A Knowledge Processing Middleware Framework and its Relation to the JDL Data Fusion Model

Fredrik Heintz and Patrick Doherty
Department of Computer and Information Science
Linköpings universitet, Sweden
{frehe, patdo}@ida.liu.se

***Abstract*— Any autonomous system embedded in a dynamic and changing environment must be able to create qualitative knowledge and object structures representing aspects of its environment on the fly from raw or preprocessed sensor data in order to reason qualitatively about the environment and to supply such state information to other nodes in the distributed network in which it is embedded. These structures must be managed and made accessible to deliberative and reactive functionalities whose succesful operation is dependent on being situationally aware of the changes in both the robotic agent's embedding and internal environments. DyKnow is a knowledge processing middleware framework which provides a set of functionalities for contextually creating, storing, accessing and processing such structures. The framework is implemented and has been deployed as part of a deliberative/reactive architecture for an autonomous unmanned aerial vehicle. The architecture itself is distributed and uses real-time CORBA as a communications infrastructure. We describe the system and show how it can be used to create more abstract entity and state representations of the world which can then be used for situation awareness by an unmanned aerial vehicle in achieving mission goals. We also show that the framework is a working instantiation of many aspects of the revised JDL data fusion model.**[1]

I. Introduction

In the past several years, attempts have been made to broaden the traditional definition of data fusion as state estimation via aggregation of multiple sensor streams.

One of the more successful proposals for providing a framework and model for this broadened notion of data fusion is the U.S. Joint Directors of Laboratories (JDL) data fusion model [1] and its revisions [2], [3], [4].

The gap between models, such as the JDL data fusion model, which describe a set of functions or processes which should be components of a deployed system to the actual instantiation of data fusion in a software architecture in this broader sense is very much an open and unsolved problem. In fact, it is the belief of the authors that architectural frameworks which support data and information fusion in this broader sense have to be prototyped, tested, analyzed in terms of performance and iterated on, in order to eventually support all the complex functionalities proposed in the JDL data fusion model.

In this paper, we will describe an instantiation of parts of such an architectural framework which we have designed, implemented, and tested in a prototype deliberative/reactive software architecture for a deployed unmanned aerial vehicle (UAV) [5], [6]. The name given to this architectural framework which supports data fusion at many levels of abstraction is DyKnow[2]. DyKnow is a knowledge processing middleware framework used to support timely generation of state information about entities in the environment in which the UAV is embedded and entities internal to the UAV itself. The latter is important for monitoring the execution of the autonomous system itself.

The DyKnow system is platform independent in the sense that the framework can be used in many different complex systems. Consequently, we believe it is of general interest to the data fusion community at large. One aspect of DyKnow which is particularly interesting is the fact that it was designed and prototyped independently of any knowledge about the JDL data fusion model. The requirements for specification were those necessary to reason about world state at very high levels of abstraction and to be able to take advantage of artificial intelligence techniques for qualitative situation assessment and monitoring of the UAV and dynamic entities in its embedded environment. It turns out that the resulting prototype can be used when implementing the JDL data fusion model and provides insight into many of the details that are important in making such architectures a reality. For example, such systems are not strictly hierarchical and often involve complex interactions among the layers. This implies that it is not feasible to specify and implement each level separately. This perceived weakness in the JDL model was in fact pointed out by Christensen in a recent panel debate concerning the JDL model [7].

A. Structure of the Paper

The paper is structured as follows. In section II, an overview of the important concepts used in the definition of the DyKnow framework is given. In section III, we consider the DyKnow framework in the context of the revised JDL data fusion model. In section IV, we describe a UAV scenario involving vehicle identification and tracking, where DyKnow has been used to advantage. In section V, some work related to the DyKnow framework is presented. In section VI, we conclude and summarize the work.

[1]This work is supported in part by the Wallenberg Foundation, Sweden and an NFFP03 research grant (COMPAS)

[2]"DyKnow" is pronounced as "Dino" in "Dinosaur" and stands for *Dynamic Knowledge and Object Structure Processing*.

II. DyKnow

The main purpose of DyKnow is to provide generic and well-structured software support for the processes involved in generating object, state and event abstractions about the external and internal environments of complex systems, such as our experimental UAV system. Generation of objects, states and events is done at many levels of abstraction beginning with low level quantitative sensor data. The result is often qualitative data structures which are grounded in the world and can be interpreted as knowledge by the system. The resulting structures are then used by various functionalities in the deliberative/reactive architecture for control, situation awareness and assessment, monitoring, and planning to achieve mission goals.

