

## A TMO Based Approach to Structuring Real-Time Agents

K. H. (Kane) Kim

DREAM Laboratory / ECE Dept.  
University of California  
Irvine, CA 92697, U.S.A.  
<http://dream.eng.uci.edu>

**Abstract:** Mobile agent structuring is an increasingly practiced branch of distributed computing software engineering. In this paper we discuss the major issues encountered in producing *real-time (RT) agents* which are designed to perform output actions in manners meeting specified timing requirements. An approach to structuring of RT agents which is an extension of a high-level real-time distributed object programming approach called the *Time-triggered Message-triggered Object (TMO)* programming scheme, is also presented. The TMO based approach is promising because it provides a sound framework in which timeliness issues can be resolved in cost-effective manners.

**Keywords:** real-time, agent, mobile, object, TMO, time-trigger, timeliness, protection, structuring, software engineering.

### 1. Introduction

*Mobile agent structuring* is an increasingly practiced branch of distributed computing software engineering [Ari98, Gia01, Kar98, Lan98, Lan99, Lin96, Min99, Pas02]. Mobile agents are software modules with the following fundamental characteristics:

- (1) Agents are dynamically created by their *employers* which are software modules and may themselves be other agents;
- (2) Agents are substantially autonomous in that they receive primarily "high-level commands" from their employers;
- (3) Agents may run in the sites where their employers are running but they are also dispatched by their employers to "visit" and run in remote sites whenever needs arise.

This agent structuring is motivated by the following factors. First, if an agent visits a remote host site and performs a substantial amount of computation which involves extensive interaction with other entities in the host site, then the agent may be able to perform the computation much more efficiently than the employer

can. This is because when the employer will do the same computation, its interaction with the entities in the remote site will involve inefficient communications and thus the computational efficiency will be degraded considerably.

Secondly, when an agent visits a remote host site, the agent may be able to obtain greater access privileges for the resources in the host site than its remotely located employer can. This is because the *host site manager (HSM)* may be willing to allow greater access privileges to an admitted agent than to external entities for security and performance reasons. Interactions between the agent and other entities in the host site do not involve long-distance communications which are less tightly protected. Also, such in-house interactions may involve shorter-term commitment of resources in the host site and thus the HSM may be more willing to allow such interactions than long-distance interactions.

Therefore, agents are meaningful additions to the client-server architecture. The type of agents which became popular first, were the web servers' display rendering agents running in the web browsers' sites. Since then, much more sophisticated types of agents have been developed. Also, agents can dispatch other agents and then the former agents are employers of the latter.

Agents normally visit only those cooperative trustable sites which can also admit trustable visitor agents only. However, in some applications, "brave" agents may be incorporated and such agents may visit potentially unfriendly sites or interact with uncertified potentially dangerous entities in such sites. Provision of such brave agents requires abilities to use defensive logic which can either filter out or enforce safe use of potentially dangerous information by the employers of the brave agents. Provision of brave agents is thus a topic for long-term research [San97, Vig97].

In this paper we discuss the major issues encountered in producing *real-time (RT) agents* which are designed to perform output actions in manners meeting specified timing requirements. Such RT agents are desired for several reasons.

- (1) RT applications impose timing requirements on certain actions [Kop97]. Untimely outputs of agents

can be useless or even poisonous in such applications.

- (2) Agents must often interact not only with their employers but also with other agents. Careless design of autonomy or inter-dependency in a cooperating community of agents can often lead to chaotic behavior of the agent community including the employers. Examples of chaotic behavior are output actions at wrong times, conflicting actions of agents, flooding the networks with useless messages, etc. Use of RT agents helps in avoiding chaotic behavior because it is easier to effect efficient coordination of agents and their employers and their existence requires careful management of resources in the sites. Their individual and collective behavior tends to be much more predictable than that of non-RT agents.
- (3) It is easier to make robust RT agents than robust non-RT agents. This is again because the sites which can run RT agents must do careful management of resources and it is easier to coordinate replicas of RT agents than replicas of non-RT agents.

However, the field of RT agents is in its infancy and there are a number of technical challenges that must be overcome before use of RT agents becomes a common practice. Such challenges are discussed in the next section (2).

