Boeing Phantom Works Fusion Architecture: A Flexible Approach for Multiple Projects and Domains

Jeffrey D. King
Advanced Network and Space Systems
The Boeing Company
Kent, WA USA
jeffrey.d.king@boeing.com

Brett Hartman
Advanced Network and Space Systems
The Boeing Company
Kent, WA USA
brett.hartman@boeing.com

Abstract: The Boeing Fusion Architecture provides a highly flexible, multi-source, easily integrated tracker for a variety of applications. Many target types can be tracked, either from sensor reports, sensor tracks, or other types of tracks. The interfaces input type, assignment method, track type, track maintenance algorithms, etc. can be selected at run time by the user. The code uses the advanced libraries and tools of Java to keep the application well positioned for quick turn around for demos, proposals and program hot starts. The architecture is a multi threaded, event driven architecture delivering real time performance for multiple customers.

Keywords: Tracking, tracks, multi-source, flexible, architecture, target, measurements.

1 Introduction

Data fusion is a complex and varied topic with a myriad of requirements, approaches, levels and solutions. This paper presents our work on a primarily Level 1 (see Figure 1) fusion architecture that has been developed organically over the course of several years for a variety of customers. The goal is to communicate the evolution, trade-offs, strengths and weaknesses of this architecture type and approach.

The standard model developed by the Joint Directors of Laboratories Sub-Panel on Data Fusion [1] provides a convenient convention to discuss different aspects of a fusion system. A summary is shown in Figure 1 and explanatory text, below.

Figure 1, Standard JDL Fusion Model.
Figures 2, High Level Flow (Use Case)

Off board sensor tracks and sensor reports are separated; their formats are translated if necessary, and their contents are placed into queues in preparation for processing.

Gating and Association algorithms relate sensor reports and sensor tracks to one another and to detected objects for each sensor. The procedure refers to pre-existing tracks (if any) available from the "Local Track Database" shown in Figure 3. The objective of the procedure is to indicate which sensor reports and sensor tracks are related to one another.

State update and estimation algorithms combine the related items. Related sensor reports are combined to form new tracks and to update pre-existing tracks for each individual sensor. Related sensor tracks are declared to be new tracks or are combined to update pre-existing tracks. The objective of estimation is to produce a single track for each object seen by each sensor.

This process is repeated in the Fused Tracker process where tracks from each sensor tracker are scored, gated, and tracks created or updated to reflect the total operational picture of all reporting sensors.

Track Maintenance is run on both the local track database and the fused track database and can respond to management instructions issued by an operator. Tracks can be split, merged, and dropped from further processing. These tasks can be done automatically via track management algorithms, or performed by an operator. An operator can also add new track or modify a track.

2 Level 1 Data Fusion Architecture

Tracking architectures come in many flavors, languages and types, but multi-source Level 1 fusion architecture boils down to one of two basic principles.

The first architectural approach is to have all measurements or tracks, from whatever sources are desired, act upon a single track database, or picture as shown in Figure 3.

The advantages of this approach are low latencies, computational efficiency and simplicity. These types of architectures tend to lend themselves to autonomous systems or platforms where a larger view of total track picture is not so important.

The second basic architectural approach is to have each measurement or track source feed a local track database and have a system, or fused track database and algorithms to meld these various sources. This is illustrated in Figure 4.

The advantages of the distributed approach, whether the actual processes are located on different machines or not, is scalability and flexibility. These types of architectures tend to be used more in higher level operational systems and networked systems.

FA began life looking like Figure 3, as the first instantiation dealt with multiple similar sensors with a single track type. The first instantiation used Multi-frame Assignment [2] and a patented association selection algorithm [3]. The architecture and code was refactored to resemble distributed architecture shown in Figure 4 as the FA began being used in network centric studies, SIAP studies and various demonstrations and proposal efforts requiring disparate sensors, track and/or measurements, and multiple track types e.g. air tracks, ground tracks, and ballistic tracks.
A layered architecture is shown in Figure 5 depicting levels of algorithms and inputs and outputs.

Over time the following design attributes have received the lion’s share of focus:

- **Interoperability**
  
  Several open interface standards guide the FA software design. External interfaces conform to CORBA, HLA, proprietary binary formats and several XML schemas. Several middleware products integrated into the FA software include both Boeing developed and commercially available types. Messaging within the application uses Java APIs compatible with "plug-n-play" algorithm implementations. The APIs also enable external systems to incorporate the FA as a component.

- **Modularity**
  
  Modern object-oriented abstractions characterize the design. The FA's modular components can be executed in separate threads and processes. The FA is equipped with several common math utilities, messaging interfaces, and an optional user interface.

