A multi-logic framework for multi-level fusion in real time data fusion applications

José Gomes de Carvalho Jr.
PEC COPPE / IPqM
Federal University of Rio de Janeiro / Navy Research Institute.
Rio de Janeiro, Brazil.
gomes@ipqm.mar.mil.br

Pablo Rangel
PESC COPPE / IPqM
Federal University of Rio de Janeiro / Navy Research Institute.
Rio de Janeiro, Brazil.
pablorangel@cos.ufrj.br

Nelson F. Ebeken
PEC COPPE
Federal University of Rio de Janeiro, Rio de Janeiro, Brazil.
nelson.ebeken@ntt.ufrj.br

Abstract – This paper presents a framework called Real Time Multi Logic Reasoner (RT-MLR) to perform high level data fusion. RT-MLR reconciles the development of Knowledge Based Systems (KBS) with the object-oriented architectures and on-line real-time monitoring requirements. This framework has been implemented with an inference engine that supports a data driven approach and knowledge expressed in First-Order Logic, Fuzzy Logic and Temporal Logic. RT-MLR has been tested face on its capability to perform data fusions in a military application, using all kinds of logics implemented. Data fusions in different levels were made through rules that express relations between objects inside naval context scenarios. Simulated and actual data were used in validation process.

Keywords: high level fusion, multiple logic reasoning, real time monitoring.

1 Introduction

Data fusion is a widely studied process that includes many techniques with different objectives [1] [2] [3]. It is usually adopted the JDL model [4] that separates the process into levels according to planned objectives. Processes at level 0 make data conversions, filtering and all managements necessary to produce data as reliable as possible at sensor level. Objects processing is done at level 1, which includes mainly estimating position, speed and acceleration of tracked objects. High level fusion processes (levels 2 and 3 of JDL model) involve situational assessment and threat refinement. These processes usually involve decisions that were previously done by humans. In military applications, decisions must comply with doctrines that specify procedures in different contexts. The process must be understandable and decisions must be justifiable. These requirements indicate the use of techniques that deals with explicit knowledge into a human comprehensible logic.

Knowledge Based Systems (KBS) are designed to provide inferences that may be explained to human operators, since they are based in logic rules.

Another characteristic of military applications is the amount of information that must be treated in real-time. These applications are generally designed in an object-oriented architecture by software engineers. However, the greatest part of intelligence embedded into the systems is provided by experts that don't know architectural details.

The main contribution of this work is a framework that allows achieving high-level fusion expressing complex domain knowledge and relations itself through rules that support multiple logics. The framework reconciles the OO modeling and the KBS approach in a data-driven reasoner. The issues of KBS are detailed in section 2, particularly the compliance with OO systems and real time monitoring. Details of the implemented framework are provided in section 3. In section 4 is explained the approach adopted to apply the framework into a naval Command, Control, Communications, Computer, Intelligence, Surveillance and Reconnaissance System (C^4ISR). Sections 5 and 6 present results and conclusions.

2 Applying knowledge based systems

Intelligent applications usually have a complex domain. In those systems is commonly adopted a hierarchical approach based on languages like UML that support object-oriented architectures. Moreover, on domains that need to establish restrictions on properties as transitivity and cardinality, ontological languages, like OWL, use to be the priority choice. These languages are particularly useful in open domain applications like those designed for the web environment and working with text mining.

Ontological languages usually have engines that make automatic inferences from the hierarchies provided by classes and from the restrictions imposed to relations. These inferences may be started by an explicit query made in the domain application or by an automatic process triggered by modifications in the working memory where facts lie. However, in complex domains reasoning is not restricted to objects in a hierarchy. So, it is always useful to have a mechanism to make inferences from business rules [5].

Another problem is the expressiveness of the knowledge represented by rules. To improve the coverage of the application domain and reduce the amount of rules needed to represent knowledge, it is useful to use rules that implement different logics. Fuzzy Logic, for example, enables a more intuitive partition of the application

domain, expressing more concisely the knowledge of human experts. Temporal logic, on the other hand, is useful for behaviors that can only be identified if we consider the events related to the time they occurred. This is particularly important when the behavior to be considered is evolutionary in nature. In a medical system, for example, the diagnosis and treatment of a patient depends on a sequence of evidences and events related in time that can characterize this or that specific pathology and recommend this or that therapeutic procedure [6]. In a C^4ISR system there are also behaviors that can be detected and countermeasures that can be suggested, which are related to temporal sequences of events.

