OPTIMIZATION OF THE MULTI-SOURCE DATA FUSION SYSTEM FOR INTEGRATION ON THE CANADIAN PATROL FRIGATE

Elisa SHAHBAZIAN, Louise BARIL, Jean-Rémi DUQUET

1. Introduction

Since 1991, the Research and Development (R&D) group at Lockheed Martin Canada (LM Canada) has been developing and demonstrating Level 1, 2, 3 and 4 data fusion, resource management and imaging technologies which will provide Observe-Orient-Decide-Act (OODA) decision making capabilities/tools in Naval and Airborne Command and Control (C2) for application on Canadian Patrol Frigates (CPF) and Canada's CP-140 (Aurora) fixed wing aircraft. Over the last three years LM Canada, in collaboration with Canada's Defence Research Establishment Valcartier (DREV), has also established a generic expert system infrastructure and has demonstrated that it is suitable for integrating these decision making technologies into real-time Command and Control System (CCS). The Multi-Source Data Fusion (MSDF) technology is the most mature among these decision making technologies and is likely to be integrated onboard a currently fielded CCS the soonest. Over the last two years the LM Canada R&D team has started the effort towards re-structuring and optimizing the proof-of-concept MSDF algorithms to establish a prototype which will be ready for integration on the existing platforms, specifically the CPF, and that can perform real-time tracking and identification by the end of the year 2000. This restructuring and optimization is occurring in phases.

First the existing proof-of-concept MSDF system was broken down into very basic modular and independent components within the generic expert system infrastructure. Each MSDF process (alignment, association, kinematic estimation, identification, etc.) consists of one or more of these basic components. This architecture is designed to enable independent modification and evaluation of each component. It is also ideal for ensuring future growth for adding additional decision support capabilities, with minimal impact on the already implemented and demonstrated system.

Next these components are analyzed, optimized and evaluated in terms of their performance, given the characteristics and amount of input sensor data and information. Initially, this is done using simulated data, and the optimization is iterated until the performance of the overall MSDF system is able to process peak loads of data with higher operational performance than the current CPF.

In the third phase, recorded data at sea will be used to validate and further optimize the MSDF system. It is clear that the behaviour of some of the algorithms will be different with this data, and this will be the most challenging aspect of this phase. At the end of this phase the MSDF system will be ready for integration on CPF. It will not only be able to process all data available on CPF, producing high quality kinematic and identification estimates, but will also be open for future evolution to more sophisticated sensor data processing, fusion of additional sources of data, higher level fusion processing, etc.

At the current time, the first two phases of this effort are close to completion. This paper includes the details, lessons learned and results of the first two phases, and describes the specific research activities envisaged in the third phase. It also describes some earlier and parallel proof-of-concept efforts towards demonstrating the future growth of this system.

2. KBS Architecture Based MSDF Design

The details of the MSDF prototype,[1,2] as well as the KBS architecture, have been published earlier.[3,4,5,6]

The KBS architecture developed at LM Canada was designed right from the start as an architecture that could support a large real-time application through all phases of its development life-cycle, from early analysis and prototyping phases to the final deployment. As such, it had to incorporate several key features to give maximum flexibility to the developers without adversely impacting performance. As a minimum, the KBS shell must provide the following:

a. Speed: The key advantage of this system is pure execution speed, as a result of its implementation as a compiled system (C++) rather than an interpreted one, and because of its optimized blackboard controller.

b. Small Overhead: Because of its streamlined design, the KBS scheduling and activation mechanisms introduce very little overhead in the system. The difference between “Total Agent CPU” and the “user CPU” has been shown to be less than 5 %.

c. Linearity: The blackboard controller design incorporates a critical mechanism, similar to the so-called RETE algorithm, which directly links each agent to its associated data types, thereby avoiding costly loops each time an agent-data pair needs to be activated. This mechanism, coupled with a design which avoids lists searches in the internal controller, ensures from a theoretical point of view that the processing time of a given system of agents will scale linearly with the number of rules and data instances present in the system, thereby allowing system scaleability (provided of course that the agents themselves are linear). This linearity has been demonstrated with run-time benchmarking of Level 2, 3 data fusion algorithms in a previous study[6] (similar to MSDF in terms of software complexity and CPU needs) with up to 1000 tracks.

These features illustrate that the KBS-based implementation will not handicap the run-time performance of the MSDF system.

Other major benefits of this architecture include modularity and the possibility of modifying each component independently, without affecting the rest of the system, as well as the ability for integrating algorithmic and rule-based decision support within the same infrastructure

Although Level 1 data fusion does not require rule-based reasoning, it is clear that the architecture is ideal for future growth into higher level fusion implementations.