Then an approach to structuring of RT agents which is an extension of a high-level real-time distributed object programming approach called the *Time-triggered Message-triggered Object* (TMO) programming scheme [Kim97, Kim00], is discussed in Section 3. The TMO scheme has been established in the last 10 years. The paper concludes in Section 4.

## 2. Challenges in realizing real-time agents

Agents may visit remote sites or reside in the same sites where their employers reside. In this section, only those agents running in remote sites are considered because the agents co-resident in the same sites with their employers do not pose any additional challenge.

### 2.1 Timely admission and activation at the destination site

Since an RT agent is subject to a deadline for its first reporting to its employer, its dispatch must be executed efficiently in a time-bounded manner. The dispatch involves the following sequence of operations:

- (1) The employer's submission to the remote host site of a request for permission to visit;
- (2) The remote HSM's reply;

- (3) Assuming that the remote HSM replied positively, the employer's supply of the code of the agent;

- (4) The remote HSM's loading of the code onto its memory, and activation of the agent.

Therefore, both the remote HSM and the communication infrastructure involved must act in time-bounded manners.

The remote HSM will accept a visit request only if it has the resources indicated by the visit requestor (i.e., employer of the proposed agent) as needed to run the proposed agent. Therefore, the resource requirements must be specified in forms understandable to both the employer and the remote HSM. Admission of an agent then involves reservation of necessary resources in the host site.

### 2.2 Timely interactions between agents and employers

Since an RT agent has deadlines to meet in making reports to its employer, it must acquire and release execution resources in the host site in timely manners within the limits agreed at the time of its admission into the host site. Conversely, the HSM and the operating system (OS) in the host site must allocate resources to the agent efficiently in the same manner as they allocate resources to other RT processes and RT objects in the site.

Whenever requested by an RT agent, the employer must respond within predetermined time bounds. This must be effected by the HSM and the OS in the site running the employer. The designers of the employer and the agent must ensure such timely interaction capabilities.

### 2.3 Timely discovery of dead or sick agents

Dead or sick RT agents must be discovered by both the HSMs of the host sites and their employers located in remote sites.

An HSM can detect a dead RT agent by operation of a time-out mechanism triggered upon absence of a farewell greeting message or a health report beyond a deadline. The HSM may also notice the lack of responses from the agent to its health inquiries. Similarly, the employer can also detect the crash of its RT agent by a time-out associated with the arrival of a (progress or final) report from the agent.

As for detecting a sick RT agent, the problem is more complicated. The HSM of the host site can notice unreasonable attempts of the agent to access certain prohibited entities. Or unreasonably late responses from the agent to its health inquiries can be noticed. In the case of the employer, it can notice the invalid contents in the reports received from the sick agent.

One can also design an agent to perform self-diagnosis at various points during its life-time. Such a self-check can reveal a sick condition.

## 2.4 Timely recovery from the failures of RT agents

Once a dead or sick agent is discovered by the HSM in the site hosting the agent or the employer, the HSM and the employer must cooperate to erase any remnants of the agent in the site. The employer must then perform some RT recovery actions. This means that the employer must be designed with the capability for taking appropriate actions upon discovering the death of an agent.

If a sick agent detects its own sick condition, it must report the detection to both the HSM in the host site and its employer. Any further actions of the agent depend on whether it was designed to possess retry capabilities or not. If the agent possesses such capabilities, it can make a retry and perform a self-diagnosis again. If the self-diagnosis produces good results, the agent reports the condition to both the HSM and the employer again and proceeds to complete the rest of its mission. If the self-diagnosis result is bad again, the agent can make some more retries if it possesses additional retry capabilities or simply take a suicide.

All the recovery actions mentioned above must be executed in time-bounded manners.

## 2.5 Protection of the host site against attacks from agents dispatched by malicious employers

Each site hosting a visiting agent must be robust in the presence of malicious employers who may dispatch malicious agents [Gra96, Gra97, Kar97, Lev95, Vig97]. The most fundamental issue here is that of authenticating requestors before admitting the agents dispatched by them. Numerous solutions for such authentication problems have been studied in recent years.