Another important aspect in the representation of knowledge is the ability to deal with values that are constantly changing, which is typical of real-time systems. Such characteristics are typical in military systems, control systems for industrial plants and systems for monitoring hospital patients. In these situations, the knowledge used in the premise of rules is constantly changing, resulting in the necessity of a constant re-evaluation of these rules.

Several models have been proposed inclusion of production rules in a systems that use procedural modeling [7,8]. Some ontology languages as LOOM [9] and OCML [10] allow the inclusion of first-order logic production rules (detailed comparisons are made in [8] and [11]). The advantage of these and other languages for specifying ontology as FLogic [12] and Ontoligua [13] is that a complete ontology specification is allowed. These ontology languages include concepts that are commonly found in languages UML-like, as classes, inheritance, polymorphism, instantiation, strong typing and encapsulation. But they also include concepts that are not found in UML, as facets of properties (transitivity and cardinality), or operations on sets such as union, intersection and complement (a comparison of correspondence between UML and ontological languages can be found in [14].

Another approach is made by frameworks that allow the specification of production rules in object-oriented architecture implemented in Java (like DROOLS, ILOG JRules, Jess and Hammurabi). These frameworks generally follow the standard JSR-94 [15]. The standard defines operations to insert, remove or modify facts in a working memory. The inferences are done by queries over the rules and the basis of facts (working memory). The idea is to separate the knowledge of the business itself in a set of rules that are easily maintained and debugged by experts of the business, without them needing to know the architecture used in the rest of the application.

The JSR-94 standard does not define syntax for the language in which the rules will be expressed. So, each framework defines a file format and syntax for the rules.

RT-MLR has another approach for coupling of procedural and declarative paradigms, as we shall see in section 3.

2.1 KBS restrictions in real-time systems

In KBS there are two key elements: the knowledge expressed in rules and the basis of facts (data known). Systems that seek to reconcile the declarative and procedural paradigms (DROOLS and Hammurapi Rules are examples) maintains a database of facts that co-exists with the application data modeled with OO techniques. It is up to the application to update the working memory used by the inference engine when a new external fact has to be included or excluded. Naturally, the activation of the inference motor also generates new facts that are automatically added to the working memory. More specifically, one fact is a true knowledge, such as:

Bob is a 20 years old single man.

The inference engines implement a forward or a backward chain strategy to search the rules that can be fired by the facts in the working memory. The inferences generate new facts that are added to the working memory. However, if there is an external change of facts, as information received from external sensors, it is necessary to remove the fact that it is no longer true and add the new fact in the working memory. Thus, if the status of Bob changes from single to married, the former must be removed and the new fact must be included in the database:

Bob is a married man of 20 years of age.

In real-time systems the information varies quickly. So, the removal and insertion of facts can make the search algorithms quite costly. Consider, for example, that the fact to be maintained is:

Bob is a married man of 20 years of age with 75 beats per minute

If the monitoring system continuously reads the heartbeat of each patient, then we have a lot of new facts constantly being removed and inserted. A monitoring system deals with hundreds of objects, each of them with dozens of attributes. Therefore, there will be thousands of constantly changing data. The efficiency of the search strategy is directly related to the amount of information in the knowledge base. So, the high frequency of changes in a real-time monitoring system turns costly insertion and removal of facts in working memory.

In the implemented framework, the data used in the rules have different granularity and the changes made in domain attributes directly triggers the rules of interest, as we shall see in section 3.

2.2 Traditional reasoning in KBS

In several KBS, after each change in the base of facts, the inference engine searches for rules that can be triggered by the inclusion of the new fact. The fired rules can add new facts to the knowledge base and fire more rules in a recurring process. This strategy of inference is called data driven or forward chain. Most implementations of inference engines (OPS5, ART, CLIPS, Jess, DROOLS and others) use the RETE algorithm [16] that is based on two observations:

- The firing of a rule usually changes only a few facts, affecting only a few rules by each of those changes.
- The same patterns appear in the left-hand side of different rules.

The RETE algorithm examines the rules and creates a rooted acyclic graph (the Rete) where nodes represent tests of one or two values which are the patterns that appear on the left-hand side of the rules. In building this graph, it is possible to identify the same patterns that appear in the different rules and simplify the graph.

The engines that use the RETE algorithm pre-process rules and build this graph. These engines also provide a mechanism to delete and add new facts in the knowledge base. After each inclusion they seek the root nodes to match the new fact, and from there, information flows down the graph until the information reaches the leaves of the network when the rules are finally fired.

