To appear in *Proceedings of the International Conference on Dependable Systems and Networks (DSN-2002),* IEEE/IFIP, June 23-26, 2002, Washington, DC.

Mr. Fusion[1]: A Programmable Data Fusion Middleware Subsystem with a Tunable Statistical Profiling Service

Andy Franz, Radek Mista, David Bakken, Curtis Dyreson, Murali Medidi[2]
School of Electrical Engineering and Computer Science
Washington State University
PO Box 642752
Pullman, WA 99164-2752 USA
Email: {afranz,rmista,bakken,cdyreson,mmedidi}@eecs.wsu.edu

Abstract

Voting is the process of combining multiple replies from replicated servers into a single reply. Data fusion is similar to but more general than voting. In data fusion, the input sources are not necessarily replicated servers, hence the inputs exhibit greater variance. Data fusion is a fundamental building block in distributed systems. It occurs in diverse contexts such as consensus, sensor networks, intrusion detection, and hierarchical resource monitoring, among others. This paper describes Mr. Fusion, a framework that provides data fusion in middleware. The heart of Mr. Fusion is a Fusion Core module that provides mechanisms for programming a wide variety of data fusion algorithms. Another part is a Fusion Status Service that monitors low-level outputs from the Fusion Core and alerts subscribers to divergent values or timings. The implementation borrows techniques from data warehousing and data mining.

1 Introduction

Voting is the process of choosing one output value from many input values, each sent by a different replica of a component such as a service implemented in software or a hardware-based sensor. Data fusion [8,16] is more general than voting in two key ways. In data fusion the values are not expected to be identical and there are not necessarily a well-defined number of replies (as with the case of a replicated server group, for example). Examples of data fusion include collating intrusion detection alerts [2], distributed sensor networks [10], ad hoc mobile network protocols (which typically aggregate many values into one, typically to save battery power on expensive transmissions), parallel neural nets trained differently to estimate the power grid safety margin [5], and hierarchical resource monitoring [16].

Middleware is a layer of software below the application but above the operating system that offers high-level programming abstractions across a network [1]. It helps mask the heterogeneity inherent in a distributed system and also helps programmers to be more productive, and helps achieve high-level interoperability.

In this paper we describe Mr. Fusion, a middleware framework supporting data fusion. The remainder of this paper is organized as follows. Section 2 describes and example application. Section 3 provides an overview of Mr. Fusion. Section 4 describes the details of the Fusion Core, which is a programmable mechanism for data fusion. Section 5 overviews the Fusion Status Service.

2 Example Application

One example application that can use Mr. Fusion is fault-tolerant middleware providing replicated servers such as with AQuA [6], Eternal [14], or ITDOS [13]. We note that a distributed sensor or other more general data fusion example would also be possible, but for brevity we use an example more familiar to DSN readers (and in fact found elsewhere in these proceedings [13]). In the replicated server scenario, each server independently executes an identical operation. The results, called *ballots*, should also be identical unless there is a value failure or inexact voting is required. The role of Mr. Fusion is to fuse the ballots into a single result and to monitor the behavior of the replicated servers to detect anomalies, such as a security violation. Mr. Fusion is configured with a set of *policies* (voting algorithms) and logic to decide which policy to use. A policy is a simple specification of how to fuse a set of ballots (e.g., a policy might be to exclude "late" ballots and choose the median). New policies and policy choice logic can be loaded into Mr. Fusion at any time and existing ones modified.

[1] Mr. Fusion is a device from Robert Zemeckis' 1985 movie, *Back to the Future*, and its sequels. It takes garbage in and produces energy for the time-travelling Delorean car of Doc Brown, the genial but mad scientist.

[2] M. Medidi is joining the WSU faculty in May 2002. His present affiliation is Department of Computer Science, Northern Arizona University.

