Using Distributed Active Real-Time Database Functionality in Information-Fusion Infrastructures

Marcus Brohede
University of Skövde
P.O. Box 408
SE-54128 Skövde
Sweden
marcus.brohede@his.se

Sten F. Andler
University of Skövde
P.O. Box 408
SE-54128 Skövde
Sweden
sten.f.andler@his.se

***Abstract*— We present a list of requirements that need to be addressed by an infrastructure for information fusion where applications have real-time requirements. The requirements are grouped into configuration requirements, temporal requirements, and robustness requirements. We show how the functionality of a distributed active real-time database system (DARTDBS) can meet many of the given requirements, and therefore, argue that it is suitable for use in an information-fusion infrastructure with real-time requirements. The design of a particular DARTDBS, the DeeDS architecture and prototype, is presented as proof of concept. A comparison with some alternative infrastructures is briefly discussed. We describe a small distributed real-time simulation experiment using DeeDS as infrastructure, and discuss open questions such as how to deal with uncertainty management of information sources, recovery of information fusion nodes, and harmonizing data structures from different information sources.**

***Index Terms*— Information fusion infrastructure, distributed active real-time database system, distributed real-time simulation.**

I. Introduction

Information fusion is the process of combining information from many different sources into new information that can be used for improved decision support. In many information fusion applications, information sources are located at geographically dispersed sites. Hardware and software differences are natural, i.e., heterogeneity is expected. The total amount of information is overwhelming and must, therefore, be controlled to fit the individual information receivers. Any infrastructure supporting the information fusion process must be dynamic and adaptive, since information receivers or producers can be added or removed dynamically. The dynamic nature of the fusion process also means that individual information receivers can change how they want the fused information to be presented, and information sources such as sensors can change how they produce data. All these factors add to the overall complexity of the information fusion process.

The information fusion process (see Fig. 1) puts temporal requirements on the underlying infrastructure, especially if any part of the information fusion process require real-time guarantees. The information fusion process includes information producers and consumers. An information producer is a source that can provide historical data (past - from databases), current state information (present - from sensors), or predictions of future behavior (future - from model-based simulation), or a sub-process that combines a number of sources into a new, fused, source. Producers and consumers are distributed over one or more nodes in a network.

In this paper we list some of these requirements and show that the functionality of a distributed active real-time database system (DARTDBS) can meet these demands. In other words, a DARTDBS could serve as a suitable information-fusion infrastructure. Moreover, we argue that adding functionality from a DARTDBS to existing infrastructures can improve their use as information-fusion infrastructures.

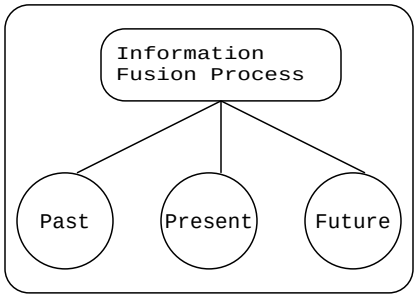

Fig. 1. Example of information fusion process

The paper is structured as follows. In section II we state basic requirements that any information-fusion infrastructure must meet, and extend with temporal requirements that are introduced if the fusion process must support real-time and embedded systems. Section III shows how the functionality of a DARTDBS could meet the requirements presented in the previous section. Section IV describes how one particular DARTDBS, called DeeDS, can be used as an information-fusion infrastructure. A sample application of DeeDS as a distributed simulation infrastructure is presented in section V. Section VI compares our proposed information-fusion infrastructure based on DeeDS to other potential approaches. Finally, section VII summarizes our contributions and details potential future work.

II. Information-fusion infrastructure requirements

Three different categories of requirements are presented: configuration, temporal, and robustness requirements. In addition to these requirements, there may be other groups of requirements or even requirements within the listed groups that are not addressed in this paper.

