Semantic Agent Technologies for Tactical Sensor Networks

Guofei Jiang[a], Wayne Chung[b] and George Cybenko[b]
[a]Institute for Security Technology Studies, Dartmouth College, Hanover, NH USA 03755
[b]Thayer School of Engineering, Dartmouth College, Hanover, NH USA 03755

Abstract

Recent advances in wireless communication and microelectronics have enabled the development of low-cost sensor devices leading to interest in large-scale sensor networks for military applications. Sensor networks consist of large numbers of networked sensors that can be dynamically deployed and used for tactical situational awareness. One critical challenge is how to dynamically integrate these sensor networks with information fusion processes to support real-time sensing, exploitation and decision-making in a rich tactical environment. In this paper, we describe our work on an extensible prototype to address the challenge. The prototype and its constituent technologies provide a proof-of-concept that demonstrates several fundamental new approaches for implementing next generation battlefield information systems. Many cutting-edge technologies are used to implement this system, including semantic web, web services, peer-to-peer network and content-based routing. This prototype system is able to dynamically integrate various distributed sensors and multi-level information fusion services into new applications and run them across a distributed network to support different mission goals. Agent technology plays a role in two fundamental ways: resources are described, located and tasked using semantic descriptions based on ontologies and semantic services; tracking, fusion and decision-making logic is implemented using agent objects and semantic descriptions as well.

Keywords: Sensor network, information fusion, peer-to-peer network, DAML, semantics, dynamic integration, web service, content-based routing

1. Introduction

Sensor networks consist of many networked sensor nodes that are capable of multiple sensing modalities. Interest in such systems is rapidly expanding in modern military and homeland security applications. While each individual node may only have limited sensing, computing, and communication capability, a large number of such nodes can coordinate among themselves and with C4ISR logic to perform large-scale sensing tasks.

At Dartmouth, we have been working with about 100 re-configurable wireless sensor platforms that include: Global Positioning System (GPS) capabilities, wireless (RF) communications, iButton and serial sensor capabilities, simple microprocessor control (Intel 8051 processor) and self-contained power. One of the sensors is depicted in Figure 1 below. Details of these devices and the ad hoc routing algorithms for establishing wireless network connectivity can be found in [1].

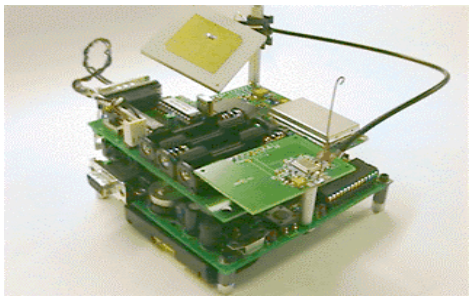

Figure 1: Dartmouth Re-configurable Wireless Sensor Platform

According to a survey on sensor networks [2], currently most research efforts focus on sensor hardware design such as the *Smart Dust* mote [3], protocols and algorithms design for different network layers, which include the physical layer, the data link layer, the network layer, the transport layer and the application layer. The positions of sensor nodes are usually not predetermined (e.g. airborne sensors), so that sensor nodes may need self-organizing routing algorithms to setup effective communication networks. Many ad-hoc routing algorithms and power-saving methods are proposed to address such issues, such as the direct diffusion algorithm [4]. At the application layer, some projects such as Cornell's COUGAR project [5] build systems to query and merge sensor's data. However, high-level sensor information systems are needed to integrate sensor networks with level 0-4 information fusion processes to support real-time sensing, exploitation, and decision-making in a rich tactical information environment. Level 0-4 information fusion processes include Object understanding, Scene/Formation understanding, Threat understanding and Effects selection and planning. Sensor information systems process and fuse data from sensor networks, transfer data into information and knowledge, and deliver the right information to the right place at the right time. We believe that sensor information systems can further leverage the power of large-scale sensor networks in tactical battlefield environments.

To this end, we have implemented our sensor networks as sensor web services that can be dynamically integrated with distributed components in a networked environment. Heterogeneous sensors' capabilities are described with DAML (DARPA Agent Markup Language) [6] for service brokering so that they can be dynamically discovered and used by various end-user and intermediate processing applications. We have developed an extensible framework with many advanced capabilities as a proof-of-concept to demonstrate several fundamental new ideas for implementing next generation sensor information systems, which integrate sensor networks with level 0-4 information fusion services in a networked environment. This prototype system is able to dynamically integrate various distributed sensors and fusion services into new applications and run them across a distributed network to support different mission goals.