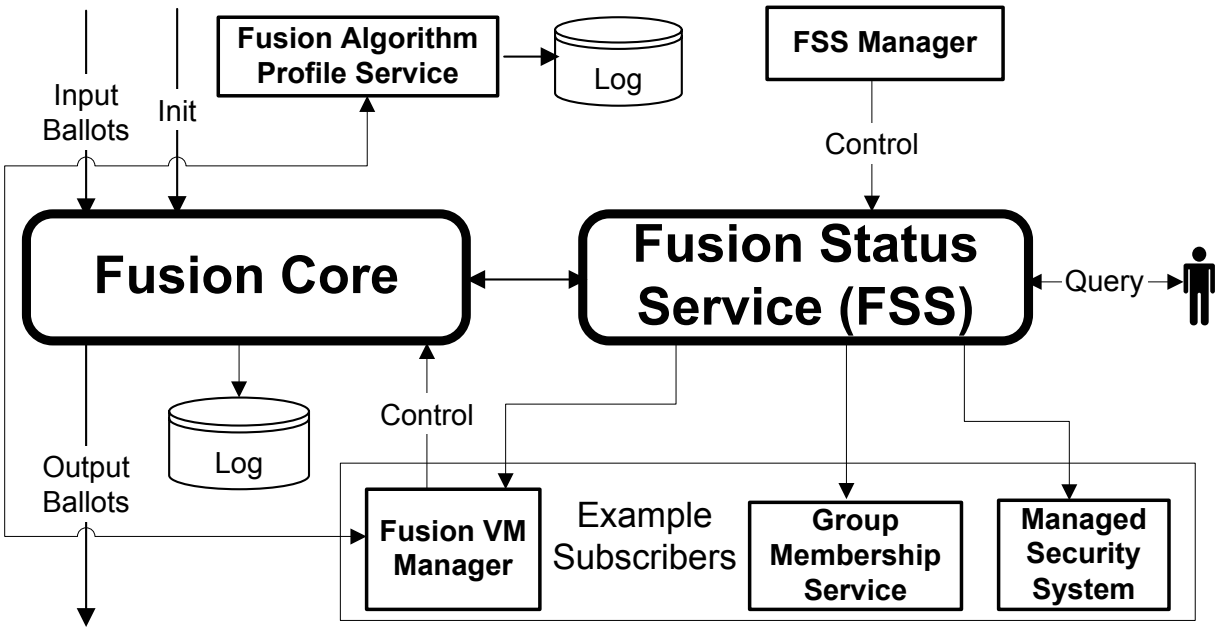

Figure 1. Mr. Fusion system architecture

3 System Architecture

The system architecture of Mr. Fusion is shown in Figure 1. It consists of two main subsystems, the Fusion Virtual Machine (FVM), which consists of the modules on the left half of the figure, and the Fusion Status Service (FSS), shown on the right half. Each subsystem consists of a main component and number of supporting components. CORBA is used for communication between the major components.

The FVM fuses application-level data. Its main component is the Fusion Core. The input to the Fusion Core is a set of ballot*s*. In the replicated server scenario, each replica produces a ballot. The core evaluates a policy given by the Fusion VM Manager. Eventually it either creates an output ballot or throws an exception. The creating of one output ballot or exception from a set of inputs is called a *fusion session*, the start of which is indicated by the *init* arrow. In the replicated server scenario, the fusion session is called a *vote* and is initialized by the client's request arriving; other more general applications can provide their own start and stop to fusion sessions by this initialization mechanism. Finally, the core also outputs information about each fusion session to the Fusion Algorithm Profile Service, indicating the success or failure of different policies used in that session (ones that were tried but could not run to completion due to the data). This can be used offline by the Fusion VM Manager to calibrate its policies; it is not discussed further for brevity.

The Fusion Status Service (FSS) is passed low-level information about value and timing errors for each fusion session by the Fusion Core. The FSS catalogs and maintains a database of this information and aggregates the data into higher granularities in spatial and temporal dimensions. The database can be accessed using either the Subscriber API or the Query API. The Subscriber API allows users to specify a set of conditions that trigger a callback or other action (three examples are presented below). Using the Query API, users can retrieve current conditions of the system using interactive queries. The FSS is controlled and tuned by the FSS Manager.