- **Scalability**
  
  The software is designed to scale up to thousands of tracks and tens of input sources. The architecture uses numerous logic "threads" and can be distributed among several CPUs. The FA's track data output rate can be adjusted by users to meet their needs.

- **Portability**
  
  Java is the FA's core programming language to maintain compatibility with a variety of operating systems and hardware platforms. The design strategy is to isolate the software from functionalities unique to particular operating systems.

- **Availability**
  
  Availability is enhanced by designing for fault tolerance and for graceful degradation during overload situations. Fault tolerance is realized by checking for bad input data early in the processing stream. Graceful degradation is attained by catching exceptions and continuing processing when possible.

- **Adaptability**
  
  Adaptation of FA software is available at both compile time and startup. Configuration files and code constructs are the means for adding sensors, modifying classification and discrimination of target features, selecting gating and association algorithms, changing track output rates, debugging, and monitoring files.

- **Reusability**
  
  Minimizing software development times is one of the objectives of using Java and reusing existing software. Java reduces the risk of memory leaks and pointer problems in developed code. Java also has standard capabilities in its Java Development Kit applicable to the FA's problem set. Additional Java capabilities are available from public domain sources. Reuse code includes freeware and algorithms developed on other projects.

- **Performance**
  
  Real-time computing performance is optimized by ensuring that input queues are not blocked for long periods of time. Users can modify output performance by setting adaptation parameters in the configuration file to output tracks either cyclically or by event-driven criteria.

2.1 **Flexibility**

The majority of the attributes above make the FA extremely flexible. This section will focus on some examples of how the design maximizes this flexibility. As more and more customers began using FA, it became quickly apparent that multiple interfaces were needed, regardless of tracking mission.

One of the first and most important architectural considerations that were incorporated was a separation of the I/O. A package was created that provided the “wiring” between the tracking algorithms and the outside world. By separating communication functionality into a distinct component, FA is able to exchange data with external systems in a consistent fashion without knowing what the external systems are.

This allows us to adapt the code to new communication requirements with minimal impact to most
of the system. It also facilitates distribution of system functionality, as the distribution details can be isolated in the communication component. Additionally, use of the communication component by other tools our group has developed (e.g., analysis tools) means that any changes to the communication requirements can be satisfied in one piece of code, rather than duplicating the effort in distinct interface software for each application.

Each local tracker pictured in Figure 4 is completely independent from the others. These can be configured at run time to accept measurements or tracks. This allows the FA to communicate with another, external network providing tracks, a sensor sending measurements and another platform sending tracks. The association algorithm for each sensor level tracker can be chosen independently. Some algorithms that are currently available are Multi-Frame Assignment (MFA), 1 to n, n to m, and external id (i.e. trusting the data source for the correct association).

Most of the current capability for tracking is accomplished using extended Kalman filters (EKF) and these filters can be tuned extensively using configuration parameters. The motion model, adaptive process noise algorithm and levels, number of states, etc. can all be set. In addition more than one model can be set up per input source, creating a flexible interacting multiple model (IMM) algorithm that can be tuned, extended, or simplified at each start up. Figure 6 shows a processing architecture view where different sensor track algorithms can be set at run time.

![Processing Architecture View](image)

One customer has implemented a real time tracking algorithm switching scheme where, depending on load and density of targets, the association algorithm can be switched to optimally use the hardware and network resources available. Some examples of parameters that can be modified to tune the sensor tracking process are:

- Updates to confirm – modifies the number of updates a sensor track must receive prior to consideration as a fused track.
- Scoring threshold – modifies the threshold at which new data is considered “gated” with existing tracks. In the real world a standard 3 sigma chi square score rarely works due to under reporting of true process noise.
- Default sigma values – used to default add process noise (R matrix) to reports or tracks for interfaces that don’t provide this information.

Static bias parameters – used to inflate or deflate incoming process noise, or put in a priori bias parameters based on earlier system calibration.

Some examples of parameters that can be modified to tune the fused tracker process are:

- Scoring threshold similar to above.
- Publish rate – the FA can be configured to publish tracks at any fixed rate, or on update.
- Debug and logging level – many debug files and messages can be configured to aid in such tasks as debugging, evaluating algorithms, or tuning performance.

A plug in track maintenance architecture has been implemented that allows any number of algorithms to set up to run on either a local track database or the fused track database. Some examples we have used to prune the track databases are timeouts, minimum and maximum track altitudes, area of interest, etc.

Plug in data filters that reside between the I/O and fusion processing have also been implemented to provide robustness. As long as predefined interfaces are adhered to, these filters can and have provided capabilities like inclusion and exclusion zones, DTED lookups, maximum latency detection, etc.