A. Configuration requirements

Configuration requirements involve the structure and connectivity of the system.

a) Heterogeneity: An information-fusion infrastructure must be capable of handling differing platforms. Given the many independent producers of information it is highly unlikely that they will all use the same hardware, software, data structures, database schema, standards, degrees of uncertainty, etc. Therefore, a useful information-fusion infrastructure must address heterogeneity.

b) Distribution: An information-fusion infrastructure must be capable of handling geographically dispersed information sources, i.e., distribution is inherent in the concept of information fusion.

c) Independence: There must be decoupling of producers and consumers of information in an information fusion process. This means that producers of information should not be required to know anything about the receiver of the information. Conversely, consumers should only need to specify which information they are interested in without specifying where this information exists. However, an information-fusion infrastructure must be capable of handling complex correlations between information sources. For example, a consumer could be interested in some information that is only available by combining input from a number of information sources. The independence requirement is vital since data sources may exist outside organizational boundaries.

d) Scalability: The potentially large number of nodes and amount of information that may be processed in an information fusion application results in a need for a scalable information-fusion infrastructure. Scalability in the number of sensors as well as nodes in the fusion process must be addressed. Often, a linear or O(n log n) behavior is expected in order to allow modular growth of the system in response to growing needs.

e) Adaptivity: An information-fusion infrastructure must adapt to changes in the node structure or how various nodes interact. For example, information producers or consumers can emerge or disappear. Structure changes cannot always be controlled, since information sources can exist outside the local organization. Changes to individual nodes in the fusion process must also be addressed, e.g., a node producing sensor data could start producing the sensor data in a different format. In addition, new information sources can have different data structures describing the same real world entity as some already known information source. In such a case it is desirable to detect this and benefit from this. Finally, consumers of fused data might change the form in which they want the information delivered.

B. Temporal requirements

For systems that store and process data as it varies over time, temporal properties of the data must be represented. Requirements that are added due to time and dependability constraints in real-time and embedded systems include resource predictability and timeliness.

f) Temporal attributes: The information fusion process brings together not only data from present data sources (sensors), but may also try to use historical data (from databases), as well as future predictions (from model-based simulations), as input to the fusion of an improved value. Therefore, an information-fusion infrastructure needs to be able to represent time and efficiently store and retrieve time series of historical data, as well as series of future predictions.

g) Resource predictability: Real-time systems must perform their designated tasks using predictable amounts of resources, e.g., processor time, memory space, communication bandwidth, etc. For an information fusion process, this means that nodes producing, consuming, or fusing information must have predictable resource usage. In other words, usage of resources such as processor time, memory space, or communication bandwidth must be bounded. One problem that arises in information fusion particularly, is that the process is often not entirely controlled by one organization. This means that the design of the information fusion application must allow functions with real-time requirements to rely on controlled nodes within organizational boundaries.

h) Timeliness: All parts of the information-fusion infrastructure that any real-time application relies on must be timely. A timely system is scheduled such that it meets all its deadlines. This requires resource predictability and sufficient efficiency. Information sources that cannot be controlled (e.g., lying outside the local organization) must not be critical to the information fusion process, but rather be seen as resources that can improve or optimize the value of the fused information.

C. Robustness requirements

These requirements affect the ability of the information fusion system system to deliver useful information to clients in the face of faulty or inaccurate information sources.

i) Fault tolerance: If the information fusion process is used by applications that affect the real world, it must be made fault tolerant. Fault tolerance increases dependability in a system, for example, by replicating critical resources.

j) Uncertainty management: Uncertainty management of information sources introduces two different problems. First, if a sensor of limited precision is known to have an uncertainty of ϵ, then a received value x is actually only known to be within $x \pm \epsilon$. Second, there may be a degree to which an information source can be trusted at all, if there are arbitrarily failures in the source's information or if the source cannot be trusted for some other reason.

III. Using a DARTDBS as an information-fusion infrastructure

We believe that all of the requirements listed in section II must be addressed by any information-fusion infrastructure

with real-time requirements. In this section we show that a DARTDBS can be implemented to meet all of the listed requirements and therefore can be used as an information-fusion infrastructure. However, existing infrastructures could incorporate some of the functionality described below to improve their ability to meet the requirements of a real-time information-fusion infrastructure.

A. Configuration: Heterogeneity

A DARTDBS could be implemented such that heterogeneity is handled. For example, differences in data types could be addressed by transforming everything stored in the database to a globally defined data structure. This conversion, which should be a part of the information fusion process, should be done without human intervention. One way to make this process automatic is to use active functionality. In addition, the DARTDBS must either be implemented on top of a platform independent middleware (e.g., Java VM) or be ported to all the platforms used by nodes in the particular information fusion system.

B. Configuration: Distribution

In addition to distributed processing, distribution of node in the information fusion process provides processing close to information sources. Something that is useful in real-time systems, since for example communication latencies are minimized. Since a DARTDBS is inherently distributed, it could easily be configured to have one node at each information source, if needed. For example, sensor data can be collected, trimmed and converted to a globally defined structure before it is sent to other parts in the fusion process.

