TOWARD A FIPA COMPLIANT MULTI-AGENT REAL-TIME ARCHITECTURE FOR INTEGRATED MODULAR AVIONICS

Johann Duscher*, Dr. Norbert Oswald, Dr. Rupert Reiger***
***EADS CRC/LG-AS, **EADS Military Aircraft**

Keywords: *modular avionics, multi-agent system, peer-to-peer, distributed blackboard*

Abstract

In our work we propose a multi-agent based real-time architecture for the implementation of multi-agent based heterogeneous Integrated Modular Avionics (IMA). It utilizes a distributed blackboard architecture with rule-based event notification in order to efficiently minimize communication overhead and to provide a flexible, scalable, and fault-tolerant peer-to-peer infrastructure for information dissemination between agents situated in arbitrary safety partitions. Moreover, the proposed architecture conforms to the widely accepted standards defined by the Foundation for Intelligent Physical Agents (FIPA) – an organization coping with multi-agent technology.

1 General Introduction

Consider an unmanned aerial vehicle (UAV) autonomously acting in a network-centric concerted actions scenario. It will be confronted with a vast of sensor data originating from its own sensors and other data sources (like other UAVs, Satellites, Ground Station, etc.). Ideally, its avionic system should process all this data in real-time and initiate appropriate actions. But because of a UAV's limited energy, intra-avionics communication bandwidth, and processing resources, this may not be feasible.

A solution for this problem is to enable avionic components to intelligently adapt to the current system needs and situation context in order to control the focus of information processing. An IMA architecture is thus required to provide a huge amount of flexibility. Another requirement that must be considered (regardless of a concrete architecture design) in order to make it a viable approach for IMA is to implement fault-tolerance and to provide support for safety partitions, which guarantee physical bounds with respect to failure propagation in case of malfunction of some subsystem(s).

1.1 The Client-Server Based Approach

Currently, there are some efforts to realize IMA using real-time operating system (e.g. LynxOS or VxWorks) and client-server based middleware technologies (e.g. RT/FT-CORBA) that enable transparent communication over the network and fault-tolerant integration of heterogeneous hardware and software components.

Those client-server based architecture approaches, however, have some drawbacks with respect to flexibility and fault-tolerance. Additionally, the CORBA standard is very complex which complicates the necessary certification of CORBA orbs, not to mention the resulting high costs.

1.2 The Peer-to-Peer Based Approach

In contrast to purely client-server based approaches some new research efforts focus on the realization of systems in a peer-to-peer fashion, because the latter approach promises considerable advantages over the former one regarding fault-tolerance, scalability, and flexibility.

However, in a peer-to-peer system new problems arise that have to be dealt with. A

UAV and, consequently, its avionic components should draw decisions based on currently available information about the global system's state and the environment. This implies that all decision-support and decision-making components should have access to all information about the current situation context they may be interested in and, hence, some repository is needed where all this information is stored to provide such an access. But in a peer-to-peer system there generally is no (or at least should not be) a central server component. Consequently, peers have to communicate relevant information to other peer nodes efficiently and in a reliable fashion, which requires an appropriate information dissemination infrastructure that covers real-time, safety, and fault-tolerance aspects.

1.3 The Multi-Agent Based Approach

A promising approach to tackle all these problems in a peer-to-peer fashion is to implement IMA using a real-time enabled multi-agent system (MAS). Multi-agent systems generally are peer-to-peer in nature, and – though no commonly agreed definition for agents exists – there is, however, often a consensus about some important properties: they are assumed to possess some degree of autonomy and intelligence, and they may act proactively and goal-oriented. Within a well-defined border this allows an agent to act and draw decisions autonomously in order to reach its goals in a flexible and intelligent way. For this reason our work focuses on a multi-agent based architecture for IMA.