Another issue is to protect the host site against the malicious behavior of an admitted agent. This is the responsibility of the HSM and the OS in the host site. Such protection problem is similar to that in the domain of web objects, which is currently an active area of research in both industry and academia.

## 2.6 Protection of agents from the snooping of malicious entities in host sites

In a sense, this protection is the responsibility of the HSM in the host site. On the other hand, the designer of an agent in this case is aware of non-negligible chances of encountering mildly malicious entities in the host site, i.e., the entities performing the snooping only. The designer must then make conscious efforts for not revealing any sensitive information to such snoopers. First of all, the employer must not send to the agent any sensitive information which is not essential in the agent's activities. Secondly, the sensitive information must be encoded to make it difficult for snoopers to decode it and make meaningful use of the information.

## 3. An approach to TMO structuring of real-time agents

### 3.1 An overview of the TMO scheme and the TMOSM architecture

High-level approaches to RT distributed programming are generally aimed for relieving the programmer from the burden of handling many details specific to execution platforms which may be composed of processors, standard peripherals, OS kernels, and middleware [Bol00, ISO, Kim97, OMG02, WOR]. The Time-triggered Message-triggered Object (TMO) programming scheme is among the most advanced approaches of such kind [Kim97, Kim00, additional references in <http://dream.eng.uci.edu/tmo/tmo.htm>]. Its development started in early 1990's with a concrete skeleton of the syntactic structure and execution semantics to support economical reliable design and implementation of RT systems.

The TMO programming scheme is a general-style component programming scheme and supports design of all types of components including distributable hard-RT objects and distributable non-RT objects within one general structure. TMO is a natural, syntactically minor, and semantically powerful extension of the conventional object(s). Typical object-oriented (OO) programmers can adopt it with relatively small efforts.

TMOs are devised to contain only high-level intuitive and yet precise expressions of timing requirements. No specification of timing requirements in (indirect) terms other than *start-windows* and *completion deadlines* for program units (e.g., object methods) and *time-windows for output actions* is required. For example, priorities are attributes often attached by the OS to low-level program abstractions such as threads and they are not natural expressions of timing requirements. Therefore, no such indirect and inaccurate styles of expressing timing requirements are associated with objects and methods.

At the same time the TMO scheme is aimed for enabling a great reduction of the designer's efforts in guaranteeing timely service capabilities of distributed computing application systems. It has been formulated from the beginning with the objective of enabling *design-time guaranteeing of timely actions*. The TMO incorporates several rules for execution of its components that make the analysis of the worst-case time behavior of TMOs to be systematic and relatively easy while not reducing the programming power in any way.

#### 3.1.1 TMO structure and design paradigms

As depicted in Figure 1, the basic TMO structure consists of four parts:

ODS-sec = object-data-store section: list of object-data-store segments (ODSS's); Each ODSS is a group of data members and is a unit

that can be locked for exclusive use by one method execution at a time as well as for shared use by multiple concurrent method executions which perform read-only operations on the data members contained; Data members are thus grouped into harmoniously sharable data store units called object data store segments (ODSSs) in a TMO.

**EAC-sec** = *environment access-capability* section: list of *gates* to remote object methods, logical communication channels, and I/O device interfaces;

**SpM-sec** = *spontaneous-method* section: list of *spontaneous methods*;

**SvM-sec** = *service-method* section.

Major features are summarized below. The second and third are the most conspicuous unique extensions of conventional object(s).