Three example subscribers that can benefit from the FSS follow. The first is a group membership service, which is used as part of the replicated server example to deliver client requests to the server replicas. Typically, a group can only expel a member when no other member (or too few of them) has received a message from it within a specified timeout period. The period should be adjusted to avoid expelling live members too often. However, if a single member of a group is suffering from performance problems, but is not yet past the threshold, then the throughput of a virtually synchronous multicast system can degrade dramatically [3]. The FSS extends support to enable a group to expel a member that has been "too slow for too long" (by various flexible definitions of this predicate) but has not yet timed out at the other members. The second example subscriber is a managed

security system that takes input from intrusion detection systems, virus checkers, operating system logs, etc., to detect and react to intrusions. Such a system may regard either value or timing errors as potential indicators of attacks or successful intrusions. Similarly, the group membership service for our replicated server example could be configured to expel a member that is returning values that are "too bad for too long", with flexible definitions of "too bad" and "too long" outlined below. A third example is the Fusion Manager, which can use the information on potential *value errors*, "significant" deviation from the values in other ballots, in assigning and adjusting the weights given to each ballot when using weighted fusion (which is similar to weighted voting).

The Fusion Manager adjusts policies for the core based on input from the FAPS, the FSS, and the core. It presently employs very simple heuristics, but we are working towards quantifying and implementing more complicated constraints such as choosing the set of policies based on application-level tradeoffs between precision and fault tolerance and performance; generally, waiting for more ballots before deciding increases fault tolerance but also increases the latency (time to output). A preliminary report is available [15].

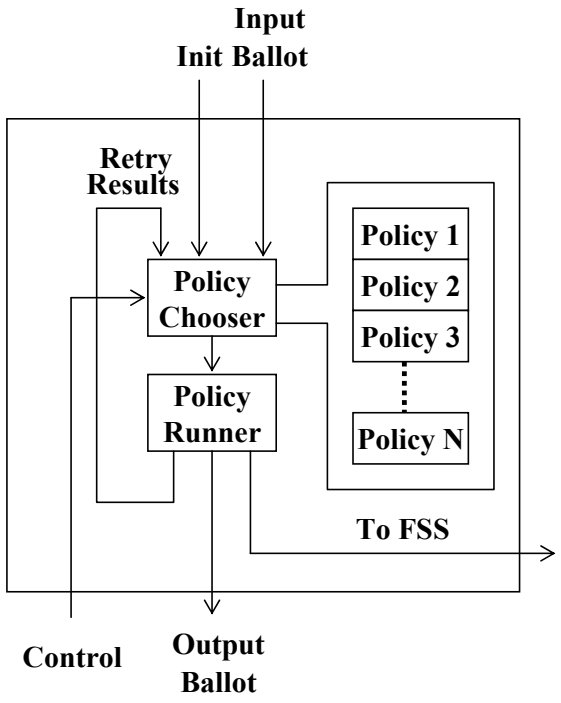

Figure 2. Fusion Core overview

4 Fusion Core

The Fusion Core is at the heart of the Fusion Virtual Machine. It is an extended, more general successor to the Voter Core in the Voting Virtual Machine (VVM) [4]. The Voting Definition Language (VDL) of the VVM has been generalized, and the **`goto`** construct for branching has been removed due to the complexity it added to the VVM's Voter Core. The core is depicted in Figure 2. We now overview its chief components and their interactions.

4.1 Specifying Fusion Policies

The policies in the Fusion Core are specified in three parts. The first two parts are called *policy wrappers*, which specify the behavior at a given level. There is an *inner wrapper* and an *outer wrapper*. The inner wrapper wraps a single fusion policy. The outer wrapper wraps both the *inner wrapper* and a number of fusion policies. The function of the outer wrapper is to choose a single fusion policy to evaluate. The third part of the structure is the singleton fusion policy itself. It has constructs that are slight generalizations of VDL's **`exclusion`** and **`collation`** states, which discard some ballots and then create the output ballot, respectively. Some examples of exclusion primitives are to exclude: furthest from mean, highest n%, and all but distance k from mean. Some examples of collation are: mean, majority, and median. Like the VDL, it also supports **`confidence`** values to indicate how good the output is believed to be.