2 Blackboards as a Means for Multi-Agent Coordination and Information Exchange

A modern avionics system for manned aerial vehicles fulfills complex mission management tasks and must deal with uncertainty (sensor data can be erroneous or inaccurate, and the environment is open, non-deterministic, and non-cooperative). Clearly, this especially accounts for UAVs where autonomous mission accomplishment is required. Managing highly complex problems like autonomous mission management in real-time is a very difficult task. In a MAS based IMA architecture this would require the agents to cooperate and coordinate with each other necessitating an appropriate infrastructure. Additionally, as already mentioned earlier, some kind of a repository is needed that provides access to each aspect of the current situation context and global system state agents might be interested in. Only this way IMA agents can optimally adapt to their needs. As we will see later the Blackboard architectural pattern is a perfect candidate for both the required data repository and the required infrastructure for highly complex problem-solving tasks. Accordingly, our proposed RT-MAS architecture for IMA builds upon a (distributed) variation of the Blackboard architectural pattern.

In order to recognize all the benefits of our suggested approach a general understanding of the Blackboard pattern is very important. Thus, we decided a brief introduction to the general pattern should follow first. Subsequently, we describe our suggested blackboard architecture which supports efficient multi-agent coordination by minimizing communication overhead necessary for information exchange through the application of a rule-based notification scheme. Finally, we show how several instances of such blackboards can be "plugged" together in order to realize a distributed blackboard which we will use as a means for an efficient and fault-tolerant infrastructure enabling information dissemination between agents located in arbitrary safety partitions of an IMA system.

2.1 The Blackboard Architectural Pattern

In general, a blackboard architecture is a data repository that represents an area of shared memory. Agents (or "knowledge sources" in terms of the Blackboard architectural pattern respectively) use this repository to share knowledge and (partial) solutions in order to cooperatively solve a complex problem. With other words, the blackboard is the source of all data on which an agent will operate and is the

destination of all conclusions from an agent. Once it finds the information it needs on the blackboard, it can proceed without any assistance from other agents. Thus, there is no direct dependency between agents, but only between their exchanged information, which leads to great flexibility. Also, a blackboard usually contains information on different abstraction levels at the same instant of time and, hence, problem solving occurs on different level of abstraction in parallel.

The loose coupling of the agents and the ability to solve problems on different abstraction levels in parallel are the key strengths responsible for the great success of the Blackboard architectural pattern in real applications. It has been widely used to tackle the problems with the characteristics of uncertainty, and non-deterministic, because there is often no direct algorithmic solution to these problems and, thus, only a best effort approach remains feasible. Applications of this pattern can be found in different kinds of software systems requiring communication, mobility, coordination, and real-time.

2.2 Rule-Based Agent Notification

2.2.1 Motivation

A common problem to solve when implementing a blackboard is how and, more important, when information is exchanged between the blackboard and cooperating agents. With other words, the performance of a blackboard implementation strongly depends on the efficiency of the communication model required to provide agents with newly available information.

A very inefficient communication model would be a pull-based approach, because agents would have to frequently query the blackboard for new information (polling) and, thereby, wasting a lot of resources. A much more efficient model is a push-based/event-driven mechanism where agents subscribe to a blackboard together with descriptions telling it what kind of information each agent is interested in. The blackboard then notifies a specific agent only when new information had been written on the blackboard it *might be interested* in. Clearly, this approach seeks to claim network resources only when they are needed. Besides reducing communication overhead such an approach also has another advantage. Agents have to interrupt their work in progress only in case new relevant information *might be* available. This idea of an event-driven blackboard is not new, but the key to an efficient implementation of a blackboard architecture suitable for distributed real-time multi-agent systems is for an agent to be able to tell the blackboard *as exactly as possible* when and what kind of information it needs such that notification really occurs only when agents are *in fact interested* in newly published information.

2.2.2 A Brief Architectural Overview

The blackboard architecture we implemented fully addresses this requirement and reduces communication overhead to a minimum, because agents can (at least theoretically) provide arbitrary complex notification rules during subscription with the blackboard. Every time the blackboard is updated these notification rules are matched against the data written on the blackboard. If some rules fire the corresponding agents are notified. Upon an agent's deregistration rules associated with that agent are removed from the blackboard again. Fig. 1 gives an overview over the three main components of our blackboard implementation:

- a *knowledge base* where the data written on the blackboard is actually stored,
- a *rule base* for the storage of the agents' notification rules,
- a *rule engine* which interprets an agents' individual notification rules and matches them against available data on the blackboard.