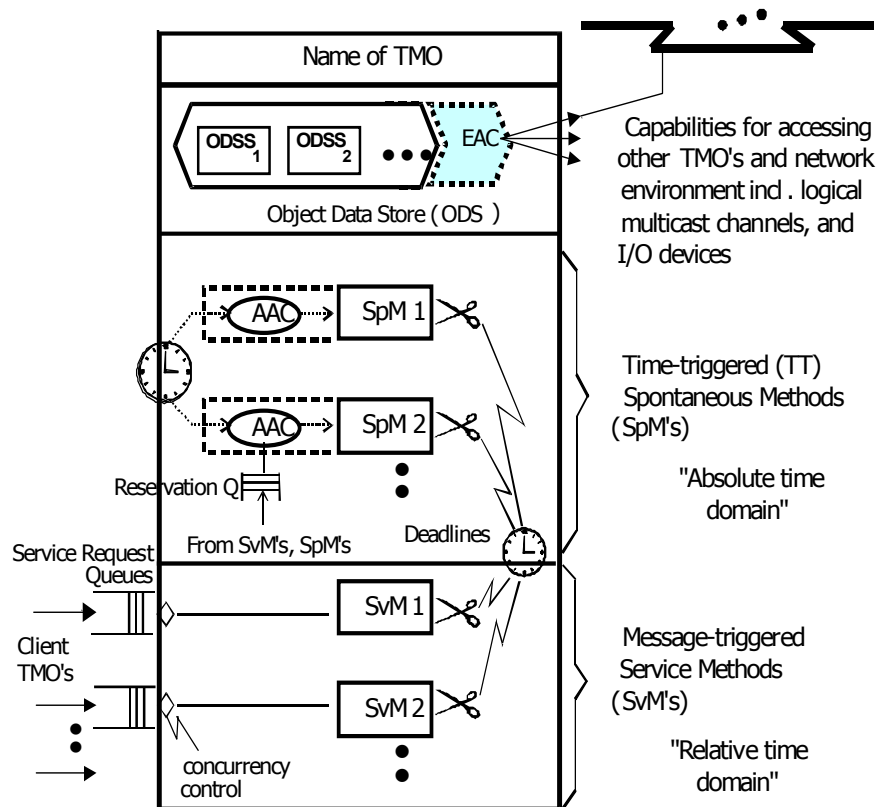


Figure 1. The basic structure of TMO (Adapted from [Kim97])

(a) *Distributed computing component:*

The TMO is a distributed computing component and thus TMOs distributed over multiple nodes may interact via remote method calls. To maximize the concurrency in execution of client methods in one node and server methods in the same node or different nodes, client methods are allowed to make non-blocking types of service requests to server methods.

(b) *Clear separation between two types of methods:*

The TMO may contain two types of methods, *time-triggered (TT-) methods* (also called the *spontaneous methods* or *SpMs*), which are clearly separated from the conventional *service methods (SvMs)*. The SpM executions are triggered upon reaching of the real-time clock at specific values determined at the design time whereas the SvM executions are triggered by service request messages from clients. Moreover, actions to be taken at real times *which can be determined at the design time* can appear only in SpMs.

(c) *Basic concurrency constraint (BCC):*

This rule prevents potential conflicts between SpMs and SvMs and reduces the designer's efforts in guaranteeing timely service capabilities of TMOs. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially*

*conflicting SpM executions are not in place.* An SvM is allowed to execute only when an execution time-window big enough for the SvM that does not overlap with the execution time-window of any SpM which accesses the same ODSSs to be accessed by the SvM, opens up. However, the BCC does not stand in the way of either concurrent SpM executions or concurrent SvM executions.

(d) *Guaranteed completion time for method execution and deadline for result return:*

The TMO incorporates deadlines in the most general form. Basically, for output actions and method completions of a TMO, the designer guarantees and advertises execution time-windows bounded by start times and completion times. In addition, deadlines can be specified in the client's calls for service methods for the return of the service results.

Triggering times for SpMs must be fully specified as constants during the design time. Those RT constants appear in the first clause of an SpM specification called the *autonomous activation condition (AAC)* section. An example of an AAC is

"for t = from 10am to 10:50am every 30min  
start-during (t, t+5min) finish-by t+10min"

which has the same effect as

```
{ "start-during (10am, 10:05am)
  finish-by 10:10am",
  "start-during (10:30am, 10:35am)
  finish-by 10:40am" }
```

An underlying design philosophy of the TMO scheme is that an RT computer system will always take the form of a network of TMOs, which may be produced in a top-down multi-step fashion [Kim97]. The designer of each TMO provides a guarantee of timely service capabilities of the object. The designer does so by indicating the *guaranteed execution time-window for every output* produced by each SvM as well as by each SpM executed on requests from the SvM and the *guaranteed completion time* (GCT) for the SvM in the specification of the SvM. Such specification of each SvM is advertised to the designers of potential client objects. Before determining the time-window specification, the server object designer must convince himself/herself that with the *object execution engine* (a composition of hardware, node OS, and middleware) available, the server object can be implemented to always execute the SvM such that the output action is performed within the time-window. The BCC contributes to major reduction of these burdens imposed on the designer.

