DyKnow Federations: Distributing and Merging Information Among UAVs

Fredrik Heintz and Patrick Doherty
Dept. of Computer and Information Science
Linköping University, 581 83 Linköping, Sweden
{frehe, patdo}@ida.liu.se

***Abstract*—As unmanned aerial vehicle (UAV) applications become more complex and versatile there is an increasing need to allow multiple UAVs to cooperate to solve problems which are beyond the capability of each individual UAV. To provide more complete and accurate information about the environment we present a DyKnow federation framework for information integration in multi-node networks of UAVs. A federation is created and maintained using a multiagent delegation framework and allows UAVs to share local information as well as process information from other UAVs as if it were local using the DyKnow knowledge processing middleware framework. The work is presented in the context of a multi UAV traffic monitoring scenario.**[1]

Keywords: distributed fusion, integrated autonomous systems, multiagent systems.

In many unmanned aerial vehicle (UAV) applications it is not enough to only have one UAV executing a mission. Sometimes no single UAV has the capability or information to perform all the required tasks and in many cases it is more efficient to use multiple UAVs to complete a mission. Therefore it would be beneficial for groups of UAVs to accomplish complex missions in a cooperative manner. Since the UAVs have their own limited field of view and sphere of influence they must share and merge information among themselves to cooperatively complete missions. The information could include plans, observations and partial world models.

Conventional approaches to merging and fusing information have focused on collecting information from distributed sources and processing them at a central location. Our goal is to allow each node to be autonomous and to do as much processing as possible locally and cooperating when necessary. This will make the processing more decentralized and remove the dependence on a central node with global information. This goal can be divided into three separate subproblems:

1) How to find and share information among nodes,
2) how to merge information from multiple sources and
3) how to cooperate and divide the sharing and merging among multiple nodes.

As part of our ongoing research in UAV technologies we have developed a stream-based knowledge processing middleware framework called DyKnow which provides design and software support for developing applications integrating sensing and deliberation [1], [2]. We believe that DyKnow provides an appropriate basis for a framework for integrating information among UAVs by sharing and merging high level information. We have previously [3] shown how DyKnow can be used to implement most of the JDL Data Fusion Model, which is the de facto standard functional fusion model [4]–[6]. This shows that DyKnow has the necessary functionality to support fusing and merging information. The focus of this paper is therefore to show how DyKnow can be extended to support finding and sharing information among UAVs.

The rest of the paper is structured as follows. Section I describes a multi platform traffic monitoring scenario and some specific use cases where distributing and merging information is required. Section II describes the knowledge processing middleware framework DyKnow which will be used as the starting point for the information integration infrastructure. Section III describes the distributing infrastructure where DyKnow instances from participating platforms are collected into a DyKnow federation. The federation is created and controlled using a FIPA compliant multiagent delegation framework. Section IV concludes the paper with a summary.

I. A Multi UAV Traffic Monitoring Scenario

Assume that two or more UAVs are given the task to monitor an urban area for traffic violations. Each UAV is equipped with the appropriate sensors and reasoning mechanisms for detecting traffic violations. This means that each UAV could monitor and detect traffic violations by itself. We have previously presented how this can be done using DyKnow [7].

To increase the size of the monitored area and to monitor several different potential traffic violations at the same time, several UAVs can be used. Even a simple approach to cooperation like dividing the area between the UAVs introduces issues related to sharing and merging information. For example, to decide how to divide the area different characteristics of the UAVs could be used, such as their speed, flying altitude, sensors and available fuel. If one UAV is responsible for dividing the area it will need to collect this information from all platforms.

Another issue is the possibility of a traffic violation beginning in one area and ending in another. This means that neither of the UAVs will see the whole event. To handle this situation the UAVs need to cooperate and share information in such a way that they can detect the traffic violation together. One approach is to let the UAV that detected the beginning of the

[1]This work is partially supported by grants from the Swedish Aeronautics Research Council (NFFP4-S4203) and the Swedish Foundation for Strategic Research (SSF) Strategic Research Center MOVIII.

potential violation request the appropriate information from the UAV in the next area. This information would have to be seen as a stream since it is not a single piece of information but rather an evolving description of the development of a complex situation. Merging such a stream with local information would allow the first UAV to detect the traffic violation even if it takes place in two different regions.