Typically, a knowledge-based system also consists exactly of these three components, so we used the efficient open source reasoning engine CLIPS to implement the blackboard (as a proof of concept). As far as we know there is no similar blackboard architecture that realized the idea of rule-based agent notification already.

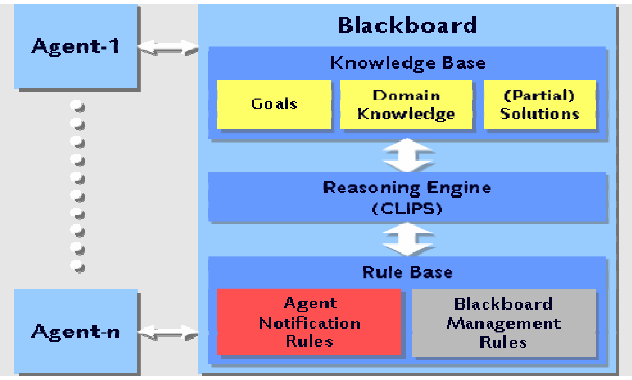

Fig. 1 A blackboard with rule-based agent notification

2.2.3 Syntax & Semantics of Notification Rules

Due to limited space we cannot discuss the rule syntax and the rule matching semantics in detail here. Instead, we provide the most important excerpt of the rule specification grammar and try to explain the rule matching semantics with the help of an exemplary rule. Readers not familiar with rule-based systems might have a look at [8] which provides a good introduction to rule-based systems.

Rules in rule-based systems generally are specified in a form equivalent to statements like

```
IF <condition> THEN <action> ENDIF
```

known from ordinary programming languages. However, in terms of rules the condition part is called *antecedent*, whereas the action part is called *consequent*. We use the following syntax to express rules (terminal symbols are colored red and written in bold style):

```
<rule_def> ::=
  ( rule <rule_name>
    <antecedent> → <consequent> )

<antecedent> ::=
  [-]<constraint> {, [-]<constraint>}*

<constraint> ::=
  <objclass_constr> | <boolean_constr>

<objclass_constr> ::=
  <variable> : <objclass_name>

<variable> ::= ?<variable_name>

<consequent> ::= <action>
```

The antecedent of a rule is a series of constraints (serving as conditions) concatenated by commas which denote the *logical AND* operator. The minus sign denotes a *logical NOT* operator.

The semantics of the rule matching process generally is as follows. If during matching a constraint does not contain any free variables then the rule engine interprets it as a boolean expression and evaluates it accordingly. However, if it does then the rule interpreter tries to instantiate all free variables by *binding* them to specific values. In case of object class constraints the interpreter tries to match the free variables with objects (data instances) of its knowledge-base. In case of other constraints the interpreter tries to *unify* the free variables with values such that the constraint is fulfilled. If a free variable cannot be instantiated the constraint evaluates to a *logical false*.

For instance, suppose the `store_mgr` of an avionics system wants to be notified about possible threats. Further assume it considers all objects simultaneously reported by infrared and radar sensors as possible threats. During registration with its local blackboard `bb_C`, the agent might subscribe with a notification rule looking similar to the one shown in Fig. 2.

```
(rule bb_C::store_mgr::report_threats
  ?IR : InfraredSensor,
  ?RAD : RadarSensor,
  ?OS1 = ?IR.objectSet,
  ?OS2 = ?RAD.objectSet,
  ?OS1 <> ∅,
  ?OS2 <> ∅,
  ?OS3 = (?OS1 ∩ ?OS2),
  ?OS3 <> ∅
→
  (inform
    :receiver store_mgr
    :content ?OS3)
)
```

