1

DyKnow: A Framework for Processing Dynamic Knowledge and Object Structures in Autonomous Systems

Fredrik Heintz and Patrick Doherty *

Department of Computer and Information Science, Linköping University

Summary. 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. These structures must be managed and made accessible to deliberative and reactive functionalities which are dependent on being situationally aware of the changes in both the robotic agent's embedding and internal environment. DyKnow is a software framework which provides a set of functionalities for contextually accessing, storing, creating and processing such structures. The system is implemented and has been deployed in 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 in execution monitoring and chronicle recognition scenarios for UAV applications.

1.1 Introduction

Research in cognitive robotics is concerned with endowing robots and software agents with higher level cognitive functions that enable them to reason, act and perceive in a goal-directed manner in changing, incompletely known, and unpredictable environments. Research in robotics has traditionally emphasized low-level sensing, sensor processing, control and manipulative tasks. One of the open challenges in cognitive robotics is to integrate techniques from both disciplines and develop architectures which support the seamless integration of low-level sensing and sensor processing with the generation and maintenance of higher level knowledge structures grounded in the sensor data.

Knowledge about the internal and external environments of a robotic agent is often both static and dynamic. A great amount of background or deep knowledge is required by the agent in understanding its world and in understanding the dynamics in the embedding environment where objects of

* Both authors are supported by grants from the Wallenberg Foundation, Sweden and NFFP 539 COMPAS.

interest are cognized, hypothesized as being of a particular type or types and whose dynamics must be continuously reasoned about in a timely manner. This implies signal-to-symbol transformations at many levels of abstraction with different and varying constraints on real-time processing.

Much of the reasoning involved with dynamic objects and the dynamic knowledge related to such objects involves issues of situation awareness. How can a robotics architecture support the task of getting the right information in the right form to the right functionalities in the architecture at the right time in order to support decision making and goal-directed behavior? Another important aspect of the problem is the fact that this is an on-going process. Data and knowledge about dynamic objects has to be provided continuously and on-the-fly at the rate and in the form most efficient for the receiving cognitive or reactive robotics functionality in a particular context.

Context is important because the most optimal rates and forms in which a robotic functionality receives data are often task and environmentally dependent. Consequently, autonomous agents must be able to declaratively specify and re-configure the character of the data received. How to define a change, how to approximate values at time-points where no value is given and how to synchronize collections of values are examples of properties that can be set in the context. By robotic functionalities, we mean control, reactive and deliberative functionalities ranging from sensor manipulation and navigation to high-level functionalities such as chronicle recognition, trajectory planning, and execution monitoring.

The paper is structured as follows. We start with section 1.2 where a larger scenario using the proposed framework is described. In section 1.3, the UAV platform used in the project is briefly described. In section 1.4, DARA, a Distributed Autonomous Robotics Architecture for UAVs is briefly described. DyKnow is an essential module in this architecture. In sections 1.5 and 1.6, the basic structure of the DyKnow framework and the dynamic knowledge and object structures is described. In sections 1.7.2 and 1.7.3, two deliberative functionalities which use the DyKnow framework are considered, chronicle recognition and execution monitoring, in addition to the dynamic object repository (DOR) described in section 1.7.1. We conclude in section 1.8 with a discussion of the role of the DyKnow framework and some related work.

1.2 An Identification and Track Scenario

In order to make these ideas more precise, we will begin with a scenario from an unmanned aerial vehicle project the authors are involved in which requires many of the capabilities discussed so far.

Picture the following scenario. An autonomous unmanned aerial vehicle (UAV), in our case, a helicopter, is given a mission to identify and track a vehicle with a particular signature in a region of a small city. 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. Additionally, assume the UAV is equipped with a GPS and INS[2] for navigating purposes and that its main sensor is a camera on a pan/tilt mount.

Let's consider the processing from the bottom up, even though in reality, there will be many feedback loops in the UAV architecture. One way for the UAV to achieve its task would be to initiate a reactive task procedure (parent procedure) which calls the systems image processing module with the vehicle signature as a parameter. The image processing module might then try to identify colored blobs in the region of the right size, shape and color as a first step. These object descriptions would have to be sent to a module in the architecture called the dynamic object repository (DOR) which is responsible for the dynamic management of such objects. Each of these *vision objects* would contain features related to the image processing task such as RGB values with uncertainty bounds, length and width in pixels, position in the image, a sub-image of the object which can be used as a template for tracking, an estimate of velocity, etc.

