# Bridging the Sense-Reasoning Gap: DyKnow - A Middleware Component for Knowledge Processing

Fredrik Heintz and Jonas Kvarnström and Patrick Doherty
Department of Computer and Information Science, Linköpings universitet,
SE-581 83 Linköping, Sweden
{frehe, jonkv, patdo}@ida.liu.se

***Abstract*— Developing autonomous agents displaying rational and goal-directed behavior in a dynamic physical environment requires the integration of both sensing and reasoning components. Due to the different characteristics of these components there is a gap between sensing and reasoning. We believe that this gap can not be bridged in a single step with a single technique. Instead, it requires a more general approach to integrating components on many different levels of abstraction and organizing them in a structured and principled manner.**

**In this paper we propose knowledge processing middleware as a systematic approach for organizing such processing. Desirable properties of such middleware are presented and motivated. We then go on to argue that a declarative stream-based system is appropriate to provide the desired functionality. Finally, DyKnow, a concrete example of stream-based knowledge processing middleware that can be used to bridge the sense-reasoning gap, is presented. Different types of knowledge processes and components of the middleware are described and motivated in the context of a UAV traffic monitoring application.**

## I. Introduction

When developing autonomous agents displaying rational and goal-directed behavior in a dynamic physical environment, we can lean back on decades of research in artificial intelligence. A great number of deliberative and reactive functionalities have already been developed, including chronicle recognition, motion planning, task planning and execution monitoring. To integrate these approaches into a coherent system it is necessary to reconcile the different formalisms used to represent information and knowledge about the world. To construct these world models and maintain a correlation between them and the environment it is necessary to extract information and knowledge from data collected by sensors. However, most research done in a symbolic context tends to assume crisp knowledge about the current state of the world while the information extracted from the environment often is noisy and incomplete quantitative data on a much lower level of abstraction. This causes a wide gap between sensing and reasoning.

Bridging this gap in a single step, using a single technique, is only possible for the simplest of autonomous systems. As complexity increases, one typically requires a combination of a wide variety of methods, including more or less standard functionalities such as various forms of image processing and information fusion as well as application-specific and possibly even scenario-specific approaches. Such integration is currently done ad hoc, partly by allowing the sensory and deliberative layers of a system to gradually extend towards each other and partly by introducing intermediate processing levels.

In this paper, we propose using the term *knowledge processing middleware* for a principled and systematic framework for bridging the gap between sensing and deliberation in a physical agent. We claim that knowledge processing middleware should provide both a conceptual framework and an implementation infrastructure for integrating a wide variety of components and managing the information that needs to flow between them. It should allow a system to incrementally process low-level sensor data and generate a coherent view of the environment at increasing levels of abstraction, eventually providing information and knowledge at a level which is natural to use in symbolic deliberative functionalities. Such a framework would also support the integration of different deliberation techniques.

The structure of the paper is as follows. In the next section, an example scenario is presented as further motivation for the need for a systematic knowledge processing middleware framework. Desirable properties of such frameworks are investigated and a specific stream-based architecture is proposed which is suitable for a wide range of systems. As a concrete example, our framework DyKnow is briefly described. The paper is concluded with some related work and a summary.

## II. A Traffic Monitoring Scenario

Traffic monitoring is an important application domain for research in autonomous unmanned aerial vehicles (UAVs) which provides a plethora of cases demonstrating the need for an intermediary layer between sensing and deliberation. It includes surveillance tasks such as detecting accidents and traffic violations, finding accessible routes for emergency vehicles, and collecting statistics about traffic patterns.

In the case of detecting traffic violations, one possible approach relies on using a formal declarative description of each type of violation. This can be done using a chronicle [7],

This work is partially supported by grants from the Swedish Aeronautics Research Council (NFFP4-S4203), the Swedish Foundation for Strategic Research (SSF) Strategic Research Center MOVIII and the Center for Industrial Information Technology CENIIT (06.09).