Fig. 2 Example notification rule

During the matching process the engine would interpret this rule as follows. By inspecting the first constraint it finds the free variable `?IR` and, in order to fulfill this constraint, it tries to find an arbitrary data instance of the specified type `InfraredSensor`. If there is such a data instance in the knowledge-base the free variable is instantiated by binding it to the corresponding data instance. Analogously, variable `?RAD` is

instantiated with a data instance of type `RadarSensor`. During inspection of the third constraint the free variable `?OS1` is found. Again the rule engine tries to instantiate it by binding values to it such that the constraint is fulfilled. In this case it simply binds `?OS1` to the values associated with the attribute `objectSet`. The subsequent constraint is handled identically. The next constraints make sure each sensor has sensed a non-empty set of objects and that their exist some objects sensed by both sensors. This is achieved by calculating the intersection of both object sets and "storing" the result in the free variable `?OS3`. In case `?OS3` is bound to a non-empty set of objects the rule fires and sends an `inform` message about these objects to the `store_mgr`. On the other hand, if any of these constraints cannot be fulfilled the rule will not fire and, hence, no message will be sent over the network.

2.3 A Distributed Rule-Based Blackboard

2.3.1 Motivation

Theoretically, a single central blackboard would be sufficient to implement the required agent coordination and problem-solving infrastructure. But besides the fact that a single blackboard might surely become a bottleneck in real-time applications IMA needs safety partitions as a physical bound regarding failure propagation for fault-tolerance reasons. If several subsystems had to rely on a single central blackboard then a failure of the blackboard component would inevitably propagate to all dependent subsystems, regardless of a concrete partitioning. However, if each partition owned a separate blackboard then even a complete partition could fail without big impact. Assumed a good partitioning other partitions may still work without failure albeit with reduced functionality and/or suboptimal performance.

Certainly, maintaining a separate blackboard for each partition may require the content of a blackboard (or at least parts of it) to be disseminated to blackboards contained in other safety partitions, because agents may depend on information provided by agents not residing locally. The following section describes a scheme for a distributed blackboard that disseminates data to dependent blackboards transparently to the agents.

2.3.2 Conceptual Details

Assume an arbitrary number of safety partitions together with their corresponding blackboards and agents. In order to maintain the advantage of a loose coupling of agents the distributed nature of the blackboard must be transparent. Otherwise, an agent would have to know all the partitions (or their blackboards respectively) that could contain information it might be interested in and issue corresponding subscriptions to all these blackboards. More trivial solutions would be to always register with all blackboards, or all agents could write their knowledge on all blackboards using an appropriate broadcast mechanism; but then we would loose all of the scalability and efficiency we expect to gain when we use blackboards together with rule-based notification. Ideally, an agent just tells its partition-local blackboard what information it is interested in, and under which circumstances it wants to be notified – regardless where the corresponding "source agent" is located; and the partition-local blackboard somehow makes sure this information is locally available every time a local agent would be interested in changes.

Fig. 3 pictures an exemplary situation where five blackboards are involved. On each blackboard A, B, and C an agent publishes a data instance 1, 2, and 3 respectively. Upon registration with blackboard D an agent informed the blackboard about its interest in data instances 1 and 2 in order to be able to calculate data instance 4, whereas another agent pointed out its interest in the data instances 1 and 3 for calculating data instance 5. As a consequence, blackboard D maintains local replicas of the data instances 1, 2, and 3. The agent on blackboard E wants to be informed about changes of the data instances 2 and 5. So E maintains replicas which stem from backboards B and D respectively. As can be seen in this example each blackboard gets the

replicas from the blackboards that contain the original data instances. For the rest of this paper we assume this always to be true.

