# A Software Framework for Heterogeneous, Distributed Data Fusion

Joshua J. Walters*
ITT AES / Naval Research Laboratory
4555 Overlook Avenue
Washington DC, 20375, USA

Simon J. Julier†
ITT AES / Naval Research Laboratory
4555 Overlook Avenue SW
Washington, DC 20375, USA

**Abstract -** ***In this paper we describe a software framework to enable heterogeneous, distributed data fusion of disparate information sources. The framework is agent-based and consists of three main elements. The first is a generalization of the target state to a container of arbitrary, uncertain attributes. The structure of this estimate can vary both across time and across different nodes in the same network. The second is the development of composable process and observation models. These make it possible to dynamically change the models at runtime to fit the current target state estimate. Finally,***

**Keywords:** tracking, data association, estimation, resource allocation, state estimates

## 1 Introduction

A key enabling technology for network centric warfare is a sensor network. The network is composed of a set of fusion nodes. Each node has the ability to sense its environment, to communicate with other nodes, and to be able to fuse the information collected locally and transmitted from other nodes. The resulting network is, in principle, scalable, flexible, and robust. Scalability arises from the fact that there is no theoretical limit on the overall size of the network. The flexibility arises because the capabilities of the network can be changed dynamically at runtime by adding and removing nodes. Finally, the robustness arises because there is no single point of failure: if a fusion node fails, the network continues operate, albeit it reduced performance.

Given these advantages, sensor networks have been the subject of intensive research. Much of this research can be categorized into three areas. The first area considers the physical implementation of such networks. Research topics include the development of compact, low cost, and low powered hardware [1], new algorithms for power management [2], and network protocols for ad-hoc networks [3]. The second area addresses the problem of developing services that can run over sensor networks. Research technologies include network distributed objects, agent-based technology [4], Fuselet Technology [5] and web services [6]. The final area considers the problem of fusion architectures and algorithms that will be run in those services. Research topics include the development of data fusion algorithms to account for double counting [7, 8] and the scheduling of broadcasts to optimize information content [9].

Although there has been a significant amount of work to develop software architectures that integrate the first two research areas, relatively little work has been carried out into how the third area can be combined with the other two. A number of software libraries, such as Bayes++ [10], have been developed to implement low-level algorithms (such as Kalman and particle filters). However, these do not consider how these algorithms would be implemented in the larger context of a dynamic and time varying system.

In this paper we consider the problem of developing a software framework to implement a wide classes of fusion algorithms in a generalized sensor fusion network. In particular, the framework has the following characteristics:

1. The target state is modeled as a container of arbitrary, uncertain attributes. The composition of the target's state can change both through time and across the network.

2. The mathematical models used to transform the target state are assembled at runtime at each node. The models are a function of the structure of the target state and contextual information such as the node's computational capabilities.

*Joshua Walters is now with DCS Corporation, Alexandria, VA. Email: joshua.walters@yahoo.com.

†Simon Julier is now with the Department of Computer Science, University College London. Email: S.Julier@cs.ucl.ac.uk

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE | 2. REPORT TYPE | 3. DATES COVERED |
|---|---|---|
| **JUL 2006** | | **00-00-2006 to 00-00-2006** |

| 4. TITLE AND SUBTITLE | 5a. CONTRACT NUMBER |
|---|---|
| **A Software Architecture for Heterogeneous, Distributed Data Fusion** | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
| | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **ITT AES / Naval Research Laboratory,4555 Overlook Avenue,Washington ,DC,20375** | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES
**9th International Conference on Information Fusion, 10-13 July 2006, Florence, Italy. Sponsored by the International Society of Information Fusion (ISIF), Aerospace & Electronic Systems Society (AES), IEEE, ONR, ONR Global, Selex - Sistemi Integrati, Finmeccanica, BAE Systems, TNO, AFOSR's European Office of Aerospace Research and Development, and the NATO Undersea Research Centre. U.S. Government or Federal Rights License**

14. ABSTRACT
**see report**

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT **unclassified** | b. ABSTRACT **unclassified** | c. THIS PAGE **unclassified** | **Same as Report (SAR)** | **8** | |

3. To optimize communication between different nodes, a publish and subscribe mechanism is used.

4. Nodes can use code mobility to share and upgrade software components.

5. The architecture imposes very few restrictions on the assumed behavior of the data fusion algorithms. Therefore, almost any kind of published data fusion algorithm can be used.