It should also be noted that the ODSS in a TMO may be a group of spontaneously changing data members. For example, an ODSS may be a nested TMO which can spontaneously change. An ODSS may also be a device driver interface.

### 3.1.2 TMO support middleware (TMOSM)

Currently one of the most advanced models of the execution engines for TMOs is the middleware architecture named the *TMO support middleware* (TMOSM) which can be implemented on widely used commercial hardware and OS platforms [Kim99]. TMOSM uses well established services of commercial OSs, e.g., process and thread support services, short-term scheduling services, and low-level communication protocols, in a manner transparent to the application TMO programmer. Prototype implementations based on several most popular OS kernels such as Windows NT/XP, Windows CE, and Linux exist. Although major structural characteristics of TMOSM have not been changed since its inception in 1998, its refinements have steadily occurred, especially in concurrency and I/O management.

A friendly programmer interface wrapping the services of TMOSM has also been developed and named the *TMO Support Library (TMOSL)* [Kim00]. It consists of a number of C++ classes and is meant to be an API-style approximation of an idealistic fully featured TMO programming language. It empowers C++ programmers with the following powerful and natural mechanisms for specification of unique and essential features of any RT distributed programs:

(1) Global time base [Kop97];

(2) Time-triggered (TT) action;

(3) Concurrency control;

(4) Interaction among RT objects and RT message communication, including blocking calls for remote methods, non-blocking calls, and direct message communication over *programmable real-time logical multicast channels* [Kin00].

### 3.2 Agent TMOs

An agent can be created and operated as depicted in Figure 2. The SpM resident in Site 1, Invest\_TMO.SpM, dispatches an agent to Site 2. The seven steps involved are as follows.

(1) Reservation for a visiting slot: Invest\_TMO.SpM issues a call for dispatching an agent using the code, Agent1.dll. The TMO execution engine in Site 1, which consists of a TMOSM instantiation, a node OS, and node hardware, executes this dispatching in several steps. The first step is to send a request to the HSM in Site 2, HSM2, for reservation of a visiting slot.

(2) Reservation confirmation: HSM2 checks its resource conditions and decides whether to accept or reject the reservation request. In Figure 2, an acceptance notice is returned to Site 1.

(3) Supply of agent code and the initial state: The TMO execution engine in Site 1 transmits to HSM2 the code of the agent being dispatched, Agent1.dll, along with the initial state information provided by Invest\_TMO.SpM in the dispatch call as a parameter.

(4) Loading and activation: HSM2 loads the received Agent1.dll onto the working memory and executes the main function (i.e., an entry function) in Agent1.dll. As a result, Agent1\_TMO is activated since the main function contains a statement for activation of the TMO. Steps 2, 3, and 4 are transparent to the designer of the employer, Invest\_TMO.SpM.

(5) Settlement report: Some time after Agent1\_TMO is activated, it sends the first report to its employer. A convenient mechanism for this report communication is the *Real-time Multicast and Memory-replication Channel* (RMMC) which is a part of the TMO [Kim00]. RMMC is a logical multicast channel and as such, can be dynamically created and removed. A TMO can establish an access gate to a desired RMMC in its ODS. In Figure 2, the employer, Invest\_TMO.SpM, waits for that first report.

(6) Command: Based on the contents of the report from Agent1\_TMO, the employer may decide to send another command via an RMMC. This kind of interactions between the employer and the agent TMO can be repeated.