The simplification provided by the RETE algorithm comes from the memory capacity of the network nodes. If only the right side value of an operator was re-calculated by the investigation of a node, then the value of the left side has not changed and you can use the value stored in the left-side-memory. If some new fact in the future changes the value of left side of this node then a new inference will be made using now the value stored in the right-side-memory. Each new fact included into the knowledge base is filtered by the network until it reaches leaf nodes, thus causing the activation of rules. However, as usually occurs in real-time systems, if several facts are generated almost at the same time (in a multi-thread monitoring system) before engine can complete inferences fired by the first one, then the decisions may not be updated with the current reality. Suppose, for example, that a new fact *F1* is generated and start shooting the inference engine. Suppose further that immediately after asynchronous updates of data occur in the database application. This is equivalent to the generation of new facts, for example, a *F2* fact. Each fact included in the working memory will fire the inference engine but the whole inference process is not concurrent, so each pair inclusion-inference will be processed sequentially. The already occurred event that generated *F2* will generate further research of the network, but at the moment it is waiting to spread the fact *F1* in the network. Eventually the fact *F2* can be used on the premise of a rule in which the fact *F1* is also used. Suppose that *F1* appears on the left side and *F2* on right side of an operator in this rule. For this rule, the engine will use the new value from *F1*, but will use the value stored for the fact *F2* in the right-side-memory, since update of *F2* was not yet provided by the application in the working memory. This can generates a not so updated decision, since the event that generated *F2* had already occurred but was not considered. Naturally, it is known that fact *F2* will be considered in the sequence and the decision of the previously investigated rule will be eventually reconsidered, but the decision at the first moment was not the most appropriate, since values that were used don't correspond to the most updated snapshot of the real-time application domain.

The proposed methodology doesn't work with a parallel memory, neither for "facts" (working memory) nor for inference in rules (right and left side operator memory). Instead, data considered are always read from the application domain, as will be explained in section 3.

2.3 Extending the expressiveness

The frameworks that combine declarative knowledge with OO-domain, which were mentioned in Section 2.1, generally include only first-order logic. A known exception is DROOLS, which recently released an extension to include reasoning with temporal logic.

The inference engine of the implemented framework works with rules expressing first-order logic, fuzzy logic and temporal logic, as we shall see in section 3.

3 The framework architecture

This section details the solutions adopted in RT-MLR .

3.1 The objectives

. The RT-MLR aims to integrate the procedural and declarative programming in real-time applications modeled with an object-oriented architecture. The main objectives are:

- Combining the hierarchical structure of OO applications with the KBS, facilitating the expression of expert knowledge in complex domains.
- Integrating the knowledge base used for inferences with the application domain database.
- Firing rules automatically after significant changes in the domain database fields.
- Increasing syntactic and semantic integration expressing rules in the language used in application domain.
- Increasing the expressiveness of system enabling inference over rules that use first order logic, fuzzy logic and temporal logic.

3.2 Application design approach

The RT-MLR offers an integrative approach for the coupling of declarative and procedural paradigms. The rules are seen as something that interacts with the domain of the problem. However, this interaction is neither so close that is necessary to know about code sequences in the domain classes nor so far that isn't necessary to know which data are modeled in the domain.

Rules are modeled as association classes that relate the domain objects. In fact, in many cases, association classes have a set of rules that allow expressing some particular aspect of domain. They relate object attributes of a particular class with object attributes of another class, respecting the separation between the declarative and procedural knowledge. The classes of rules have a simple construction that does not require deep knowledge about the rest of the modeling and does not require a proprietary

syntax. The rules are written in their own programming language, using logical operators that are resources provided by the framework. Thus, Boolean logic operators, fuzzy logical operators and temporal operators are provided as framework resources to be called in the body of the rules. Moreover, rules use the syntax of the native language (Java, for example). Java is cited as an example because the package was developed for Java, but nothing prevents there are other implementations for C + + or any other object-based language.

3.3 Multi logic reasoning

For first-order logic, RT-MLR works with the closed world assumption in a non monotonic logic and also with the unique name assumption. Marc Dörflinger [17] proposed an inference motor for Courteous Logic [18] using labels to determine precedence of rules, which is an important issue in non monotonic logics. Into RT-MLR the precedence is only defined by the sequence the rules are added to a rule set. Those rules that are inserted later have higher precedence. As in Courteous logic, the hierarchy solves the problem of mutual exclusion, which is the existence of conflicting conclusions. In RT-MLR the final conclusion is from the highest priority rule. The sets of rules are created first and then each rule is added to one of the sets. In a typical application, it is probably necessary to create only one set for all rules of first-order logic and temporal logic. The set exists only to define the precedence of the rules. In the case of fuzzy rules, each group should aggregate the rules that relate to the same output variables (variables that appear at the conclusion of the rules).