We make two assumptions:

1. There is an agreed upon standard for the naming and interpretation of each attribute. This standard should be derived from an ontology.

2. Each attribute is a modeled as a real-valued scalar and the uncertainty associated with each state can be described as a Gaussian Mixture Model.

The second assumption is made to simplify the initial design of the framework; future revisions will relax this assumption.

The structure of this paper is as follows. The problem statement is discussed in more detail in Section 2. An overview of our framework is provided in Section 3. The target state representation is discussed in Section 4. The operations within a single node are described in Section 5. The issue of the interaction between multiple nodes is described in Section 6. An example of this architecture is discussed in Section 7. Conclusions are drawn in Section 8.

## 2 Problem Statement

The structure of the network is shown in Figure 1: the network is composed of a set of *fusion nodes* and *communication links* between those nodes. Each node maintains its own estimate of the Common Operational/Tactical Picture (COTP). The COTP consists of a set of entities. Each entity is a container of attributes. There are many kinds of attributes including kinematics (e.g., position, velocity), identification (e.g., entity type, entity class), and sensor signatures (e.g., radar cross section, color).

The internal operation of each sensor is described by the event-driven loop shown in Figure 2. The target state is initialized. The node then waits until it receives an event. Some types of events (such as sensor observations) provide sources of new information. Other types (such as timeouts) are control signals and provide no new information. The target state is predicted forwards to the event time. If the event provides new information, this information is fused into the target state. If the event does not provide any new

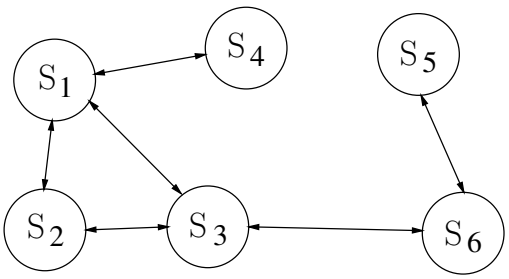


Figure 1: The distributed data fusion network architecture.

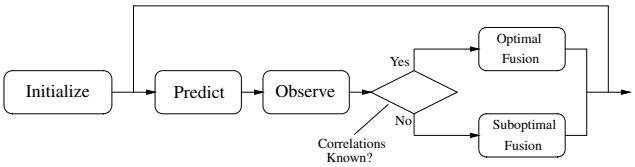


Figure 2: The generalized fusion architecture.

information, the estimated target state is set to the predicted target state. In either case, the loop returns to its waiting condition again.

The way in which this loop is instantiated on each node depends on four factors:

1. **Local sources of information.** Different fusion nodes have different means of acquiring information locally. Some nodes are equipped with sensors that can directly measure various target attributes with various levels of fidelity. Other nodes have indirect sources of information. For example, a node could use a terrain traversability database to constrain the motion of a target of a particular class type. Other types of nodes have no local source of information. Instead these can act as routers or fusion hubs, fusing estimates from several different nodes.

2. **Local processing capabilities.** Sensing nodes span from small, unattended grounds sensors through UAVs to entire command centers. Computational and storage costs can vary by many orders of magnitude and, as a result, some types of calculations simply cannot be carried out on some types of nodes. As an example, certain types of sensor data can be collected and transmitted very easily but are computationally very difficult to process[1]. Therefore, some nodes might simple broadcast data they collect. Other nodes could

[1] A proximity detector provides 1 bit of information for "detect" or "no-detect". However, if this information is to be used to refine the $(x, y)$ position of a target, the calculations involved become non-trivial.