Therefore the first step towards optimization of the MSDF prototype was to decompose it into agents. Figure 1 shows a high level diagram of how MSDF was decomposed into agents within the KBS architecture. It illustrates the fact that the MSDF system can be viewed as a small number of independent domains, consisting of a number of sequential steps:

a. Data reception, preparation and buffering

b. Data processing (i.e., the fusion processes)

c. Track management

d. Data output mechanism (not represented in Figure 1).

The end result of this first step was a new prototype, Data Fusion on Blackboard (DFBB).

The designer can use three potential options to make optimal use of the processor (and other system's resources) to obtain a faster execution, and ultimately guarantee real-time performance of the system within this infrastructure.

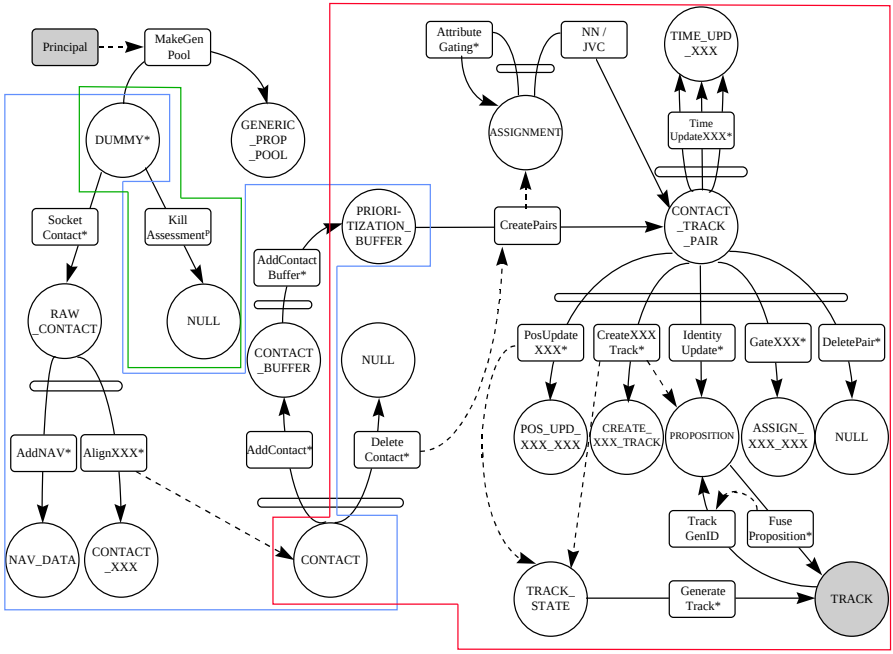

Figure 1: High-Level Data Flow Diagram of Agents Present in the MSDF System. Circles represent data types present on the KBS, while squares represent the agents that act on the data. Symbols XXX are used when agents/data types are present under several flavors depending on the context of operation (e.g. sensor name, track and contact types, etc.)

The first is intrinsic to the KBS and involves the regular blackboard scheduler together with the fine granularity of each individual agent; the second deals with agents multithreading, which is very robust and user-friendly on the KBS; and the third uses the real-time features of the operating system, which are still available to the developer through the KBS layer. Because of the nature of Data Fusion algorithms, and also because the timing constraints are not too stringent, our efforts will focus on the first method, namely run-time optimization of individual agents.

3. Initial Optimization Efforts

The first necessary condition that has to be met by any real-time application is run-time efficiency, that is, it has to ensure at least average real-time performance. This step involves the optimization of the DFBB to support real-time processing of all data available on a platform, specifically the CPF.

The four domains of MSDF are more or less independent and could in principle be suitable for process prioritization schemes and real-time scheduling. However, initial review of the DFBB code shows that three domains out of four are low consumers of CPU resources, and the remaining one (Data Processing) consists of a relatively small number of sequential steps which would benefit very little from a sophisticated scheduling mechanism. Moreover, CPF real-time constraints on input data (typically several tenths of a second) do not justify the use of hard real-time features (or even a strict real-time operating system). For these reasons, before real-time scheduling and prioritization issues are even considered, code optimization must be pushed to the limit to increase run-time performance as much as possible.

The code optimization is done by iteratively profiling the software, evaluating the bottlenecks and re-designing/re-coding to remove/reduce CPU utilization by such components, taking advantage of the various intrinsic facilities of the KBS architecture and other methods.