3 Applicability

The FA has been used in numerous studies, demonstrations, proofs of concepts, rapid prototyping, proposals, and operations as shown in Figure 7.
Some of the studies include testing new algorithms, rapidly prototyping an extension to a current platform to expand into new markets, and providing a network centric tracker to determine feasibility of networking systems together. Proposals have ranged from maritime to air to space types of tracking.

Many different projects and research require data fusion in some form and the FA provides a quick way to integrate this capability. Interfaces exist for files of various formats, several types of middleware using XML and various binary interfaces. This, along with the extensive library of built in track types allows most users to plug the FA into their architecture with a minimum of effort.

The primary design focus of the FA was not initially performance, as far as maximizing load and minimizing latency, but to create a plug and play architecture to facilitate best of breed algorithm insertion. However, depending on association algorithm chosen and hardware platform, hundreds of tracks from multiple sources have been supported. Figure 8 shows a typical performance curve for randomly distributed targets.

MFA doesn’t scale as well as the simpler association algorithms, obviously, so choosing this type of algorithm for expected high loads, depending on hardware available, may result in poor performance. This particular study showed an n to m association algorithm performing slightly better than a simpler, 1 to n. This was studied and attributed to the time required to propagate candidate tracks to the time of the reports so many more times in the 1 to n case, even though the selection process was more efficient.

To combat the inherent inefficiencies of the architecture the FA has been designed to be highly multithreaded to take advantage of today’s multiprocessor systems. Figure 9 depicts how the processes are mapped to different threads.

4 Performance

The FA was designed and built over a number of years by primarily research and development funds. As such personnel has come and gone as well as customers. A daunting challenge has been to keep the FA a cohesive unit with the ever changing requirements. To combat this developers have developed an aggressive approach to factoring the FA when the design begins to resemble a “tar baby” with additional capabilities just tacked on. New class structures, managers, and object oriented design principals have been reapplied to give the FA a solid design base.

Early on the decision was made to place the FA under configuration control, regardless of the added development time. This decision has paid many dividends as new developers have come on board and been able to work on their own branch without repercussion. Another benefit is some customers have required some “off the wall” processing or algorithm modifications and those have been

Figure 7, FA Applicability

Figure 8, Association Algorithm Performance

Figure 9, Fusion Architecture Threads
able to be saved in a separate branch without affecting the main algorithms.

6 Future Work

The FA continues to evolve in several directions. Several network centric demonstrations, proposals, and an operational customer are currently using and requesting continual improvements.

Particle filtering is an important consideration in data fusion currently and we currently have stand alone instances to solve several different problem types. We plan to incorporate this particle filtering capability into the mainstream FA. This will allow even greater flexibility to meet customer needs for non-Gaussian tracking problems.

The FA is extremely tunable and flexible, but most of these parameters are set at run-time. For real time 24/7 operations we plan to allow more on the fly changes to facilitate real world tuning without a restart. Examples of these parameters would be timeout values, Kalman filter tuning parameters, additional input data filters, etc.

One of the limitations the FA currently has is it can only communicate with one middleware type at a time. Some work has been done on a messaging gateway product that can communicate with any number of middleware types simultaneously. Incorporating this capability would make the FA even more flexible and allow quicker insertion into various demonstrations, proofs of concept studies and proposals.

Our group has other work in Fusion levels 2, 3, and 4 that have not been integrated into a seamless, multilevel fusion capability. Further inserting the FA into a larger, more comprehensive fusion suite has been on the horizon for some time now.

7 Conclusion

The FA is a flexible primarily Level 1 data fusion architecture that fits the needs of many customers. The architecture, design and programming language allow for quick insertion into a variety of environments.

Designing a fusion application requires careful attention to the requirements of the customer. If varied customers are to be satisfied, the designers have many trade offs to consider. Would building an application per customer best fit the needs of the users? Are there expertise and resources available to maintain multiple code bases? We chose to keep the code base in a single, configurable application with an architecture weighted towards flexibility and adaptability above all other attributes.

Building a robust application over a number of years with inconsistent funding and changing personnel is a difficult proposition. Having one or two people consistently on the program, preferably a software architect is helpful. Placing the product under configuration control is highly recommended as changes can be backed out easily and branches can be used to support differing user requirements.

The FA has been highly successful in supporting many customers and has saved The Boeing Company much time and resources that can be channeled into other studies. By designing for flexibility and aggressively pursuing a reusable architecture the FA will continue to support its customers for years to come.

References

[1] E Waltz and J Llinas, Multisensor Data Fusion; Artech House, 1990. The original JDL model is considered dated and has been modified by some authors. A Google of JDL Fusion Model or similar terms will return many recent references.