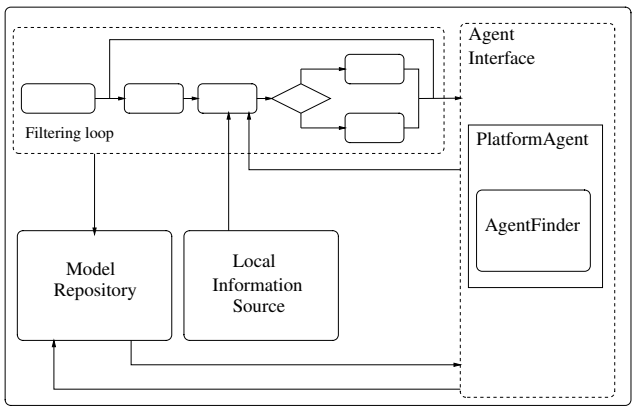


Figure 3: The structure of each agent.

process that data, but with different levels of fidelity.

3. **Target state representations used.** The target state representation is not fixed and immutable. It can change over time and across the network [11]. Some of these choices are limited by computational costs, some by communication overhead and some by security issues. For example, some attributes are privileged and can only be distributed to a subset of nodes in the network [12].

4. **Available bandwidth.** The bandwidth is a function of many factors including the hardware equipped on the node, current environmental conditions, and the current sensor geometry. Decreasing bandwidth causes a trade-off between the fidelity of the state estimate that is distributed and the rate at which it is distributed. As an example, the target estimate is a Gaussian mixture model which consists of multiple modes. As the bandwidth decreases, the update rate can only be maintained if the distributed track state merges modes and / or deletes attribute values.

Any software framework for generalized distributed data fusion must address these four factors. We propose to use an agent-based architecture.

# 3 Overview of the Approach

Each sensor node is modeled as a software agent. The reason for using agents is that they offer methods of communication across the network, lookup services, and advertisement capabilities. The structure of each agent is shown in Figure 3. The fusion loop from Figure 2 is augmented by a model repository and an agent interface layer. The model repository stores the mathematical components used to construct the different types of mathematical models. The agent interface layer mediates all inter agent communication. Most of this work is carried out through the *PlatformAgent*. The ***PlatformAgent*** represents the agent stored on a local platform and performs all operations associated with publishing and collecting observation models, observation data, and other agent interaction messages.

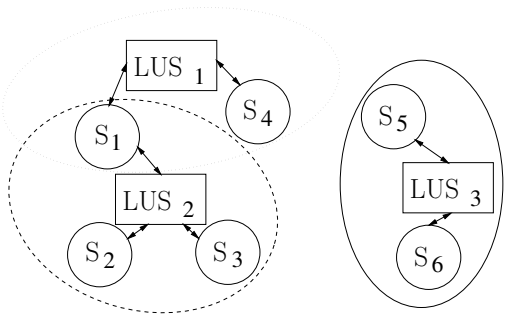


Figure 4: The communication topology for the LUS. Like the data fusion network, this is fully distributed and can have an arbitrary network topology.

All agents coordinate their activities through the distributed network of Look-Up-Services (LUSs) illustrated in Figure 4. The LUS facilitates coordination between nodes by maintaining a list of available nodes and their capabilities. These capabilities are expressed as a set of *advertisements* that include what information a node can provide (from its onboard sensors and track estimates), what models it contains, and what processing capabilities it has.

Nodes communicate with one another through a publish and subscribe mechanism. Suppose a node $X$ wishes to receive the set of attributes $\{A_X\}$ from a node $Y$. $X$ passes its registration request to $Y$ and $Y$ only delivers $\{A_X\}$.

To implement the agent services we use the CoABS Grid [4], hereafter referred to as The Grid. The Grid was originally developed for DARPA as a framework to control large systems of software agents. In particular, it is optimized to aid communication and intelligence gathering in military domains, an environment where bandwidth availability and connected hardware is constantly changing. The Grid wraps Sun's Jini services [13] and allows software agents to register and advertise capabilities to the LUS. These advertisements are customizable and change dynamically as an agent's capabilities change. To support changing the capabilities of an agent, the Grid supports *code mobility*: one agent can download portions of the codebase of another agent to permit it to use the new capabilities. We exploit this to allow agents to share models, attributes,