The remainder of the paper is organized as follows: In Section 2, we give an overview of our system architecture as well as a dynamic information flow framework. From Section 3 to Section 5, technical details about the main components are discussed, which includes sensor web services, a web services composer and knowledge base, and peer-to-peer fuselets networks. In Section 6, we introduce a generic fusion engine, called TRAFEN, and message-oriented middleware, which are currently under development to extend our sensor information system framework.

2. System Overview

2.1 Prototype architecture

Our prototype system consists of several main components: physical sensors, sensor web services, composer and knowledge base, peer-to-peer fuselet network, signal processing fuselets, tracking application, and end-user view. The system architecture is shown in Figure 2. In this section we give a brief introduction to these components and architecture. The technical details about these components will be discussed in the remaining sections.

Physical sensors: Several acoustic sensors are implemented to support the ground-vehicle tracking scenario. These sensors are composed of microphones connected to laptop computers. We implemented software with the Java Sound API to acquire the sound signal from microphones and formatted the signal into a WAVE-format data stream. The laptop running the software pushes the WAVE data stream to MySQL databases periodically. A hierarchical sensor ontology and acoustic sensor ontology have been created with DAML to describe the properties of each physical sensor. Sensors are marked up with these ontologies using tools such as RDF Instance Creator (RIC). As a result, these sensor instances can be dynamically discovered by a non-functional property search during service brokering.

Sensor web services: Sensor web service is used as a "wrapper" to interface between other components and the physical sensors. From one perspective, sensor web services can use any specific and internal approaches to communicate with the physical sensors. From another perspective, with cutting-edge Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL) technologies, sensor web services can be dynamically interfaced and invoked by other components on the network. DARPA Agent Markup Language – Service Ontology (DAML-S) has been used to describe and markup the sensor web services, using WSDL as the service grounding. The associated sensor's non-

functional properties are included and described in the “service profile” part of the DAML-S ontology. The sensor DAML-S instances are published in a knowledge base for service brokering.