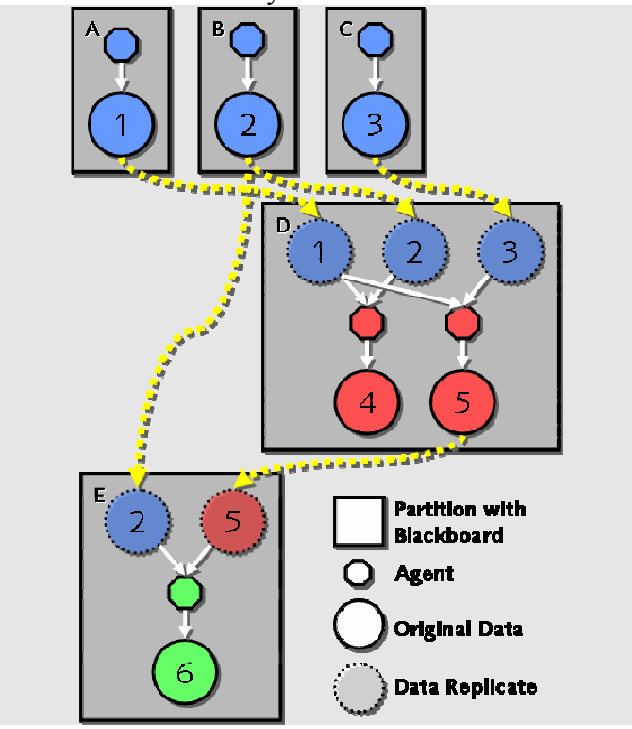

Fig. 3 Data exchange between blackboards

The fact that a blackboard has to make sure requested replicas from data instances of other blackboards are maintained locally implies that a blackboard knows where to get replicas from. For this purpose there exists a globally known list of mappings that allows a blackboard requiring a specific data instance copy to find the blackboard containing the original one. Of course, a replicate of this globally known mapping must be available for each partition because, otherwise, there would again be one central component all safety partitions had to rely on. For maintaining these replicas in a consistent way well known algorithms can be used. Note that such a mapping may also be determined ahead of time and remain unchanged during runtime if agents are not able to move from partition to partition. In this case the globally known mapping is not subject to changes, so a static copy of the mapping list for each partition suffices and no algorithm for maintaining consistency between these copies is needed.

2.3.3 Decomposing & Distributing Notification Rules to Minimize Communication Overhead

Suppose an agent registers with its local blackboard together with its specific set of notification rules. Each rule matches some data instances which must be locally available in order to make rule firing possible at all. However, it may be the case that some or all of these instances are not maintained locally. Even if all required replicas are maintained locally (because other agents already registered with the same local blackboard and initiated gathering replicas through their notification rules) then it is still not guaranteed that each data instance is updated as frequently as needed, because the other agents may need updates only in special situations. As a consequence, the blackboard must ensure not only the existence of local replicas but also appropriate update rates.

A rather trivial solution to this problem is for the blackboard to register with blackboards maintaining the original data instances and tell them that it wants to be notified with replicas anytime a change to one of them occurs. But this would again result in a loss of the advantages the rule-based notification approach could provide in order to minimize inter-blackboard communication overhead. Instead, we suggest to apply a rule splitting algorithm, which is able to generate a set of sub-rules from a given rule. More specifically, given a rule that matches (and hence depends) on a set of data instances and given a specific partitioning of this set, then the rule will be split into sub-rules, such that they each only depend on data instances of a specific partition.

For instance, suppose the `store_mgr` agent registers with its local blackboard `bb_C` and provides the notification rule shown in Fig. 2. Blackboard `bb_C` then in turn detects that the rule depends on data instances of type `InfraredSensor` which are provided on blackboard `bb_A`, and `RadarSensor` instances which are maintained on blackboard `bb_B`. The splitting algorithm would then generate the two sub-rules depicted in Fig. 4.

The first one will be sent to `bb_A`, the second one to `bb_B`, and the original rule in Fig.

2 will be added to the `store_mgr`'s local blackboard `bb_C`. As a consequence, the first two rules will fire every time the `objectSet` of a sensor instance changes and is not empty. Other changes of the sensors' attributes, however, will not trigger an update of the corresponding replicas on blackboard `bb_C`. Note that some conditions of the original rule depend on information stemming from both blackboards `bb_A` and `bb_B`. Thus, these conditions do not reflect in the sub-rules, but are contained only in the original rule.

```
(rule bb_A::store_mgr::report_threats
  ?IR : InfraredSensor,
  ?OS1 = ?IR.objectSet,
  ?OS1 <> ∅
→
  (inform
    :receiver bb_C
    :content ?IR)
)

(rule bb_B::store_mgr::report_threats
  ?RAD : RadarSensor,
  ?OS2 = ?RAD.objectSet,
  ?OS2 <> ∅
→
  (inform
    :receiver bb_C
    :content ?RAD)
)
```

Fig. 4 Sub-rules for blackboards bb_A, bb_B

As can be inferred by looking at this example such a rule splitting algorithm will eventually have to transform the antecedent of a given rule into an equivalent expression which can easily be split into several independent parts. In [1] such a decomposition algorithm is described. It is based on the manipulation of queries given as relational expressions. We believe a rule splitting algorithm suitable for our needs will likely be based on the same principles.

