A Review on System Architectures for Sensor Fusion Applications

Wilfried Elmenreich

Vienna University of Technology
Treitlstrasse 3, 1040 Vienna, Austria
wil@vmars.tuwien.ac.at

Abstract. In the literature there exist many proposed architectures for sensor fusion applications. This paper briefly reviews some of the most common approaches, i. e., the JDL fusion architecture, the Waterfall model, the Intelligence cycle, the Boyd loop, the LAAS architecture, the Omnibus model, Mr. Fusion, the DFuse framework, and the Time-Triggered Sensor Fusion Model, and categorizes them into abstract models, generic and rigid architectures. While an abstract model does not guide the designer in the concrete implementation, the generic architectures provide a generic design but leave open several design decisions regarding operating system, hardware, communication system, or database system. Rigid architectures specify at least some of these aspects and therefore provide existing hardware designs, tools, and source code at the cost of flexibility.

1 Introduction

Sensor fusion, "*the combining of sensory data or data derived from sensory data such that the resulting information is in some sense better than would be possible when these sources were used individually*" [1], encompasses a wide variety of different application types (e. g., automation, automotive driver assistance systems, autonomous robots, C^{3}I (command, control, communications, and intelligence)). An example for sensor fusion applications are current innovations in automotive electronic driver assistant systems [2].

Due to the fact that sensor fusion models heavily depend on the application, there exists no generally accepted model of sensor fusion. According to Kam, Zhu, and Kalata, it is unlikely that one technique or one architecture will provide a uniformly superior solution [3]. Thus, there exist numerous architectures and models for sensor fusion in the literature. In order to use sensor fusion for an application, it is of interest which models and architectures can be used as design patterns.

It is the objective of this paper to review several sensor fusion models and architectures that have been used for sensor fusion. The different approaches will be assessed with respect to their eligibility for real-time applications.

The rest of the paper is structured as follows: The following section briefly describes nine sensor fusion architectures or architectures that have been used to implement sensor fusion applications. Section 3 introduces a classification and discusses the implications of design decisions and design freedom for an implementation. The paper is concluded in Section 4.

2 Architectures for Sensor Fusion

2.1 The JDL Fusion Architecture

A frequently referred fusion model originates from the US Joint Directors of Laboratories (JDL). It was proposed in 1985 under the guidance of the Department of Defense (DoD). The *JDL model* [4] comprises five levels of data processing and a database, which are all interconnected by a bus. The five levels are not meant to be processed in a strict order and can also be executed concurrently. Figure 1 depicts the top level of the JDL data fusion process model. The elements of the model are described in the following:

Sources: The sources provide information from a variety of data sources, like sensors, *a priori* information, databases, human input.

Source preprocessing (Level 0): The task of this element is to reduce the processing load of the fusion processes by prescreening and allocating data to appropriate processes. Source preprocessing has later been labelled *level 0* [5].

Object refinement (Level 1): This level performs *data alignment* (transformation of data to a consistent reference frame and units), *association* (using correlation methods), *tracking* actual and future positions of objects, and *identification* using classification methods.

Situation refinement (Level 2): The situation refinement tries to find a contextual description of the relationship between objects and observed events.

Threat refinement (Level 3): Based on *a priori* knowledge and predictions about the future situation this processing level tries to draw inferences about vulnerabilities and opportunities for operation.

Process refinement (Level 4): Level 4 is a meta process that monitors system performance (e. g., real-time constraints) and reallocates sensor and sources to achieve particular mission goals.

Database management system: The task of the database management system is to monitor, evaluate, add, update, and provide information for the fusion processes.

Man-machine interaction: This part provides an interface for human input and communication of fusion results to operators and users.

The JDL model has been very popular for fusion systems. Despite its origin in the military domain it can be applied to both military and commercial applica-

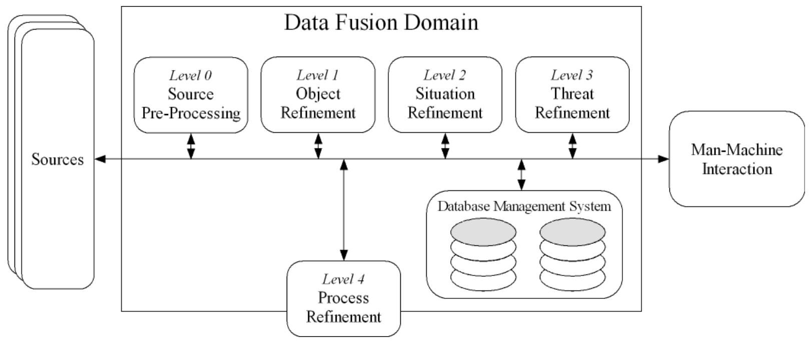

Fig. 1. JDL fusion model (from [4])

tions. The JDL model also has categorized processes related to a fusion system. However, the model suffers from the following drawbacks:

- It is a data-centered or information-centered model, which makes it difficult to extend or reuse applications built with this model.
- The model is very abstract, which makes it difficult to properly interpret its parts and to appropriately apply it to specific problems.
- The model is helpful for common understanding, but does not guide a developer in identifying the methods that should be used [4] – thus, the model does not help in developing an architecture for a real system.

The basic JDL model has also been improved and extended for various applications. Waltz showed, that the model does not address multi-image fusion problems and presented an extension that includes the fusion of image data [6]. Steinberg, Bowman, and White proposed revisions and expansions of the JDL model involving broadening the functional model, relating the taxonomy to fields beyond the original military focus, and integrating a data fusion tree architecture model for system description, design, and development [7].

2.2 Waterfall Fusion Process Model

The waterfall model, proposed in [8], emphasizes on the processing functions on the lower levels. Figure 2 depicts the processing stages of the waterfall model. The stages relate to the levels 0, 1, 2, and 3 of the JDL model as follows: Sensing and signal processing correspond to source preprocessing (level 0), feature extraction and pattern processing match object refinement (level 1), situation assessment is similar to situation refinement (level 2), and decision making corresponds to threat refinement (level 3).

Being thus similar to the JDL model, the waterfall model suffers from the same drawbacks. While being more exact in analyzing the fusion process than other models, the major limitation of the waterfall model is the omission of any feedback data flow. The waterfall model has been used in the defense data fusion community in Great Britain, but has not been significantly adopted elsewhere [5].

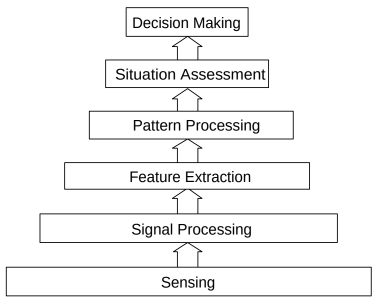

Fig. 2. The waterfall fusion process model (from [8])

2.3 The Intelligence Cycle

Another approach to model a fusion application is to line out its cyclic character. Representatives of such an approach are the intelligence cycle [9] and the Boyd control loop [10].

The *Intelligence Cycle* [9] comprises the following five stages:

Planning and Direction: This stage determines the intelligence requirements.
Collection: Gathering of appropriate information, e. g., through sensors.
Collation: Here the collected information is lined up.
Evaluation: The actual fusion is done and the information gets analyzed.
Dissemination: Dissemination distributes the fused intelligence.

2.4 Boyd Model

Boyd has proposed a cycle containing four stages [10]. This *Boyd control cycle* or *OODA loop* (depicted in figure 3) represents the classic decision-support mechanism in military information operations. Because decision-support systems for situational awareness are tightly coupled with fusion systems [11], the Boyd loop has also been used for sensor fusion. Bedworth and O'Brien compared the stages of the Boyd loop to the JDL [5]:

Observe: This stage is broadly comparable to source preprocessing in the JDL model.
Orientate: This stage corresponds to functions of the levels 1, 2, and 3 of the JDL model.
Decide: This stage is comparable to level 4 of the JDL model (Process refinement).
Act: This stage has no direct counterpart in the JDL model.

The Boyd model represents the stages of a closed control system and gives an overview on the overall task of a system, but the model lacks of an appropriate structure for identifying and separating different sensor fusion tasks.