3.1. DFBB Benchmarking

The initial DFBB system processed a 100 seconds scenario in about 100 seconds (i.e., average real-time) for 100 targets, while the 200-targets scenario requires about 3.5 times the amount of CPU to process a scenario of the same duration.

This is not surprising, since a few agents are clearly expected to behave in a non-linear way. In fact, all the agents participating in updating tracks, the application of the Kalman filter and the identity update are expected to show a linear behaviour (i.e., linear against the number of tracks), while those performing the gating are expected to behave roughly as "Ntr2", since the gating process involves "Ntr" x "Nir" pairs (where "Nir" is the average number of input reports in a data set which is proportional to "Ntr").

Those expectations are confirmed by a closer inspection at code profiling results for the individual agents. Several tools are available for this task, depending on the level of investigation taking place. Standard profiling tools are available on Unix, such as gprof, giving various degrees of details about the internal calls performed in each agent, with various timing accuracies as well. For the time being, a minimally intrusive way of probing the cumulative CPU used by each agent is of interest. For

this investigation, a timing tool is available on the KBS to monitor the user process time spent between the start and the end of each agent with minimal overhead, using C "Times" functions. The nominal precision on each agent execution (time / call) is 1 millisecond.

Results are presented in Table 1 for most agents involved in the fusion process in DFBB.

Table 1. DFBB Benchmark Results on a 450MHz Pentium Processor running under Solaris 2.6. A very large fraction of the CPU is used by only 6 agents (highlighted).

Scenario :	**50 Targets**		**100 Targets**		**200 Targets**	
Agent Name :	**num. of calls**	**Total Time (secs)**	**num. of calls**	**Total Time (secs)**	**num. of calls**	**Total Time (secs)**
AddContact	14171	0.070	26808	0.230	51214	0.260
AlignIFFContact	4389	0.170	8454	0.480	16307	0.900
AlignSG150Contact	4663	0.240	8760	0.600	16629	0.980
AlignSPS49Contact	5119	0.210	9594	0.550	18278	0.990
AttributeGating	**7120**	**0.760**	**8831**	**2.190**	**9358**	**7.760**
CreatePairs	**7142**	**1.510**	**8854**	**3.620**	**9370**	**10.310**
CreateRBTrack	59	0.010	117	0.010	248	0.010
DeleteContact	14171	0.100	26808	0.300	51214	0.700
DeletePair	**55984**	**0.660**	**153937**	**1.640**	**550216**	**6.640**
ExtAdap KalmanRB_RB	**14112**	**2.170**	**26691**	**4.050**	**50966**	**7.490**
FuseProposition	2534	0.110	4870	0.180	9472	0.410
GateRB_RB	**41813**	**3.700**	**127129**	**11.690**	**499002**	**46.870**
GenerateTrack	14171	0.230	26808	0.490	51214	0.770
IdentityUpdate	14112	0.320	26691	0.600	50966	1.370
NearestNeighbour	7120	0.520	8831	0.720	9358	1.270
Principal	1	0.130	1	0.140	1	0.140
SocketContact	7143	0.580	8855	0.930	9371	1.620
TimeUpdateRBTrack	**55925**	**5.290**	**153820**	**14.630**	**549968**	**54.720**
TrackGenID	1615	0.620	2516	0.910	3174	1.310
CPU agent total time		**17.42**		**43.98**		**144.52**
User CPU time (sec):		18.67		46.07		152.46
System CPU time (sec):		1.27		2.89		6.00
Execution Time (min):		0:20.40		0:49.36		2:38.85
Overall CPU use (%):		97.7 %		99.1 %		99.7 %

The following observations follow directly from the data displayed in Table 1:

a. A very large fraction of the CPU time (above 90% for 200 tracks) is spent in six agents. These agents are all on the critical path and cannot be pushed aside or executed out of sequence by some process scheduling scheme. In order to reduce average run-time comfortably below the 100-second duration of the scenario, the first step is clearly to optimize those agents to increase execution speed and, if possible, linearize them with respect to the number of tracks "Ntr" to reduce their impact on the worst-case scenario and improve scaleability (for Ntr > 200).

b. The most time-consuming agents, as identified in Table 1, are all (except one) agents that show a non-linear execution time against the number of tracks processed by the system. The non-linear agents are: *TimeUpdateRBTrack, GateRB_RB, CreatePairs, DeletePair, AttributeGating*; Among those, we can identify two categories:

    1) The number of calls to the agent is roughly constant, but the agent internal algorithms involve input data of the type "Ntr" x "Nir" (e.g., track-input report pairs), and requires a processing time roughly proportional to "Ntr2 ". The agents falling in this category are "AttributeGating" and "CreatePairs".