In a distributed architecture it is also possible to use replicas to increases fault tolerance. For example, by using semi-active replication such as Leader-Follower [2], data from important information sources can be made fault tolerant. Fault tolerance is particularly interesting for information fusion applications where the fusion process itself must not fail, i.e., the use of fused data is critical. For example, a decision support system may be required to deliver proposals in time for some human (or computer) operator to make a well-founded decision. By failing to meet the deadline, a severe penalty may be suffered (financial or human loss).

C. Configuration: Independence

To achieve independence between producers and consumers of information there is a need for an intermediate part in the fusion process. A database provides a robust way to store information and decouples readers and writers, i.e., by storing and retrieving information in a database producers and consumers are kept independent of each other.

D. Configuration: Scalability

Processing of information can be done at any node in a DARTDBS. First, information can be processed directly on the node where data enters the information fusion process, e.g., where sensor data is collection. Second, filtering of data at consumer nodes is also possible, for example, receive only one third of the sensor updates for a specific sensor. Lastly, reformatting or merging of information can be done by the producer, consumer, or an intermediate node, e.g., combine two independent sensors to get an improved precision on a real world object. Reformatting of data allows information to fit a predefined structure suitable for a stakeholder. For example, displaying sensor readings on a hand held device can be done differently compared to a large desktop display. Merging information means that data from information sources are brought together to form new, fused information with improved value as compared to using the values separately. For example, consider bringing together a radar view, camera view, and infrared camera view into a fused view suitable for a command center.

Finally, a DARTDBS represents a design that can be made to scale if care is take in the design. By adding more processing power, i.e., nodes, when adding producers or consumers the architecture can exploit parallelism in the information fusion process. Also, additional intermediate nodes for merge processing can be added to off-load heavy processing nodes. For example, if node a produced the above mentioned command center view and also collected a number of sensor readings this task could be shared between the two nodes a and b to reduce a's load. However, design decisions such how to store data, i.e., whether the database should be partitioned, partial, or fully replicated must be decided.

E. Configuration: Adaptivity

Active functionality, which is incorporated in a DARTDBS, is essential in our proposed information-fusion infrastructure. Based on active rules or triggers in the database this is what makes automatic processing at the different database nodes (producer, consumer, intermediate) possible. Filtering is one type of processing suitable for triggers. For example, allow position updates from cars to enter the database every 400 ms, even if there exists more frequent updates from the sensors. Another example on active behavior is to deliver an alarm if a value in the database exceeds some threshold. In addition, the system can through the triggers react to more complex event combinations such as sequences of events or non-occurrences of events. Finally, triggers can be constructed such that they react on changes from both information sources and processes that use the fused information.

F. Temporal: Temporal attributes

Temporal attributes can be supported directly by a DARTDBS by allowing time series to be represented in the database schema. Predictions on future behavior, e.g., event occurrences, or trends, can be included by allowing information sources to be simulators. In this way historical data can be retrieved from the database, as well as current data from sensors, along with predictions generated from simulations.

G. Temporal: Resource predictability

To support resource predictability, a DARTDBS must define its functionality in such a way that it allows resource-

predictability implementation of a defined subset of the functions. This must be true for individual database functions (store/retrieve objects, begin/end transactions) as well as for the active functionality (event composition, ECA rule evaluation), distribution (replication, conflict detection/resolution), and scheduling (admission control, overload resolution). These properties are part of the DeeDS architecture described below.

H. Temporal: Timeliness

To support timeliness, A DARTDBS must support predictability and be sufficiently efficient to be able to meet its specified deadlines. Ways to achieve this are main-memory residence of critical data, storing replicas of data where it is needed, and allowing independent update of data. Again, these are properties that are included in the DeeDS architecture described below.

I. Robustness: Fault tolerance

In a DARTDBS, fault tolerance can be supported by redundancy in the form of replicated data and processes, with voting or primary-backup. Requests and corresponding results are stored in the database, allowing for continued operation in case a node is lost. The active functionality of the DARTDBS can be used to monitor that state of replicas and trigger recovery.

J. Robustness: Uncertainty management

Uncertainty management is beyond the scope of the current paper, but is the subject of much research in information fusion. Active rules could be used to monitor the uncertainty in critical data and trigger action to raise an alert or take action to restore precision to a given level.