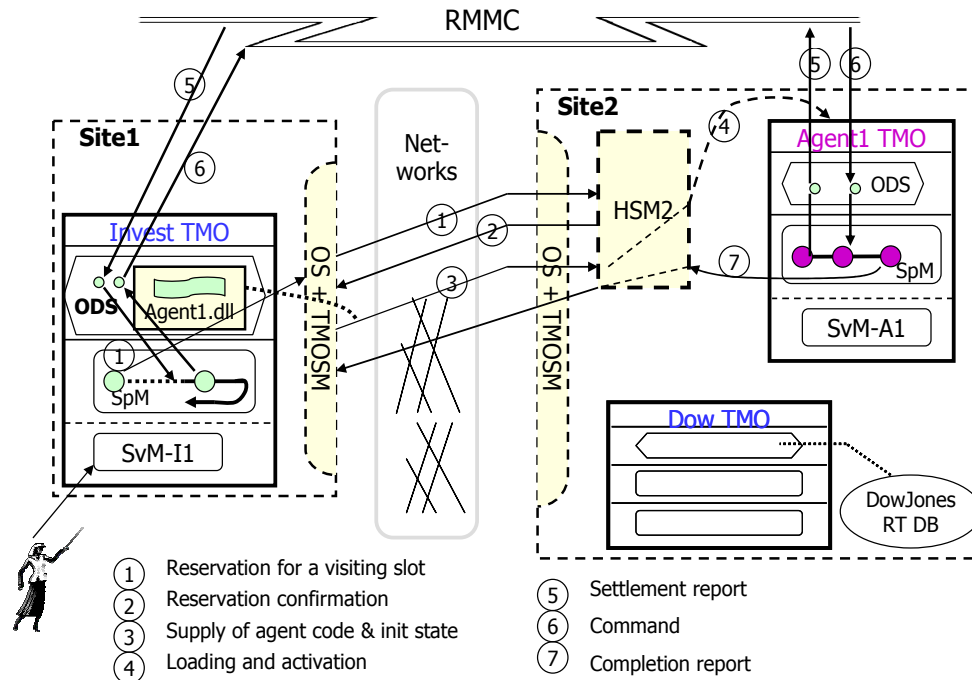


Figure 2. An agent TMO and its employer

(7) Completion report: When Agent1\_TMO has finished its mission, it can either send an explicit notice to HSM2 or simply terminate in which case the TMO execution engine in Site 2 will inform HSM2. HSM2 then notifies Site 1 of the termination of Agent1\_TMO.

The first four steps are for establishment of an agent at a remote site. In TMO execution environments, the conditions already exist for executing these four steps in timely manners. Similarly, RMMCs ensure timely interactions between the employer and the agent.

In Figure 2, much of the agent's mission is carried out by Agent1\_TMO.SpM. This SpM may be executed just once or periodically. Note that between periodic executions, the employer can call for the SvM-A1 service which may have been designed to accept new commands. In other words, the RMMC is not the only mechanism facilitating the transfer of commands from the employer to the agent.

Similarly, the employer may live for a single execution or periodic executions of Invest\_TMO.SpM. If a periodic execution is involved, then between periodic executions of the employer SpM the agent can call for the SvM-I1 service which may have been designed to accept new reports.

### 3.3 Implementation issues

The framework presented in the preceding section (3.2) is promising for meeting the timing requirements in

cost-effective manners. However, its complete and efficient implementation requires a considerable amount of research in the future. The most urgent challenge to be overcome is to establish a rigorous approach for representation of resources required to run an agent. A desired representation must enable efficient on-line decision by the HSM on whether to admit the proposed agent or not while keeping a maximal set of agents admitted safely all the time.

Also, the framework presented in the preceding section does not provide any new mechanisms for protection of the host site. This issue discussed in Section 2.5 remains as a challenging topic for future research.

TMOs can be built in any of the basic object programming languages, e.g., C++, Java, and C#. Detailed implementation structures for agent TMO support facilities will differ to some extent, depending on which base programming language is used. A prototype implementation of the facilities depicted in Figure 2 is under way in the author's laboratory.

Based on the structure discussed up to now, RT agents with considerable intelligence can be conceived. For example, RT agents that can adapt to dynamically changing health or resource conditions of the host site and if necessary, migrate into a new host site and efficiently discover resources in the new site, are conceivable.