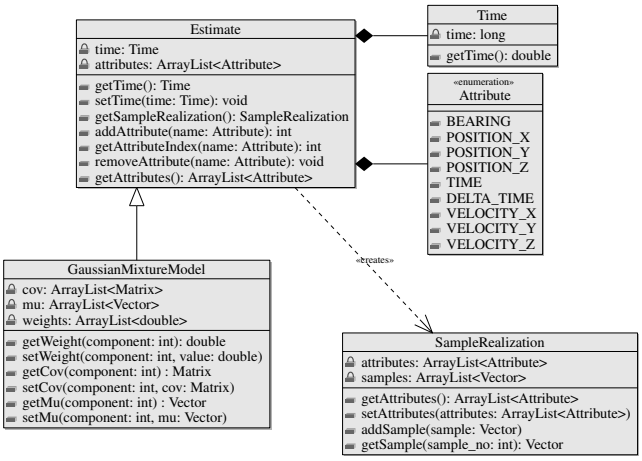


Figure 5: The generalized estimate representation. The estimate consists of a set of *Attribute*s, the *GaussianMixtureModel*, and the *Time*.

and their current states with other agents.

To map these capabilities into the fusion algorithms, we describe the architecture in three parts: the representation of the state estimate, the operations which occur within a node, and the operations that occur between nodes.

# 4 Estimate Representation

All uncertain quantities in the architecture are referred to as *Estimate*s. These include the target state, sensor observations, and information acquired from other sources such as databases. Because the state space representation can vary for the same target between different nodes, each estimate must maintain the metadata $s(k)$ to specify the structure of the estimate. This information is contained in the ***Estimate*** class, illustrated in Figure 5. This aggregates three separate classes: a container of *Attribute*s, the *GaussianMixtureModel* that specifies the probability distribution associated with the estimate, and the *Time* when the observation was taken.

Each *Attribute* instance stores a specific value about an entity. As explained above, we assume that there is an agreed upon standard for the naming and interpretation of each attribute. These standards should be defined by an ontology. Dorion's Command and Control Information Exchange Data Model (C2IEDM) [14], for example, considers the battlespace to be populated by a set of OBJECT-ITEMs. Attributes associated with each OBJECT-ITEM include the OBJECT-ITEM-LOCATION (where is it?), the OBJECT-ITEM-STATUS (how hostile is it?), and its OBJECT-ITEM-AFFILIATION (how is it affiliated?). By explicitly encoding the attribute information contained within the estimate, any node can understand what attribute information is available in the estimate that is supplied to it.

The ***GaussianMixtureModel*** class encodes the probability distribution used to encode the attribute values. In general, the uncertainty in one attribute is correlated with the uncertainty in another attribute. Therefore, the probability must be maintained within a single, joint structure. The *Estimate* class maintains a mapping of attributes values to indices within the *GaussianMixtureModel*.

The final class, ***Time***, specifies the time at which the estimate was calculated. *Time* could be based, for example, on the last time that an observation was received. In a general distributed data fusion network there is a significant issue with clock synchronization [15] and so the time value should, itself, maintain uncertainty. For this initial design we neglect this uncertainty but note that data fusion algorithms have been developed to be robust to measurements with unknown but bounded time delays [16].

The set of attributes contained within an *Estimate* can change through time. When an *Attribute* is to be added to an estimate, a new instance is created and registered with the *Estimate* using the *addAttribute* method. This method assigns a new index to the attribute, stores its value in the attribute list, and resizes the *GaussianMixtureModel* to include the new attribute. When an *Attribute* is deleted using the *removeAttribute*, it is removed from the attribute list, the *GaussianMixtureModel* is resized and all indices updated.

# 5 Operations Within a Single Node

The operation within a node consists of the steps outlined by the filtering loop shown in Figure 2: the estimate is initialized, predicted and updated. However, these steps can only be achieved by constructing the appropriate mathematical models and using them to transform the uncertainty associated with an estimate. In turn, these depend on the structure of the target estimate and the capabilities of the node.