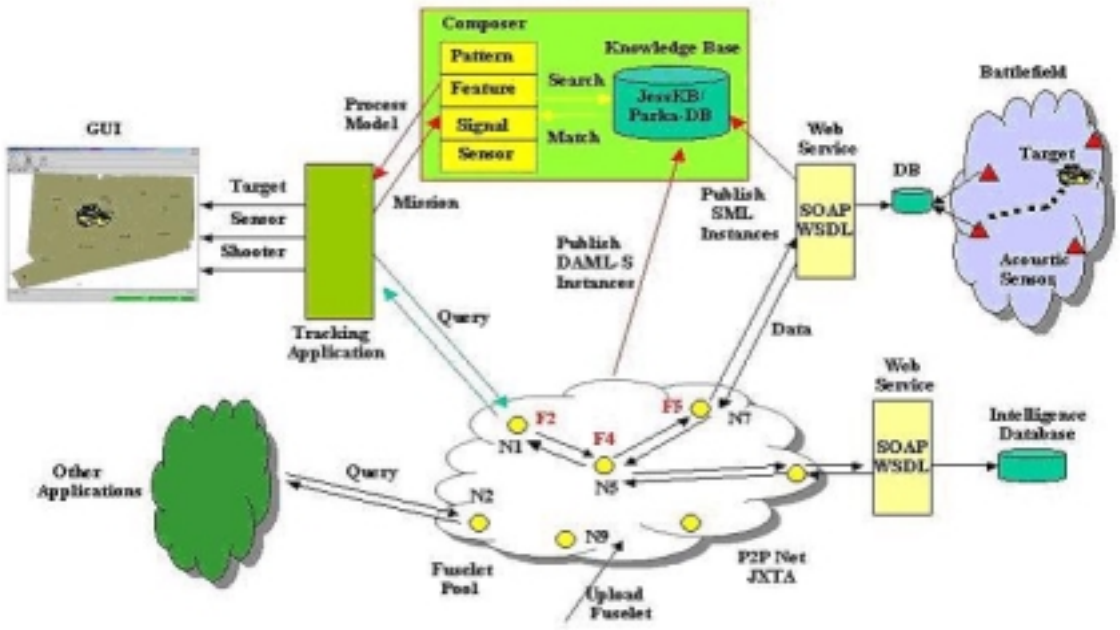

Figure 2: Prototype architecture

Composer and knowledge base (KB): The service composition system serves as a human-machine interface to combine a human’s domain knowledge together with a machine’s searching and matching capability. It includes both a composer and an inference engine. The composer is the user interface that handles the communication between the human operator and the engine. The inference engine stores the information about the services in its Knowledge Base and has the capability to find matchable services. In our system the DAML-S instances for sensors and fuselets are loaded to the inference engine, which is a DAML reasoner built in Prolog. The ontological information written in DAML is converted to RDF triples and loaded to the Knowledge Base, JessKB, in our system. The engine has built-in axioms for DAML inferencing rules, which are applied to the facts in the KB to find the entailments. The composer presents a user interface where an operator can view the available services, browse their detailed information, filter the services by entering constraints on the non-functional properties, and compose different services by chaining them. This system supports searching and matching of both non-functional and functional properties. The composer and KB were implemented by Evren Sirin and Jim Hendler at University of Maryland as a joint effort.

Peer-to-Peer fuselet network: Fuselets are lightweight data fusion algorithms that can be dynamically deployed in a network environment. They are hosted on fuselet servers and these fuselet servers are organized with a peer-to-peer network to enhance reliability and scalability. The P2P network is implemented using Sun Microsystem’s JXTA platform. A content-based routing mechanism is implemented to support dynamic and reliable query routing in this network. As a result, queries with common sub-expressions can be dynamically merged along the route to the sensors so that the data stream will not be pulled and processed multiple times along the route back to the clients. The fuselet server consists of three major components: the gateway module, the fuselets module and the routing module. The gateway module is responsible for parsing queries, forwarding queries and data, loading fuselets, invoking fuselets, and managing the fuselets’ execution. The server is implemented as a generic web service to interface with other nodes. The fuselets module is responsible for managing the dynamic fuselets library on the server. The routing module is responsible for discovering new peers, updating routing tables and maintaining the dynamic P2P network.

Signal processing fuselets: Fuselets are reusable, shareable and linkable computing services hosted on the P2P network. Several fuselets are implemented for our tracking application, such as Band-pass filtering, Root Mean Square (RMS) power calculator and sensor service invoker. The complied code of these fuselets is deployed to the fuselets servers for use. The gateway module of these servers can dynamically load the code based on incoming requests. Meanwhile the routing module automatically distributes its new fuselet indexing information across P2P network and content-based routing tables are updated. Fuselets are described with the DAML-S service ontology and their markup instances are published in the Knowledge Base for service brokering.

Tracking application and user view: The tracking application dynamically assembles all components in the prototype to support a "target tracking" mission. The application is designed to demonstrate the semantic integration and interoperability process. Acoustic signals generated by the moving target are acquired by several acoustic sensors in the field and are stored in databases. The application can pull the signal data from the databases using sensor web services. Band-pass filter fuselets are used to process the sound signals and retain signals of certain frequency signatures while dismissing others. The RMS power calculator fuselets are used to estimate average power of retained signals over a certain period of time. With the average signal power and the sensor location, a multi-sensor fusion algorithm is employed to determine the target position. The estimated position of the target, as well as those of our own assets and sensors, is shown on a Graphical User Interface with digital maps.

2.2 Dynamic workflow

In our system, sensors and fuselets are described using DAML-S service ontologies with WSDL as the service grounding. Non-functional properties of sensors are described with hierarchical sensor ontologies and they are included in the "service profile" of the DAML-S ontologies. Their instances are published in the Knowledge Base for service brokering. Various sensors acquire data from battlefields and output data to databases. The data can be pulled by various applications through sensor web services. Meanwhile various fuselets are developed and deployed to the peer-to-peer fuselets network. Every node in the network distributes its fuselet indexing information to other peers and content-based routing table is created based on these indexing information.

For a specific mission in a field, operators use the composer to search the Knowledge Base and discover the sensors capable of sensing events for that mission. E.g. in our tracking application, acoustics sensors are discovered in that field. With their knowledge, operators know that they can accomplish the mission by fusing the data from several sensors in that field and processing that data through signal processing fuselets. With the composer, operators view the available fuselet services, browse their detailed information, search the fuselet services by entering constraints on the non-functional properties, and compose a process model by chaining fuselets services and sensors. While the operator inputs specific knowledge to compose a process model for a mission, the composer itself supports non-functional properties search and inputs-outputs match to chain services.