From the perspective of the UAV, these objects are only cognized to the extent that they are moving colored blobs of interest and the feature data being collected should continue to be collected while tracking those objects perceived to be of interest. What objects are of interest? The parent procedure might identify that or those objects which are of interest based on a similarity measure according to size, color and movement. In order to do this, the DOR would be instructed to create one or more *world objects* and link them to their respective vision objects. At this point the object 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 features at various levels.

A world object could contain additional features such as position in a geographic coordinate system rather than the low-level image coordinate. Generating a geographic coordinate from an image coordinate continuously, called *co-location* is a complex process that involves combining dynamic data about features from several different objects such as the camera object, helicopter object and world objects, together with data from an onboard geographical information system (GIS) module which is also part of the architecture. One would require a computational unit of sorts that takes streamed data as input and outputs a new stream at a higher level of abstraction representing the current geographical coordinate of the object. This co-location process must occur in real-time and continually occur as the world object is tracked. This

[2] GPS and INS are acronyms for global positioning system and inertial navigation system, respectively.

implies that all features for all dynamic objects linked to the world object in focus have to be continually updated and managed.

At this point, the parent task may want to make a comparison between the geographical coordinate and the position of that coordinate in terms of the road system for the region, information of which is stored in the onboard GIS. This indexing mechanism is important since it allows the UAV to reason qualitatively about its spatial surroundings. Let's assume this is done and after some period of tracking and monitoring the stream of coordinates, the parent procedure decides that this looks like a vehicle that is following the road. *On-road* objects might then be created for each of the world objects that pass the test and linked to their respective world objects. 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 other building structures in the vicinity of the road. At this point, streams of data are being generated and computed for many of the features in the linked object structures at many levels of abstraction as the helicopter tracks the on-road objects.

The parent procedure could now use static knowledge stored in onboard knowledge bases and the GIS together with this dynamic knowledge to hypothesize as to the type of vehicle. The hypothesis would of course be based on the linkage structure for an on-road object and various features at different levels of abstraction. Assume the parent procedure hypothesizes that the on-road object is a car. A *car object* could then be created and linked to the existing linkage structure with additional high-level feature information about the car.

Whether or not the sum of streamed data which makes up the linkage structure represents a particular type of conceptual entity will only ever remain a hypothesis which could very well change, based on changes in the character of the streams of data. Monitors, users of these structures, would have to be set up to observe such changes and alert the parent procedure if the changes become too abnormal relative to some criteria determined by the parent procedure. Abnormality is a concept that is well-suited for being reasoned about at a logical level and the streamed data would have to be put into a form amenable to this type of processing.

How then can an architecture be set up to support the processes described in the UAV scenario above? This is the main topic of this paper and in it we propose a software system called the *DyKnow Framework.*[3]

1.3 The WITAS UAV Platform

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

The WITAS[4] Unmanned Aerial Vehicle Project [4, 5] is a long-term basic research project whose main objectives are the development of an integrated hardware/software VTOL (Vertical Take-Off and Landing) platform for fully-autonomous missions and its future deployment in applications such as traffic monitoring and surveillance, emergency services assistance, photogrammetry and surveying.

The WITAS Project UAV platform we use is a slightly modified Yamaha RMAX (Fig. 1.1). It has a total length of 3.6 m (including main rotor), a maximum take-off weight of 95 kg, and is powered by a 21 hp two-stroke engine. Yamaha equipped the radio controlled RMAX with an attitude sensor (YAS) and an attitude control system (YACS).