This traffic monitoring scenario is an instance of a class of scenarios where multiple platforms must cooperate to complete complex missions. To succeed they need to collect, share and merge information. A solution which handles the issues introduced in this scenario will also provide a solution for many other interesting scenarios. For example, instead of having homogeneous platforms covering different parts of an area there could be heterogeneous platforms with complementing sensors each providing different types of information. Another example is to increase the accuracy in the monitoring by having several homogeneous or heterogeneous platforms covering the same area. It is also possible to replace traffic monitoring with scanning an area for injured people to do a rescue mission or to look for troops and military equipment to do a military surveillance mission. The traffic monitoring instance is chosen because we could fly it today with our current platforms and testing facilities.

II. DyKnow

DyKnow is a middleware framework for describing, implementing and interacting with stream-based knowledge processing applications [1], [2]. Processing of streams is done at many levels of abstraction starting with low level quantitative sensor data and often resulting in qualitative data structures which are grounded in the world and can be interpreted as knowledge by an agent. For the result to allow an agent to react in time to changes in the environment, the processing must be done in a timely manner.

Conceptually, knowledge processing middleware processes streams generated by different components in a distributed system. These streams may be viewed as time-series and may start as streams of observations from sensors or sequences of queries to databases. Eventually, they will contribute to more refined, composite streams. Processes combine such streams by computing, synchronizing, filtering and approximating to derive higher level abstractions. Each process is associated with quality of service properties such as maximum delay and strategies for calculating missing values, which together define the properties of the information derived by the process.

It is important to realize that knowledge is not static, but is a continually evolving collection of structures which are updated as new information becomes available from sensors and other sources. Therefore, the emphasis is on the continuous and ongoing knowledge derivation process, which can be monitored and influenced at runtime. The same streams of information may be processed differently by different parts of the architecture by tailoring knowledge processes relative to the needs and constraints associated with the tasks at hand. This allows DyKnow to support easy integration of existing sensors, databases, reasoning engines and other knowledge processing services.

A. DyKnow Applications

A *DyKnow application* consists of a set of *knowledge processes* processing *streams*. A knowledge process may take streams as input and provides one or more *stream generators* which can be subscribed to. Each subscription creates a distinct asynchronous stream with its own constraints according to a declarative *policy*. These constraints can for example specify how to approximate missing values, that certain samples are filtered out or that the stream should contain samples added with a regular sample period. Each stream is associated with one or more *labels* that can be used to refer to it.

A knowledge process with no inputs is called a *primitive knowledge process* and it can be seen as an interface to an external information source, such as a sensor or a database. Knowledge processes with inputs are called *dependent knowledge processes*. To model the processing of a dependent knowledge process a *computational unit* is introduced. A computational unit is used by a dependent knowledge process to process the data from the input streams and to create a new fluent generator. A computational unit can encapsulate any computation on one or more streams. Examples of computational units are filters, such as Kalman filters, and other sensor processing and fusion algorithms.

The DyKnow application used for the traffic monitoring scenario consists of four primitive processes, the image processing system, the helicopter and camera state estimations and a geographical information system (GIS). The output of the image processing system is a stream of blobs, fused from color and thermal images, estimating the states of potential vehicles. A set of computational units further process the stream of blobs together with information about the road network from the GIS, generating streams of car state estimations and qualitative spatial relations between cars. Figure 1 provides an overview of the incremental processing for the traffic monitoring scenario.