3.4 On line monitoring

One of the goals of the RT-MLR is to adapt the inference engine for real-time object-oriented applications. Some simplifying assumptions are made. The main assumption is that there is not a base of facts (working memory) parallel with database application. Figure 1 illustrates the concept.

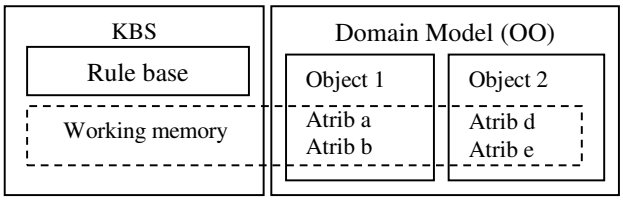


Figure 1 – The working memory in RT-MLR

The knowledge base consists of attributes that have already been defined in the application domain and which are not replicated on a parallel basis. The approach is a closed world of knowledge, where facts are values of attributes and it is assumed that these facts ever exist with some value.

The second important feature concerns to the values used for inferences. The RT-MLR does not use values stored in a replicated working memory for inferences. The rules use the attributes of objects in the real-time application domain as the facts to be considered for inferences. Thus, changes of values are frequent and generated by asynchronous data acquisition processes in the domain application. Therefore, to assume the best inference as possible at each moment, the algorithm pays the price of always reading the current values in the domain attribute at the precise time that the rule is being investigated. The rules don't have an associated memory. RT-MLR directly associates attributes in the domain applications with rules using those attributes. Thus, only rules that use those attributes are investigated when any significant change is detected in the value of the attributes.

The attributes of domain classes that are used in the classes of rules should not be defined by the application as attributes of Java basic types. They must be defined by inheriting from classes provided by RT-MLR that encapsulate the basic types of Java (numeric, Boolean, etc.). When the values of these attributes are changed by any calling object in the domain application, the rules associated with that particularly attribute are directly investigated. Furthermore, users can establish a confidence interval for attribute values in order to define the minimum variation accepted as relevant to rules investigation. Thus, little variances in values, that aren't considered relevant, will not cause rules investigations. RT-MLR is multi-thread safe, as usual in real-time domains.

Fuzzy logic inference

Fuzzy Logic is a theory proposed by Zadeh [19] that works with linguistic concepts. Logic rules use these linguistic concepts to inference. The linguistic concepts are modeled with membership functions that establish a degree of truth or degree of membership of the linguistic concept to a value domain. The membership, ranging from 0 to 1, is used by the inference engine to fire the rules. By dealing only with concepts such as near, far, critical, slow, fast, big or small, the rules allow a greater expression to transcribe the knowledge of common sense.

For fuzzy logic, the investigation of the rules into RT-MLR is also linked to the changing values of the variables that appear in the premises of the rules. The left side of each fuzzy rule is evaluated based on the fuzzy variables and fuzzy operators. If the membership of the premise is greater than zero, the rule is fired, generating fuzzy an output function for each fired rule. The output functions of the fired rules are aggregate and the final function is then deffuzified to produce a scalar value for the output variable. For each fuzzy output variable is necessary to define previously a set of rules. All the fuzzy rules of the set are investigated when one of then is fired, since the values in the premises of these rules may remain unchanged but their current values may also fire the respective rules, then causing an influence in the final value of the output variable.

Temporal logic inference

There are several approaches proposed in the literature to consider the time domain in reasoning models. The RT-MLR uses the concept of Interval Temporal Logic [20, 21]. The basic element used is the event, which can be viewed as a fact associated with a given time. In fact, the event is a value with a time stamp associated. Since events must be related with attributes previously defined with the framework data types, is not necessary for the application to define explicitly an event. Instead of this, the events will be automatically defined and memorized by the framework as the temporal operators dealing with events are instantiated on logical rules. There is a set of time operators that performs operations like *ThereWasEqual (A,B,T1,T2)*. This operator returns true if in any time in the past, between time T1 and Time T2, the value of A was equal to the value of B. Like this, there are other comparing operators, counting operators, medium value operators and operators that establish relations like before, after, during and until, always associated with time intervals in the past. Since the operators appear in rules, the interesting events are registered by the framework, exactly by the maximum time they are needed to remain memorized for investigation purposes.

There is not a set of temporal logic rules. Temporal logic operators are used in first order logic rules, since their results are also Boolean or numeric.