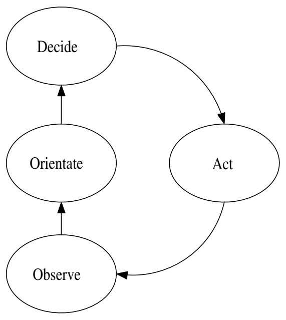

Fig. 3. The Boyd (or OODA) loop

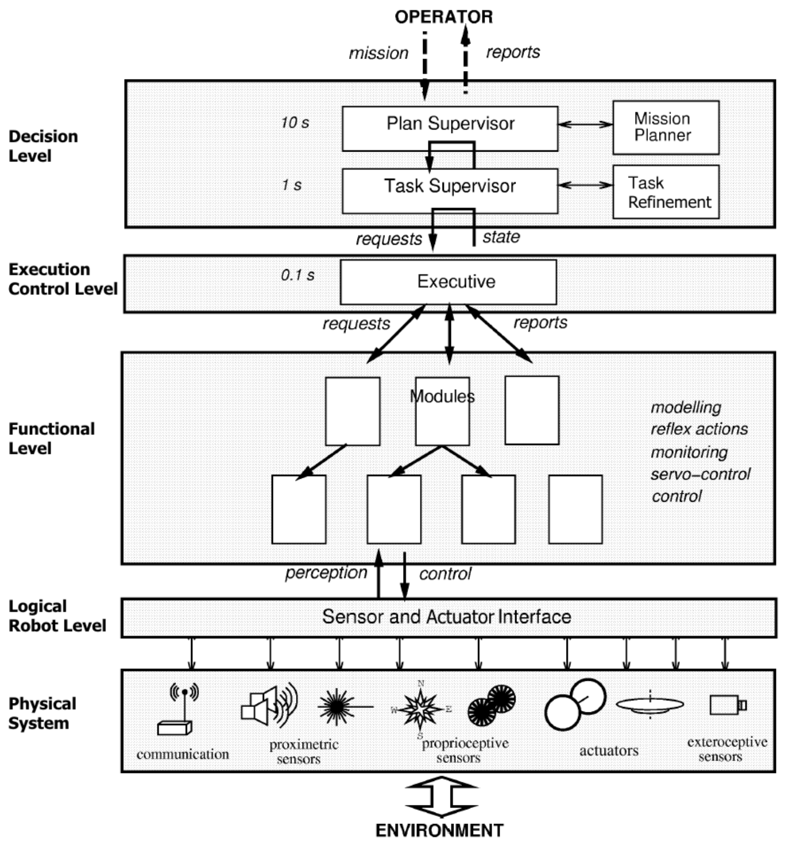

Fig. 4. LAAS Architecture (from [12])

2.5 The LAAS Architecture

The LAAS (Laboratoire d'Analyse et d'Architecture des Systèmes) architecture [12] was developed as an integrated architecture for the design and implementation of mobile robots with respect to real-time and code reuse. Due to the fact that mobile robot systems often employ sensor fusion methods, we briefly discuss the elements of the LAAS architecture (depicted in figure 4).

The architecture consists of the following levels [12]:

Logical robot level: The task of the logical robot level is to establish a hardware independent interface between the physical sensors and actuators and the functional level.

Functional level: The functional level includes all the basic built-in robot action and perception capabilities. The processing functions, such as image processing, obstacle avoidance, and control loops, are encapsulated into separate controllable communicating modules.

Execution control level: The execution control level controls and coordinates the execution of the functions provided by the modules according to the task requirements.

Decision level: The decision level includes the capabilities of producing the task plan and supervising its execution while being at the same time reactive to other events from the execution control level. Depending on the application, the decision level can be composed of several layers that provide different representation abstractions and have different temporal properties.

The LAAS architecture maps low-level and intermediate-level sensor fusion to modules at the functional level. High-level sensor fusion is represented in the decision level. The timing requirements are different at the decision level and the functional level. In contrast to the JDL model, the LAAS architecture guides a designer well in implementing reusable modules as part of a real-time application.

2.6 The Omnibus Model

The Omnibus model [5] has been presented in 1999 by Bedworth and O'Brien. The model was created after analyzing the strengths and weaknesses of existing models and integrates most of the beneficial features of other approaches.

Figure 5 depicts the architecture of the Omnibus model. Unlike the JDL model, the Omnibus model defines the ordering of processes and makes the cyclic nature explicit. It uses a general terminology that does not assume that the applications are defense-oriented. The model shows a cyclic structure comparable to the Boyd loop, but provides a much more fine-grained structuring of the processing levels. The model is intended to be used multiple times in the same application recursively at two different levels of abstraction. First, the model is used to characterize and structure the overall system. Second, the same structures are used to model the single subtasks of the system.