Fig. 1.1. The WITAS RMAX Helicopter

The hardware platform consists of three PC104 embedded computers (Fig. 1.2). The primary flight control (PFC) system consists of a PIII (700Mhz) processor, a wireless Ethernet bridge and the following sensors: a RTK GPS (serial), and a barometric altitude sensor (analog). It is connected to the YAS and YACS (serial), the image processing computer (serial) and the deliberative computer (Ethernet). The image processing (IP) system consists of a second PC104 embedded computer (PIII 700MHz), a color CCD camera (S-VIDEO, serial interface for control) mounted on a pan/tilt unit (serial), a video transmitter (composite video) and a recorder (miniDV). The deliberative/reactive (D/R) system runs on a third PC104 embedded computer (PIII 700MHz) which is connected to the PFC system with Ethernet using CORBA event channels. The D/R system is described in more detail in the next section.

For further discussion, it is important to note that computational processes are executed concurrently on distributed hardware. Data flow is both synchronous and asynchronous and the concurrent distributed nature of the hardware platform contributes to diverse latencies in data flow throughout the system.

[4] WITAS (pronounced *vee-tas*) is an acronym for the Wallenberg Information Technology and Autonomous Systems Laboratory at Linköping University, Sweden.

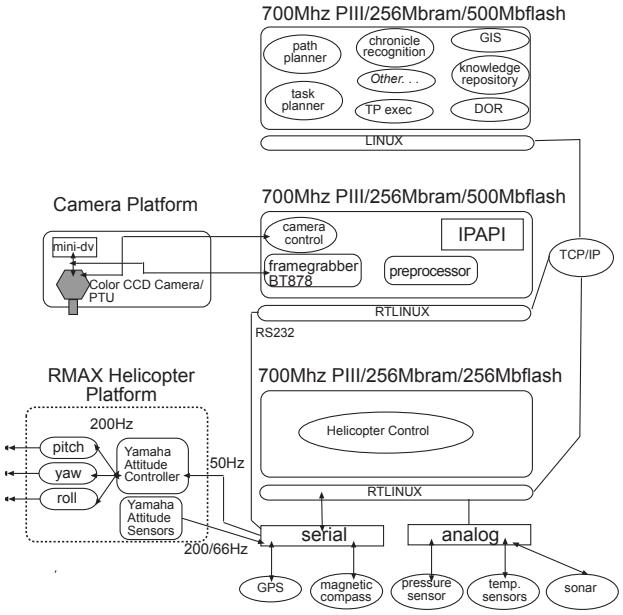

Fig. 1.2. DARA Hardware Schematic

1.4 DARA: A Distributed Autonomous Robotics Architecture

The DARA system [6] consists of both deliberative and reactive components which interface to the control architecture of the primary flight controller (PFC). Current flight modes include autonomous take-off and landing, predefined and dynamic trajectory following, vehicle tracking and hovering. We have chosen real-time CORBA [16][5] as a basis for the design and implementation of a loosely coupled distributed software architecture for our aerial robotic system.

The communication infrastructure for the architectures is provided by CORBA facilities and services. Fig. 1.3 depicts an (incomplete) high-level schematic of some of the software components used in the architecture. Each of these may be viewed as a CORBA server/client providing or requesting services from each other and receiving data and events through both real-time and standard event channels.

The modular task architecture (MTA) which is part of DARA is a reactive system design in the procedure-based paradigm developed for loosely coupled heterogeneous systems such as the WITAS aerial robotic system. Reactive be-

[5] We are currently using TAO/ACE. **T**he **A**ce **O**rb is an open source implementation of CORBA 2.6.

haviors are implemented as *task procedures* (TP) which are executed concurrently and essentially event-driven. A TP may open its own (CORBA) event channels, and call its own services (both CORBA and application-oriented services such as path planners) including functionalities in DyKnow.

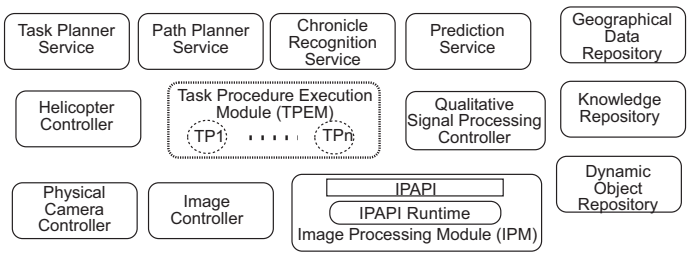

Fig. 1.3. DARA Software Schematic

1.5 DyKnow

Given the distributed nature of both the hardware and software architectures in addition to their complexity, one of the main issues is getting data to the right place at the right time in the right form and to be able to transform the data to the proper levels of abstraction for use by high-level deliberative functionalities and middle level reactive functionalities. DyKnow is designed to contribute to achieving this.

