different agents. In concurrent agent systems, it would be very helpful to be able to express the computation flow descriptions for different agents. Interaction diagrams are used which are easily expressed and understood by system designers. The linear representation of these diagrams facilitates the automation of the processes of diagram manipulation for design, report generation and testing. In particular, testing can be performed by comparing agent execution traces against specifications expressed in terms of interaction diagrams.

However, existing interaction diagram formalisms support quite rigid temporal order constraints only. If the execution trace does not exactly match the specified order, it is considered unacceptable. In other words, there is no way of specifying multiple acceptable traces without writing multiple versions of fixed execution traces. This can become a difficult job, especially in multi-agent systems with high interaction rates. In such systems, the number of acceptable interactions can be quite large and there is a need to more concisely express such flexibility in a single interaction diagram. Our research is a step towards achieving this goal.

In the next section, we give a brief description of interaction diagrams as proposed in [6], and we outline their drawbacks. Then, we describe how to add more flexible temporal characteristics to existing interaction diagram formalisms in section 3. We then show some examples to demonstrate its expressive power. Finally, a number of conclusions are drawn and future research is outlined.

EXISTING INTERACTION DIAGRAMS

An interaction diagram is a graphical representation of the computation threads and communications in a multi-agent system and the like. It is a graph showing the process of each agent symbolically as one or more vertical bars, and the messaging between agents as horizontal arrows between these bars [6]. A sample interaction diagram is shown in Figure 1, describing three agents A, B and C and the message sequences among them. There are a number of properties that are worth mentioning with respect to the meaning of this interaction diagram according to the formalism in [6].

A *fragmentation* is an algebraic representation of an interaction diagram. In order to convert an interaction diagram into a fragmentation, we need to decompose it into its graphical elements, which correspond to *fragments*. A number of graphical elements have been proposed in [6]. Figure 2 shows some types of those fragments. These fragments correspond to the beginning of an agent thread, the end of an agent thread, an agent sending a message, and an agent receiving a message. Underneath every fragment is its corresponding algebraic form. A combination of those algebraic atoms can be combined (forming a fragmentation) to describe a particular interaction diagram.

There are a number of different types of fragmentation, namely with respect to the order in which the fragments in the diagram are stated in the algebraic form. We are mainly concerned with *computation flow fragmentation* [6], which orders the fragments by grouping those of the same agent together. Each agent's fragments are ordered according to a top-to-bottom sequence (or temporal sequence). This represents the computation flow of an agent. The order of fragments in a fragmentation is significant as it reflects the temporal order [5]. The computation flow fragmentation for Figure 1 is as follows.

For agent A: <beg(A), snd(A,m1), rcv(A,m4), end(A)>
For agent B: <beg(B), rcv(B,m1), snd(B,m2), rcv(B,m3), snd(B,m4), end(B)>
For agent C: <beg(C), rcv(C,m2), snd(C,m3), end(C)>

According to [6], when actions concern a single agent, the order of actions has significance for the interpretation of the diagram. The meaning of a single agent computation flow is that the order in which the fragments occur reflects the order in which the corresponding actions take place. In other words, they state the **exact temporal order** in which the fragments or messages **should** take place. There is no flexibility in expressing other temporal constraints between different fragments. This is a severe drawback to the expressive power of

This paper appears in *Issues and Trends of Information Technology Management in Contemporary Organizations,* the proceedings of the Information Resources Management Association International Conference.

Figure 1: Simple interaction diagram

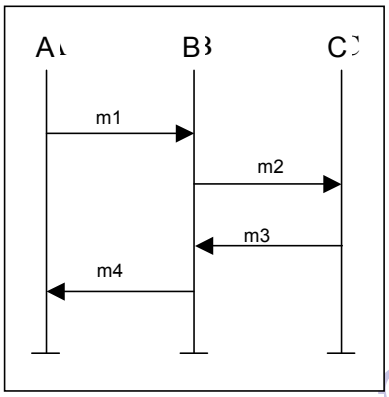
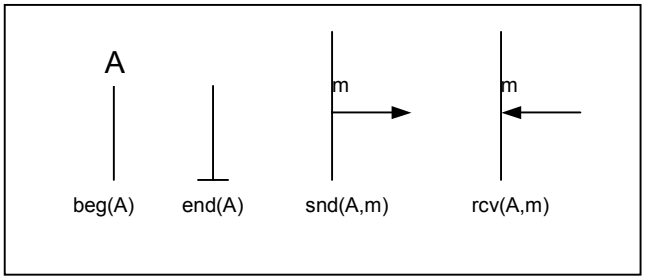