4 Performing data fusion with RT-MLR

Using RT-MLR to make data fusion doesn't mean to establish a clear hierarchy in data fusion. In fact, for the RT-MLR such a hierarchy does not exist from a conceptual point of view. The main goal is to use the RT-MLR to implement inferences from features and behaviors detected in a C^4ISR with sets of logic rules that capture relationships based on expert knowledge. From a classical approach, those inferences may be classified as level 1, level 2 or level 3 data fusion processes, but this doesn't mind from an architectural point of view.

The architecture of system where fusion was applied imposes some restrictions. For example, object fusion must be done at track level, not at measurement level, since each sensor has a node that performs level 0 fusion, provides Kalman filter estimation for object kinematic and delivers tracks to a central fusion node. The RT-MLR lies in this node besides a fusion application that specifies the rules and reports conclusions to a visualization node.

Level 1 fusion example

The fusion node receives periodically the individual tracks from each sensor node and performs a central fusion to decide which tracks from different radar nodes refer to the same targets. This fusion is based on fuzzy sets.

Each new track received by fusion node is maintained in a separate list by the fusion application. The application instantiates a set of rules each time a message with a new track is received. Each instantiated rule establishes a fusion relation between the new track and another track from other radar. This associates the new track with all the other tracks from the other sensors. Thereafter those rules will be investigated each time an actualization of data track is received from sensor node.

The track data must be aligned in time before comparison. So, for each compared track, the fusion rule calls a method that aligns the track's kinematics for the current time. The association rule uses the aligned attributes to investigate the correlation of tracks. The following attributes are used: position, speed, course and size.

The correlation measure is done by the membership value of each fuzzy set. The fuzzy sets are defined accordingly to the precision of each sensor. The fuzzyfied value is the module of the difference (Δ) between each attribute. The fuzzy sets are presented in figure 2.

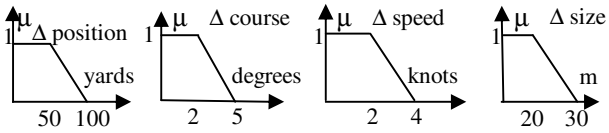
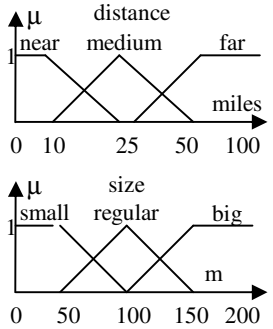


Figure 2 –Fuzzy sets used for track fusion

The support values shown in figure 2 are just examples, since the actual values depend on the precision of the involved sensors. The membership values obtained by entering the attribute values into the fuzzy functions are the measures of similarity between tracks.

The attribute size is not directly provided by radar nodes. Radar nodes just send a counter of how many times the target was illuminated by radar wave. This counter and the distance from target to radar are correlated with the target size. A fuzzy rule set with nine rules is provided to obtain the size in meters from the fuzzy sets and rules in figure 3.



c \ d	near	med	far
lw	S	S	R
av	R	R	B
hi	R	B	B

Figure 3 –Fuzzy sets and rules used for size estimation

The membership values obtained from the sets in figure 2 are then contrasted with a threshold in order to decide if the attributes from the different tracks belong to the same object. In other works, the threshold reflects the precision of sensors [22]. In this work we use thresholds that vary with the condition of measurements made by radars. Each radar node has a Kalman filter (KF) that provides the estimation used as input for our fusion model. The KF also provides a performance measure that is the error variance (σ) of the estimation done by filter. The standard deviation

($\sigma^{1/2}$) is used to calculate the thresholds for track position (μ_{TP}), track course (μ_{TC}) and track speed (μ_{TS}). The threshold for track size (μ_{TZ}) comes from the standard deviation of size that is calculated by the model illustrated in figure 3. The fusion rule uses a first order logic that deals with fuzzy values. Its schema is presented if figure 4.

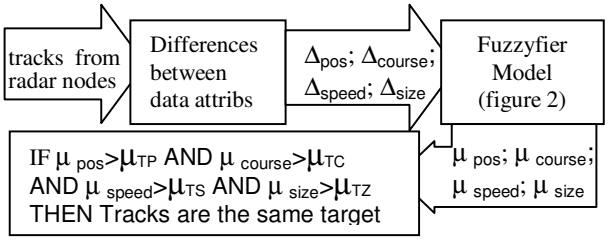


Figure 4 – Track fusion decision rule

Level 2 fusion example

A level 2 data fusion example was implemented using the RT-MLR framework. The objective is offers a measure for the hostility of each unknown track. The measure varies from 0 to 100 in a scale that increases as the hostility increases. Differently from the previous explained track fusion rules, this data fusion is totally fuzzy. A set of fuzzy rules were created, associating rules defined by a domain expert to define the hostility. The model is an example of level 2 fusion because it correlates tracks with aspects of scenario, like commercial routes. An example was created with rules that consider the following variables: the angle between the track course and the direction of a commercial route; the proximity of the track position and the axis of the route; the size of the track; and, the velocity of the track. Fuzzy sets were created and the supports were defined by a domain expert (figure 5).