Ontologically, we view the external and internal environment of the agent as consisting of entities representing physical and non-physical objects, properties associated with these entities, and relations between entities. We will call such entities *objects* and those properties or relations associated with objects will be called *features*. Features may be static or dynamic and parameterized with objects. Due to the potentially dynamic nature of a feature, that is, its ability to change value through time, a *fluent* is associated with each feature. A fluent is a function of time whose range is the feature's type. For a dynamic feature, the fluent values will vary through time, whereas for a static feature the fluent will remain constant through time.

Some examples of features would be the *estimated velocity* of a world object, the *current road segment* of an on-road object, and the *distance* between two car objects. Each fluent associated with these examples implicitly generates a continuous stream of time tagged values of the appropriate type.

Additionally, we introduce *locations*, *policies*, *computational units* and *fluent streams* which refer to aspects of fluent representations in the actual software architecture. A *location* is intended to denote any pre-defined physical

or software location that generates feature data in the DARA architecture. Some examples would be onboard or offboard databases, CORBA event channels, physical sensors or their device interfaces, etc. In fact, a location will be used as an index to reference a representational structure associated with a feature. This structure denotes the process which implements the fluent associated with the feature. A fluent implicitly represents a stream of data, a *fluent stream*. The stream is continuous, but can only ever be approximated in an architecture. A *policy* is intended to represent a particular contextual window or filter used to access a fluent. Particular functionalities in the architecture may need to sample the stream at a particular rate or interpolate values in the stream in a certain manner. Policies will denote such collections of constraints. *Computational units* are intended to denote processes which take fluent streams as input, perform operations on these streams and generate new fluent streams as output. Each of these entities are represented either syntactically or in the form of a data structure within the architecture and many of these data structures are grounded through sensor data perceived through the robotic agent's sensors. In addition, since declarative specifications of both features and policies that determine views of fluent streams are 1st-class citizens in DyKnow, a language for referring to features, locations, computational units and policies is provided, see [13] for details.

One can view DyKnow as implementing a distributed qualitative signal processing tool where the system is given the functionality to generate dynamic representations of parts of its internal and external environment in a contextual manner through the use of policy descriptors and feature representation structures. The dynamic representations can be viewed as collections of time series data at various levels of abstraction, each time series representing a particular feature and each bundle representing a particular history or progression. Another view of such dynamic representations and one which is actually put to good use is to interpret the fluent stream bundles as partial temporal models in the logical sense. These partial temporal models can then be used on the fly to interpret temporal logical formulas in TAL (temporal action logic) or other temporal formalisms. Such a functionality can be put to good use in constructing execution monitors, predictive modules, diagnostic modules, etc. The net result is a very powerful mechanism for dealing with a plethora of issues associated with focus of attention and situational awareness.

1.6 Dynamic Object Structure in DyKnow

An ontologically difficult issue involves the meaning of an object. In a distributed architecture such as DARA, information about a specific object is often distributed throughout the system, some of this information may be redundant and it may often even be inconsistent due to issues of precision and approximation. For example, given a car object, it can be part of a linkage

structure which may contain other *objects* such as on-road, world and vision objects. For an example of a linkage structure see Fig. 1.4. In addition, many of the features associated with these objects are computed in different manners in different parts of the architecture with different latencies. One candidate definition for an object could be the aggregate of all features which take the object as a parameter for each feature. But an object only represents some aspects of an entity in the world. To represent that several different objects actually represent the same entity in the world, links are created between those objects. It is these linkage structures that represent all the aspects of an entity which are known to the UAV agent. It can be the case that two linkage structures in fact represent the same entity in the world but the UAV agent is unable to determine this. Two objects may even be of the same type but have different linkage structures associated with them. For example, given two car objects, one may not have an on-road object, but an off-road object, as part of its linkage structure. It is important to point out that objects as intended here have some similarities with OOP objects, but many differences.