4.2 The Wrappers

The inner wrapper controls when a particular fusion algorithm can be executed, namely when enough ballots have arrived to make the policy eligible to be used. The inner wrapper uses a table-driven approach where values are provided for percentages, quantities, time, and timeout values. These different fields help determine when appropriate number of ballots have arrived and when timeouts should be called.

The outer wrapper is used to describe which policies are eligible to be used, and in what order. It also employs a table-driven approach similar to the inner wrapper. It holds more general fields than the inner wrapper so that inner wrapper values can be bypassed when necessary. The wrapper's fields let it choose and skip appropriate policies and also can be programmed to defer using a given algorithm later when conditions are more appropriate for it.

4.3 The Policy Chooser

The role of the Policy Chooser is to take the information on the start of a fusion session and the current policy information and choose a policy to attempt to run. Once a policy is chosen, control is handed to the Policy Runner.

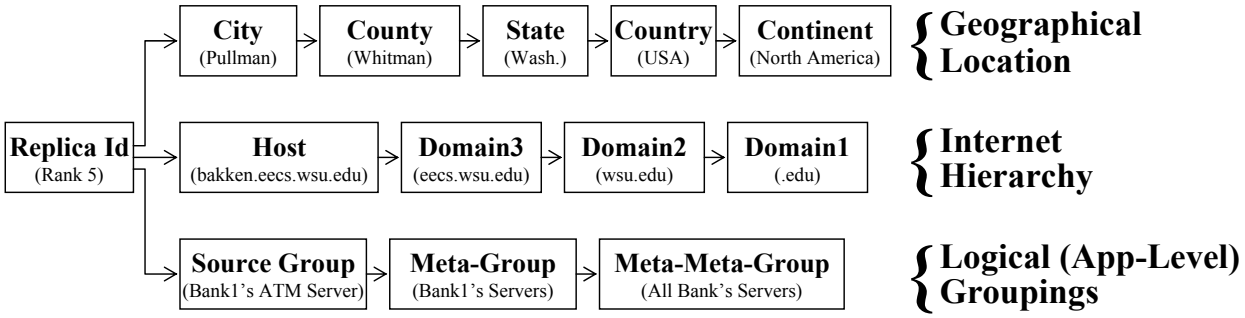

Figure 3. FSS spatial categories example

A policy is chosen by the value or values that are stored inside the policy wrappers type field. Policies may be enabled or disabled (included or excluded from being eligible for execution) based on their outer wrappers. When a policy disabled it cannot be activated again until the fusion session has been completed. The policies are also in a specific order that has been predetermined by the Fusion Manager. The Chooser simply chooses a policy based what the policy parameters are, on its order, and if it is currently able to run. If all the policies are turned off, then the Chooser drops to a default policy that is then run. This policy always has a result to return.

4.4 Multidimensional Data Fusion

Mr. Fusion supports multidimensional data fusion, operating on more than one parameter for a given ballot. For example, a CORBA method with a return value and two **`out`** parameters would have three values in each ballot that could be voted on, as would a sensor providing three distinct values.

Mr. Fusion supports multidimensional data fusion in the following four ways, listed most restrictive first.

1. The output ballot must be one of the input ballots, which in turn must be "equal" to a given threshold of the total ballots (typically over two-thirds).
2. The output ballot is the one of the input ballots that received the best "score" by a per-parameter ranking of the ballots.
3. The output ballot contains values from input ballots, but the parameters are chosen as "best" from each parameter and thus the output ballot may not be one of the input ballots.
4. The output ballot has no restrictions on it, for example its values may not be found in any input ballot but rather may computed by an operation such as **`mean`**.