3 The FIPA Compliant Multi-Agent Architecture for IMA

In this section we describe our proposed FIPA compliant multi-agent architecture for IMA by way of a simplified exemplary MAS which is shown in Fig. 5. It is composed of two safety partitions each containing a Blackboard Agent, a separate GPS Agent in each partition, and a Mission Mgmt Agent in partition 2.

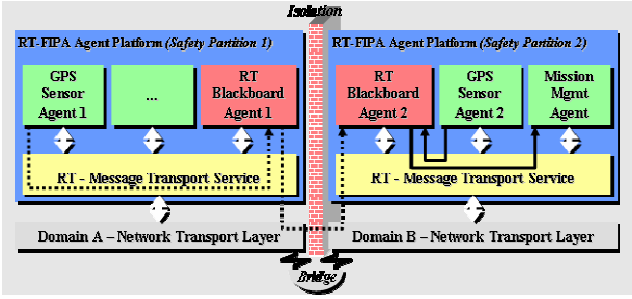

Fig. 5 Exemplary FIPA RT-MAS for IMA with two safety partitions

The Mission Mgmt Agent in partition 2 wants GPS data produced by GPS Agent 1 and 2 to be placed on its partition-local blackboard for fault-tolerance reasons. Hence, its registration message will contain notification rule(s) matching on sensor data provided by GPS Agent 1 and 2. Upon reception the blackboard system will analyze the rule set and, eventually, register itself with Blackboard Agent 1 in a transparent way. As a consequence, every time the GPS Agent 1 updates its partition-local blackboard the new information is sent to the blackboard in partition 2. The Mission Mgmt Agent will finally be notified about changes to both types of sensor data. In this example the Mission Mgmt Agent prefers GPS data provided by the GPS Agent 2 but in case of a malfunction of GPS Agent 2 it could still use the GPS data disseminated by the blackboard in partition 1.

Generally, a FIPA compliant MAS for IMA will implement the *FIPA Abstract Architecture Specification* as defined in [4] like every other FIPA compliant platform. However, it will likely provide additional services or ACL (agent communication language) extensions in order to be suited for real-time applications (see [5,6] as examples). Additionally, each safety partition running agents that implement IMA functionality will also host a blackboard like it is described in the previous section.

4 Conclusions

In this paper a flexible, and fault-tolerant multi-agent based FIPA compliant architecture for IMA has been proposed which utilizes a rule-based notification scheme together with a distributed blackboard to efficiently disseminate data between different safety partitions. Agents cooperate and coordinate their actions solely by exchanging information using their partition-local blackboards. During registration, agents provide notification rules specifying as exactly as possible when and what information they would like to be notified of in case of changes regarding the blackboard's content. The distributed blackboard mechanism transparently makes sure information required by an agent is supplied on its local blackboard. While in this manner all agents may have access to arbitrary parts of the global system state the proposed mechanism minimizes communication between partitions by splitting the agents notification rules into sub-rules and, accordingly, spreading them to blackboards contained in other safety partitions. Consequently, information is exchanged between blackboards only when required.

First test results obtained from our prototypical implementation of our idea of a blackboard with rule-based notification suggest this concept can indeed save a lot of communication overhead, especially in cases where agents must cooperate in order to solve a common problem. In such cases, agents depend on partial solutions provided by other agents and, thus, they often have to wait until they are available and possibly fulfill some criterions. Rules can be specified which fire not until all required partial solutions are written on the blackboard. This way, network resources are not wasted for unnecessary communication and, hence, are available for other information, which is likely to be exchanged more frequently. Sensor data, for instance, may be needed throughout the system which requires them to be disseminated to all partitions with high frequency.