Figure 2: Graphical elements (fragments)



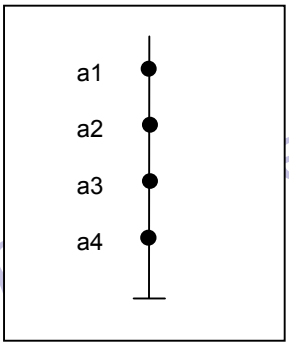
the framework. What we would like to achieve is a formal framework for supporting more flexibility in the temporal constraints.

In order to explain the drawback more clearly, we will give a simple example of one agent. Suppose we would like to express the following temporal order constraints on the events of the diagram in Figure 3, where an event is e.g. snd or rcv, etc. (for clarity purposes, we have omitted stating what these events really are because for our purpose, the other agents are not important).

We will give an example from a coffee making machine agent. Suppose an agent needs to perform the following tasks:

- First, send a message to the resources agent requesting sugar and coffee (call this event a1).
- Then, receive a message from the resources agent stating sugar is ready (call this event a2). Also receive another message from the resources agent stating coffee is ready (call this event a3). It does not matter which of these happens first, but they both need to be completed before moving to the next step.
- Now, send a message to the water agent requesting hot water to be poured in the cup (call this event a4).

Figure 3: Single agent



More abstractly, we have the following constraints:

- a1 must be before a2, a3 and a4
- the order of a2 and a3 is not important
- a4 must take place after both a2 and a3.

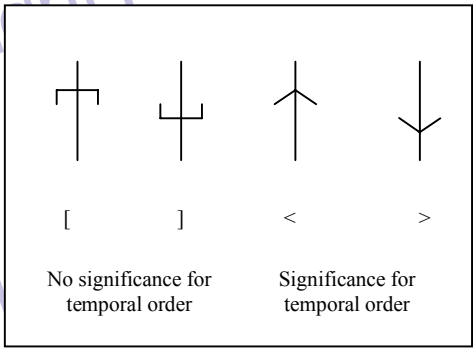
In fact, we can see the problem as follows. In a multi-agent system design and communication specification stage, instead of specifying a single valid execution sequence that must be followed, we might like to specify a number of valid sequences. Recall that interaction diagrams can be used for checking agent execution traces against specified order constraints. In the example above, the designer's intention is to consider both the event sequence a1, a2, a3, a4 and the event sequence a1, a3, a2, a4 as valid. Using the old formalism, we need to provide two different (rigid) interaction diagrams. If the execution trace satisfies one of these diagrams, then the trace is correct. This technique becomes less practical in more complex settings where the number of possible acceptable sequences of events is large (eg. in multi-agent settings with complex interaction protocols). In such a situation, a separate interaction diagram needs to be provided for each valid sequence, and checking needs to be carried out against all interaction diagrams until one (or none) matches. In the next section, we present an extension to the existing framework which supports flexible constraints to be expressed in an interaction diagram.

PROPOSED ENHANCEMENT

In this section, we propose the enhancement of the current interaction diagram formalism as follows. We would like to distinguish between two types of temporal order constraints; namely, groups of events among which the order is important, and groups of events among which the order is not important.

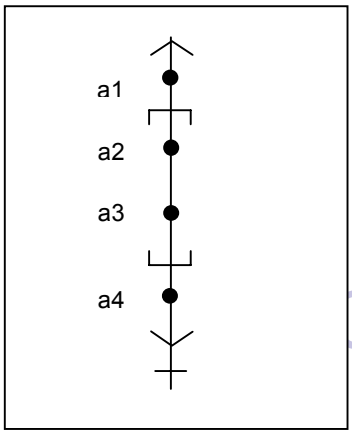
We will append the set of graphical elements with the fragments shown in Figure 4. A couple of fragments, corresponding to the '<' and '>' symbols (we will call them *angular brackets*), represent the start and end of a group of fragments among which the top-to-bottom order is the actual temporal order (i.e. as in the original formalism). This temporal order also states that, in addition to the specified temporal order, no external event is allowed to interleave within those events. The other couple of fragments, corresponding to the '[' and ']' symbols (we call them *square brackets*), represent the start and end of a group of fragments among which the top-to-bottom order does not necessarily reflect the actual temporal order. Throughout the paper, we will use the term "*bracket*" to refer to all types of brackets.