A. Knowledge Processing Middleware

Conceptually, DyKnow processes data streams generated from different sources in a distributed architecture. These streams may be viewed as representations of time-series data and may start as continuous streams from sensors or sequences of queries to databases. Eventually they will contribute to definitions of more complex composite knowledge structures. Knowledge producing processes combine such streams, by abstracting, filtering and approximating as we move to higher levels of abstraction. In this sense, the system supports conventional data fusion processes, but also less conventional qualitative processing techniques common in the area of artificial intelligence. The resulting streams are used by different reactive and deliberative services which may also produce new streams that can be further processed. A knowledge producing process has different quality of service properties, such as maximum delay, trade-off between data quality and delay, how to calculate missing values and so on, which together define the semantics of the chunk of knowledge created. The same streams of data may be processed differently by different parts of the system relative to the needs and constraints associated with the tasks at hand.

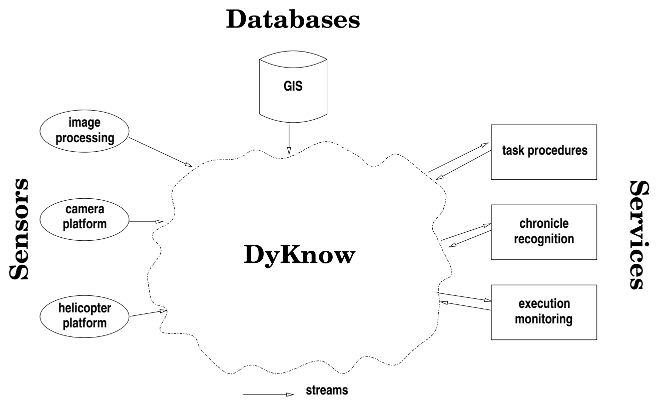

Fig. 1. An instantiation of the DyKnow knowledge processing middleware.

In Fig. 1 an example of a concrete instantiation of the DyKnow framework that we use in our experimental UAV architecture is shown. There are three virtual sensors, the image processing subsystem, the camera platform and the helicopter platform. We have a geographical information system (GIS) which is a database that contains information about the geography, such as road structures and buildings, of the region we are flying in. The services include the reactive task procedures which are components linking the deliberative services with the camera and helicopter controllers, a chronicle recognition engine for reasoning about scenarios, and a temporal logic progression engine that can be used for execution monitoring and other tasks based on the evaluation of temporal logic formulas.

B. Ontology

Ontologically, we view the external and internal environment of the agent as consisting of physical and non-physical *entities*, *properties* associated with these entities, and *relations* between these entities. The properties and relations associated with entities will be called *features*. Features may be static or dynamic. Due to the potentially dynamic nature of a feature, that is, its ability to change values through time, a *fluent* is associated with each feature. A fluent is a function of time whose range is the feature's type. Some examples of features are the *velocity* of an object, the *road segment* of a vehicle, and the *distance between* two car objects.

C. Object Identifiers and Domains

An *object identifier* refers to a specific entity and provides a handle to it. Example entities are "the colored blob", "the car being tracked" or "the entity observed by the camera". The same entity in the world may have several different identifiers referring to it and a composite entity (consisting of a set of entities) can be referred to with a single identifier. Three examples of this are shown in Fig. 2. In the first example we have two representations of the same entity, in this case **blob1** and **blob2** which could be blobs extracted from two different pictures by the image processing system, that we may or may not know refer to the same entity. In the second example we have **blob3** and **car1** which represents two different aspects of the same entity. An example of object identifiers referring to a composite entity may occur when several object identifiers refer to the same entity at different levels of abstraction, such as the car entity referred to by **car2** and the hood and wheel entities referred to by **hood** and **wheel**.

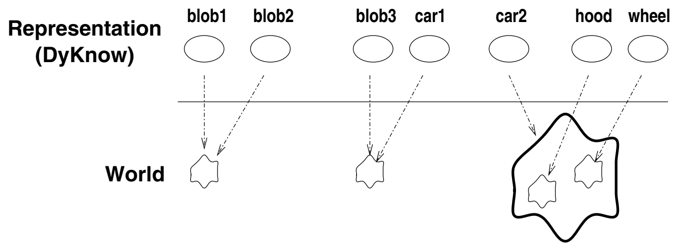

Fig. 2. Examples of relations between object identifiers and entities.

An agent will most often not know the exact relation between object identifiers, whether they refer to the same entities or not, because they are generated for different reasons and often locally. In Section II-F we present some mechanisms for relating them. The basic constraints placed on object identifiers are that they are unique and only assigned to an entity once (single assignment).

An *object domain* is a collection of object identifiers referring to entities with some common property, such as all red entities, colored blobs found in images or qualitative structures such as the set of cars identified in a mission. An object identifier may belong to more than one domain and will always belong to the domain "top". Object domains permit multiple inheritance and have a taxonomic flavor. The domains an object identifier belongs to may change over time, since new information provides new knowledge as to the status of the entity. This makes it possible to create domains such as "currently tracked entities" or "entities in regions of interest".