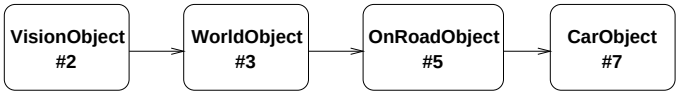

Fig. 1.4. An example object linkage structure

To create and maintain these object linkage structures we use hypothesis generation and validation. Each object is associated with a set of possible hypotheses. Each possible hypothesis is a relation between two objects associated with constraints between the objects. To generate a hypothesis, the constraints of a possible hypothesis must be satisfied. Two different types of hypotheses can be made depending on the types of the objects. If the objects have different types then a hypothesis between them is represented by a link. If they have the same type then a hypothesis is represented by a codesignation between the objects. Codesignations hypothesize that two objects representing the same aspect of the world are actually identical, while a link hypothesizes that two objects represent different aspects of the same entity.

A link can be hypothesized when a reestablish constraint between two existing objects is satisfied or an establish constraint between an object and a newly created object is satisfied. In the anchoring literature these two processes are called reacquire and find [3].

Since the UAV agent can never be sure its hypotheses are true, it has to continually verify and validate them against its current knowledge of the world. To do this, each hypothesis is associated with maintenance constraints which should be satisfied as long as the hypothesis holds. If the constraints are violated then the hypothesis is removed. The maintenance and hypothesis generation constraints are represented using the linear temporal logic (LTL)

with intervals [15] and are checked using the execution monitoring module which is part of the DyKnow framework. For a more detailed description see [14].

1.7 Applications using DyKnow

In the following subsections, we will show how the DyKnow framework can be used to generate fluent streams for further processing by two important deliberative functionalities in the DARA system, chronicle recognition and execution monitoring. Both are implemented in the UAV system. Before doing this, we provide a short description of the Dynamic Object Repository (DOR), an essential part of the DARA which uses the DyKnow framework to provide other functionalities in the system with information about the properties of dynamic objects most often constructed from sensor data streams.

1.7.1 The Dynamic Object Repository

The Dynamic Object Repository (DOR) is essentially a soft real-time database used to construct and manage the object linkage structures described in section 1.6. The DOR is implemented as a CORBA server and the image processing module interfaces to the DOR and supplies vision objects. Task procedures in the MTA access feature information about these objects via the DyKnow framework, creating descriptors on-the-fly and constructing linkages. Computational units are used to provide values for more abstract feature properties associated with these objects. For example, the co-location process involving features from the vision, helicopter and camera objects, in addition to information from the GIS, use computational units to output geographical coordinates. These are then used to update the positional features in world objects linked to the specific vision objects in question.

Objects are referenced via unique symbols which are created by the symbol generation module which is part of the DOR. Each symbol is typed using pre-defined domains such as car, world-object, vision-object, vehicle, etc. Symbols can be members of more than one domain and are used to instantiate feature representations and as indexes for collecting information about features which take these symbols as arguments. Since domains collect symbols which *reference* a certain type of object, one can also conveniently ask for information about collections or aggregates of objects. For example, "take all vision objects and process a particular feature for each in a certain manner".

1.7.2 An Application to Chronicle Recognition

Chronicles are used to represent complex occurrences of activity described in terms of temporally constrained event structures. In this context, an event is

defined as a change in the value of a feature. For example, in a traffic monitoring application, a UAV might fly to an intersection and try and identify how many vehicles turn left, right or drive straight through a specific intersection. In another scenario, the UAV may be interested in identifying vehicle overtaking. Each of these complex activities can be defined in terms of one or more chronicles. In the WITAS UAV, we use the CRS chronicle recognition system developed by France Telecom. CRS is an extension of IxTeT [8]. Our chronicle recognition module is wrapped as a CORBA server.