Figure 4: Ordering fragments



A sample interaction diagram using some of the proposed fragments is shown in Figure 5. This interaction diagram expresses the temporal constraints mentioned in the previous section, which the previous formalism failed to express. Simply, the group inside the square bracket-shaped fragments have no specific temporal order. However, event a1 must precede this group, and a4 must take place after it. The following is the corresponding modified computation

Figure 5: Single agent diagram appended with proposed symbols



flow fragmentation. (Note that a1, a2, etc, are placeholders for normal fragments such as snd and rcv).

Note that this fragmentation is semantically equivalent to the following fragmentation, that is changing the order of events inside the square brackets does not affect the meaning of the diagram:

These fragmentations have the following properties:

1. An opening fragment of a particular type should appear before any closing fragment of that type.
2. At any stage of parsing the fragmentation, the number of opening fragments must be greater than or equal to the number of corresponding closing fragments of the same type.
3. At the end of the interaction diagram (or the corresponding fragmentation), the number of opening and closing fragments should be equal (i.e. each opening fragment should have a corresponding closing fragment).
4. No two groups overlap, i.e. if a bracket of group g1 opens, and within that a bracket of group g2 opens, the closing bracket of group g2 must close before the closing bracket of group g1 does. That is each bracket is matched with the nearest corresponding bracket.
5. An *entity* is either an atomic event, a group of events enclosed within a matched pair of brackets, or a group of entities enclosed within a matched pair of brackets. All entities are treated the same with respect to the higher-level group they belong to.
6. The temporal relation between any two events is determined by the type of the nearest complete pair of brackets (angular or square) which contains them.

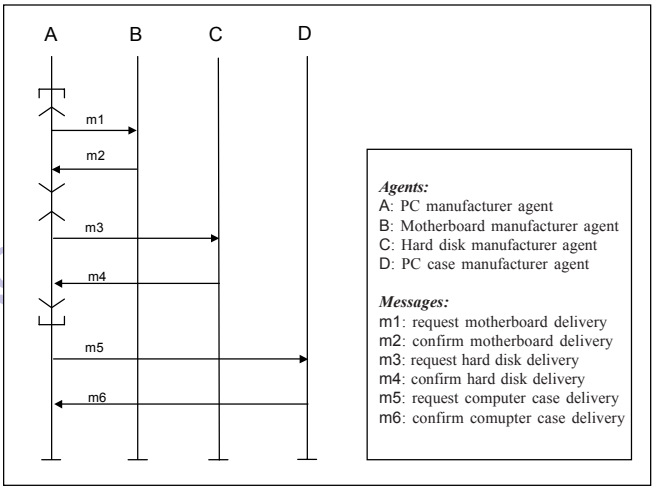
EXAMPLE AND OBSERVATIONS

In this section, we show another example to further illustrate our notation. We will exploit the multi-agent-based supply chain automation domain. This domain is a typical example of situations in which there is a need for flexible specification of multi-agent interactions. Suppose we have four agents:

- **Agent A**: Representative of a personal computer (PC) manufacturing company. This company does not manufacture all computer parts, but rather purchases them from known partners (shown below). There are partners from which this agent purchases motherboards, hard disks, and computer cases.
- **Agents B, C, and D**: Representatives of manufacturers for PC motherboard, hard disks, and computer cases, respectively.

Figure 6 shows the interactions between the different agents, with the meaning of different messages passed between them. Note that since the default case is "order is important", there is no need to include angular brackets at the beginning and the end. The meaning of the diagram with respect to agent A is that the process of ordering motherboards and the process of ordering hard disks can take place in any order. However, ordering the PC cases should take place after the completion of both these processes (suppose there is no point storing computer cases for long periods of time until the other components arrive).