## 5.1 Constructing the Mathematical Models

As explained above, the structure of the target state varies both through time and across the network. However, maintaining an exhaustive list of all possible mod-

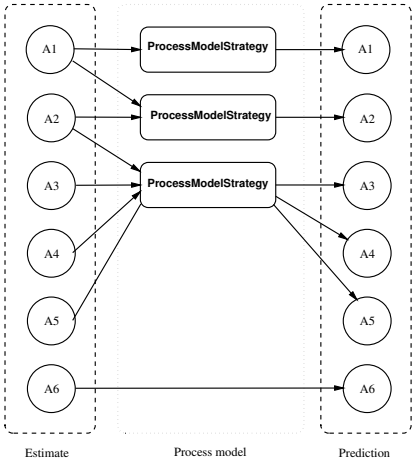


Figure 6: The composable process model is a set of *ProcessModelStrategy* objects. Each object acts on an input set of attributes and yields an output set. For those attributes without an explicit model (such as attribute A6) a constant-value mapping is provided.

els for all possible structures of the target state can become prohibitively expensive.

#### 5.1.1 Composable Models

We overcome these difficulties by making the models *composable*. The principle is illustrated in Figure 6 for a process model. The process model takes the state estimate at a time step $k$ and generates a new state estimate, with the same structure, at time step $k+1$. The model is composed of a set of *ProcessModelStrategy* objects. Each *ProcessModelStrategy* takes an input of a set of attributes and yields an output set of attributes. The input and output attribute sets for a *ProcessModelStrategy* do not have to be the same. For example, the *ProcessModelStrategy* might take position and velocity attribute information and predict a new position based on the assumption that the velocity is constant. Furthermore, default mappings can be provided if none are defined.

The same approach can be provided composable observation and initialization models.

However, to compose an appropriate *ProcessModel*, the appropriate strategy components must be chosen.

#### 5.1.2 Identifying the Correct Model Strategy Components

The choice of strategies to construct the process model is influenced by three factors. First, the collection of strategies are chosen such that all of the required attributes are updated. Second, it is possible to use models of different fidelity to describe the behavior of the *same* set of attributes. For example, a model of aircraft motion could assume constant velocity or it could use complicated aerodynamic equations to predict the aircraft's path. Finally, it is possible to have conflicts — two *ProcessModelStrategy* objects could attempt to update the same attribute value. Our solution is to use a *RuleEngine* to pick and choose the collection used. Each *ModelStrategy* advertises its input attributes and output attributes that define its inputs and outputs. In addition, each strategy can advertise something about its computational costs as well. The *RuleEngine* then picks and chooses the combination of models that provide the minimum cost. Our current strategy for avoiding conflicts is to use the value from the strategy object with the lowest cost.

### 5.2 Projecting Uncertainty Through the Models

The previous subsection described how strategy objects can be assembled to dynamically build models at runtime. However, the filtering algorithm propagates estimates with uncertainties associated with them. One means of combining the two together is to assume that *ModelStrategy* objects have to, for example, be able to provide Jacobians of their computations to be used in an extended Kalman Filter. However, in this paper we advocate the use of numerical techniques such as the Unscented Transform [17] or particle filters [18]. The reason for using these techniques is that each *ModelStrategy* object can be considered to be a "black box": it takes a set of inputs and transform them, but no knowledge of the internal structure is required. Furthermore, using a set of discrete samples, almost any mathematical quantity about the transformation can be calculated and thus almost any data fusion algorithm can be used.

The numerical scheme is implemented using the ***SampleRealization*** class. This class contains a set of (weighted) samples drawn from the *Estimate* together with an *Attribute* list to specify the indices for each attribute. The process and other models are applied to each sample in turn and the statistics of the transformed set are calculated.

## 6 Interaction Between the Nodes

Nodes do not operate in isolation; rather they function in a network and they must be able to exchange models and data.