Although the hierarchical separation of the sensor fusion tasks is very sophisticated in the Omnibus model, it does not support a horizontal partitioning into tasks that reflect distributed sensing and data processing. Thus, the model does not support a decomposition into modules that can be separately implemented, separately tested, and reused for different applications.

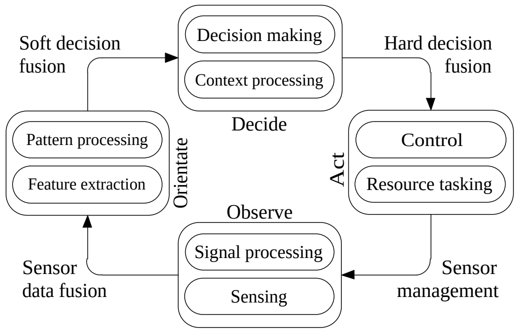

Fig. 5. The Omnibus model (from [5])

2.7 Mr. Fusion

Mr. Fusion [13] is a middleware framework supporting data fusion. Mr. Fusion is not exactly tailored to the communication and processing of sensor measurements but aims at data at application level, as for example the output from several network servers.

The architecture consists of two main subsystems, a *fusion core* running a Fusion Virtual Machine (FVM) and a Fusion Status Service (FSS). The communication between the main components is done via CORBA (Common Object Request Broker Architecture). The FVM gathers so-called ballots, i. e., messages from the replicas and evaluates a given policy in order to create an output ballot or an exception. The FSS monitors the output from the fusion core and collects information about value and timing errors for each fusion session into a database. Information from this database is used by a component named Fusion VM Manager in order to eventually adjust a policy for the FVM.

2.8 DFuse Framework

The DFuse framework for distributed data fusion [14] has been designed to support data fusion applications in heterogeneous ad hoc wireless sensor networks. DFuse models an application as a task graph of data sources, fusion points and data sinks. DFuse assumes data sources to have data available, whenever it is required. If requested data does not arrive at a fusion point in time due to extended computation time or communication failures, the fusion points may perform the fusion over an incomplete set of data.

Figure 6 depicts the two main components of the DFusion architecture, the fusion module implementing the fusion API and the placement module that tries to find a good mapping of the fusion functions within the sensor network. DFuse provides an automatic deployment of applications on the network. An application is launched by passing the task graph and fusion code to a designated root node. The DFuse architecture then performs a distributed algorithm that automatically deploys the application onto the network nodes.

While being a very powerful approach, DFuse requires an underlying hardware and middleware that provides support for timestamping data and a reliable transport layer. Therefore, DFuse cannot be deployed in small wireless sensor architectures such as the Mica platform [15]. Kumar et al. present a case study running on a set of iPAQ 3870 handheld computers providing each at least 32 MB RAM and a 206 MHz StrongARM processor.

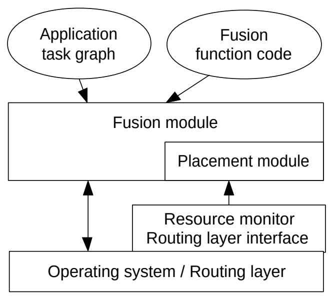

Fig. 6. DFuse Architecture (from [14])

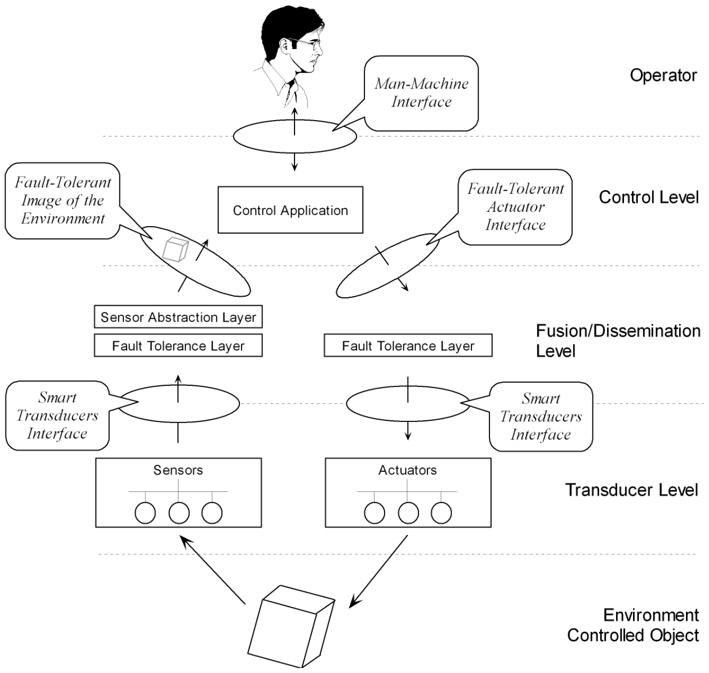
Operator
Man-Machine Interface
Control Level
Fault-Tolerant Image of the Environment
Control Application
Fault-Tolerant Actuator Interface
Sensor Abstraction Layer
Fault Tolerance Layer
Fault Tolerance Layer
Fusion/Dissemination Level
Smart Transducers Interface
Smart Transducers Interface
Sensors
Actuators
Transducer Level
Environment Controlled Object