D. Approximating Fluents

The fluents associated with features are the target of the representation in DyKnow. A feature has exactly one fluent in the world which is its true value over time. The actual fluent will almost never be known due to uncertain and incomplete information. Instead we have to create approximations of the fluent. Therefore, the primitive unit of knowledge is the *fluent approximation*. In DyKnow there are two representations for approximated fluents, the *fluent stream* and the *fluent generator*. The fluent stream is a set of observations of a fluent or samples of an approximated fluent. The fluent generator is a procedure which can compute an approximated value of the fluent for any time-point. Since a fluent may be approximated in many different ways each feature may have many approximated fluents associated with it. The purpose of DyKnow is to describe and represent these fluent generators and fluent streams in such a way that they correspond to useful approximations of fluents in the world.

There are two types of fluent approximations, primitive and computed fluent approximations. A primitive fluent approximation acquires its values from an external source, such as a sensor or human input, while a computed fluent approximation is a function of other fluent approximations. To do the actual computation a procedural element called a *computational unit* is used. The computational unit is basically a function taking a number of fluent approximations as input and generating a new fluent approximation as output.

Since a fluent generator represents a total function from time to value and a fluent stream only represents a set of samples a fluent generator created from a fluent stream must be able to estimate the value at any time-point whether or not a sample exists at that time-point. Since this estimation can be made in many different ways, depending on how the samples are interpreted, it is possible to create many different fluent generators from a single fluent stream. From each of these fluent generators we can generate many different fluent streams by sampling the fluent generator at different time-points. How these transformations are done are described by declarative policies. The *fluent generator policy* specifies a transformation from a fluent stream to a fluent generator, and the *fluent stream policy* specifies a transformation from a fluent generator to a fluent stream. A fluent generator policy may be viewed as the context in which the observations in a fluent stream are interpreted. The resulting fluent approximation is the meaning of the feature in that context.

We are primarily interested in distributed systems where the sources of data often determine its properties such as quality and latency. These and other characteristics such as access and update constraints must be taken into account when generating and using fluent approximations associated with specific data sources. *Locations* are introduced as a means of indexing into data sources which generate fluent approximations associated with specific features. A feature may be associated with several fluent approximations located in different places in the architecture, but each fluent approximation must be hosted by exactly one location. By representing these different places with locations we make it possible to model and reason about them.

E. States and Events

Two important concepts in many applications are states and events. In DyKnow a *state* is a composite feature which is a coherent representation of a collection of features. A state synchronizes a set of fluent approximations, one for each component feature, into a single fluent approximation for the state. The value of the new fluent approximation, which actually is a vector of values, can be regarded as a single value for additional processing. The need for states is obvious if we consider that we might have several sensors each providing a part of the knowledge about an object, but whose fluent approximations have different sample rates or varying delays.

A concrete example is that we have streams of positions given in pixel coordinates and streams of camera states describing the position and orientation of the camera. In order to find out what coordinate in the world a pixel position corresponds to we need to synchronize these two streams. If we have a position at time-point t we want to find a camera state which is also valid at time-point t. In the simplest case there exists such a sample, but in a more general (and realistic) case we have to either find the "best" camera state in the stream or estimate what the camera state was at time-point t from the observed samples.

The problem of creating coherent states from data streams is non-trivial and can be realized in many different ways. In DyKnow the synchronization strategy is described by a policy called the *state policy*. If the existing pre-defined synchronization strategies are not adequate for an application then a computational unit can be created and used as a general mechanism for extracting states.

An *event* is intended to represent some form of change or state transition. Events can either be primitive, e.g. a sample received from a sensor can be seen as an event, or generated, e.g. the event of the approximated fluent f reaching a peak in its value. Generated events can either be extracted from fluent approximations or computed from other events. In DyKnow it is possible to define primitive events on approximated fluents, mainly *change events* such as fluent approximation f changed its value with more than 10% since the last change event.

DyKnow currently has support for two types of computed events. The first is the evaluation of linear temporal logic (LTL) formulas becoming true or false. The second is the recognition of scenarios, called chronicles, composed of temporally related events, expressed by a simple temporal constraint network. An LTL formula is evaluated on a state stream containing all the features used by the LTL formula, so the state extraction mechanism mentioned above is a prerequisite for the LTL formula evaluation. The chronicle recognition engine, on the other hand, takes events representing changes in fluent approximations as input and produces other events representing the detection of scenarios as output. These can be used recursively in higher level structures representing complex external activity such as vehicle behavior.

F. *Objects, Classes and Identity*

Grounding and anchoring internal representations of external entities in the world is one of the great open problems in robotics. Consequently, middleware systems for knowledge processing must provide suitable support for the management of representations and their relation to the external entities they represent.