Then the mission application receives the process model output from the composer and formulates it as a SOAP query message. The query message is routed across the P2P fuselets network based on content-based routing and eventually the sensor web services are invoked. Data streams are pulled from the sensor and enclosed in a SOAP message as data attachments. Then the returned data message is routed back through signal processing fuselets sequentially for data processing. Eventually the processed data is returned to the mission application and results are computed and output to a graphical user view. In this way, distributed semantic services are dynamically discovered, composed and executed to support a dynamic integration and accomplish a mission "on the fly".

3. Sensor Web Services

Building a system capable of semantic brokering requires a standard ontology to describe the resources in the network. In our prototype, we created sensor ontologies using DAML to describe the capabilities of sensor networks. The general sensor ontology includes major properties such as sensor location and sensing mechanism. Each major property can be specified with low-level properties. For example, the following is a property named sensing mechanism. Every sensor has to choose at least one of eight sensing mechanisms to describe itself.

```
<daml:Class rdf:ID="SensingMechanismType">
                    <daml:oneOf rdf:parseType="daml:collection">
                    <SensingMechanismType rdf:ID="ElectricMagnetic"/>
                              <SensingMechanismType rdf:ID="Mechanical"/>
<SensingMechanismType rdf:ID="Biological"/>
                              <SensingMechanismType rdf:ID="Chemical"/>
                              <SensingMechanismType rdf:ID="Radioactive"/>
                              <SensingMechanismType rdf:ID="Cyber"/>
                              <SensingMechanismType rdf:ID="Optical"/>
                              <SensingMechanismType rdf:ID="Other"/>
                    </daml:oneOf>
</daml:Class>
```

Sensors are marked up with the ontology via some tools such as RIC. The sensor instances can be dynamically discovered by non-functional property searchs in service brokering.

Bound with popular HTTP protocol, web server and XML technology, Web Service architecture provides an open, standard and extensible interface for communication among different applications and components. Sensor web service, as shown in Figure 3, is used as "a wrapper" to interface between other components and the physical sensors. On one side, sensor web service can use any specific and internal approaches to communicate with the physical sensors. On another side, with cutting-edge SOAP and WSDL technologies, sensor web service can be dynamically interfaced and invoked by other components in the network. In our system, after sensor web service receives incoming SOAP requests for acoustic sensor data, it uses Java database connectivity (JDBC) to pull the data from the database and encloses the audio data as a MIME data attachment in its response SOAP message.

Figure 3: Sensor Web Services

Though WSDL describes the programming interface of a web service, it doesn't include any semantics of services, parameters, or conditions. For example, what does the name of a parameter mean for that service? In our prototype, DAML-S ontology is employed to describe the semantics of services. DAML-S is a set of ontologies to establish a framework within which the semantics of web services may be described. DAML-S partitions a semantic web service description into three components: profile, process model, and grounding. ServiceProfile describes what the service does by specifying the input and output types, preconditions, and effects. ProcessModel describes how the service works; each service is either an AtomicProcess that is executed directly or a CompositeProcess that is a combination of other subprocesses. Grounding contains the details of how an agent can access a service by specifying a communications protocol, parameters to be used in the protocol, and the serialization techniques to be employed for the communication. As we mentioned above, sensors are marked up with sensor ontology and their instances are included in the "service profile" part of DAML-S. Sensor's WSDL is used as a service grounding approach in the "grounding" part of DAML-S. The sensor DAML-S instances are published in the Knowledge Base for service brokering.

4. Composer and Knowledge Base

The service composition system has two basic components: composer and inference engine. The inference engine stores the information about the services in its Knowledge Base (KB) and has the capability to find matching services. The composer is the user interface that handles the communication between the human operator and the engine, which is shown in Figure 4. The inference engine is a DAML reasoner built on Prolog. Ontological information written in DAML is converted to RDF triples and loaded to the KB. The engine has the built-in axioms for DAML inferencing rules. These axioms are applied to the facts in the KB to find the entailments such as the class inheritance relation between two classes that are separated several steps away in the hierarchy tree. The service and sensor descriptions are both expressed using DAML statements so no special handling is required to distinguish these different resource types. This system supports searching and matching of both non-functional and functional properties. See details in [7].