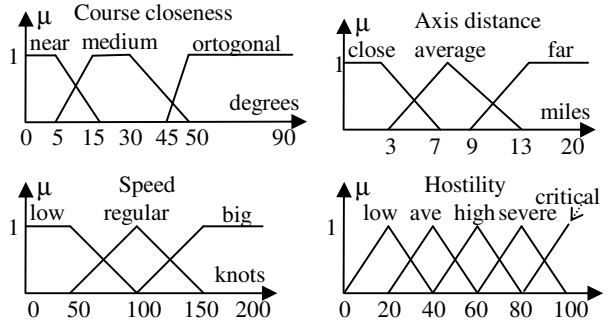


Figure 5 – Fuzzy sets used for hostility estimation

A fuzzy rule that reflects the expert belief that the track is probably a freighter may be created relating a track and a route to define the hostility. The rule may be:

```
IF course closeness IS near AND axis distance IS
close AND speed IS regular AND size IS big THEN
hostility IS low
```

Other rules may consider different aspects of scenario to conclude something about hostility. For instance, an expert can create a rule associating a track and the ship where the system is installed, specifically to detect and classify targets flying straight to the ship. The rule may be:

```
IF course closeness IS near AND distance IS
close AND size IS small AND speed IS big THEN
hostility is severe.
```

So, many different rules may be activated at the same time, combining the results to infer a final hostility for tracks.

Level 3 fusion example

It was also prepared an example to implement a level 3 fusion feature using the RT-MLR framework. The objective is estimating time that will be necessary to achieve a combat position with an enemy track. The calculus uses temporal logic considering events happened in the past, like amount of course changes in the last 10 minutes and emissions detected in the last 30 minutes. The rule uses temporal operators like these:

```
IF ThereWasEqual(track.emiting,true,1800,0)AND
Count(track.changingCourse,true,600,0)>2
THEN interceptionFireCircleTime(me,track)
```

On the JDL model such reasoning may be considered of level 3, since it provides a prevision about future events from the past behavior of a threat.

Test architecture

Simulation architecture was created to provide sensor tracks for fusion models and to enable the scenario visualization of fusion performance. The architecture has two radar nodes, one visualization node and one fusion node. The radar nodes receive simulated measurements (radar dots) generated by the visualization node. The simulated dots were created by an operator with a desired kinematics. They represent the measurements made by sensors added by a Gaussian process noise and a Gaussian measurement noise. The noise range for each sensor is selected by the operator. The operator also asks for the radar nodes to create tracks. After tracks are created and associated with the dots, a Kalman filter provides kinematic estimations. The estimated tracks are sent to the visualization node and to the fusion node. Data fusion is processed by a monitoring application that uses the RT-RML to implements its logic approach. The results of data fusion provided by fusion node are then sent to visualization node to be exhibited.

5 Test results

Three scenarios were created to test level 1 track fusion model. First scenario has two vessels separated by 5 miles and moving in intersect courses. Second scenario has a small ship and a helicopter, in the same course. The helicopter has a bigger speed and overlaps the ship's position. Third scenario has two ships in maneuvering trajectories (fig. 6).

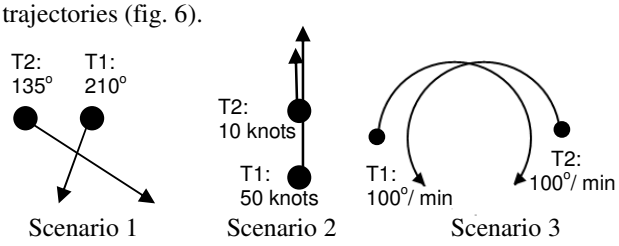


Figure 6 – Test scenarios for track fusion

The tests were performed adding Gaussian noise to dot position in order to vary the quality of Kalman Filter estimation. Two test batteries were performed, using lower (σ=5 meters) and higher (σ =20 meters) Gaussian noises. Each scenario has two objects that were created and sent to two sensor nodes. So, four different tracks would appear in visualization node before fusion. Fusion node rules try to associate all four tracks and should associates two and don't associate the other two. The average values of confusion matrix taken over 20 executions were presented in table 1. Hypothesis h=1 means tracks fusion.