### 3.4 Incorporation of rule declarations into agent TMOs

Rule-declarations are higher-level abstractions than conventional procedural program-segments. Therefore, in some applications, it may be beneficial to allow incorporation of *event-condition-action* (ECA) statements into the TMO at the same level as SvMs and SpMs. This means to support the following type of statements.

```
whenever cond1(data1) and cond2(data2)
do <action-1>;
```

The following is an abstract example.

```
whenever cond1 (sensor data during last interval)
and cond2 (abridged history of sensor data)
do <action-1>;
```

Figure 3 shows such an extended TMO. In principle, it should not be too difficult to support certain rule declarations in the extended TMO. Suppose that we make it mandatory in the extended TMO programming to design each ODSS as a protected object and also design it to have access methods. If a condition phrase in a certain rule declaration is related to an ODSS, then the safest thing to do is to check for the existence of the specified condition whenever the content of the ODSS changes. A practical approach to realizing this is to check for the existence of the condition immediately before closing the execution of a read-write access method of the ODSS.

Invocation of such a checking routine can be largely mechanized. It should be possible to build a tool which is capable of inserting such condition-checking logic into ODSS access methods.

Once a specified condition is found, then the relevant action phrase should be invoked. Whether such execution of an action phrase can or should be done immediately in all cases or it can or should be delayed until the current TMO method execution is completed, is a topic for further research.

Another conceivable approach to supporting rule declarations without requiring the programmer to provide access methods for every ODSS is to have a tool insert relevant condition-checking logics into the exit paths in every TMO method. Suppose a TMO method, SpM1, accesses ODSS1 for read-write purpose and ODSS1 appears in one condition phrase, COND1, in a rule declaration. Then if SpM1 has just one exit point, the tool will insert logic for checking COND1 immediately before the exit point. Whether such late checking is sufficient or not is again a topic for further research.

#### 4. Conclusion

RT agent programming is in its infancy. A number of basic challenging issues that need to be resolved before RT agent programming become a common practice were discussed in this paper.

The TMO based approach for structuring of RT agents which has been presented in this paper is

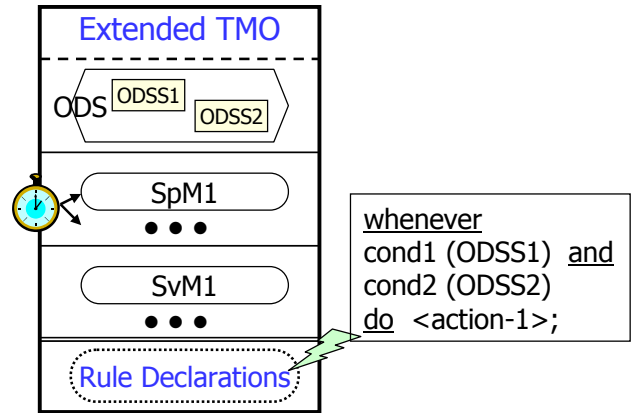


Figure 3. The TMO extended with the rule section

promising because it provides a sound framework in which timeliness issues can be resolved in cost-effective manners. However, the experimental research with this approach is at an early stage. Also, much more research is needed in the area of facilitating protection of the host site against undesirable behaviors of sloppy or even malicious agents.

**Acknowledgments:** The author wishes to acknowledge helpful discussions with Yuqing Li in the UCI DREAM Lab. The research work reported here was supported in part by the NSF under Grant Numbers 99-75053 and 02-04050 (NGS) and 00-86147 (ITR), and in part by the US DARPA under Contract F33615-01-C-1902 monitored by AFRL. No part of this paper represents the views and opinions of any of the sponsors mentioned above.

#### References

- [Ari98] Aridor, Y., and Lange, D.B., "Agent Design Patterns: Elements of Agent Application Design", *Proc. 2nd Int'l Conf. on Autonomous Agents (Agents '98)*, ACM Press, 1998, pp. 108-115.
- [Bol00] Bollella, G., and Gosling, J., "The Real-Time Specification for Java", *IEEE Computer*, June, 2000, pp. 47-54.
- [Gia01] Giampapa, J.A., Juarez-Espinosa, O., and Sycara, K., "Configuration Management for Multi-Agent Systems," *Proc. ACM 5th Int'l Conf. on Autonomous Agents (Agents 2001)*, June, 2001, pp. 230-231.
- [Gra96] Gray, R.S., "Agent Tcl: A flexible and secure mobile-agent system", *Proc. the USENIX 1996 Tcl/Tk Workshop*, July, 1996, pp. 9-23.
- [Gra97] Gray, R.S., et al., "D'Agents: Security in a multiple-language, mobile-agent system", in Giovanni Vigna, ed., '*Mobile Agents and Security*', Lecture Notes in Computer Science, Vol. 1419, Springer-Verlag,

1997ISBN: 3540647929.