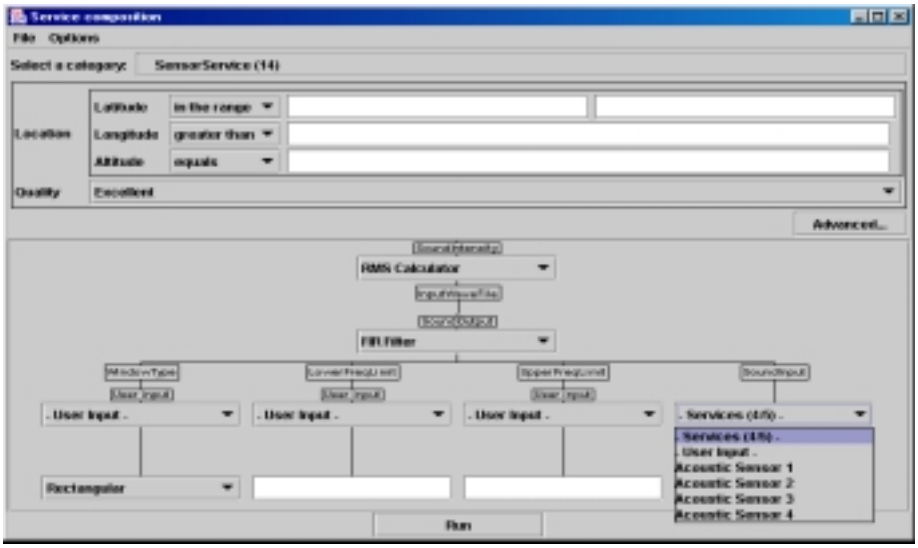

Figure 4: Composer

In the sensor network environment, there are two types of services: sensor web services and fuselet services. While sensor web services provide sensor data, fuselet services are reusable, shareable and linkable computing services that process sensor data. The composition of semantic services in the sensor network is accomplished as follows: Each sensor is associated with a semantic web service that contains the DAML-S description of the service. This service description contains a link to the description of the sensor that is marked up in DAML language. These sensor and fuselet DAML-S descriptions are loaded to an inference engine and the composer program presents a user interface where an operator can view the available services, browse their detailed information, compose different services by chaining them and filter the services by entering constraints on the non-functional attributes of the service such as the properties of its corresponding sensor.

During the composition process, the composer allows the user to create a composition of services by presenting the available choices at each step. The user is first presented with all the available services registered to the engine. When a service is selected from the list, a query is sent to the KB to retrieve the information about the inputs of the service if any exist. For each of the inputs, a new query is run to get the list of the possible services that can supply the data for this input. Composer also shows different types of services available in the system and filters the results based on the constraints defined by the user on the attributes of a service.

After the composition, the composer outputs a query that outlines the sequence of chained services and their parameters. The query is submitted to the Peer-to-Peer fuselet network that uses content-based routing to deliver the query message.

At each node along the route to the end sensors, the service and parameter information is parsed from the query message and used to locate dynamically and load the expected service. The following is an example of the query message format:

```
<passMessage>
        <classname NameofClass="Service1"
                Method="Service1_Invoke">
                <Param>Service1_Param</Param>
                <classname NameofClass="Service2"
                        Method="Service2_Invoke">
                        <Param>Service2_Param</Param>
                </classname>
        </classname>
</passMessage>
```

5. Peer-to-Peer Fuselet Network

The peer-to-peer fuselet network is designed to create an open, transparent network of interconnected nodes that contain fuselets services. Fuselets are lightweight data fusion algorithms that can be deployed in a network environment. They are hosted on fuselets servers, nodes, and these fuselets servers are organized with a peer-to-peer network to achieve reliability and scalability. The P2P network is implemented on Sun Microsystem's JXTA platform. Content-based routing mechanisms are implemented to support dynamic and reliable query routing in this network. The framework of a P2P node is shown in Figure 5. The framework consists of three major components: the gateway module, the fuselet module, and the routing module. The gateway module is responsible for parsing queries, forwarding queries, forwarding data, loading fuselets, invoking fuselets, and managing fuselet execution. It is implemented as a generic web service to interface with other nodes. The fuselet module is responsible for managing the dynamic fuselet library on the server. The routing module is responsible for discovering new peers, updating its routing tables, and maintaining the dynamic P2P network.