Table I1 – Track fusion evaluation measures

	h=1	h=0
Same	T_P	F_N
Different	F_P	T_N

$$error = \frac{F_P + F_N}{n} \qquad prec = \frac{T_P + T_N}{n}$$

$$cover = \frac{T_P + F_P}{n}$$

$$pconf = \frac{T_P}{T_P + F_P} \qquad nconf = \frac{T_N}{T_N + F_N} \qquad \sup = \frac{T_P}{n}$$

		Error	Prec	Pconf	Nconf	Sup	Cover
Scenario 1	**σ = 5**	0.01	0.99	1	0,99	0.49	0.49
	σ = 20	0.04	0.96	0.99	0.92	0.46	0.46
Scenario 2	**σ = 5**	0.02	0.98	1	0.96	0.48	0.45
	σ = 20	0.04	0.96	1	0.97	0.46	0.46
Scenario 3	**σ = 5**	0.01	0.99	1	0,99	0.49	0.46
	σ = 20	0.02	0.98	1	0.96	0.48	0.49

Actual data

The track fusion process was also tested whit actual data recorded by a frigate operating in the Brazilian coast. Data came from RTN30X tracker radar and from Scanter1000 navigation radar. Data provided by Scanter1000 came from two targets. One of these targets was also tracked by the RTN30X. The common target is a ship running in a 7.5 miles trajectory with speed varying from 10 to 20 knots and with a sharply 90 grade turn in the middle of trajectory. The fusion process involved three tracks from two radars and the obtained results are shown in table 2.

Table II – Actual data fusion evaluation measures

Error	Prec	Pconf	Nconf	Sup	Cover
0,001	0.978	1	0,977	0.503	0.503

Tracks from different target were never fused and tracks from same target were not fused only 9 times in a 556 sample universe. The non-fusing situations occurred during the turn of the vessel due to the much worse course estimation from Scanter1000 compared with RTN30X.

Level 2 fusion results

One test scenario with three tracks was created to test the level 2 hostility fusion model. First track is a freighter moving near a commercial route. Second track is a fighter in a course of interception. The third is a medium size ship out of commercial routes. Figure 7 shows the scenario.

Test was performed using different Gaussian noises for each sensor (σ=5 meters for one radar and σ =20 meters for the other). Rules calculated hostility during the time of test (around 3 minutes). Mean and standard deviation of hostility values were calculated for each track.

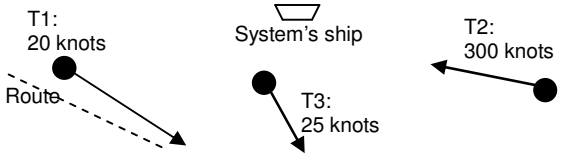


Figure 7 – Test scenario for hostility estimation rules

One test battery was performed using different Gaussian noises for each sensor (σ=5 meters for one radar and σ =20 meters for the other). Hostility was calculated by rules during the time of test (around 3 minutes). Mean and standard deviation of hostility values were calculated for each track. Table 2 shows the results.

Table III – Hostility evaluation

		Hostility	
		Mean	SD
Track 1	**σ = 5**	19,5	1,3
	σ = 20	20,1	2,1
Track 2	**σ = 5**	95	0
	σ = 20	95	0
Track 3	**σ = 5**	40	0
	σ = 20	40	0

Conclusions varied only for track 1 because this is the only one which is affected by two different rules. The other ones fired only one rule each, causing that the deffuzified value corresponds to the maximum of output fuzzy function.

Level 3 fusion results

For this test was created a scenario with three tracks. During the test the tracked threat was programmed to make the course changes and these events were registered by RT-MLR. The events corresponding to the detected emissions were artificially generated, since there is no sensor to provide information about electromagnetic emissions in the simulator. The range of threat's weapons is also manually provided, since, in an actual environment, this information should be read from a data bank after a meta-classification of the threat.

After the RT-MLR detected the events related in the rule, the time prevision became to be calculated and the information was sequentially actualized on the visualization node attached to the tracked symbol.

6 Conclusion

This paper presented an integrated approach for fusing context information with a model that allows the reasoning based on multiple-logic rules and supports concurrent processing of real-time applications. Knowledge was modeled in rules as association classes in the domain

space. The framework created (RT-MLR) to support multiple-logic reasoning deals with rules in native application language (Java), so rules doesn't need to be previously interpreted, increasing development time and accelerating learning curves. RT-MLR has an automated activation rule system, fired by significant changes in domain value attributes that enable user tuning rule activation. RT-MLR increases knowledge expressiveness supporting fuzzy and temporal reasoning beyond traditional first order logic.