B. Anchoring

An important issue in many applications, including traffic monitoring, is to classify objects and to reason about the identity of an object. One type of reasoning is to connect data from sensors with symbolic representations of objects and to maintain the connection over time. This is called *anchoring* [8]. An example is to determine if an object being tracked by a UAV is actually a car, that is, to anchor a symbolic representation of a car to a stream of estimations extracted by an image processing system. Our approach is based on using temporal logic to describe the normative behavior of different types of objects and, based on the behavior of an observed object, hypothesizing its type [7], [9]. For example, in our traffic monitoring domain the object being tracked is assumed to be a physical object in the world, called a *world object*. Then, if the position of this world object is consistently on a road, it can be hypothesized to be an *on road object*, i.e. an

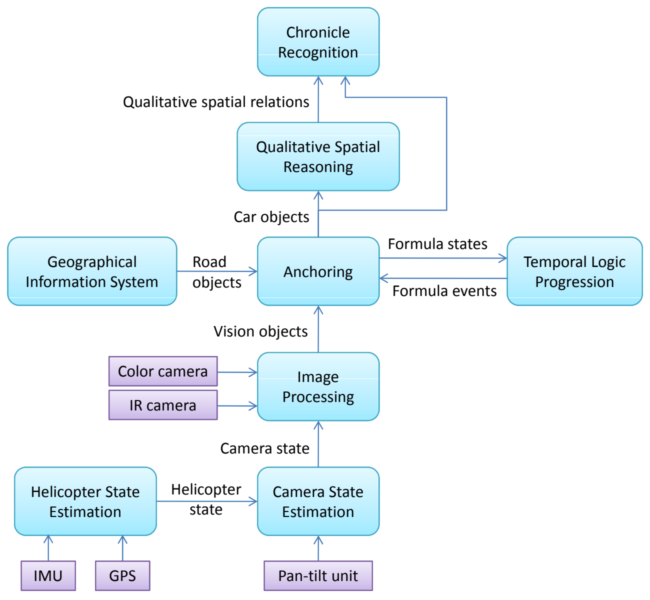

Figure 1. An overview of how the incremental processing required for the traffic surveillance task could be organized.

object moving within the road system. By further monitoring the behavior and other characteristics such as speed and size of an on road object it could be hypothesized whether it is a car, a truck, a motorcycle, or another type of vehicle.

C. Scenario Recognition

Another important functionality is to describe and recognize complex events and scenarios. To describe a complex event in DyKnow a formal representation called a *chronicle* is used [10]. A chronicle is a description of a generic scenario whose instances we would like to recognize. A chronicle corresponds to a simple temporal network [11] where the nodes are events and the edges are metric temporal constraints between event occurrences. In this context, an event is defined as a change in the value of a property or relation.

The chronicle recognition engine we use takes a stream of time-stamped event occurrences and detects all chronicle instances in the stream in a tractable and efficient manner. This is possible since chronicles correspond to simple temporal networks and there exists polynomial algorithms for checking their consistency. An instance is detected if the stream contains a set of event occurrences which satisfy all the constraints in a chronicle model. Recognized instances of a chronicle can be used as events in another chronicle, thereby enabling recursive chronicles.

The chronicles the traffic monitoring application use primitive events which capture the structure of the road network, qualitative information about cars such as which road segment they are on and qualitative spatial relations between cars such as beside and behind. These events are computed from the streams of estimated car states by computational units.

III. Distributing Information using DyKnow

From the point of view of DyKnow multiple nodes could be viewed as a single system, since it is designed for a distributed environment and does not differentiate between streams based on where they are hosted. At the same time there are a number of opportunities for improving the support provided by DyKnow.

For example, DyKnow assumes there is a global specification with unique names of computational units, stream generators and streams. In a distributed system without global control it is non-trivial but doable to support unique names, for example, by relying on a common naming schema or a common service for creating new names. When having a global specification it would also become much harder to implement the different nodes independently. When a new node is added its specification must be merged with the current global specification and when a node is removed the specification must be subtracted. Keeping the specification updated would require a lot of communication and processing.

Another issue is that internal structures and representations used by one node should not necessarily be public to all other nodes. Most of each local specification will be irrelevant to other nodes and some should even be kept secret. By only sending the relevant information the communication overhead is reduced and the robustness is increased since the system does not require stable communication all the time.