We require a mechanism for reasoning about the relation between object identifiers, including finding those object identifiers which actually codesignate with the same entity in the world. When two object identifiers are hypothesized as referring to the same entity in the world, a link is created between them. The collection of object identifiers referring to the same entity in the world and the links between them is called an *object linkage structure*. This represents the current knowledge about the identity of the entity.

We have separated the object identity (i.e. which entity in the world an object identifier refers to) from the object state. Classes provides a mechanism for specifying certain relationships between the two, by regulating the minimum state required for certain classes of object identifiers. Links provides the mechanism for describing relations between object identifiers, i.e. to reason about the identity of object identifiers.

The object linkage structure makes it possible to model each aspect of an entity as a class and then provide the conditions for when an instance of the class should be linked to an instance of another class. For example, in the traffic domain we model the blobs extracted by the image processing system as separate object identifiers belonging to the class VisionObject and objects in the world as object identifiers belonging to the class WorldObject. We also provide a link type between these classes in order to describe the conditions for when a vision object should be hypothesized as being a world object. This simplifies the modeling since each aspect can be modeled separately, it also simplifies the classification, tracking and anchoring of the objects.

To describe a collection of object identifiers representing an aspect of an object, a *class* is used. A class describes what fluent approximations all instances should have and includes four constraints, the *create*, *add*, *codesignate*, and *maintain constraints*, that regulate the membership of the class. If a create constraint is satisfied then a new object identifier is created and made an instance of the class. If the add constraint for an object identifier is satisfied then it is considered an instance of the class and it is added to the class domain. A codesignation constraint encodes when two objects of the class should be considered identical. The maintain constraint describes the conditions that always should be satisfied for all instances of a class. If the maintain constraint is violated the object identifier is removed from the class. Constraints are LTL formulas that only use the fluent approximations required by the class and link definitions.

A link type represents the potential that objects from two classes might represent the same entity. The link specification contains three constraints, the *establish*, *reestablish*, and *maintain constraints*. A link specification might also contain fluent approximations representing specific properties that result from the entities being linked together. If an establish constraint, defined on objects from the linked-from class (a link is directed), is satisfied then a new instance of the linked-to class is created and a link instance is created between the objects. An example of this is given in Fig. 3 if read from left to right. The establish constraint represents the conditions for assuming the existence of another, related, aspect of an entity. For example, in our application we assume all vision objects are related to a world object, therefore a new world object is created if a vision object is not already linked to one. A reestablish constraint encodes when two existing objects, one from each class, should be linked together. An example of this is given in Fig. 4 if read from left to right. When a link instance is created a maintain constraint, which is a relation between the two objects, is set up in order to monitor the hypothesis that they are actually referring to the same entity in the world. If it is violated then the link instance is removed which is the case in Fig. 4 if read from right to left.

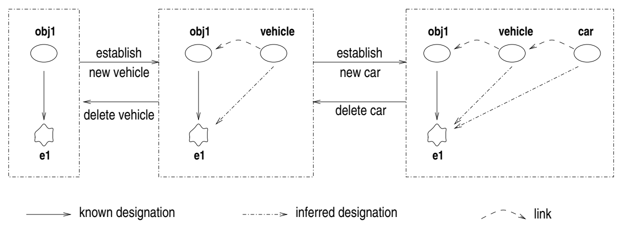

Fig. 3. An example of creating and deleting a linked object.

For a more detailed account of object linkage structures in DyKnow, see [8].

G. *Implementation*

All of the concepts described above are implemented in C++ using the TAO/ACE [9] CORBA implementation. The DyKnow implementation provides two services. The Domain

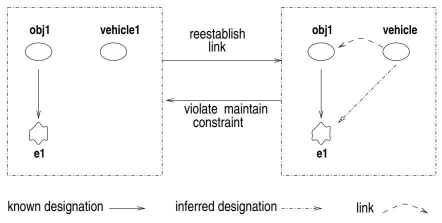

Fig. 4. An example of reestablishing a link and violating its maintain constraint.

and Object Manager (DOM) and the Dynamic Object Repository (DOR). The DOM is a location that manages object identifiers, domains, classes and objects. The DOR manages fluent approximations, states and events.

III. JDL Data Fusion Model

The JDL data fusion model is the most widely adopted functional model for data fusion. It was developed in 1985 by the U.S. Joint Directors of Laboratories (JDL) Data Fusion Group [1] with several recent revisions proposed [2], [3], [4].

The data fusion model originally divided the data fusion problem into four different functional levels [1], later a level 0 [2] and a level 5 [3] was introduced. The levels 0-4 as presented in [2] and level 5 as presented in [3] are shown in Fig. 5.