4 Future Work

Before we are able to implement a demonstrator of our proposed architecture some issues have to be addressed first.

In section 2.3.3 we talked about a rule decomposition algorithm that is able to generate the sub-rules which can be disseminated to other partitions. We did not, however, provide an appropriate algorithm. This is due to the fact that we have to figure out first how powerful the expressivity for conditions within the antecedent of a (splittable) rule has to be at least/most, because on the one hand very complicated expressions may be difficult to split and, more important, they may have a drawback on computation time during the rule matching process. A significant slowdown is not feasible, however, because it would directly slow down information exchange. On the other hand, if expressiveness is too restrictive not much network traffic can be saved because rules cannot be specified as exactly as necessary which would result in more agent notifications than theoretically required.

Additionally, another very important issue has to be considered in advance. How to evaluate distributed rules correctly? After decomposition, the sub-rules are sent to the corresponding blackboards. Their task is to collect consistent local results and to forward them to their "superordinate" blackboard. In [1] it is stated that *"a distributed evaluation algorithm is correct if and only if, first, it reports all rules whose conditions have occurred, (i.e. whose conditions are true globally), and, second, it does not report rules whose conditions have not occurred."*. We have to further investigate if and how we can ensure this requirement in our special distributed blackboard context, because due to lack of a global clock and communication delay, consistent rule evaluation is not trivial. For example, Chakravarthy et al. identified four different strategies for the detection of *composite events*[1] – Recent, Chronicle,

[1] Note that (composite) events can be defined to occur when the condition of a rule matching on (several) data instances is fulfilled.

Continuous, and Cumulative – because they were unable to find a strategy suitable for all applications (see [11]). Moreover, it might be necessary to implement communication protocols for knowledge exchange between blackboards that preserve the casual order of messages (e.g. like those defined in [3]).

Finding answers to these questions should be subject of our future efforts. A lot of work has already been done in the fields dealing with "causal relationships" [7], "detection of composite events" [10], "detection of global predicates" [2,9], "active databases" [1,10], and "monitoring of distributed applications" [1]. These areas are very closely related to our problem domain. So finding answers should be merely a matter of finding appropriate existing theories and solutions and adapting them to our needs, if necessary.

References

[1] Ing-Miin Hsu, Mukesh Singhal and Ming T. Liu; *Distributed rule monitoring in active databases and its performance analysis*. Dep. of Computer and Information Science, The Ohio State University, 1992.

[2] R. Cooper, K. Marzullo; *Consistent detection of global predicates*. Proc. ACM/ONR Workshop on Parallel and Distributed Debugging, Santa Cruz, California, pp. 163-173, 1991

[3] R. José de Araújo Macêdo; *Casual order protocols for group communication*, Proc. of Brazilian Symp. nn Computer Networks (SBRC), Universidade Federal da Bahia, 1995

[4] *FIPA Abstract Architecture Specification*, http://www.fipa.org/specs/fipa00001/index.html

[5] Lekshmi S. Nair; *Extending ACL to support communication in a real-time multi-agent system*, University of Rhode Island, 2000

[6] SIMBA – multi-agent system based on ARTIS, http://www.dsic.upv.es/users/ia/sma/tools/simba/index.html

[7] R. Schwarz, F. Mattern; *Detecting causal relationships in distributed computations: in search of the holy grail*, Dep. of Computer Science, University of Kaiserslautern/Saarland, 1994

[8] M. D'Hondt; *Hybrid aspects for integrating rule-based knowledge and object-oriented functionality*. PhD thesis, Vrije Universiteit Brussel, 2004.

[9] Guy Dumais; *Detection of separable predicates on series-parallel systems*. Concordia University, Montréal, Québec, Canada, 1998

[10] S. Chakravarthy, V. Krishnaprasad, E. Anwar, S.-K. Kim; *Composite events for active databases: semantics, contexts and detection*. Proc. of the 20th VLDB Conference Santiago, Chile, 1994