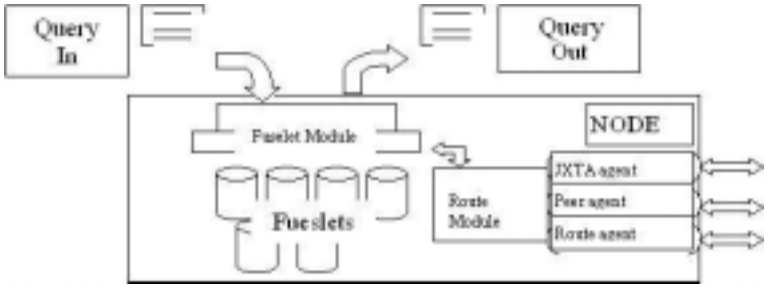

Figure 5: The framework of a P2P fuselets node

Content-based query routing: P2P fuselet network is an overlay network built on Internet infrastructure. Content-based query routing is implemented on top of IP routing. The routing service of each node discovers its peers and establishes content-based routing tables. The routing table includes fuselets indexing information and IP numbers of their physical hosts.

The composite query consists of a series of specially formatted XML tags and is generally passed using SOAP messages. After a fuselet node receives the composite query, it searches its content-based routing table to see which node has the fuselet to execute the first service in the query. If the current node has the exact fuselet, it removes the first process from the composite query and forwards the rest of the query to the next node, and waits for a response. Once the response is received, it is used as a parameter to invoke the fuselet service. The result is enclosed in a SOAP message as an attachment and sent back to the previous calling node. At one point, if one node splits the query to several parallel sub-queries and forwards them to several different nodes, multi-threads are created to wait for the responses from these nodes. Only after all responses are received as parameters, the fuselet on the current node is invoked.

The query does not specify the node but the name of the fuselet services required. Content-based routing dynamically searches and resolves the fuselet name to a physical node that runs the fuselets. In the SOAP message, each fuselet service is represented with the name of the class and the name of the method, along with parameters. Sequential classes and methods in the query are organized with a tree structure. The outer most tag is used to identify this specific SOAP message. Queries with common sub-expressions can be dynamically merged along the route to the sensors so that the data stream will not be pulled and processed multiple times along the route back to the clients. This dynamic content-based routing process can dramatically reduce both network traffic and computational loads in the fuselet network. See details on query routing in [8].

Dynamic fuselets loading: The current framework is designed as a generic fuselets container so that semantic fuselet services can be dynamically integrated. The fuselet system consists of individual fuselet classes and a dynamic loader. The dynamic loader is responsible for managing the class loader within the Java Virtual Machine. The fuselets are algorithm classes that can be invoked by other nodes or end-users in the network. They can be added and removed from the node dynamically and can also be added to multiple nodes for redundancy.

The Dynamic loader makes use of the java reflection library. Java reflection is designed to allow for the dynamic loading and instancing of classes that are not known until run time. This feature is very helpful in designing modular code and dynamic code, since code can be written and tested for classes and methods that have not been created or are not yet known by the programmer. With reflection, the dynamic loader casts a generic class object to match the class specified in the query, under the "NameofClass" field. Meanwhile, the dynamic loader can also cast a generic method list to the methods listed in the query. Once the class is instanced and the method is determined, reflection is invoked and the necessary parameters are passed to the call. As long as the inputs and outputs of a fuselet are kept consistent, the underlying methods and implementation can be modified at will.

Fuselets are described in DAML-S ontology with WSDL as the service grounding. DAML-S instances are published in the Knowledge base for service brokering. After a fuselet is uploaded to a node, the routing module of the node distributes its fuselets indexing information to its peers and the routing tables are updated.

6. The TRAFEN Fusion Engine

The TRAFEN fusion engine and related message-oriented publish/subscribe middleware are currently under development to extend our existing prototype of sensor information systems to support high-level information fusion. The extended framework is shown in Figure 6, where the data-driven part is the existing prototype architecture shown in Figure 2. As we discussed above, with the composer and knowledge base, the operators view the available sensors and fuselet services, browse their detailed information, search the fuselet services by entering constraints on the non-functional properties, and compose a workflow by chaining fuselets services and sensors. Agents are independent components (e.g. the tracking application in Figure 2) that periodically send the composed queries to the sensor networks. After agents receive processed data from the sensor networks, they further process the data with their specific logic to form observation messages or events. These messages are published into a message-oriented middleware with specific topics. On the other side, TRAFEN, the fusion engine subscribes to messages/events from the middleware. These messages possibly originate from various agents in the battlefields, from intelligence database, and/or human observations etc... Multiple hypothesis tracking algorithms (MHT) are fed with these ad-hoc messages and sort out the hidden relationship among them to build various hypotheses. Hypothesis results and predictions can be used to support decision-making processes. Moreover, they can be published back to the middleware as message sources for other fusion engines. Sensor networks can also subscribe to these hypotheses and predictions as feedback to optimize their sensing tasks. Technical details about message-oriented middleware and TRAFEN engine are discussed in the following sections.