IV. The DeeDS DARTDBS

The DeeDS database [3] is an in-house developed DARTDBS prototype. In this section we discuss how DeeDS can address the requirements described in section II and how particular issues listed in section III such as data storage model and how replication is addressed can be solved.

The heterogeneity requirement is partly met by DeeDS through a generic operating systems interface called DeeDS operating systems interface (DOI) that provides a simple way to port DeeDS to different platforms, combined with a simple marshalling and unmarshalling functionality for adapting replicated data to the different platforms. A more general way to support heterogeneity is to replace the communication part of DOI with real-time CORBA (e.g., the TAO [4] implementation). Another alternative could be to use real-time Java and remote method invocation (RMI).

The need for distribution and independence in information fusion processes are met by the DeeDS database. DeeDS is a distributed database and as such it can have database nodes at distributed locations; information sources can have database nodes locally. Moreover, DeeDS advocates a whiteboard architecture where all communication between applications is performed through the database operations. Bounded-time replication [5] of updates in the database make ensures that real-time requirement are not compromised by the replication protocol.

If all data is not available locally it may be required that other nodes are queried in order to complete a transaction. This could be a problem if the network is non-real-time or unreliable, since this endangers predicability of the database, i.e., real-time guarantees cannot be given. In addition, access times to data in main memory differs in orders of magnitude compared to accesses to secondary storage (i.e., disk). Since worst case access must be considered when designing a real-time database using disk storage as in traditional databases would lead to overly pessimistic worst-case execution times [6]. Therefore, DeeDS store critical real-time data in main memory. However, since the amount of data in typical information fusion applications is huge it is not practical to have all data in main memory. By segmenting the database, critical data can be stored in segments available in main memory while other segments can be directed to secondary storage such as disk.

The transaction model in DeeDS assumes full replication; all data is replicated to all nodes. However, a database that is fully replicated does not scale [7]. To resolve this, scalability in terms of memory usage in DeeDS is achieved by the use of *virtual full replication* through segmentation [8]. Virtual full replication means that every data object that applications on a certain node require is guaranteed to be accessible locally on that node. In essence, no distributed queries to other nodes are required and therefore it appears to the applications as if the nodes is fully replicated.

DeeDS exploits virtual full replication and local commit to avoid unpredictable delays during network partitions. This guarantees predictability when reading and writing to the database. By making sure that all data objects are replicated a sufficient level of fault tolerance can be achieved, with a lower memory consumption compared to a fully replicated database. Even though critical data resides in main memory, a diskless distributed recovery algorithm [9] makes recovery possible.

Since all transactions commit locally and results are eventually propagated and integrated on all other replicas, the global database state in DeeDS is said to be eventually consistent [10]. That is, given that no more transactions enter the system, the database will eventually converge to a globally consistent state. The relaxation of consistency potentially introduces conflicts. These conflicts must be i) detected and ii) resolved. DeeDS makes use of a modified variant of version vectors and logfilters to detect conflicts. Conflict resolution is application dependent and can, for example, be to always take the latest value, highes value, or the value originating from prioritized node.

Adaptivity is achieved by using active behavior in the form of event-condition-action (ECA) rules in the database. The generation of both simple and complex events, as well as the actual monitoring of events can be done while consuming a predictable amount of resources [11].

Previous work has shown that the DeeDS prototype is suitable for use as infrastructure for distributed simulations [**?**]. An example of the use of DeeDS in a distributed simulation scenario, with real-time fusion of information produced

and consumed by simulators and visualizers, is presented in Section V.

To summarize, we have shown that a DARTDBS, DeeDS, can serve as part of an information-fusion infrastructure with real-time requirements. The main design goal for the DeeDS architecture is to remove potential sources of unpredictability. This includes disk access, network communication, and distributed transactions. In addition to this, functionality such as active behavior through ECA rules and virtual full replication are essential elements of the architecture.

V. A REAL-TIME SIMULATION SCENARIO

Within the WITAS UAV project [12], the primary goal is to build an autonomous aerial vehicle (UAV) that perform tasks according to high-level commands. In a scenario with multiple UAVs along with multiple operators, centralized coordination and other information producers and consumers, there is a need for information fusion. The applicability for fusion processes are many. For example, consider a coordination central responsible for deploying UAVs for surveying disaster areas (areas where it is too risky for humans to venture). The information gathered from the various UAVs has to be merged into something that is graspable for a human operator at the command center. Moreover, the various UAVs could carry different types of sensors, e.g., camera, infra-red camera, or radar, thus adding to the need for information fusion.