Therefore an extension of DyKnow is preferred where each node can be developed and used independently and then connected on demand. When nodes are connected parts of their local DyKnow specifications is shared among them.

A. Design Requirements

When designing a framework for distributing information among multiple nodes there are several important issues that need to be considered.

First, how to discover and broker information among a group of agents. Such a mechanism should be able both to find an agent who either has or can produce a particular piece of information and to announce to interested agents when a particular piece of information is available. The mechanism should also allow for an efficient transfer of information between nodes.

Second, how to refer to a piece of information when communicating with other agents, i.e. how to handle naming issues. One solution is to agree on a common ontology among the agents. This ontology is required to be able to refer to particular pieces of information. In the simplest case it could be assumed that all agents share a common static ontology. In a more general case only a small common ontology could be assumed which is then extended on an on-demand basis.

Third, how to negotiate with other agents to make them generate desired information. In the simplest form this mechanism would request the production of a piece of information from an agent. In the general case an agent could refuse to perform the request due to limited resources or other commitments. There might also be several agents that could produce the same information but with different quality. In this case an agent would have to reason about the different options and negotiate with the agents to find an agent who is willing to produce the information with good enough quality.

To make these requirements more explicit three different use cases are presented. Together they cover most of the functionality required for the multi platform monitoring scenario. It is important to note that the DyKnow federation framework does not solve all of the problems, instead it provides an integrated framework with basic support for solving these problems.

1) Explicit Ask and Tell to Divide the Monitoring Area: To divide an area to be monitored among a group of UAVs they need to negotiate. A simple approach would be to appoint one of them the leader. This leader has then to find out which UAVs are available and collect information about them. The information could for example be available sensors and maximum speed and flying altitude. Using this information the leader can partition the area among the UAVs and inform them about their responsibilities.

This use case gives an example where a node needs to find which other nodes are available, ask for specific pieces of information from each of the nodes, compute the result and then inform the other UAVs about the result.

2) Continuous Information Streaming and Merging to Detect Traffic Violations: When monitoring a traffic violation occurring in two areas covered by different UAVs there will be an interval where none of the UAVs have a complete picture of the situation. This means that each UAV only has a limited view and therefore has to cooperate with each other in order to cover the situation.

A concrete use case is when a UAV has detected the beginning of a potential traffic violation involving two cars and one of the cars leaves the view field of the first UAV and enters the view field of the second UAV. Now, the first UAV has to continuously get relevant updates from the second UAV about the car it can no longer see.

The information provided by the second UAV could be on many different abstraction levels. A high level in this case could be to send a stream of car states with the best current estimation of the car. Using these car states and the car states it produces itself about the other car, the first UAV can merge the information in order to monitor the potential traffic violation. It is important to notice that this is an ongoing activity where each new car state should be transmitted to the first UAV to be merged which each car state it produces locally.

An alternative approach is to merge the information on a lower level. Instead of letting the second UAV produce car states it could send over more primitive information. The lowest possible level would be to send over the raw sensor data, such as images. This will in most cases not be appropriate since there is a limited bandwidth and, in the general case, the other UAV might not have the capability to process the sensor data.

To find an appropriate abstraction level of the communication many factors must be taken into account. The most important ones are the processing capability of the involved platforms, the available bandwidth and the current load on the involved platforms. In general we believe that the higher the abstraction level the less information needs to be transmitted and the easier it is to merge it with the existing information.

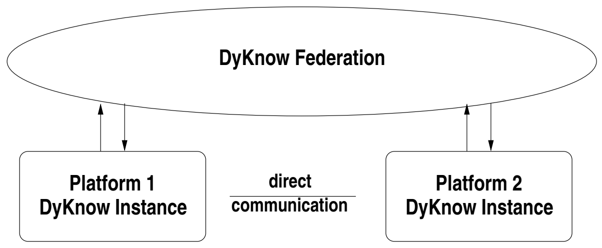

Figure 2. A high level overview of a DyKnow federation.