Fig. 7. Data flow in the time-triggered sensor fusion model

2.9 Time-Triggered Sensor Fusion Model

The Time-Triggered Sensor Fusion Model [16] proposes the implementation of a sensor fusion application on top of the Time-Triggered Architecture [17].

The Time-Triggered Architecture proposes a strictly synchronous design, where each task and communication activity is planned *a priori* in a static schedule. All distributed nodes are synchronized to a global time base, which enables the nodes to perform coordinated actions like measurement or actuator settings. Furthermore, the design supports an easy verification of the timing constraints.

The Time-Triggered Sensor Fusion Model describes a set of jobs that represent all necessary activities like measurement, data processing, decision, and actuation. The jobs are represented as vertexes in a distributed graph, whereas each communication activity is represented by an edge between the service providing linking interface (SPLIF) of the job that provides the data and the service requesting linking interface (SRLIF) of the job that receives the data. A physical node may host one or several jobs, thus two logically different tasks may be split up into two jobs but still executed on the same microcontroller subsequently.

The job graph is furthermore structured hierarchically into three levels in order to distinguish between transducers (direct interfaces to the environment), fusion and dissemination activities, and decision activities.

Figure 7 depicts a control loop modelled by the time-triggered sensor fusion model. Interfaces are illustrated by a disc with arrows indicating the possible data flow directions across the interface. Physical sensors and actuators are located on the borderline to the process environment and are represented by circles. All other components of the system are outlined as boxes. The model distinguishes three levels of data processing with well-defined interfaces between them. The

Table 1. Properties of transducer, fusion/dissemination, and control level

Level	Task	Implementer	Knowledge
Transducer level	Deliver sensor measurements, instrument actuators	Transducer manufacturer	Internals of sensor/actuator
Fusion/Dissemination level	Gather, process, and represent sensor information; disseminate control decisions to actuators	System integrator	Sensor fusion algorithms, fault tolerance concepts
Control level	Find a control decision, navigation and planning	Application programmer	Mission goals, control theory, decision finding
Operator	Definition of goals	—	Conceptual model of system

transducer level contains the sensors and actuators that interact directly with the controlled object. A *smart transducer interface* provides a consistent borderline to the above *fusion/dissemination level*. This level contains fault tolerance and sensor fusion tasks. The *control level* is the highest level of data processing within the control loop. The control level is fed by a dedicated view of the environment (established by transducer and fusion/dissemination level) and outputs control decisions to a *fault-tolerant actuator interface*. User commands from an operator interact with the control application via the *man-machine interface*.

The breakdown into these three levels is justified by the different tasks the three levels have to fulfill and the different knowledge necessary for designing the corresponding hard- and software. Table 1 describes the task and the attributes of the different levels. The following sections describe the three levels in detail.

Prerequisites for implementing an application in the Time-Triggered Sensor Fusion Model are a deterministic time-triggered communication system that supports coordinated task execution and a known upper bound for the computation time of each job in the real-time control loop. Thus, the Worst-Case-Execution Time (WCET) has to be determined for each job, an overview of appropriate methods and tools for WCET estimation can be found in [25].

There are several protocols available for wired time-triggered systems [18–22] but only a few solutions are available for wireless systems. Notable exceptions are Kim and Li's work on time-triggered tasks [23] that have also been implemented on the wireless Mica platform [15] and the work on time-triggered wireless communication by Huber and Elmenreich [24].