    2) The agent execution time is roughly constant, but the number of calls to the agent increases like "Ntr2". This category includes DeletePair, GateRB_RB and TimeUpdateRBTrack.

From the preliminary analysis and observations above, taking each agent independently, the following optimization strategy to reach average real-time performance was selected:

a. *TimeUpdateRBTrack*: This agent is the most demanding in the whole system, thanks to both internal processing needs and a large number of calls. It is used both for the gating process and the positional track update process (as part of the Kalman filter process); these processes can be analyzed separately:

    1) Track update: At the end of the fusion process, each contact is used to update the state vector of one of the tracks in the system. The track is time-updated in the process, resulting in ~50 000 calls to TimeUpdateRBTrack for the 200 seconds scenario. The number of calls is linear with Ntr and does not cause abusive overhead in this process.

    2) Position update: before the gating process, the MSDF algorithm selects a sample of tracks, and propagates their state vector to the time of each contact received to form a contact-track pair. This translates into a

number of calls of order "Ntr^2", for a total of ~480 000 agent calls to TimeUpdateRBTrack during the 200 second scenario. This latter number could be reduced by a factor of ~6 if the tracks were time updated to an average time instead of the individual times of the input reports, thereby making the number of calls to the agent linear with "Ntr"

3) Once this agent has been linearized, another quantum leap in speed will be gained by the use of XY coordinates for tracking, instead of the current RB coordinates. The current agent TimeUpdateRBTrack spends significant processing time converting the RB track state to an intermediate XY state vector, and back to RB.

b. *GateXY_XY* will replace the current RB_RB version. This by itself will do little to improve run-time performance since the gating agents do not require RB to XY conversions. However, in the current implementation, all gating agents compute their statistical distance via a call to a single, generic method that performs several complex matrix operations (i.e., matrix reduction, transposition, multiplication and inversion). This improves code readability but only at the expense of significant CPU overhead. It is hard to predict the cumulative gain expected by all these optimizations; a factor of 2 is certainly an underestimation and a factor or 5 is not out of reach.

c. *ExtAdapKalmanRB_RB* is a linear agent, but suffers both from time-consuming RB to XY conversions and from extensive use of matrix operations used to calculate the Kalman gain and the resulting track state update. This agent is already linear in "Ntr" and should drop by a (very conservative) factor of 2 at least in the final implementation.

d. A significant speed increase can be achieved just by implementing a better object creation strategy in the MSDF system. Most of the dynamic memory allocation can be replaced by the use of persistent objects created upon system initialization, for instance by replacing contact-track pair objects by a single, persistent pair list. An immediate effect would be the disappearance of the agents *DeletePair* and *DeleteContact*, two of the major - non-linear - CPU contributors identified above. This would result in an immediate gain of about 40 seconds out of 320, for the 200-targets scenario. Similar object creation is also hidden inside other agents (e.g., *ExtAdapKalmanRB_RB*, which instantiates a new TrackState object for each track update) and can be improved, with significant gains in terms of run-time performance.

The sole implementation of about half of the strategies stated above decreased significantly the CPU time needed by those processes, prior to performing any deeper

investigation to streamline and optimize the individual agents (e.g. through internal code profiling). After a change of coordinate system, and with most generic matrix operations expanded, one gets the figures presented in Table 2, where the results of benchmarking before and after optimization are shown side-by-side for 200-targets scenario.

Even though object creation/deletion strategies and agents linearization still remain to be applied, overall CPU needs of DFBB agents has already been divided by three, allowing the system to achieve average real-time performance on the presented scenario. Further optimization is expected to bring the current figure down by another factor of two.

Table 2. Comparison of DFBB before and after the first round of optimisation for a 100-seconds, 200-targets scenario on a 450MHz Pentium Processor, showing the main CPU-demanding agents.

Scenario :	**Before Optimisation (R-B Tracking)**		**After Optimisation (X-Y Tracking)**	
Agent Name :	**num. of calls**	**Total Time (secs)**	**num. of calls**	**Total Time (secs)**
ExtAdapKalman	50743	8.100	50764	7.510
TimeUpdate Track	891431	102.140	895377	17.000
Gate	840688	101.040	844613	12.400
CreatePairs	9326	22.350	9326	21.950
DeletePair	891681	10.110	895606	9.940
AttributeGating	9315	12.770	9315	2.510
total main 6 agents		*256.51*		*71.31*
% of total agent CPU		*95 %*		*85 %*
total all other agents		*15.04*		*14.01*
Total Agents CPU		**271.55**		**85.32**

4. Supporting R&D and Future Plans

In parallel with this real-time performance optimization efforts, there are a number of projects at LM Canada which look at the optimization of algorithm performance, development of alternate algorithms which have higher performance, development of

strategies for fusion management (level 4 fusion) to activate different algorithms depending to different context, etc.