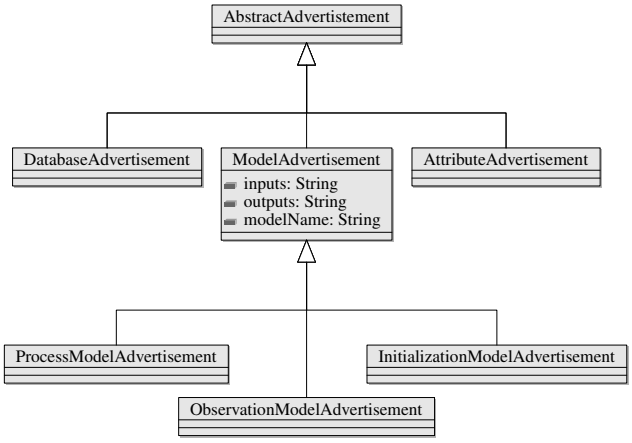


Figure 7: The Advertisement architecture.

## 6.1 Negotiation Between the Nodes

Fusion nodes exchange data and models dynamically via the Remote Method Invocation (RMI) capabilities that underly the Grid. For the fusion nodes to understand what data and/or models need to be exchanged, they use the Grid to advertise the capabilities of the agent and query the LUS to ascertain the capabilities of other agents and services.

When a fusion node starts up, the first thing it does is try to register with an instance of the Look UP Service (LUS) on the CoABS Grid. If the node cannot connect to an instance of a LUS, the fusion node has two strategies it can adopt. The first is that the node can create its own LUS. As explained above, multiple redundant LUSs can run on the same network without difficulty. However, some nodes (such as embedded sensors) might not have the capability to start their own LUS. In this case, the node operates in a standalone mode and polls for an LUS intermittently. Once a connection has been made, the node advertises its capabilities to the LUS. It also downloads from the LUS a list of capabilities of all of the nodes within its range.

Each node contains a rule engine that is written specifically for that type of fusion node. This rule engine contains all of the rules needed to query the LUS for other nodes that it can communicate with and have data and/or models that it can make use of locally. Alternatively, it could request that another node in the network perform a calculation for it, thus offloading some of its computational burden.

All of the fusion nodes derive their advertisements from the *AbstractAdvertisement* class of the Grid, from which a class hierarchy (Figure 7) of class objects is created that allows various types of meta data to be posted.

There are three main types of advertisements that are used in the architecture: *AttributeAdvertisement*s, *DatabaseAdvertisement*s, and *ModelAdvertisement*s. The ***ModelAdvertisement*** is further developed into advertisements for each of the types of models that are used by the fusion algorithms. These advertisements allow other fusion nodes to query the lookup service for raw data, new models for processing that data, and to discover what types of databases are on the network for querying.

Each fusion node locally keeps an object called an *AgentFinder*. The *AgentFinder* implements the Grid interface *ServiceListener*. The *ServiceListener* interface provides, via an event based mechanism, the ability to discover when new agents register or deregister from the network, and when the advertisements for any agent change. The *AgentFinder* keeps a local cache of this information and thus avoids the need for additional queries across the network. From its rule engine the fusion node will determine the best way to request data and models from other fusion nodes and then send out a request message requesting either data, or data and models. These requests are stored locally by a fusion node, and each time an observation/measurement is taken by one of its sensors, it will check a local table containing these requests in order to determine if other fusion nodes have requested the data. The node will determine which of the other nodes have requested the data, publish it, and if a model was also requested that will be included with the data. Through RMI, the other fusion node will not have to have that model's codebase locally; it will be able to download it via the capabilities of RMI and use it locally, even though it did not originally exist locally for that remote node.

## 6.2 Handling Changes in Network Topology

Over time the network topology will change as sensor nodes are added and removed. All agents in the network retain some knowledge of their topology through querying of the LUS(s), however what happens if no LUS is available? A LUS could be viewed as a point of failure thus creating a more centralised network. Without the LUS, the organization of the sensor nodes and distribution of their states would be very difficult to manage. So in order to maintain the decentralized nature of the network, i.e. it has no central node, the architecture must be robust enough to detect if an agent can see a LUS, and in the case of the LUS's absence, start its own LUS. Multiple instances of LUSs can share information across the network.