3) Temporary Loss of Communication during Information Streaming: To communicate among platforms some form of radio based communication will most likely be used. Unfortunately these communication channels are not always available and reliable. A common use case is that the communication is interrupted for a few seconds or even minutes due to interference or radio shadow. This introduces three challenges. First, the sender must detect the temporary loss and buffer the data that would have been sent until the connection is restored. Second, it must detect when the connection has been restored so that the buffered data can be transmitted. Third, the receiver must be able to handle that the information it is waiting for is delayed for potentially several minutes. It also has to handle the potentially large burst of delayed data when it becomes available again.

B. DyKnow Federation Overview

To fulfill the requirements we propose to connect nodes having local DyKnow applications in a DyKnow federation, like federated databases [12], [13]. The federation is used to find other DyKnow applications which can provide a particular piece of information and to ask queries about information available at other nodes. To support efficient continuous streaming of information between nodes we propose to create direct communication channels between pairs of nodes. These channels are set up through the federation framework but are then under the control of the participating nodes. A high level overview of a DyKnow federation is shown in Figure 2.

The DyKnow federation framework uses an existing delegation framework [14], where each DyKnow application, from now on called a *DyKnow instance*, becomes a service. A DyKnow federation is managed through speech act-based interactions between these services.

C. The Delegation Framework

To support cooperative goal achievement among a group of agents a delegation framework has been developed [14]. It provides a formal approach to describing and reasoning about what it means for an agent to delegate an objective, which can be either a goal or a plan, to another agent. The concept of delegation allows for studying not only cooperation but also mixed-initiative problem-solving and adjustable autonomy.

By delegating a partially specified objective the delegee is given the autonomy to complete the specification itself. By making the objective more specific the autonomy is limited. If the delegated objective is completely specified then the agent has no autonomy. By allowing agents and human operators to partially specify an objective mixed-initiative is supported.

An agent is a complex entity providing a set of *services*. A service is a particular task that can be done by an agent. Agents communicate with each other using the standardized agent communication language FIPA ACL [15], which is based on speech acts. Each agent is FIPA compliant and is implemented using the Java agent development framework JADE [16].

Each UAV platform has a delegation framework layer consisting of a set of agents communicating using FIPA ACL and a legacy layer implementing platform specific functionalities. The interface between the two layers is the *Gateway Agent*, which provides a FIPA ACL interface to the platform specific legacy system. In our UAV platform, which is implemented using CORBA, this involves invoking methods on different CORBA objects.

All communication between a platform and agents external to the platform goes through a single agent called the *Interface Agent*. The Interface Agent provides a single entry point to the platform which makes it possible to keep track of all communication, authenticate incoming messages and perform access control to the platform.

To find services in the delegation framework a *Director Facilitator* (DF) is used. It is a database of all the services provided by the different platforms.

D. DyKnow Federation Components

A DyKnow federation consists of three components: DyKnow agents, export proxies and import proxies. A DyKnow agent is a agent which makes a local DyKnow instance available as a service. The export and import proxies are used to mediate streams through direct communication between two DyKnow instances. Apart from these DyKnow federation specific components, the framework also uses the interface and gateway agents from the delegation framework. The JADE agents communicate using the FIPA ACL while two DyKnow instances can communicate directly through the export and import proxies after setting up a subscription. An overview of the components is given in Figure 3.

To make a DyKnow instance available to other platforms it must be integrated in the delegation framework. This is done in three steps:

1) by implementing a DyKnow agent which provides the dyknow service,
2) by extending the interface agent to provide the dyknow service and
3) by extending the gateway agent which mediates between FIPA ACL messages and the DyKnow implementation on the platform to handle DyKnow related messages.