Despite the levels of data fusion do not matter because the rule-based approach works on whatever granularity and domain, RT-MLR was used to support a level 1 to 3 fusion application based on knowledge rules for a C4I real-time application. A fuzzy inference set of rules was first used to infer objects' sizes. After that, sizes, speeds, courses and positions of tracks were used in a level 1 track to track fusion model. This fusion is based on comparing fuzzy measures with thresholds dynamically adjusted according to covariance matrix error estimations provided by KF. Results show that the model fuses correctly the targets the greatest part of the time. The model couldn't fuse the targets only when differences of targets' measures are above estimations errors provided by KF. Level 2 fusion was performed using navigation routes and targets' routes to infer a hostility grade. Level 3 fusion was performed using past behaviors of targets captured with temporal logic rules to infer future times of combat position achievement.

Level 2 and level 3 may be mightier if more high level information can be provided (e.g.: data base information) and other resources, like Bayesian inference, can be aggregate to the inference system. These will be probably the next steps in the RT-MLR development.

References

[1] D. Hall, and S. McMullen, *Mathematical Techniques in Multisensor Data Fusion*, Artech House Publications, 2004.

[2] Y. Bar-Shalom and W. Blair, *Multitarget-Multisensor Tracking: Applications and Advances*. 2000

[3] O. Kessler, *Functional Description of the Data Fusion Process*, Warmininster, PA: Office of Naval Technology, Naval Air Development Center, 1992

[4] A. Steinberg, C. Bowman and E. White Jr., *Revisions to the JDL Data Fusion Model*, 3rd NATO/IRIS conferation, Quebec City, Canada, 1998

[5] R. Rosati. *On combining description logic ontologies and nonrecursive datalog rules*. In Proc. of 2nd Int. Conf. on Web Reasoning and Rule Systems , 2008.

[6] J. Juarez et al., *Computing context-dependent temporal diagnosis in complex domains*. Expert Syst. Appl. 35, 3 (Oct. 2008), 991-1010. 2008.

[7] S. Bhansali and B. Grosof, *Extending the SweetDeal Approach for E-Procurement using SweetRules and RuleML*, Proc. International Conference on Rules and Rule Markup Languages (RuleML-2005). 2005.

[8] Ó. Corcho and A. Gómez-Pérez. *A Roadmap to Ontology Specification Languages*. In Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management, October 02 – 06. 2000

[9] R. MacGregor *Inside the LOOM clasifier*. SIGART bulletin. #2(3):70-76. June, 1991.

[10] E. Motta, *Reusable Components for Knowledge Modelling*. IOS Press. Amsterdam. 1999.

[11] M. Ahmad and R. Colomb. *Managing ontologies: a comparative study of ontology servers*. In Proceedings of the Eighteenth Conference on Australasian Database - Volume 63 - Ballarat, Victoria, Australia. 2007.

[12] M. Kifer et al. *Logical Foundations of Object-Oriented and Frame-Based Languages*. Journal of the ACM. 1995.

[13] A. Farquhar et al. *The Ontolingua Server: A Tool for Collaborative Ontology Construction*. Proceedings of KAW96. Banff, Canada, 1996.

[14] K. Baclawski et al. *Extending the UML for Ontology Development.* International Journal of Software and Systems Modeling, 1(2): 142-156. 2002.

[15] Q. Mahmoud, *Getting Started With the Java Rule Engine API (JSR 94): Toward Rule-Based Applications* web: http://java.sun.com July 26, 2005

[16] C. Forgy, "*Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem*", Artificial Intelligence Journal, 19, pp 17–37, 1982

[17] M. Dörflinger *Interpreting Courteous Logic Programs,* Diploma Thesis .2005

[18] B. Grosof, *Courteous logic programs: Prioritized conflict handling for rules*, Tech. report, IBM T.J. Watson Research Center, IBM Research Report RC 20836. 1997.

[19] L. Zadeh, *Fuzzy Sets*, Information and Control, vol 8, pp 338-353, 1965

[20] B. Bennett and A. Galton. *A unifying semantics for time and events*. Artif. Intell. 153, 1-2, 13-48. 2004.

[21] J. Allen. *An interval-based representation of temporal knowledge*. In Proc. of the 7th International Joint Conference on Artificial intelligence. 1981.

[22] M. Tummala and S. A. Midwood Multi Sensor Data Fusion Using Fuzzy Association Techniques. 1998