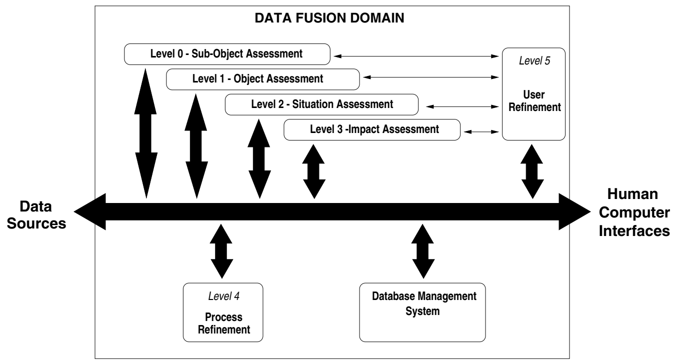
DATA FUSION DOMAIN
Level 0 - Sub-Object Assessment
Level 1 - Object Assessment
Level 2 - Situation Assessment
Level 3 -Impact Assessment
Level 5
User Refinement
Data Sources
Human Computer Interfaces
Level 4
Process Refinement
Database Management System

Fig. 5. Revised JDL data fusion model from [3].

In this section we will go through each of the levels and describe how the implementation of its functionalities can be supported by DyKnow. It is important to realize that DyKnow does not solve the different fusion problems involved, but rather provides a framework where different specialized fusion algorithms can be integrated and applied in a data fusion application.

A. Level 0 Sub-Object Assessment

On this level, fusion on the signal and sub-object level should be made. Since the object identifiers can refer to any entity, including sensors and entities which may be an object on its own or not, we can represent and work on features such as “signal from sensor S” and “property of blob found by image processing system”. Fusion on this level would be implemented by computational units. The purpose of the computational units is to reduce the noise and uncertainty in the fluent approximations in order for the higher layers to get the best possible approximations to work with. The sub-object features are used mostly at level 1 to create coherent object states.

B. Level 1 Object Assessment

On this level, sub-object data should be fused into coherent object states. In DyKnow there are mainly two functionalities used, state aggregation and the creation of object linkage structures. A state collects a set of sub-object features into a state which can be used as a single value similar to the value of a struct in C. Linkage structures are then used to reason about the identity of objects and to classify existing objects.

In the linkage structure two special cases of data fusion are handled. The first is the fusion of codesignated objects, i.e. when two or more objects from the same class are hypothesized as actually being the same entity, where the knowledge related to each of these objects has to be fused into a single object. There are two modes of doing this fusion; it can either be done continuously, so that all the individual object instances still exist, but their content is continually fused into a new object, or it can be a one-shot fusion where all knowledge at the moment of the codesignation is fused into a single new object and the old objects are deleted.

The second special case is the fusion of several different objects from different classes into a single object. This is the case when an object is linked-to from more than one object of different classes. For example, assume our robot has both a sonar and a camera, each sensor provides sub-object fluent approximations containing the sensor readings related to entities in the world. If the entity sensed by the sonar and the entity sensed by the camera are hypothesized as being the same entity, the position according to the camera fluent approximation and the position according to the sonar fluent approximation must be merged into a single position fluent approximation, representing the combined knowledge about the entity. In DyKnow this would be done using a computational unit which takes two fluent streams as input one with camera positions and one with sonar positions and using an appropriate algorithm computes a new fluent approximation, the combined position in the world. The stream will be generated as long as the hypothesis that the three objects are the same is maintained.

In DyKnow fluent approximations from level 1 mainly interact with level 2 by providing coherent object states for computing and detecting situations. Level 3 is also very important since it is responsible for checking the hypothetical object linkage structures by continually checking the impact of new observations on the current hypotheses. Since the computations on this level can be time consuming, the interactions with level 4 and level 5 are also important in order to maintain a steady update of the most important fluent approximations for the moment as decided by the system and the user.

C. Level 2 Situation Assessment

On this level, relations between objects fused together on the previous levels should be detected as well as more complex situations being represented and recognized. The detection of events, both primitive and computed, are important tools to model situations. Computed events can e.g. be temporal logic formulas or chronicles describing temporal relations between events. In this fashion different features are fused together over time in order to extract more abstract situations that are features in themselves. Collections of objects can also be aggregated into states in order to synchronize them to a coherent situation, just as collections of fluent approximations can be collected into states.

Properties, relations, states and events are all represented by fluent approximations in DyKnow. Sets of entities belonging to concepts such as "the set of all cars that have been observed to make reckless overtakes in the last 30 minutes" can be described and maintained through the use of domains described by classes. Classes function as classification procedures which add all object identifiers which satisfy the associated add constraint to the domain and keep them as members as long as the maintain constraint is not violated. By belonging to a class certain fluent approximations related to the object identifier are guaranteed to exist and to have certain properties described by the maintain constraint.