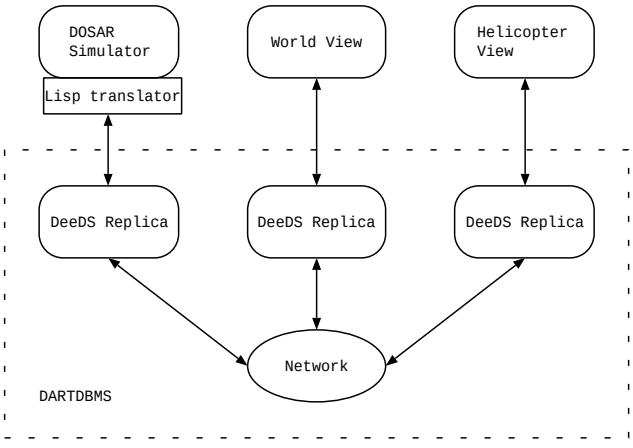

Fig. 2. Small proof-of-concept implementation

We set up a number of nodes in a small experiment as can be seen in Fig. 2. The main purpose with this experiment was to show that using an active database as communication channel does not impose any substantial overhead compared to peer-to-peer communication. In this small scenario a simulator, called DOSAR, generates information about one unmanned aerial vehicle (UAV) and a number of cars. The UAV is given orders such as “fly to the red brick building” through computer evaluated voice commands. The UAV then evaluates this request and based on artificial intelligence tries to carry out the task. The simulation state is visualized by two different consumer applications, World View and Helicopter View. The data sent to the Helicopter View is filtered so that no changes in the simulated world occurring outside the helicopter’s camera view is delivered. The World View receives all updates in the simulation. The tests carried out with the database as distributed simulation infrastructure was part of a demonstration of the WITAS project 2003. Updates in the simulator were transmitted every 400 ms and there were no indications that the communication through the database was a bottleneck.

Further development of the prototype will include multiple simulators where data is merged into one or many visualizers. This will be a more interesting test, since the real value of our architecture manifests itself when there are multiple producers as well as consumers of information. By making them independent of each other, developers do not have to setup the different communication channels themselves; communication and storage is implicit in the architecture. The proof of concept implementation has shown, so far, that our proposed architecture works. However, for example scalability has not yet been tested.

VI. RELATED WORK

Our DARTDBS approach differs from other potential information-fusion infrastructure such as HLA, CORBA, and Java RMI. For example, High Level Architecture (HLA) [13] could provide an information-fusion infrastructure supporting simulations. HLA is capable of handling both distribution and heterogeneity. However, HLA does not provide a fault tolerant architecture [14] making it less suitable when the fusion process must be reliable. In addition, HLA uses a centralized approach built around a central unit called the run-time infrastructure (RTI), which is a potential bottleneck.

Other infrastructures such as Common Object Request Broker Architecture (CORBA), or JAVA have real-time variants (Real-Time CORBA, and Real-Time Java) and they also handle distribution and heterogeneity, but the do not provide a clear way to keep producers and consumers independent of each other.

VII. CONCLUSION

The major contributions in this paper are:

- We give a summary of requirements that must be addressed by information-fusion infrastructure serving applications with real-time requirements.
- We show how the functionality of a DARTDBS can meet these requirements.
- We present a proof-of-concept design using a specific DARTDBS called DeeDS.

Information fusion puts requirements such as heterogeneity, distribution, independence between producers and consumers, scalability, uncertainty management, etc., on the infrastructure. Therefore, any suitable information-fusion infrastructure must address these requirements to allow for successful information fusion. In addition, if the information fusion process is used in embedded and real-time systems, requirements such as predictability, timeliness, and fault tolerance are added.

It has been argued that an information-fusion infrastructure based on the functionality of a DARTDBS can be made fault tolerant. Furthermore, such an infrastructure provides a robust

way to store data (both fused and raw), is scalable, supports real-time requirements, and provides a foundation for recovery.

A. Future work

When using the DeeDS database architecture as part of an information-fusion infrastructure there are open questions that need to be investigated. This includes, for example, recovery of nodes in the fusion process, and harmonizing data structures with different absolute and relative temporal validity.