Message-Oriented Middleware (MOM): Messaging is a method of communication between software components or applications. MOM is a middleware communication mechanism that allows different loosely coupled applications to communicate with each other in an asynchronous connectionless way. The sender and the receiver don't have to be available at the same time in order to communicate via MOM. They don't need to know anything about each other

except the message format. There are two general types of messaging: message queuing and publish-subscribe. Message queuing is a point-to-point communication model. Publish-subscribe provides delivery to more than one receiver at a time. It is particularly useful when many receivers need the same data, when high performance is required, or when the data has to be delivered in real-time. Theoretically, messages are clustered in the MOM with specific topics or queues. In our framework, we are building a semantic MOM based on Sun's Java Messaging Service. Both topics and message contents are marked up with DAML so that they can have embedded semantics. Agents push their observation messages to the MOM with specific topics. On the other side, the TRAFEN engine receives all messages within the same topic from the MOM and starts the multiple hypotheses tracking process. By clustering ad-hoc messages with specific topics on the MOM, the entire set of targets and measurements are divided into independent groups. Instead of solving one large problem, a number of smaller problems are solved in parallel. Therefore, we can manage the size of the hypothesis space and computing complexity of each MHT algorithm.

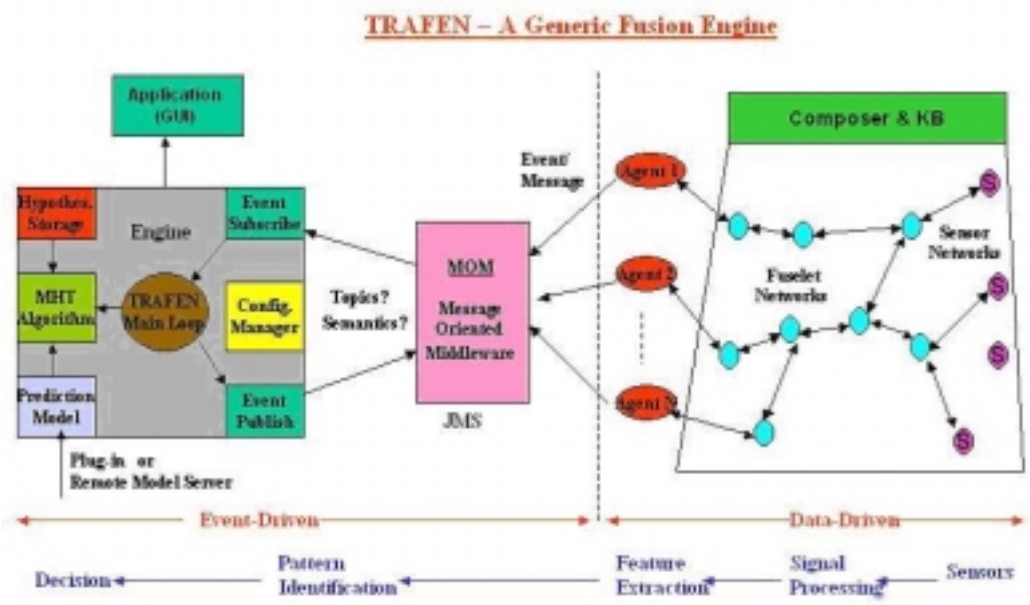

Figure 6: The TRAFEN fusion engine

TRAFEN: TRAFEN is designed as a generic fusion engine to implement multiple hypothesis tracking/fusion processes. After TRAFEN receives observation messages/events from the MOM, it uses Bayesian formulation to determine the probabilities of alternative data-target association hypotheses. A hypothesis is consisted of a sequence of related events along the time. Data association algorithms are used to cluster events for different hypotheses. Multiple target tracking algorithms such as Reid's algorithm [9] recursively calculate the probability on how likely the new observations are associated with every existing hypotheses. Then the new observation is added into the hypothesis with the maximum likelihood and the set of hypotheses are updated. Tracking target's prediction model is needed in order to compute the conditional probability on how likely an observation is associated with the existing hypotheses. A prediction model describes the state transition of a tracking target, which evolves with time according to specific known laws such as the kinematical constraint of the tracking target. State transition model can be described with state equations, hidden Markov models and rules-based models etc. A Kalman filter [10] (or other least-squares methods) may be used to predict the track state at future times.