Apart from the input provided by fluent approximations at level 1, the interactions of level 2 are mainly with level 3 where fluent approximations representing complex situations can be used to maintain object linkage structures as well as create new object identity hypotheses. For instance the example given in [2] about the detection of a missing SA-6 unit in a battery can be handled by a create constraint on the SA-6 class triggered by the detection of an incomplete SA-6 battery. Given a computed event that is detected when an incomplete battery is found, this event could be used to trigger the creation of a new SA-6 instance. In this case a monitor could also be set up to make sure the complete SA-6 battery is detected since all units have been found. This monitoring would be handled by level 3 data fusion.

D. Level 3 Impact Assessment

On this level, objects and situations should be used to assess the impact on the current actions and plans of the agent. To assess the impact, different types of monitoring are done, among others the execution monitoring of plans and behaviors and the monitoring of object hypotheses. To implement these monitors the different event detection mechanisms can be used. Currently, we use LTL formulas to model the temporal aspects of execution and hypothesis validation.

Level 3 interacts with both level 1 and level 2 since the fluent approximations produced on those levels are the ones used as input to impact assessment. The detection of violations of monitored constraints will lead to changes at the lower levels.

E. Level 4 Process Refinement

On the fourth level the system should adapt the data acquisition and processing to support mission objectives. In DyKnow this usually corresponds to changing what fluent approximations and classes are currently being computed. This is related to focus of attention problems where the most important fluent approximations should be computed while less important fluent approximations have to stand back in times of high loads. To support focus of attention, fluent approximations and class specifications can be added and deleted at run-time.

Another tool used for refinement are the policies supplied with the fluent approximations. By changing the policies of the fluent approximations the load can be reduced. For example, if the current policy for a fluent approximation of the position given by the sonar sensor is to sample it 10 times a second and the latency on the higher level approximations computed from this is more than 100ms then the sample rate could be lowered to e.g. 5 times a second until the load goes down again. It is also possible to setup filters to remove certain samples or events. For example, instead of receiving all samples, only receive a sample when the value has changed with more than 10% compared to the last change. Changes in policies can be made dynamically and can later be changed back to the original policy.

Level 4 interacts with all the other levels since it controls the context within which those are being computed as well as controlling what is actually being computed.

F. Level 5 User Refinement

On the fifth level the system should determination who queries information and who has access to information and adapt data retrieved and displayed to support cognitive decision making and actions. In DyKnow this level is very similar to the process refinement level. The main difference is that a user instead of the system itself is controlling the quality and amount of data being produced. Conceptually there is no difference in DyKnow who controls the fluent approximations. Users also have to possibility to input observations to fluent streams and in that way provide expertise about the current situation.

It is also possible to create special fluent approximations which are only used to support the cognition of the user, such as complex event descriptions or temporal logic formulas expressing conditions that the user wants to monitor. For example, instead of keeping track of a number of indicators the user can express in a LTL formula the normal conditions for all the indicators, i.e. that everything is in order. If this formula becomes false then an alarm can be triggered that forces the user to look at the individual indicators to find out the source of the problem. We believe that the complex event descriptions and temporal logics supported by DyKnow are useful tools to describe high level views of a system which are suited for a human operator.

IV. Example Scenario

Picture the following scenario. An autonomous unmanned aerial vehicle (UAV), in our case a helicopter, is given a mission to identify and track vehicles with a particular signature in a region of a small city in order to monitor the driving behavior of the vehicles. If the UAV finds vehicles with reckless behavior it should gather information about these, such as what other vehicles they are overtaking and where they are going in crossings. The signature is provided in terms of color and size (and possibly 3D shape). Assume that the UAV has a 3D model of the region in addition to information about building structures and the road system. These models can be provided or may have been generated by the UAV itself.

One way for the UAV to achieve its task would be to initiate a reactive task procedure (parent procedure) which calls an image processing module with the vehicle signature as a parameter. The image processing module will try to identify colored blobs in the region of the right size, shape and color as a first step. The fluent approximations of each new blob, such as RGB values with uncertainty bounds, length and width in pixels and position in the image, are associated with a vision object (i.e. an object identifier which is an instance of the class VisionObject). The image processing system will then try to track these blobs. As long as the blob is tracked the same vision object is updated. From the perspective of the UAV, these objects are only cognized to the extent that they are moving colored blobs of interest and the fluent approximations should continue to be computed while tracking.