One important issue is how to refer to information among platforms. A DyKnow instance will contain a set of labels referring to streams. The easiest approach would be to use

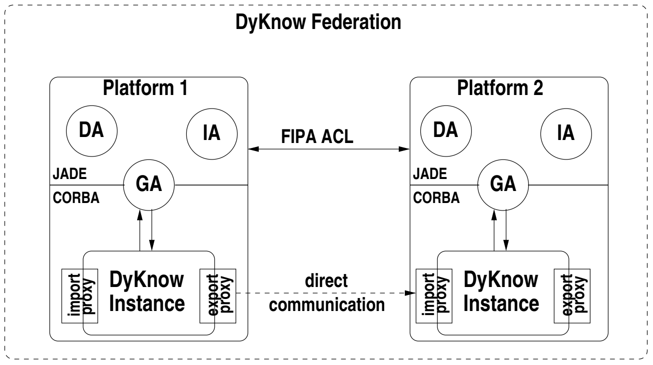

Figure 3. An overview of the components of a DyKnow federation. A DA is a DyKnow agent, an IA is an interface agent and a GA is a gateway agent.

these labels directly. One problem with this approach is that the delegation framework layer then must know what labels each of the other platforms have in their local DyKnow instances. This is not a major issue if all platforms are built by the same people, but in a more general setting this would not be feasible. A more feasible approach is to agree on a set of labels with a certain meaning among a group of agents called *semantic labels*. These semantic labels can then be translated by each agent to local DyKnow labels using whatever procedure necessary. This is a first step towards introducing a common ontology of information among a group of agents. The benefits are that each group of agent can use their own set of semantic labels, with a meaning they have agreed upon, and that the labels in the local DyKnow instances are isolated from each other.

1) DyKnow Agent: The interface agent will forward requests to the dyknow service to the DyKnow agent, which is then responsible for fulfilling the request by using the gateway agent to access the local DyKnow instance. The following requests can be made to a DyKnow agent:

1) Request a stream satisfying a policy, which makes the DyKnow instance produce a stream according to the policy, if possible, and export it;
2) request a stream corresponding to a semantic label, which translates the semantic label to a label, looks it up in the local DyKnow instance and exports the matching stream if it exists;
3) request the latest value, the value at a time-point or a trajectory between two time-points for a semantic label, which translates the semantic label to a label and looks it up in the local DyKnow instance and returns all answers from all matching streams; and
4) ask for all semantic labels that satisfy a formula written in the FIPA content language SL [17].

The first two requests will set up export proxies exporting the requested stream and then inform the requester about how to access the stream from the proxy. The next request will return the requested value directly. The last request will result in an inform message about available semantic labels matching the formula.

Each request can be either *local* or *global*. If the request is local only the DyKnow instance directly accessible to the DyKnow agent will be queried. If the request is global then all other platforms will be queried as well. This is done by the DyKnow agent asking the Director Facilitator about all agents providing the dyknow service and forwarding the request to each of them. It will then aggregate the result and send it to the original requester.

It is also possible to inform a DyKnow agent about some information. A FIPA ACL *inform* message contains the semantic label of the information and the value. The DyKnow agent returns whether the information was accepted by the local DyKnow instance or not.

2) Export Proxy: An export proxy is a component used by a platform to export one or more streams. To export a stream an internal subscription is made by the proxy which then makes the stream available to other platforms in an implementation specific way.

3) Import Proxy: An import proxy is a component used by a platform to import one or more streams. Each imported stream will be provide as a stream generator in the local DyKnow instance, like any other stream generator. How the stream is imported is an implementation detail which must be coordinated with the export proxy. Different pairs of proxies can use different methods to communicate.

E. DyKnow Federation Functionalities

1) Adding and Removing Nodes: To add a node the interface agent of that node has to register its dyknow service in the Director Facilitator. When this is done the node is available. To leave a DyKnow federation it is enough to unregister the dyknow service. This does not necessarily close all ongoing streams to or from the node since the proxies talk directly with each other when the streaming has been set up.