### 6.3 Communication Between Sensor Nodes

The process by which nodes communicate is simple. They already maintain local stores of dynamically updated information concerning the other nodes within their topology, from which a node can choose to request data and/or algorithms from another node. A simple rule engine is used, where the rules for use of data and algorithms is known *a priori*. The rule engine will determine the following:

- What data from the remote agent is readily usable locally?
- What data from the remote agent used in conjunction with local data, can be combined to create new attributes in the state estimate? For example, agent A contains estimates on range of a target. Agent B contains estimates on bearing of a target. If an algorithm for combining range and bearing estimates that returns an estimation of position is available, then create a new attribute field in the local state.
- What algorithms can be downloaded from the remote agent?
- If a sophisticated algorithm cannot be used locally due to computational constraints, can the remote agent perform the calculations?

It is interesting to note that an algorithm that is downloaded in order to give new capabilities locally can be kept locally and used more than once. For instance, a sensor that detects bearing and range is fused with another sensor's data adding a position attribute to its state. The sensor didn't originally possess a position algorithm; it was provided by the other sensor and downloaded through the mobility of code in the Grid. Should the two agents lose network connectivity to each other, the sensor can still use the algorithm locally, updating the position attribute as new observations on range and bearing are obtained. From a data fusion stand-point, this estimate would, over time, diverge and statistically more noise would enter into the calculation. However, should the two nodes come into contact again, the estimates can readily be updated, once again.

## 7 Example

Consider the scenario shown in Figure 8. A single target is being tracked by two sets of sensors: an unattended seismic ground sensor (USGS) and an unmanned aerial vehicle (UAV). The target is moving

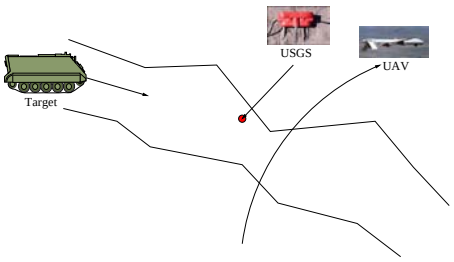


Figure 8: The scenario.

through a mountain pass, which is being monitored at first only by the USGS. The USGS observes seismic data and, from this, it can estimate the range to the target and its spectrogram. The USGS state vector consists of the following states:

$$\text{USGS} = (\text{range, spectrogram})$$

This state is updated at the command center where a UAV is dispatched for imagery reconnaissance. The UAV carries LIDAR and EO/IR imagery sensors. It contains attributes for imagery both infrared and visual, position estimated from the imagery and the estimated GPS position of the UAV, and a range-to-target measured in kilometers from the LIDAR. At first, since the USGS is within the mountain pass, until the UAV is overhead it will not be able to communicate directly with the USGS. Once it has reached the pass, and is able to see the target with its sensors, the UAV will create a state estimate consisting of the following values:

$$\text{UAV} = (\text{target\_xyz, uav\_xyz, range, ir\_image, visual\_image})$$

Note that both sensors start by containing only the attributes that their respective onboard sensors can measure. Over time, as the two nodes continue to track the target, they will exchange estimates from their states. If it is determined that data from one node can be used by another node, they will exchange that data. This may cause the state of one of the nodes to be updated with a new value not previously contained within the state. Referring to our example, the UAV contains the attribute for position within its state, and the seismic sensor contains a range-to-target. These are values that can readily be fused together to create new estimates of position and range. The UAV would provide the USGS with this data, and, if necessary, a model for calculation. At this point the seismic sensor would add an attribute for position within its state and the state would become:

$$\text{USGS} = (\text{range, spectrogram, target\_xyz})$$

At the same time, the UAV's state becomes more accurate since its position attribute is being fused with

the range attribute from the USGS. Notice that these two sensor nodes do not need to exchange data from the imagery, nor signature and intensity of the signal contained within the spectrogram, since these values are not readily fusable with any of the other values contained within either node.

# 8 Conclusions

This paper has described an agent-based architecture for heterogeneous distributed data fusion algorithms. The algorithm generalizes the notion of a target state to a dynamically adjustable collection of attributes with uncertain values. Mathematical models can be constructed in real time on the basis of their inputs and outputs and their computational costs. We have discussed how advertisements can be used to refine the communication between agents and we have illustrated these results in a target tracking example.