The TRAFEN engine consists of several main components: MHT algorithm, Prediction model, Hypothesis management and storage, Configuration manager and Event subscriber/publisher. Various MHT algorithms are implemented as modules in our engine architecture. For a specific fusion task, users specify the event publish/subscribe topics and the MHT algorithm via the configuration manager. They are initialized in the TRAFEN main loop when an engine starts. Every engine can have a pool of prediction models for various tracking targets. Every prediction model is implemented as a plug-in module, which can also be invoked as a remote service. After the main loop parses the incoming observation message, a specific prediction model is dynamically loaded with the tracking target information from the message.

Hypothesis management involves several operations of hypothesis maintenance. Observations not assigned to existing tracks initiate new tentative tracks. A tentative track becomes a confirmed track upon satisfying some quality tests. Low quality tracks, as determined by the update history, are deleted.

7. Conclusions

With a rapid expanding network of sensors in the battlefield, one critical challenge is how to integrate dynamically networked sensors with multi-level information fusion processes to support real-time sensing, exploitation and decision-making. During the fusion process, sensor data is exploited and transferred to information and knowledge. Further the right information is delivered to the right place at the right time. While most research focuses on low-level protocols, routing algorithms, and design of sensor networks, we develop an extensible prototype framework with basic capability to demonstrate next generation sensor information system. In our system, sensor networks can be dynamically discovered, composed and integrated with distributed information fusion services to support various missions.

Acknowledgements: This research was a joint effort by Dartmouth College and University of Maryland (UMD) at College Park. Evren Sirin and Jim Hendler at UMD implemented the composer. At Dartmouth, Annarita Giani, Yong Sheng, Glenn T. Nofsinger, Han Li, Diego Hernando, Paul Thompson participated in the implementation of the prototype system. The authors are grateful for their contributions. This research was partially supported by: Defense Advanced Research Projects Agency projects F30602-00-2-0585 and F30602-98-2-0107; the Office of Justice Programs, National Institute of Justice, Department of Justice award number 2000-DT-CX-K001 (S-1). Points of view in this document are those of the authors and do not necessarily represent the official position of the sponsoring agencies or the U.S. Government.

References

[1] Michael Corr, Masters Thesis, Thayer School of Engineering, Dartmouth College, Hanover, NH 2001 http://actcomm.dartmouth.edu/~mgcorr/papers/Corr_Thesis.pdf

[2] I. Akyildiz, W. Su, Y. Sankarasubramanian and E. Cayircl, “A Survey on Sensor Networks”, *IEEE Communication Magazine*, August, 2002.

[3] J. M. Kahn, R.H. Katz, and K.S.J. Pister, “Next Century Challenges: Mobile Networking for Smart Dust,” *Proc. ACM MobiCom ’99*, Washington, DC, 1999.

[4] C. Intanagonwiwat, R. Govindan, and D. Estrin, “Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks,” *Proc. ACM MobiCom’00*, Boston, MA, 2000.

[5] Cornell COUGAR Device Database Project: http://www.cs.cornell.edu/database/cougar/index.htm

[6] DARPA Agent Markup Language: http://www.daml.org

[7] E. Sirin, J. Hendler, B. Parsia, "Semi-automatic Composition of Web Services using Semantic Descriptions." Accepted to "Web Services: Modeling, Architecture and Infrastructure" workshop in conjunction with ICEIS2003, 2002.

[8] G. Jiang and G. Cybenko, “Query Routing Optimization in Sensor Communication Networks”, *IEEE 41st Conference on Decision and Control*, Las Vegas, December, 2002.

[9] D. Reid, “An Algorithm for Tracking Multiple Targets”, *IEEE Trans. On Automatic Control*, Vol. AC-24, No. 6, December 1979.

[10] R.E. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” *J. Basic Eng*., vol. 82-D, 1969.