[ISO] Series of *Proc. ISORC (IEEE CS Int'l Symp. on Object-oriented Real-time distributed Computing)*, IEEE CS Press: '98, '99, 2000, 2001, and 2002.

[Kar97] Karjoth, G., Lange, D.B., and Oshima, M., "A Security Model for Aglets", *IEEE Internet Computing* 1, 4, July/August, 1997, pp.68-77.

[Kar98] Karnik, N.M., and Tripathi, A.R., "Design Issues in Mobile-Agent Programming Systems", *IEEE Concurrency*, Vol. 6, No. 3, July-September 1998, pp. 52-61.

[Kim97] Kim, K.H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No.8, August 1997, pp. 62-70.

[Kim99] Kim, K.H. Ishida, Masaki, Liu, Juqiang, "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", *Proc. 2nd IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing (ISORC '99)*, St. Malo, France, May, 1999, pp.54-63.

[Kim00] Kim, K.H., "APIs for Real-Time Distributed Object Programming", *IEEE Computer*, June 2000, pp.72-80.

[Kop97] Kopetz, H., 'Real-Time Systems: Design Principles for Distributed Embedded Applications', *Kluwer Academic Publishers*, ISBN: 0-7923-9894-7, Boston, 1997.

[Lan98] Lange, D.B., and Oshima, M., '*Programming and Deploying Java Mobile Agents with Aglets*', Addison-Wesley Longman, Reading, Mass., 1998, ISBN 0-201-32582-9.

[Lan99] Lange, D.B. and Oshima, M., "Seven Good Reasons for Mobile Agents", *Comm. ACM*, 42(3), March 1999, pp.88-89.

[Lev95] Levy, J.Y., and Ousterhout, J.K., "Safe Tcl toolkit for electronic meeting places", *Proc First USENIX Workshop on Electronic Commerce*, July 1995, pp. 133-135.

[Lin96] Lingnau, A., and Drobnik, O., "Making Mobile Agents Communicate: A Flexible Approach", *Proc. 1st Annual Conf. on Emerging Technologies and Applications in Communications (etaCOM'96)*, May 1996.

[Min99] Minar, N., et al., "Hive: Distributed agents for networking things", *Proc. ASA/MA'99 (1st Int'l Symp. on Agent Systems and Applications and 3rd Int'l Symp. on Mobile Agents)*, 1999.

[OMG02] Object Management Group, "Chapter 24. Real-time CORBA", in *CORBA Specification, Version 2.6.1*, [http://www.omg.org/technology/documents/formal/corba\\_2.htm](http://www.omg.org/technology/documents/formal/corba_2.htm), May, 2002.

[Pas02] Pasquale, J., et al, "Improving Wireless Access to the Internet By Extending the Client/Server Model," *Proc. European Wireless Conference*, Florence, Italy, Feb. 2002.

[San97] Sander, T., and Tschudin, C.F., "Protecting mobile agents against malicious hosts", Chapter 4 in Giovanni Vigna, ed., '*Mobile Agents and Security*', Lecture Notes in Computer Science, Vol. 1419, Springer-Verlag, 1997, ISBN: 3540647929, pp. 44-61.

[WOR] Series of *Proc. WORDS (IEEE CS Workshop on Object-oriented Real-time Dependable Systems)*, IEEE CS Press: '94, '96, '97, '99, '99F, 2001, and 2002.

[Vig97] Vigna, G., ed., '*Mobile Agents and Security*', Lecture Notes in Computer Science, Vol. 1419, Springer-Verlag, 1997, ISBN: 3540647929.

Proceedings

# 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2002)

4–6 November 2002

Washington, DC

*Sponsored by*

IEEE Computer Society

*In cooperation with*

The Information Technology Research Institute  
Wright State University



Los Alamitos, California

Washington • Brussels • Tokyo

---