We will be extending this architecture in two respects. First, we shall integrate our architecture using Fuselet Technology [5]. Second, we shall extend the representation of state to support complicated attribute types such as non-numeric, discrete quantities.

# References


[1] K. S. J. Pister, J. M. Kahn, and B. E. Boser. Smart dust: Wireless networks of millimeter-scale sensor nodes. Technical report, Robotics and Intelligent Machines Laboratory,University of California at Berkeley, 1999. Highlight Article in 1999 Electronics Research Laboratory Research Summary.

[2] Mario Zagalj, Jean-Pierre Hubaux, and Christian Enz. Minimum-energy broadcast in all-wireless networks:: Np-completeness and distribution issues. In *International Conference on Mobile Computing and Networking: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 172–182, Atlanta, Georgia, USA, 23–28 September 2002.

[3] J. J. Garcia-Luna-Aceves and Ewerton L. Madruga. The core assisted mesh protocol. *IEEE Journal on Selected Areas in Communications, Special Issue on Ad-Hoc Networks*, 17(8):1380–1394, August 1999.

[4] GlobalInfotek. Control of agent based systems [online, cited 03 January 2006]. Available from: `http://coabs.globalinfotek.com`.

[5] Fuselet Technology Overview [online, cited 14 February 2006]. Available from: `http://www.fuselet.org/tech-overview`.

[6] SOAP Version 1.2 Part 0: Primer [online, cited 14 February 2006]. Available from: `http://www.w3.org/TR/soap12-part0/`.

[7] S. Grime and H. F. Durrant-Whyte. Data fusion in decentralized sensor fusion networks. *Control Engineering Practice*, 2(5):849–863, 1994.

[8] S. J. Julier and J. K. Uhlmann. General Decentralized Data Fusion With Covariance Intersection (CI). In D. Hall and J. Llinas, editor, *Handbook of Data Fusion*. CRC Press, Boca Raton FL, USA, 2001.

[9] Ramana Rao Kompella and Alex C. Snoeren. Practical Lazy Scheduling in Sensor Networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 280–291, 5–7 November 2003.

[10] Bayes++ [online, cited 14 February 2006]. Available from: `http://bayesclasses.sourceforge.net/Bayes++.html`.

[11] S. J. Julier and H. F. Durrant-Whyte. A horizontal model fusion paradigm. In *The Proceedings of the SPIE AeroSense Conference: Navigation and Control Technologies for Unmannged Systems*, volume 2738, pages 37–48, Orlando, FL, USA, April 1996.

[12] Radiant mercury. Technical report, SSC San Diego, San Diego, CA, USA, November 2003. Available from: `http://www.fas.org/irp/program/disseminate/radiant_mercury.pdf` [cited 31 January 2006].

[13] Jini.org — The Community Resource for Jini Technology [online, cited 14 February 2006]. Available from: `http://www.jini.org`.

[14] Eric Dorion, Christopher J. Matheus, and Mieczyslaw M. Kokar. Towards a formal ontology for military coalitions operation. In *Proceedings of the 10th Internatoinal Command & Control Research and Technology Symposium, The Future of C2: Coalition Interoperability*, McClean, VA, USA, June 2005.

[15] P. Blum, L. Meier, L. Thiele. Improved Interval-Based Clock Synchronization in Sensor Networks. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks (IPSN'04)*, pages 349–358, Berkeley, CA, USA, 26–27 April 2004.

[16] S. Julier and J. K. Uhlmann. Fusion of time delayed measurements with uncertain time delays. In *2005 American Control Conference*, Portland, OR, USA, 8 – 10 June 2005.

[17] S. J. Julier, J. K. Uhlmann and H. F. Durrant-Whyte. A new approach for the nonlinear transformation of means and covariances in filters and estimators. *IEEE Transactions on Automatic Control*, 45(3):477–482, March 2000.

[18] Branko Ristic, Sanjeev Arulampalam, and Neil Gordon. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, 2004.