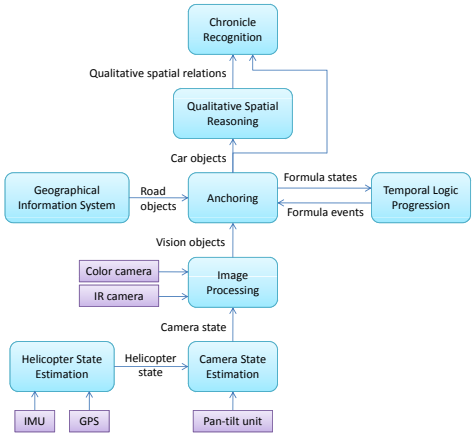


Fig. 1. An overview of how the incremental processing for the traffic surveillance task could be organized.

which defines a class of complex events using a simple temporal network where nodes correspond to occurrences of high level qualitative events and edges correspond to metric temporal constraints between event occurrences. For example, to detect a reckless overtake, qualitative spatial events such as $beside(car_1, car_2)$, $close(car_1, car_2)$ and $on(car_1, road_7)$ might be used. Creating such high-level representations from low-level sensory data, such as video streams from color and infrared cameras, involves a great deal of work at different levels of abstraction which would benefit from being separated into distinct and systematically organized tasks.

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

At the lowest level, a helicopter state estimation system uses data from inertial measurement unit and GPS sensors to determine the current position and attitude of the helicopter. The resulting information is fed into a camera state estimation system, together with the current angles of the pan-tilt unit on which the cameras are mounted, to generate information about the current camera state. The image processing system uses the camera state to determine where the camera is currently pointing. Video streams from the color and thermal cameras can then be analyzed in order to extract vision objects representing hypotheses regarding moving and stationary physical entities, including their approximate positions and velocities.

Each vision object must be associated with a symbol for use in higher level services, a process known as *anchoring* [5, 11]. In the traffic surveillance domain, identifying which vision objects correspond to vehicles is also essential, which requires knowledge about normative sizes and behaviors of vehicles. One interesting approach to describing such behaviors relies on the use of formulas in a metric temporal modal logic, which are incrementally progressed through states that include current vehicle positions, velocities, and other relevant information. An entity satisfying all requirements can be hypothesized to be a vehicle, a hypothesis which is subject to being withdrawn if the entity ceases to satisfy the normative behavior and thereby causes the formula progression system to signal a violation.

As an example, vehicles usually travel on roads. Given that image processing provided absolute world coordinates for each vision object, the anchoring process can query a geographic information system to determine the nearest road segment and derive higher level predicates such as $on\text{-}road(car)$ and $in\text{-}crossing(car)$. These would be included in the states sent to the progressor as well as in the vehicle objects sent to the next stage of processing, which involves deriving qualitative spatial relations between vehicles such as $beside(car_1, car_2)$ and $close(car_1, car_2)$. These predicates, and the concrete events corresponding to changes in the predicates, finally provide sufficient information for the chronicle recognition system to determine when higher-level events such as reckless overtakes occur.

In this example, we can identify a considerable number of distinct processes involved in bridging the gap between sensing and deliberation and generating the necessary symbolic representations from sensor data. However, in order to fully appreciate the complexity of the system, we have to widen our perspective somewhat. Looking towards the smaller end of the scale, we can see that what is represented as a single process in Figure 1 is sometimes merely an abstraction of what is in fact a set of distinct processes. Anchoring is a prime example, encapsulating tasks such as the derivation of higher level predicates which could also be viewed as a separate process. At the other end of the scale, a complete UAV system also involves numerous other sensors and information sources as well as services with distinct knowledge requirements, including task planning, path planning, execution monitoring, and reactive procedures.

Consequently, what is seen in Figure 1 is merely an abstraction of the full complexity of a small part of the system. It is clear that a systematic means for integrating all forms of knowledge processing, and handling the necessary communication between parts of the system, would be of great benefit. Knowledge processing middleware should fill this role, by providing a standard framework and infrastructure for integrating image processing, sensor fusion, and other data, information and knowledge processing functionalities into a coherent system.

## III. Knowledge Processing Middleware

Any principled and systematic framework for bridging the gap between sensing and deliberation in a physical agent qualifies as knowledge processing middleware, by the definition in the introduction. We now consider the necessary and desirable properties of any such framework.

The first requirement is that the framework should *permit the integration of information from distributed sources*, allowing this information to be processed at many different levels of abstraction and finally transformed into a suitable form to be used by a deliberative functionality. In the traffic monitoring scenario, the primary input will consist of low level sensor data such as images, a signal from

a barometric pressure sensor, a GPS (Global Positioning System) signal, laser range scans, and so on. But there might also be high level information available such as geographical information and declarative specifications of traffic patterns and normative behaviors of vehicles. The middleware must be sufficiently flexible to allow the integration of all these different sources into a coherent processing system. Since the appropriate structure will vary between applications, a general framework should be agnostic as to the types of data and information being handled and should not be limited to specific connection topologies.