2) Asking for Explicit Information: If a platform needs a particular piece of information, knows its semantic label and knows which platform can provide the information then a request is sent to the interface agent of that platform with the semantic label as the argument. The information will then be sent back via a FIPA *propose* message from its DyKnow agent handling the request. Using this method a platform could ask for the latest value of a stream, the value at a particular time-point or the trajectory of values between two time-points.

If a platform only knows the semantic label of the information, but not which platform is hosting the information, then it has to make a global request instead of a local. This will cause the DyKnow agent to query all platforms providing a dyknow service for the semantic label. This could give any number of answers. If the platform gets more than one answer then it has to either select one of the values or merge them together.

To implement the first use case, explicitly asking for information about platforms, this functionality would be used. The leader UAV would make a global request for the current value of the semantic labels max_speed, fuel and so on.

3) Explicit Sharing of Information: If a platform knows who would need a particular piece of information then it could explicitly communicate it to that platform. This is done by looking up the dyknow service in the DF and sending an inform message to the interface agent found. The interface agent will forward the message to the DyKnow agent which may or may not accept the information.

To continue the first use case, this functionality would be used by the leader to either make a local inform to each platform about its region or a global inform about all assignments of regions.

4) Streaming Information: Setting up a stream from one platform to another is different from requesting a particular piece of information directly. Instead of sending something back, the agent receiving the request will set up an export proxy which will start streaming the information to the import proxy of the sending agents. How this is done is implementation dependent.

When proxies are set up the platform that made the request can access the stream as if it was a local stream generator through the import proxy. The import proxy could be viewed as a sensor which providing information.

This would be the main functionality required to implement the second use case, to provide continuous information about tracked vehicles from one UAV to another. The UAV receiving the stream would then have to fuse this stream with its own stream of car estimations in order to do the qualitative spatial reasoning and chronicle recognition.

5) Finding Information: To find a DyKnow instance which has the capability to create a stream according to a specific policy the dyknow service is used. If more than one platform can provide the dyknow service then the platform has to choose which one it would like to use. It could also choose to use more than one of them in order to provide redundancy. By merging the result from these different sources it could also increase the accuracy.

If a platform does not know the semantic label of the information it can make a request for all semantic labels which match an SL formula. SL is a first order content language developed by FIPA and used by JADE. To be able to write SL formulas an ontology must be created for the DyKnow federation. This is done within the JADE framework by providing a concept for each semantic label. For example, if the ontology contains the concepts `Car`, `Color` and `OnRoad`, formulas using these can be written. To find all semantic labels of blue cars on road 7 the formula `(all ?x (and (Car ?x) (Color ?x blue) (OnRoad ?x road7)))` could be used. It is then up to each DyKnow agent to interpret the formula and find all matching semantic labels. If a platform would like to know if any other platform has found the same car it is tracking then it could use this functionality to find potential matches.

6) Managing Unreliable Communication: One important issue is the need to handle communication which is not stable. By not being stable we mean that two nodes might lose communication between each other. The proxies have the responsibility for managing the communication. How this is implemented will depend on the method used to communicate between two proxies.

a) Missing Information: If the communication channel is unreliable some standard form of resend protocol will have to be set up between the proxies.

b) Reordering Information: If the communication between two proxies may reorder the information and the subscription policy on the import side requires ordered information then this must be handled by the import proxy. It can be done by buffering and reordering the data at the expense of a higher latency.

c) Temporary Loss of Communication: The export proxy will have to keep track of the status of the communication channel and if there is a temporary loss then it must buffer the data. When the channel is restored the data should be transmitted. An option is to inform an export proxy that data is only useful if it is not older than t seconds. This means the export proxy can shed old data.

d) Permanent Loss of Communication: If a permanent loss of communication is detected then the exporting platform will inform the importing platform about this using the JADE framework if possible. A permanent loss could also be caused by a platform leaving the DyKnow federation. In this case there should be a time-out and if no communication with the other platform has been possible within this time-period then it is removed from the federation and all streams closed.