Now one can hypothesize, if the establish constraint of the vision to world object link is satisfied, that the blob actually represents an object in the world by creating a representation of the blob in the world. New fluent approximations, such as position in geographical coordinates, are associated with the new world object. The geographic coordinates provide a common frame of reference where positions over time and from different objects can be compared. To represent that the two objects represents two aspects of the same entity the vision object is linked to the world object. Since the two objects are related, the fluent approximations of the world object will be computed from fluent approximations of the linked-from vision object. When the vision object is linked to a world object the entity is cognized at a more qualitative level of abstraction, yet its description in terms of its linkage structure contains both cognitive and pre-cognitive information which must be continuously managed and processed due to the interdependencies of the fluent approximations at various levels. We have now moved from level 0 to level 1 in the data fusion model.

Since links only represent hypotheses they are always subject to becoming invalid given additional observations. Therefore the UAV agent continually has to verify the validity of the links. This is done by associating maintenance constraints with the links. If the constraint is violated then the link is removed, but not the objects. A maintenance constraint could compare the behavior of the objects with the normative and predicted behavior of these types of objects. This monitoring of hypotheses at level 3 in the data fusion model uses fluent approximations computed at all the lower levels.

The next qualitative step in creating a linkage structure in this scenario would be to check if the world object is on or close to a road, as defined by a geographical information system (GIS). In this case, it would be hypothesized that the world object is an on-road object, i.e. an object moving along roads with all the associated normative behavior. An on-road object could contain more abstract and qualitative features such as position in a road segment which would allow the parent procedure to reason qualitatively about its position in the world relative to the road, other vehicles on the road, and building structures in the vicinity of the road. At this point, as shown in Fig. 6, streams of data are being generated and computed for many of the fluent approximations in the linked object structures at many levels of abstraction as the helicopter tracks the on-road objects.

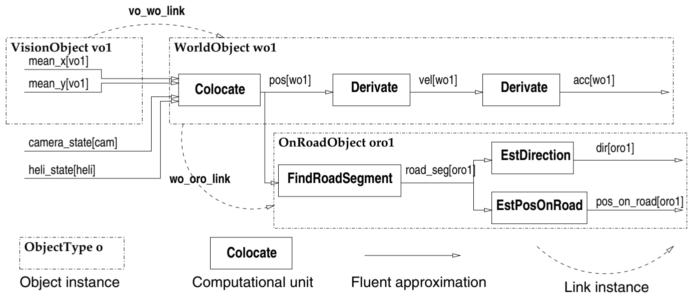

Fig. 6. The objects, link instances and fluent approximations after the world object has been hypothesized as an on road object.

Using on-road objects, we can define situations describing different traffic behaviors such as reckless driving, reckless overtakes, normal overtakes and turning left and right in crossings. All of these situations are described using chronicles, which are represented by simple temporal constraint networks where events are represented with nodes and temporal constraints are attached to edges between nodes. The chronicles are recognized online by a chronicle recognition engine.

We can now define a class RecklessBehavior which has $\Diamond(\mathsf{reckless\_overtake}(this) \vee \mathsf{reckless\_driving}(this))$ as the add constraint, which is satisfied if an on-road object is observed doing a reckless overtake or driving recklessly. A maintain constraint for this class could be, $\Box\Diamond_{[0,1800]}(\mathsf{reckless\_overtake}(this) \vee \mathsf{reckless\_driving}(this))$, which is violated if the object is not observed doing any reckless driving within 30 minutes (the time-unit is seconds in the formulas). By creating a fluent stream with all overtake, turn left, and turn right events related to an object in the RecklessBehavior domain using a set subscription, which creates a single fluent stream containing samples from certain fluent approximations for all objects in a given domain, the system is able to produce the required information and successfully carry out the mission. Fluent approximations are now maintained at levels 0, 1, and 2 in the JDL model, and continually monitored by fluent approximations at level 3.

All fluent approximations, classes, links, events and chronicles are configured by a parent task procedure at the beginning of the scenario. Thus if the situation changes the task procedure has the option of modifying the specifications associated with the task at hand. It is also possible to set up monitors checking the current delays in computing different fluent approximations in order to monitor the real-time behavior of the system. If the latency goes above a certain threshold the task procedure has the option of either removing fluent approximations it deems as less important or changing policies in such a way that the amount or quality of the data produced is reduced. These are all examples of process refinement at level 4 of the data fusion model. It is equally possible for a user to monitor the development of the situation and manually change the policies in order to influence the system in a desired direction. This would be an example of level 5 user refinement.

V. Related Work

The DyKnow framework is designed for a distributed, real-time and embedded environment [10], [11] and is developed on top of an existing middleware platform, real-time CORBA [12], using the real-time event channel [13] and the notification [14] services.

Different aspects of the framework borrow and extend ideas from a number of diverse research areas primarily related to real-time, active, temporal, and time-series databases [15], [16], [17], data stream management [18], [19], and knowledge representation and reasoning [20].

One of the many differences between DyKnow and mainstream database and data stream approaches is that we use a data model based on the use of features and fluents which integrates well between quantitative and qualitative constructions of knowledge structures.