To continue with the traffic monitoring scenario, there is a natural abstraction hierarchy starting with quantitative signals from sensors, through image processing and anchoring, to representations of objects with both qualitative and quantitative attributes, to high level events and situations where objects have complex spatial and temporal relations. Therefore a second requirement is the *support of quantitative and qualitative processing* as well as a mix of them.

A third requirement is that *both bottom-up data processing and top-down model-based processing should be supported*. Different abstraction levels are not independent. Each level is dependent on the levels below it to get input for bottom-up data processing. At the same time, the output from higher levels could be used to guide processing in a top-down fashion. For example, if a vehicle is detected on a particular road segment, then a vehicle model could be used to predict possible future locations, which could be used to direct or constrain the processing on lower levels. Thus, a knowledge processing framework should not impose strict bottom-up nor strict top-down processing.

A fourth requirement is support for *management of uncertainty* on different levels of abstraction. There are many types of uncertainty, not only at the quantitative sensor data level but also in the symbolic identity of objects and in temporal and spatial aspects of events and situations. Therefore it is not realistic to use a single approach to handling uncertainty throughout a middleware framework. Rather, it should allow many different approaches to be combined and integrated into a single processing system in a manner appropriate to the specific application at hand.

Physical agents acting in the world have limited resources, both in terms of processing power and in terms of sensors, and there may be times when these resources are insufficient for satisfying the requests of all currently executing tasks. In these cases a trade-off is necessary. For example, reducing update frequencies would cause less information to be generated, while increasing the maximum permitted processing delay would provide more time to complete processing. Similarly, an agent might decide to focus its attention on the most important aspects of its current situation, ignoring events or objects in the periphery, or to focus on providing information for the highest priority tasks or goals. An alternative could be to replace a resource-hungry calculation with a more efficient but less accurate one. Each trade-off will have effects on the quality of the information produced and the resources used. Another reason for changing the processing is that it is often context dependent and as the context changes the processing needs to change as well. For example, the processing required to monitor the behavior of vehicles following roads and vehicles which may drive off-road is very different. In the first case simplifying assumptions can be made as to how vehicles move, while these would be invalid if a vehicle goes off-road. To handle both cases a system would have to be able to switch between the different processing configurations. A fifth requirement on knowledge processing middleware is therefore support for *flexible configuration and reconfiguration* of the processing that is being performed.

An agent should not depend on outside help for reconfiguration. Instead, it should be able to reason about which trade-offs can be made at any point in time, which requires introspective capabilities. Specifically, the agent must be able to determine what information is currently being generated as well as the potential effects of any changes it may make in the processing structure. Therefore a sixth requirement is for the framework to provide a *declarative specification of the information being generated and the information processing functionalities that are available*, with sufficient content to make rational trade-off decisions.

To summarize, knowledge processing middleware should support declarative specifications for flexible configuration and dynamic reconfiguration of context dependent processing at many different levels of abstraction.

## IV. Stream-Based Knowledge Processing Middleware

The previous section focused on requirements that are necessary or desirable in any form of knowledge processing middleware, intentionally leaving open the question of how these requirements should be satisfied. We now go on to propose one specific type of framework, *stream-based* knowledge processing middleware, which we believe will be useful in many applications. A concrete implementation, DyKnow, will be discussed later in this paper.

Due to the need for incremental refinement of information at different levels of abstraction, we model computations and processes within the stream-based knowledge processing framework as active and sustained *knowledge processes*. The complexity of such processes may vary greatly, ranging from simple adaptation of raw sensor data to image processing algorithms and potentially reactive and deliberative processes.

In our experience, it is not uncommon for knowledge processes at a lower level to require information at a higher frequency than those at a higher level. For example, a sensor interface process may query a sensor at a high rate in order to average out noise, providing refined results at a lower effective sample rate. This requires knowledge processes to be decoupled and asynchronous to a certain degree. In stream-based knowledge processing middleware, this is achieved by allowing a knowledge process to declare a set of *stream generators*, each of which can be *subscribed* to by an arbitrary number of processes. A subscription can be viewed as a continuous query, which creates a distinct asynchronous

*stream* onto which new data is pushed as it is generated. The contents of a stream may be seen by the receiver as data, information or knowledge.