1) Recovery of information sources: Recovery of information sources is important when the information fusion process must be fault tolerant, for example, when feeding an decision support system with suggestions on actions. If there is substantial loss of value in the application using the information fusion system due to a crash in some part of the information fusion process then the information fusion process must have predictable recovery time. The recovery method depends on whether the part of the information fusion process is managed by the local organization or if it is external. If the part is external, one way to increase fault tolerance is to have multiple nodes doing the retrieval of data from the producing node. In this way a failure on on one of these retrieving processes can be handled. However, failure on the actual information producer cannot be handled in this case, i.e., the information fusion process must be design so that it does not rely sole on this producer. If, on the other hand, control over the information producer exists. Multiple instances (replicas) of both the producer of information and connections to the information-fusion infrastructure can be made redundant to achieve an increased fault tolerance. However, more work needs to be done in this area. For example, we need to find guarantees that any part required in the information fusion process is recovered in bounded time.

2) Harmonizing different data structures: The issue of harmonizing different data structures needs to be further investigated. Research on this issue has been conducted for a long time in the distributed-database community, for example on whether global schemas should be used. However, when faced with real-time requirements, new and interesting problems arise. Not only must data be polished and objects with different attribute names for the same attribute merged, but temporal validity of data must be considered. For example, one information source could update its sensor values at twice the rate compared to another information source. Even though they produce information based on the same real world entity the information fusion process must consider the time and validity of the data. As future work it would be interesting to see if there are ways to incorporate this type of information into the database schema to help the information fusion process.

REFERENCES

[1] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[2] Powell David, editor. *DELTA-4 - A Generic Architecture for Dependable Distributed Computing*. Springer Verlag, 1991.

[3] Sten F. Andler, Jörgen Hansson, Joakim Eriksson, Jonas Mellin, Mikael Berndtsson, and Bengt Eftring. DeeDS towards a distributed and active real-time database system. *ACM SIGMOD Record*, 25(1):38–40, March 1996.

[4] Douglas C. Schmidt, David L. Levine, and Sumedh Mungee. The Design of the TAO Real-Time Object Request Broker. *Computer Communications, Elsivier Science*, 1(4), April 1999.

[5] Johan Lundström. A conflict detection and resolution mechanism for bounded-delay replication. Master's thesis, University of Skövde, September 1997.

[6] John A. Stankovic, Sang H. Son, and Jörgen Hansson. Misconceptions about real-time databases. *IEEE Computer*, June 1999.

[7] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. *SIGMOD Record*, 25(2):173–182, June 1996.

[8] Gunnar Mathiason and Sten F.Andler. Virtual full replication: Achieving scalability in distributed real-time main-memory systems. In *Proc. of the Work-in-Progress Session of the 15th Euromicro Conf. on Real-Time Systems*, pages 33–36, Porto, Portugal, July 2003.

[9] Sten F. Andler, Egir Örn Leifsson, and Jonas Mellin. Diskless real-time database recovery in distributed systems. In *Work in Progress at Real-Time Systems Symposium (RTSS'99)*, 1999.

[10] Sanny Gustavsson and Sten F. Andler. Continuous consistency management in distributed real-time databases with multiple writers of replicated data. In *Proc. of the 13th Int'l Workshop on Parallel and Distributed Real-Time Systems 2005 (WPDRTS'05)*, Denver, CO, USA, April 2005.

[11] Jonas Mellin and Sten F. Andler. A formalized schema for event composition. In *8th Int'l Conf on Real-Time Computing Systems and Applications (RTCSA 2002)*, pages 201–210, Tokyo, Japan, March 2002.

[12] Patrick Doherty, Gösta Granlund, Krzysztof Kuchcinski, Erik Sandewall, K. Nordberg, Erik Skarman, and Johan Wiklund. The WITAS Unmanned Aerial Vehicle Project. In W. Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-00)*, pages 747–755. IOS Press, 2000.

[13] Judith S. Dahmann, Richard M. Fujimoto, and Richard M. Weatherly. The Department of Defense High Level Architecture. In *Proceedings of 1997 Winter Simulation Conference*, pages 142–149, 1997.

[14] Johannes Lüthi and Clemens Berchtold. Concepts for Dependable Distributed Discrete Event Simulation. In R. Van Landeghem, editor, *Proc. of the Int'l European Simulation Multi-Conference*, pages 59–66, 2000.