As an example, suppose we would like to recognize vehicles passing through an intersection. Assume cars are being identified and tracked through the UAV's camera as it hovers over a particular intersection. Recall that the DOR generates and maintains linkage structures for vehicles as they are identified and tracked. It can be assumed that the following structured features exist:

pos = position(DOR, policy1, car1)
roadseg = road_segment(DOR, roadSegment(pos), policy2, car1)
incross = in_crossing(DOR, inCrossing(roadseg), policy3, car1)

pos is a feature of a car object and its fluent stream can be accessed via the DOR as part of its linkage structure. roadseg is a complex feature whose value is calculated via a computational unit roadSegment which takes the geographical position of a world object associated with the car object as argument and uses this as an index into the GIS to return the road segment that the vehicle is in. Similarly, incross is a complex feature whose value is produced using a computational unit that takes the roadseg fluent stream as input and returns a boolean output stream, representing whether the car is in a crossing or not, calculated via a lookup in the GIS.

For the sake of brevity, a car is defined to pass through an intersection if its road segment type is not a crossing then it eventually is in a road segment that is a crossing and then it is again in a road segment that is not a crossing. In this case, if the fluent stream generated by incross generates samples going from false to true and then eventually true to false within a certain time frame then the car is recognized as passing through a crossing. The chronicle recognition system would receive such streams and recognize two change events which match its chronicle definition and thereby recognize that the car has passed through the crossing.

The stream itself requires some modification and policy3 specifies this via a *monotonic time* constraint and a *change* constraint. The monotonic time constraint would make sure the stream is ordered, i.e. the time stamp of events increase monotonically. The change constraint specifies how change is defined for this stream. There are several alternatives which can be used:

- *any change policy* – any difference between the previous and current value is a change;
- *absolute change policy* – an absolute difference between the previous and current value larger than a parameter *delta* is a change;
- *relative change policy* – a normalized difference between the previous and current value larger than a parameter *delta* is a change.

There are obvious variations on these policies for different types of signal behavior. For example, one might want to deal with oscillatory values due to uncertainty of data, etc. The example used above is only intended to provide an overview as to how DyKnow is used by other modules and is therefore simplified.

1.7.3 An Application to Execution Monitoring

The WITAS UAV architecture has an execution monitoring module which is based on the use of a temporal logic, LTL (linear temporal logic with intervals [15]), which provides a succinct syntax for expressing highly complex temporal constraints on activity in the UAV's internal environment and even aspects of its embedding environment. For example safety and liveness conditions can easily be expressed. Due to page limitations we can only briefly describe this functionality. Essentially, we appeal to the intuitions about viewing bundles of fluent streams as partial models for a temporal logic and evaluating formulas relative to this model. In this case though, the model is fed piecewise (state-wise) to the execution monitor via a state extraction mechanism associated with the execution monitor. A special progression algorithm [15] is used which evaluates formulas in a current state and returns a new formula which if true on the future states would imply that the formula is true for the complete time-line being generated.

The DyKnow system is ideal for generating such streams and feeds these to the execution monitor. Suppose we would like to make sure that two task procedures (all invocations) in the reactive layer of the DARA, called A and B, can never execute in parallel. For example, A and B may both want to use the camera resource. This safety condition can be expressed in LTL as the temporal formula **always**¬(∃x∃y tp_name[x]="A" ∧ tp_running[x]=true ∧ tp_name[y]="B" ∧ tp_runing[y]=true), where "**always**" in the formula is the modal operator for "at all times". To monitor this condition the execution monitor requires fluent streams for each of the possible instantiations of the parameterized features tp_name and tp_running which can be generated by the reactive layer of the DARA. These are fed to the instantiated execution monitor which applies the progression algorithm to the temporal formula above relative to the fluent streams generated via the DyKnow framework. This algorithm is run continuously. If the formula evaluates to false at some point, an alert message is sent to a monitor set up by the functionality interested in this information and modifications in the system configuration can be made.

1.8 Related Work

The DyKnow framework is designed for a distributed, real-time and embedded environment [18, 19] and is developed on top of an existing middleware platform, real-time CORBA [20], using the real-time event channel [12], the

notification [11] and the forthcoming real-time notification [9] services. One of the purposes for this work is in the creation of a knowledge processing middleware capability, i.e. a framework for interconnecting different knowledge representation and reasoning services, grounding knowledge in sensor data and providing uniform interfaces for processing and management of generated knowledge and object structures. The framework is quite general and is intended to serve as a platform for investigating a number of pressing issues associated with the processing and use of knowledge on robotic platforms with soft and hard real-time constraints. These issues include anchoring, or more generally symbol grounding, signal to symbol transformations, information fusion, contextual reasoning, and focus of attention. Examples of application services which use the middleware capabilities are execution monitoring services, anchoring services and chronicle recognition services.