IV. Summary

A DyKnow federation framework for information integration in a distributed multi-node network of UAVs has been presented. This type of framework is required to develop complex multiagent systems where agents have to cooperate to solve problems which are beyond the capability of any individual agent. The framework allows agents to share and merge information to provide more complete and accurate information about the environment.

The starting point is the stream-based knowledge processing middleware framework DyKnow. DyKnow contextually processes streams on many different levels of abstraction, from low level sensor streams to qualitative streams that can be interpreted as knowledge by an agent, integrating sensing and reasoning. This flexible stream-based processing is well suited for applications which require situation awareness since it allows for timely and continuous processing of information.

To support sharing of information in a multiagent system an extension of DyKnow is made by integrating it with a FIPA compliant delegation framework. The extension allows an agent to share parts of its local DyKnow instance with other agents in a DyKnow federation. The basic interaction and sharing is made on an agent level using the standardized FIPA ACL agent communication language. To increase the efficiency, direct communication is supported for continuous streaming of information between nodes. In either case the federation is used to find information and to set up the distribution.

Distributing and merging of information among multiple agents has been widely studied in many respects. This work does not extend any single of these approaches, but rather provides a complete integrated system for doing knowledge processing both on the agent level and the multiagent level. The contribution is how the versatile and useful knowledge processing middleware framework DyKnow can be extended to cover an even larger set of issues and allow it to integrate not only sensing and reasoning on a single platform, but also sharing and merging of information among multiple platforms. This paper provides the motivation, requirements and initial design of such an integrated framework which is being implemented and tested on our existing UAV platforms.

The next step is to allow the platforms to reason about the information that could be produced and investigate how to collaborate in order to accomplish complex missions.

In summary we believe that the DyKnow federation framework provides appropriate support for dynamically sharing and merging information in a distributed network of platforms. Since the federation approach is very general and it builds on a formal delegation framework it should be applicable to a wide range of very complex multi platform scenarios.

References

[1] F. Heintz and P. Doherty, "DyKnow: An approach to middleware for knowledge processing," *Journal of Intelligent and Fuzzy Systems*, vol. 15, no. 1, pp. 3–13, nov 2004.

[2] ——, "A knowledge processing middleware framework and its relation to the JDL data fusion model," *Journal of Intelligent and Fuzzy Systems*, vol. 17, no. 4, pp. 335–351, 2006.

[3] ——, "A knowledge processing middleware framework and its relation to the jdl data fusion model," in *Proceedings of the Eighth International Conference on Information Fusion (Fusion'05)*, E. Blasch, Ed. ISIF, IEEE, AES, july 2005.

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

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

[6] 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*, 2004.

[7] F. Heintz, P. Rudol, and P. Doherty, "From images to traffic behavior - a uav tracking and monitoring application," in *Proceedings of the Tenth International Conference on Information Fusion*, 2007.

[8] S. Coradeschi and A. Saffiotti, "An introduction to the anchoring problem," *Robotics and Autonomous Systems*, vol. 43, no. 2-3, 2003.

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

[10] M. Ghallab, "On chronicles: Representation, on-line recognition and learning," in *Proc. KR*, 1996.

[11] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, pp. 61–95, 1991.

[12] D. Heimbigner and D. Mcleod, "A federated architecture for information management," *ACM Trans. Inf. Syst.*, vol. 3, no. 3, pp. 253–278, 1985.

[13] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Comput. Surv.*, vol. 22, no. 3, pp. 183–236, 1990.

[14] P. Doherty and J.-J. C. Meyer, "Towards a delegation framework for aerial robotic mission scenarios," in *CIA*, 2007, pp. 5–26.

[15] FIPA, "Foundation for intelligent physical agents (FIPA) ACL message structure specification," http://www.fipa.org/.

[16] D. G. by Fabio Luigi Bellifemine, Giovanni Caire, *Developing Multi-Agent Systems with JADE*, 1st ed. Wiley, Mars 2007.

[17] FIPA, "Foundation for intelligent physical agents SL content language specification," http://www.fipa.org/.