The first two ways are useful in situations where the output ballot must be one of the inputs, which is true for example for most uses of actively replicated servers (it is dangerous to assume otherwise unless it is established that the semantics of the particular application do not require it). The first way is further restrictive in situations where Byzantine fault tolerance is required. It is the strategy for example hardcoded in [13]. The third and fourth ways relax the restrictions of the first two. All of these 4 strategies use a generalized and flexible technique to rank the parameters based on either value or time, or both, and a number of variations of each strategy is possible.

5 Fusion Status Service

The Fusion Status Service (FSS) is a performance monitoring service for the Fusion Core and Fusion Virtual Machine. The FSS provides an *aggregate* view of the performance of each policy, vote, and voting group member. This information is important to building a more secure and more reliable fusion system.

5.1 FSS

The FSS is a *multidimensional database* (MDB) [11,12]. An MDB is a tool that allows a user to "fly-around" and get different views of the same aggregate data (i.e., sum or count data). A multidimensional database has one or more *dimensions*. The FSS MDB for the replicated server scenario has three dimensions: Time, Source, and the Status of each vote.

Each dimension consists of a hierarchy of related *categories*. A category is a system of measurement, and is configurable by creating a text input file describing the category. For example, the Source dimension is used for grouping replicas by spatial position. Figure 3 shows example high-level categories in the spatial dimension. The categories are Geographic Location, Internet Hierarchy, and Logical Grouping; all start with the base Replica ID and generalize from there in the ways depicted in the figure. Each box in Figure 3 represents a category or subcategory, where the name of the category or subcategory is given in bold font. Example values are shown in parentheses below this name

A category consists of individual measurements, which are called *units*. Units high in the hierarchy are

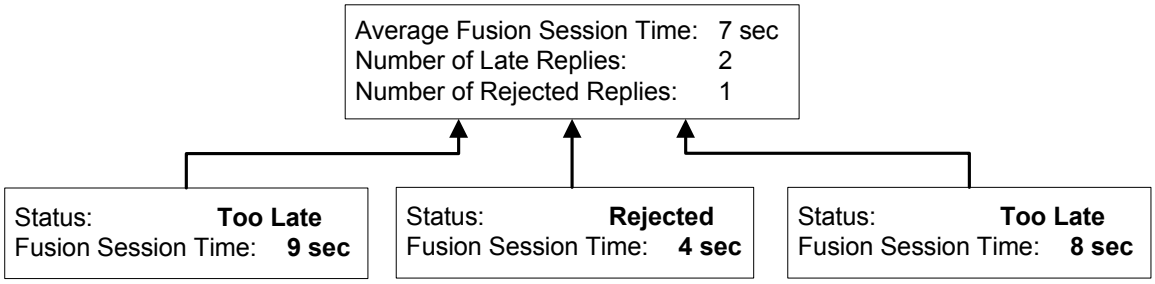

Figure 4. Aggregating Fusion Core data

aggregations of those lower in the hierarchy. The utility of the hierarchical organization is that the user can easily navigate among high and low precision views of the same aggregate data using *drill-down* and *roll-up*. Drill-down is an operation that increases the precision of aggregate data being viewed while roll-up decreases the precision. The FSS builds, maintains, and manages, an MDB of fusion session information. Figure 4 shows an example of aggregating the statuses and averaging the fusion session time for a group of replicas. These values can be aggregated further to calculate the total number of late and rejected replies and the total average of the fusion

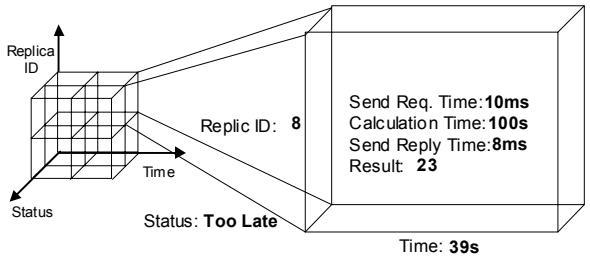

Figure 5. Depiction of the FSS MDB

session time of all the replicas in the entire fusion group, which might span multiple hosts. Figure 5 shows the three dimensions in the FSS MDB, and the value at a single unit in the hierarchy with the coordinates Time: 39s, Status: too late, and Replica ID: 8.