Figure 6: PC manufacturing multi-agent example with flexible order constraints



The computation flow fragmentation of the multi-agent system in Figure 6 is as follows:

- *For agent A*: <beg(A), [<snd(A,m1), rcv(A,m2)>, <snd(A,m3), rcv(A,m4)>], snd(A,m5), rcv(A,m6), end(A)>
- *For agent B*: <beg(B), rcv(B,m1), snd(B,m2), end(B)>
- *For agent C*: <beg(C), rcv(C,m3), snd(C,m4), end(C)>
- *For agent D*: <beg(D), rcv(C,m5), snd(C,m6), end(C)>

Note, however, that in the above fragmentation, the process of ordering motherboards must completely finish (i.e. the request as well the reply must both take place), before the process of ordering hard disks starts. This seems unnatural. One alternative approach is to replace the notation "[<snd(A,m1), rcv(A,m2)>, <snd(A,m3), rcv(A,m4)>] " with the following "< [snd(A,m1), snd(A,m3)], [rcv(A,m2), rcv(A,m4)] > ", which means that sending out the first two orders can happen in any order, and receiving the corresponding responses may also happen in any order, with the only condition being that *both* orders should be sent out before any order gets received. In reality, on the other hand, we might want to express that they can both happen in any order as long as no reply occurs before a request without disallowing a response to be received before the second order is sent. This is one of the limitations of our formalism. This limitation is due to the fact that we do not allow interleaving between events belonging to different bracket pairs. We treat all events within a bracket pair as a single entity with respect to all other events outside that pair.

CONCLUSIONS

Existing interaction diagram formalisms and their corresponding linear notations (known as fragmentations) require a number of strict interaction diagrams to be written in order to allow for different execution sequences. This is because each interaction diagram is capable of expressing a single, strict valid execution sequence. In this paper, we have presented a formalism for describing flexible interaction diagrams which are able to express many possible execution sequences using one interaction diagram. Our formalism describes interaction diagrams which have a mix of two types of temporal relationships between events belonging to a single agent, namely ordered and unordered temporal relationships. An ordered temporal relationship means that events should take place in the order specified. An unordered temporal relationship states that the order of executing two events is not important. We have showed how combinations of these two types

of relationships could be used to represent more complicated scenarios through a linear fragmentation.

Future studies will include further extending our framework to accommodate more complex temporal relationships such as conditional temporal ordering and interleaving events.

ACKNOWLEDGEMENT

Part of this work was done when the second author was at Swinburne University of Technology. The second author would also like to thank Khaled Anton Kattan for valuable discussions and implementations surrounding an earlier version of this paper.

REFERENCES

1. B. Bauer. Extending UML for the Specification of Agent Interaction Protocols, OMG document ad/99-12-03, FIPA submission to the OMG's Analysis and Design Task Force (ADTF) in response to the Request of Information (RFI) entitled "UML2.0 RFI", December 1999.
2. N. R. Jennings. An Agent-based Approach for Building Complex Software Systems. Communications of the ACM, 44(4):35-41, 2001.
3. N. R. Jennings, M. Wooldridge. Applications of Intelligent Agents. In: N.R. Jennings and M. Wooldridge (eds.): Agent Technology: Foundations, Applications, and Markets, pages 3-28, 1998.
4. D. Kinny. The AGENTIS Agent Interaction Model. In Proceedings of the 5th International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1999.
5. C. K. Low, R. Ronnquist and T. Y. Chen. An Automated Tool (IDAF) to Manipulate Interaction Diagrams and Fragmentations for Multi-Agent Systems. International Journal of Software Engineering and Knowledge Engineering, 9(1):127-149, 1997.
6. R. Ronnquist and C. K Low. Formalisation of Interaction Diagrams. In Proceedings of the 3rd Asia-Pacific Software Engineering Conference, Seoul, Korea, IEEE Computer Society Press, pages 318—327, Dec. 1996.
7. M. Wooldridge. Intelligent Agents. In: Grehard Weiss (ed.): Multiagent Systems. MIT Press, April 1999.
8. M. Wooldridge, N. R. Jennings and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design, Journal of Autonomous Agents and Multi-Agent Systems 3:285-312, 2000.