The time-triggered approach works well with typical sensor fusion algorithms, such as:

– Kalman Filtering [26] requires periodic sets of measurements. The measurements have to be taken at the same instant and the communication system should avoid out-of-sequence behavior of messages.
– Abstract reliable sensors [27] and the confidence-weighted averaging algorithm [1] require measurements to be taken at approximately the same instant with respect to the change rate of the measured variable.

3 Comparison

3.1 Classification

The presented sensor fusion architectures can be roughly classified into the following categories:

Abstract Models: These approaches serve as a way to think of or explain an aspect of a fusion system without guiding the engineer in its implementation. As a consequence, a fusion system may contain references to several abstract models. Members of this group are the Waterfall model, the Boyd control loop.

Generic Architectures: A generic architecture gives an outline how to implement an application, but for example does not specify which operating system, hardware, communication system or database should be used. Examples for this group are the JDL model and the Omnibus Model.

Rigid Architectures: These systems guide the engineer well in its implementation, but at the cost that several design decisions have already been taken. While new systems can be realized quickly by taking advantage of existing hardware designs, tools, and source code, the cost of migrating a design from one rigid architecture to another is unnecessarily high. Examples for this group are the LAAS architecture, Mr. Fusion, DFuse, and the Time-Triggered Sensor Fusion Model.

The three categories should not imply a valuation. Abstract models are very important to understand and model the problem statement at the beginning. Using a generic architecture will provide the necessary designer's freedom if a special solution for a special problem is required. On the other hand, selecting a rigid solution will be often the best way to avoid unnecessary re-implementations of already available solutions.

3.2 Real-Time Support

Typically, sensor fusion applications interact with a real environment where it is necessary to fulfill some timing constraints, e. g., for a timely reaction on a particular situation.

For most architectures, especially the abstract and generic ones, it is possible to support real-time behavior, if the implementation of the system is provided with the respective means, like timestamping, deterministic communication, etc. Thus, these architectures neither support nor hinder real-time behavior. However, in order to reduce system complexity such real-time issues should be intrinsic to the architecture.

We will quickly review the rigid architectures regarding this issue:

The LAAS architecture provides real-time support within the functional level by the Generator of Modules (GenoM) [28]. Modules are annotated by the user stating period, delay, and priority properties. GenoM creates the concrete real-time architecture for the application.

Mr. Fusion has been designed at a higher network level, where several sources of indeterminism like possible network delays or unpredictable execution time of the virtual machines jeopardize hard real-time behavior. Therefore, Mr. Fusion is not suited to real-time control applications.

The DFuse framework does not provide predictable timing due to the nature of the underlying wireless communication network. One cannot predict how the heuristic placement algorithm assigns the roles in the networks. Moreover, a wireless transmission may be arbitrary delayed due to inference from other wireless nodes. Therefore, the DFuse framework will not fulfill hard real-time requirements.

The Time-Triggered Sensor Fusion Model is very rigid regarding the timing assumptions and therefore well apt to design real-time fusion applications. However, in applications with soft real-time requirements, the approach still requires a strict analysis and design of the communication schedule. Although this is supported by tools like [29], this strictness comes with some overhead.

4 Conclusion

The large number of proposed sensor fusion architectures makes it difficult for a system engineer to decide which model best fits his or her needs.

While some fusion models are too vague in order to support and guide an implementation, the more concrete models propose different interfaces that do not interoperate with each other. Some systems, especially the ones that stem from the robotic domain are in principle compatible regarding their basic data items and the role of time (that is supporting real-time communication and being implementable on small embedded devices). In contrast, high-level network systems such as Mr. Fusion are tailored to their specific application requirements.

For the future it would be advantageous to elaborate ways that provide interoperation between components of existing fusion architectures instead of creating even more isolated systems anew.