The KBS architecture is ideally suited for supporting all of these concurrent activities, permitting iterations of algorithmic and real-time optimization indefinitely, until the desired performance is achieved of each individual platform.

The next step for the CPF is to use recorded data at sea and use it to validate and further optimize the MSDF system. It is clear that the behaviour of some of the algorithms will be different with this data, and this will be the most challenging aspect of this phase. In this phase too, the algorithmic developments of the parallel research efforts will be very useful, as a variety of algorithms to perform each MSDF task will be available for experimentation. At the end of this phase the MSDF system will be ready for integration on CPF.

Acknowledgements

The authors would like to thank the DREV DFRM research team and the rest of LM Canada's R&D team, all of whom had significant contributions into decision support technologies research and demonstrations for Canada's defence platforms since 1990, and specifically the application of these technologies to CPF.

References:

1. Bégin F., E. Boily, T. Mignacca, E. Shahbazian and P. Valin, "Architecture and Implementation of a Multi-Sensor Data Fusion Demonstration Model within the Real-time Combat System of the Canadian Patrol Frigate," *AGARD symposium on Guidance and Control for Future Air-Defence Systems* AGARD-CP-555 (Copenhagen, 17-20 May 1994), 28.1-28.8.
2. Valin P., J. Couture, and M.-A. Simard, "Position and Attribute Fusion of Radar, ESM, IFF and Data Link for AAW missions of the Canadian Patrol Frigate," in Multisensor Fusion and Integration for Intelligent Systems (MFI'96) (Washington, D.C. December 8-11 1996), 63-71.
3. Macieszczak, M., P.Bergeron, M. Mayrand, J. Couture, and J.R. Duquet, "Fast Multithreaded Agent-Blackboard Expert System: A Solution for a Next Generation Command and Control System," in 1998 Systems Engineering and Software Symposium (New Orleans: May 13-15, 1998).
4. Shahbazian E., J.-R. Duquet, M.-A. Simard, "Literature Survey on Computer based Decision Support for Command and Control Systems," in FUSION 99 (Sunnyvale, CA, 6-8 July 1999), vol. 2, pp. 926-933

5. Bergeron P., J. Couture, J.-R. Duquet, M. Macieszczak, and M. Mayrand, "A New Knowledge-Based System for the Study of Situation and Threat Assessment in the Context of Naval Warfare," *FUSION '98* (Las Vegas, July 6-9, 1998).
6. Duquet J.-R., P. Bergeron, D.E. Blodgett, J. Couture, M. Macieszczak, and M. Mayrand, "Analysis of the functional and real-time requirements of a Multi-Sensor Data Fusion (MSDF) / Situation and Threat Assessment (STA) / Resource Management (RM) system," in *Sensor Fusion: Architectures, Algorithms, and Applications II*, *SPIE Aerosense'98* (Orlando, 13-17 April 1998), 198-209.

ELISA SHAHBAZIAN received her PhD degree in High Energy Physics from McGill University in 1983. In 1991 she became responsible for the Data Fusion section of the R&D Department at LM Canada and since 1994 she is responsible for conception, prioritisation, and co-ordination of all R&D activities in Canada for development of intelligent decision support technologies for C4I applications (Data Fusion - levels 1, 2, 3 & 4, Resource Management, Imaging, etc.), and the engineering infrastructure for the establishments of these technologies onboard the Naval and Airborne platforms of Canada. Dr. Shahbazian leads the LM Canada's R&D department, consisting of 14 PhD Scientists and Engineers, who perform the research in various decision support technology domains in teams consisting of internal researchers, graduate students as well as satellite university teams. Address: Lockheed Martin Canada, 6111 Royalmount Ave., Montréal, Québec, H4P 1K6, Canada; tel: (514) 340-8310, extensions 8343, 8537, 8547, fax: (514) 340-8318; E-mail: elisa.shahbazian@lmco.com

LOUISE BARIL is researcher at Lockheed Martin Canada. E-mail: loiuse.baril@lmco.com.

JEAN-RÉMI DUQUET is researcher at Lockheed Martin Canada. E-mail: jean-remi.duquet@lmco.com.