The Time dimension has categories for Minute, Hour, and Day. Finally, the Status dimension groups the replicas depending on status of the votes produced by them. The unit of this dimension is a status of the result generated by the replica (for example an error code or other lateness metric).

5.2 Example

Figure 6 shows the FSS MDB interface built using the prototype developed in the incomplete data cube project [7, 8]. This interface is a web accessible PERL script (a Java applet interface is also available). In the example, data has been collected from three hosts (dyreson.eecs.wsu.edu, dyreson2.eecs.wsu.edu and bakken.eecs.wsu.edu) over the period of time that spans from the year 1995 until 1997. The example shows the result of a query that retrieves all the network-related problems for the host 'dyreson.eecs.wsu.edu' during the year '1997'. The set of possible network-related problems consists of two elements: 'Slow Network' and 'Unreachable Host'. According to the query result the host had one 'Slow Network' error and no 'Unreachable Host' errors during the entire year (the counts are artificially low because the data is test data).

Data can be retrieved from the FSS MDB for any specified categories and units. For example, in the categories chosen are 'Host', 'Years', and 'Network Reason' for the Replica, Time, and Status dimensions, respectively. The corresponding units chosen are 'dyreson.eecs.wsu.edu', '1997' and 'All'. A user interested in further details could roll-up to look at an overall count for all hosts or drill-down to obtain a count for each month in 1997.

6 Acknowledgements

This research was funded by the DARPA OASIS program via a subcontract to NAI Labs, contract F30602-00-C-0183. We thank Hien Tran and Ty Palmer for their help with this various aspects of Mr. Fusion plus the NAI Labs ITDOS team for their feedback and encouragement on this research.

References

[1] Bakken, D. "Middleware", Chapter in *Encyclopedia of Distributed Computing*, J. Urban and P. Dasgupta, Eds., Kluwer Academic Publishers, 2002, to appear. Available at http://www.eecs.wsu.edu/~bakken/middleware.pdf.

[2] Bass, T. "Intrusion Detection Systems and Multisensor Data Fusion", *Communications of the ACM*, 43(4), April 2000, 99-105.

[3] Birman, K., Hayden, M., Ozkasap, O., Xiao, Z., Budia, M., and Minsky, Y. "Bimodal Multicast", *ACM Transactions on Computer Systems*, 17(2), May 1999, 41-88.