We are not aware of any similar frameworks, but the framework itself uses ideas from many diverse research areas mainly related to real-time, active, temporal, and time-series database [7, 17, 21], data stream management [1, 2, 10], and work in the area of knowledge representation and reasoning.

The main differences between DyKnow and the database and data stream approaches are that we have a different data model based on the concepts of features and fluents and we have many views or representations of the same feature data in the system each with different properties depending on the context where the feature is used as described by a policy.

References

1. 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*, August 2003.
2. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of 21st ACM Symposium on Principles of Database Systems (PODS 2002)*, 2002.
3. S. Coradeschi and A. Saffiotti. An introduction to the anchoring problem. *Robotics and Autonomous Systems*, 43(2-3):85–96, 2003.
4. P. Doherty. Advanced research with autonomous unmanned aerial vehicles. In *Proceedings on the 9th International Conference on Principles of Knowledge Representation and Reasoning*, 2004.
5. P. Doherty, G. Granlund, K. Kuchcinski, E. Sandewall, K. Nordberg, E. Skarman, and J. Wiklund. The WITAS unmanned aerial vehicle project. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 747–755, 2000.
6. 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 *Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems*, 2004.
7. Joakim Eriksson. Real-time and active databases: A survey. In *Proc. of 2nd International Workshop on Active, Real-Time, and Temporal Database Systems*, 1997.

8. M. Ghallab. On chronicles: Representation, on-line recognition and learning. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR-96)*, 1996.
9. Pradeep Gore, Douglas C. Schmidt, Chris Gill, and Irfan Pyarali. The design and performance of a real-time notification service. In *Proc. of the 10th IEEE Real-time Technology and Application Symposium*, may 2004.
10. The STREAM Group. STREAM: The Stanford stream data manager. *IEEE Data Engineering Bulletin, 26(1)*, 2003.
11. R. Gruber, B. Krishnamurthy, and E. Panagos. CORBA notification service: Design challenges and scalable solutions. In *17th International Conference on Data Engineering*, pages 13–20, 2001.
12. Tim Harrison, David Levine, and Douglas C. Schmidt. The design and performance of a real-time CORBA event service. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA-97)*, volume 32, 10 of *ACM SIGPLAN Notices*, pages 184–200, New York, October 5–9 1997. ACM Press.
13. Fredrik Heintz and Patrick Doherty. DyKnow: An approach to middleware for knowledge processing. *Journal of Intelligent and Fuzzy Systems*, 2004.
14. Fredrik Heintz and Patrick Doherty. Managing dynamic object structures using hypothesis generation and validation. In *Proceedings of the AAAI Workshop on Anchoring Symbols to Sensor Data*, 2004.
15. K. Ben. Lamine and F. Kabanza. Reasoning about robot actions: A model checking approach. In *Advances in Plan-Based Control of Robotic Agents*, LNAI, pages 123–139, 2002.
16. Object Computing, Inc. *TAO Developer's Guide, Version 1.3a*, 2003. See also `http://www.cs.wustl.edu/~schmidt/TAO.html`.
17. Gultekin Özsoyoglu and Richard T. Snodgrass. Temporal and real-time databases: A survey. *IEEE Trans. Knowl. Data Eng.*, 7(4):513–532, 1995.
18. Douglas C. Schmidt. Adaptive and reflective middleware for distributed real-time and embedded systems. *Lecture Notes in Computer Science*, 2491:282–??, 2002.
19. Douglas C. Schmidt. Middleware for real-time and embedded systems. *Communications of the ACM*, 45(6):43–48, June 2002.
20. Douglas C. Schmidt and Fred Kuhns. An overview of the real-time CORBA specification. *IEEE Computer*, 33(6):56–63, June 2000.
21. Duri Schmidt, Angelika Kotz Dittrich, Werner Dreyer, and Robert W. Marti. Time series, a neglected issue in temporal database research? In *Proceedings of the International Workshop on Temporal Databases*, pages 214–232. Springer-Verlag, 1995.