References

1. W. Elmenreich. *Sensor Fusion in Time-Triggered Systems.* PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.
2. A. Kirchner, A. Obojski, H. Philipps, C. Rotaru, D. Stueker, and K. Weiss. Development of an environmental server for advanced driver assistance systems. In *5th Eur. Congr. and Exhibition on Intel. Transportation Systems and Services*, 2005.
3. M. Kam, X. Zhu, and P. Kalata. Sensor fusion for mobile robot navigation. *Proceedings of the IEEE*, 85(1):108–119, Jan. 1997.
4. J. Llinas and D. L. Hall. An introduction to multi-sensor data fusion. *Proceedings of the International Symposium on Circuits and Systems*, 6:537–540, May–June 1998.
5. M. D. Bedworth and J. O'Brien. The omnibus model: A new architecture for data fusion? In *Proceedings of the 2nd International Conference on Information Fusion (FUSION'99)*, Helsinki, Finnland, July 1999.
6. E. Waltz. The principles and practice of image and spatial data fusion. In *Proceedings of the 8th National Data Fusion Conference, Dallas*, 1995.
7. A. N. Steinberg, C. L. Bowman, and F. E. White. Revisions to the JDL data fusion model. In *Proceedings of the 1999 IRIS Unclassified National Sensor and Data Fusion Conference (NSSDF)*, May 1999.
8. M. Markin, C. Harris, M. Bernhardt, J. Austin, M. Bedworth, P. Greenway, R. Johnston, A. Little, and D. Lowe. Technology foresight on data fusion and data processing. Publication, The Royal Aeronautical Society, 1997.

9. A. N. Shulsky. *Silent Warfare: Understanding the World of Intelligence*. Brassey's, New York, 1991.
10. J. R. Boyd. A discourse on winning and losing. Unpublished set of briefing slides, Air University Library, Maxwell AFB, AL, USA, May 1987.
11. T. Bass. Intrusion detection systems and multisensor data fusion: Creating cyberspace situational awareness. *Comm. of the ACM*, 43(4):99–105, May 2000.
12. R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *International Journal of Robotics Research*, 17(4):315–337, April 1998.
13. A. Franz, R. Mista, D. Bakken, C. Dyreson, and M. Medidi. Mr. fusion: A programmable data fusion middleware subsystem with a tunable statistical profiling service. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'02)*, pages 273–278, 2002.
14. R. Kumar, M. Wolenetz, B. Agarwalla, J. Shin, P. Hutto, Arnab Paul, and U. Ramachandran. DFuse: A framework for distributed data fusion. In *Proceedings of the Intl. Conference on Embedded Networked Sensor Systems*, pages 114–125, 2003.
15. J. L. Hill and D. E. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November/December 2002.
16. W. Elmenreich and S. Pitzek. The time-triggered sensor fusion model. In *Proceedings of the 5th IEEE International Conference on Intelligent Engineering Systems*, pages 297–300, Helsinki–Stockholm–Helsinki, Finland, September 2001.
17. H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1):112–126, January 2003.
18. H. Kopetz et al. Specification of the TTP/A protocol. Research Rep. 61/2002, TU Vienna, Inst. of Comp. Engineering, Vienna, Austria, September 2002. Version 2.00.
19. TTAGroup. *Specification of the TTP/C Protocol V1.1*, 2003. Available at http://www.ttagroup.org.
20. Flexray Consortium. *FlexRay Communications System Protocol Specification Version 2.1*, 2005. Available at http://www.flexray.com.
21. F. Hartwich, B. Müller, T. Führer, and R. Hugel. Time triggered communication on CAN. In *Proc. of the Intl. CAN Conference*, Amsterdam, The Nederlands, 2000.
22. H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer. The Time-Triggered Ethernet (TTE) design. In *Proceedings of the Intl. Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, pages 22–33, Seattle, USA, May 2005.
23. K. Kim and Y. Li. Toward easily analyzable sensor networks via structuring of time-triggered tasks. In *Proceedings of the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, pages 344–351, May 2003.
24. B. Huber and W. Elmenreich. Wireless time-triggered real-time communication. In *Proceedings of the Second Workshop on Intelligent Solutions for Embedded Systems (WISES'04)*, pages 169–182, Graz, Austria, June 2004.
25. P. Puschner and A. Burns. A review of worst-case execution-time analysis. *Journal of Real-Time Systems*, 18(2/3):115–128, May 2000.
26. R. E. Kalman. A new approach to linear filtering and prediction problems. *Transaction of the ASME, Series D, Journal of Basic Engineering*, 82:35–45, March 1960.
27. K. Marzullo. Tolerating failures of continuous-valued sensors. *ACM Transactions on Computer Systems*, 8(4):284–304, November 1990.
28. S. Fleury, M. Herrb, and R. Chatila. Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 842–848, Grenoble, France, September 1997.
29. W. Elmenreich, C. Paukovits, and S. Pitzek. Automatic generation of schedules for time-triggered embedded transducer networks. In *Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA05)*, volume 2, pages 535–541, Catania, Italy, September 2005.