VI. Conclusions

We have presented a knowledge processing middleware framework which provides support for many of the functionalities specified in the revised versions of the JDL data fusion model. DyKnow supports on-the-fly generation of different aspects of an agent's world model at different levels of abstraction. Contextual generation of world model is absolutely essential in distributed contexts where contingencies continually arise which often restrict the amount of time a system has for assessing situations and making timely decisions. It is our belief that autonomous systems will have to have the capability to determine where to access data, how much data should be accessed and at what levels of abstraction it should be modeled. We have provided initial evidence that such a system can be designed and deployed.

We believe that DyKnow provides the necessary concepts to integrate existing software and algorithms related to data fusion and world modelling in general. The location provides an interface to existing data and knowledge in databases, sensors and other programs. The computational units encapsulate individual algorithms and computations on data and knowledge while the fluent streams provide the means of communication. To aid the interaction with high level services DyKnow provides object, state, and event abstractions. The system has been tested in a number of complex scenarios involving our experimental UAV platform and has provided great insight into what will be required for the realization of advanced distributed data fusion services. Observe that the focus here is not on individual data fusion techniques but the infrastructure which permits use of many different data fusion techniques in a unified framework.

References

[1] F. White, "A model for data fusion," in *Proc. of 1st National Symposium for Sensor Fusion*, vol. 2, 1988.

[2] A. Steinberg and C. Bowman, "Revisions to the JDL data fusion model," in *Handbook of Multisensor Data Fusion*, D. Hall and J. Llinas, Eds. CRC Press LLC, 2001.

[3] E. Blasch and S. Plano, "Level 5: User refinement to aid the fusion process," in *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications*, B. Dasarathy, Ed., 2003.

[4] J. Llinas, C. Bowman, G. Rogova, A. Steinberg, E. Waltz, and F. White, "Revisions and extensions to the JDL data fusion model II," in *Proc. of the 7th Int. Conf. on Information Fusion*, P. Svensson and J. Schubert, Eds., 2004.

[5] P. Doherty, P. Haslum, F. Heintz, T. Merz, P. Nyblom, T. Persson, and B. Wingman, "A distributed architecture for autonomous unmanned aerial vehicle experimentation," in *Proc. of the 7th International Symposium on Distributed Autonomous Robotic Systems*, 2004.

[6] P. Doherty, "Advanced research with autonomous unmanned aerial vehicles," in *Proc. of International Conference on Principles of Knowledge Representation and Reasoning*, 2004.

[7] "Panel discussion on challenges in higher level fusion: Unsolved, difficult, and misunderstood problems/approaches in levels 2-4 fusion research," in *Proc. of the 7th Int. Conf. on Information Fusion*, P. Svensson and J. Schubert, Eds., 2004.

[8] F. Heintz and P. Doherty, "Managing dynamic object structures using hypothesis generation and validation," in *Proc. of the AAAI Workshop on Anchoring Symbols to Sensor Data*, 2004.

[9] Object Computing, Inc., "*TAO Developer's Guide, Version 1.3a*," 2003, see also http://www.cs.wustl.edu/~schmidt/TAO.html.

[10] D. Schmidt, "Adaptive and reflective middleware for distributed real-time and embedded systems," *Lecture Notes in Computer Science*, vol. 2491, 2002.

[11] ——, "Middleware for real-time and embedded systems," *Communications of the ACM*, vol. 45, no. 6, 2002.

[12] D. Schmidt and F. Kuhns, "An overview of the real-time CORBA specification," *IEEE Computer*, vol. 33, no. 6, 2000.

[13] T. Harrison, D. Levine, and D. Schmidt, "The design and performance of a real-time CORBA event service," in *Proceedings of OOPSLA'97*, 1997.

[14] R. Gruber, B. Krishnamurthy, and E. Panagos, "CORBA notification service: Design challenges and scalable solutions," in *17th International Conference on Data Engineering*, 2001.

[15] J. Eriksson, "Real-time and active databases: A survey," in *Proc. of 2nd International Workshop on Active, Real-Time, and Temporal Database Systems*, 1997.

[16] G. Özsoyoglu and R. Snodgrass, "Temporal and real-time databases: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 7, no. 4, 1995.

[17] D. Schmidt, A. K. Dittrich, W. Dreyer, and R. Marti, "Time series, a neglected issue in temporal database research?" in *Proc. of the Int. Workshop on Temporal Databases*, 1995.

[18] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: A new model and architecture for data stream management," *VLDB Journal*, 2003.

[19] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proc. of 21st ACM Symposium on Principles of Database Systems*, 2002.

[20] R. Brachman and H. Levesque, *Knowledge Representation and Reasoning*. Morgan Kaufmann, 2004.