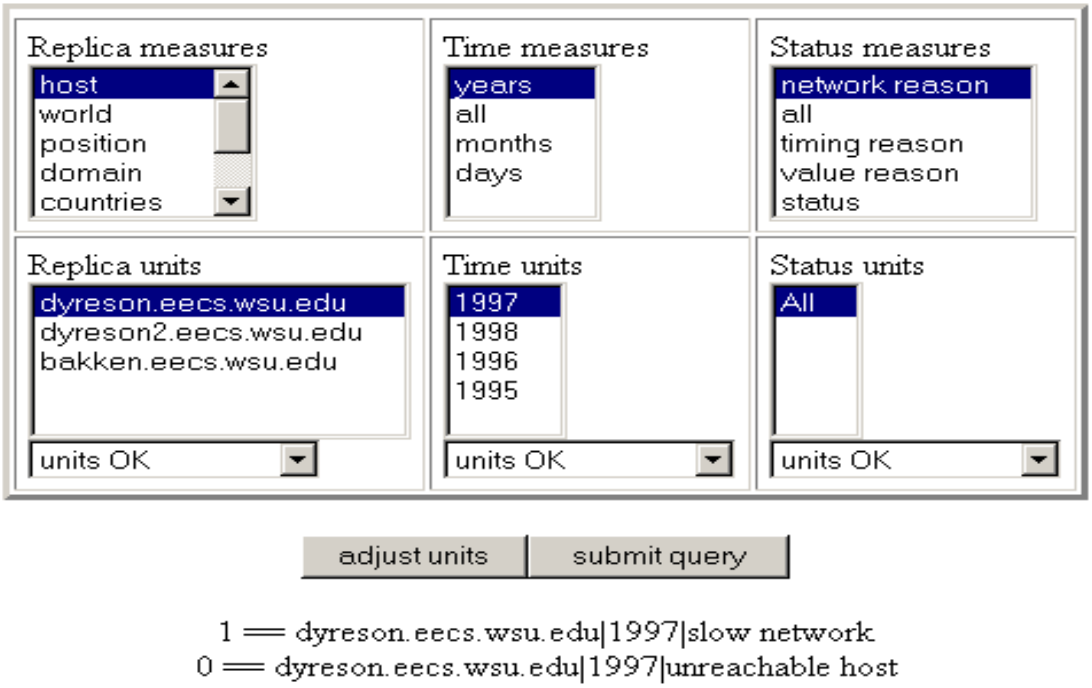

Figure 6 FSS MDB interface

[4] Bakken, D. and Zhan, Z. and Jones, C. and Karr, D. "Middleware Support for Voting and Data Fusion", in *Proceedings of the International Conference on Dependable Systems and Networks (DSN-2001)*, IEEE/IFIP, Göteborg, Sweden, July 1-4, 2001, 453-462.

[5] Chen, L. "Neural Network Approaches for Estimating Margins in Power System Voltage Security Analysis", MS Thesis, Washington State University, August 2000.

[6] Cukier, Michel and Ren, Jennifer and Sabnis, Chetan and Henke, David and Pistole, Jessica, and Sanders, William, and Bakken, David and Berman, Mark and Karr, David and Schantz, Richard, "AQuA: An Adaptive Architecture That Provides Dependable Distributed Objects", in *Proceedings of the Seventeenth Symposium on Reliable Distributed Systems* (SRDS-17), IEEE, October 1998, 245-253.

[7] Dyreson, C. "Information Retrieval from an Incomplete Data Cube." In *Proceedings of the Conference on Very Large Databases* (VLDB), Mumbai, India, September 1996, pp. 532-543.

[8] Dyreson, Curtis, "Using an Incomplete Data Cube as a Summary Data Sieve." *Data Engineering Bulletin,* 20(1). March 1997, 19-26.

[9] Hall, D. and Llinas, J. "Handbook of Multidata Sensor Fusion", CRC Press, 2001.

[10] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. and Pister, K. "System Architecture Directions for Networked Sensors", in *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, 93-104, ACM SIGPLAN, November 2000.

[11] Kimball R. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley & Sons, 1996.

[12] Mendelzon, A. (2001). Data Warehousing and OLAP: a Research-Oriented Bibliography (in progress). http://www.cs.toronto.edu/~mendel/dwbib.html. Current as of November, 2001.

[13] McDonnel, D and Neibuhr, B. and Matt, B. and Sames, D. and Tally, Gregg and Wang, S. and Whitmore, B and Bakken, D. "Developing a Heterogeneous Intrusion-Tolerant CORBA System", to appear in *Proceedings of the International Conference on Dependable Systems and Networks* (DSN-2002), Washington, DC, June 2002.

[14] Narasimhan, P., Moser, L., and Melliar-Smith, M. "Strong Replica Consistency for Fault-Tolerant CORBA Applications", *Journal of Computer System Science and Engineering*, Spring 2002, to appear.

[15] Parameswaran, R., Blough, D., and Bakken, D. "A Preliminary Investigation of Precision vs. Fault Tolerance Trade-offs in Voting Algorithms", Fast Abstract in *Supplement of the International Conference on Dependable Systems and Networks (DSN-2001)*, Göteborg, Sweden, July, 2001.

[16] van Renesse, R. "Scalable and Secure Resource Location", In *Proceedings of the Hawaii International Conference on System Sciences*, January, 2000, Maui, Hawaii.

[17] Walz, E. and Llinas, J. "Multisensor Data Fusion", Artech House, Boston, 1990.