Decoupling processes through asynchronous streams minimizes the risk of losing samples or missing events, something which can be a cause of problems in query-based systems where it is the responsibility of the receiver to poll at sufficiently high frequencies. Streams can provide the necessary input for processes that require a constant and timely flow of information. For example, a chronicle recognition system needs to be apprised of all pertinent events as they occur, and an execution monitor must receive constant updates for the current system state at a given minimum rate. A push-based stream system also lends itself easily to "on-availability" processing, i.e. processing data as soon as it is available, and the minimization of processing delays, compared to a query-based system where polling introduces unnecessary delays in processing and the risk of missing potentially essential updates as well as wastes resources. Finally, decoupling also facilitates the distribution of processes within a platform or between different platforms, another important property of many complex autonomous systems.

Finding the correct stream generator requires each stream generator to have an identity which can be referred to, a *label*. Though a label could be opaque, it often makes sense to use structured labels. For example, given that there is a separate position estimator for each vehicle, it makes sense to provide an identifier $i$ for each vehicle and to denote the (single) stream generator of each position estimator by $position[i]$. Knowing the vehicle identifier is sufficient for generating the correct stream generator label.

Even if many processes connect to the same stream generator, they may have different requirements for their input. As an example, one could state whether new information should be sent "when available", which is reasonable for more event-like information or discrete transitions, or with a given frequency, which is more reasonable with continuously varying data. In the latter case, a process being asked for a subscription at a high frequency may need to alter its own subscriptions to be able to generate stream content at the desired rate. Requirements may also include the desired approximation strategy when the source knowledge process lacks input, such as interpolation or extrapolation strategies or assuming the previous value persists. Thus, every subscription request should include a *policy* describing such requirements. The stream is then assumed to satisfy this policy until it is removed or altered. For introspection purposes, policies should be declaratively specified.

While it should be noted that not all processing is based on continuous updates, neither is a stream-based framework limited to being used in this manner. For example, a path planner or task planner may require an initial state from which planning should begin, and usually cannot take updates into account. Even in this situation, decoupling and asynchronicity are important, as is the ability for lower level processing to build on a continuous stream of input before it can generate the desired snapshot. A snapshot query, then, is simply a special case of the ordinary continuous query.

### A. Knowledge Processes

For the purpose of modeling, we find it useful to identify four distinct types of knowledge processes: Primitive processes, refinement processes, configuration processes and mediation processes.

*Primitive processes* serve as an interface to the outside world, connecting to sensors, databases or other information sources that in themselves have no explicit support for stream-based knowledge processing. Such processes have no stream inputs but provide a non-empty set of stream generators. In general, they tend to be quite simple, mainly adapting data in a multitude of external representations to the stream-based framework. For example, one process may use a hardware interface to read a barometric pressure sensor and provide a stream generator for this information. However, greater complexity is also possible, with primitive processes performing tasks such as image processing.

The remaining process types will be introduced by means of an illustrating example from the traffic monitoring scenario, where car objects must be generated and anchored to sensor data which is mainly collected using cameras. Note that this example is not purely theoretical but has been fully implemented and successfully used in test flights in an experimental UAV platform [13].

In the implemented approach, the image processing system produces *vision objects* representing entities found in an image, called *blobs*, having visual and thermal properties similar to those of a car. A vision object state contains an estimation of the size of the blob and its position in absolute world coordinates. When a new vision object has been found, it is tracked for as long as possible by the image processing system and each time it is found in an image a new vision object state is pushed on a stream.

Anchoring begins with this stream of vision object states, aiming at the generation of a stream of *car object* states providing a more qualitative representation, including relations between car objects and road segments. An initial filtering process, omitted here for brevity, determines whether to hypothesize that a certain vision object in fact corresponds to a car. If so, a car object is created and a *link* is established between the two objects. To monitor that the car object actually behaves like a car, a maintain constraint describing expected behavior is defined. The constraint is monitored, and if violated, the car hypothesis is withdrawn and the link is removed. A temporal modal logic is used for encoding normative behaviors, and a progression algorithm is used for monitoring that the formula is not violated.

Figure 2 shows an initial process setup, existing when no vision objects have been linked to car objects. As will be seen, processes can dynamically generate new processes when necessary. Figure 3 illustrates the process configuration when VisionObject#51 has been linked to CarObject#72 and two new refinement processes have been created.

The first process type to be considered is the *refinement process*, which takes a set of streams as input and provides

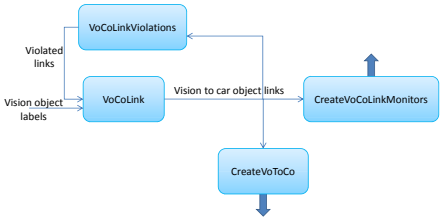


Fig. 2. The initial set of processes before any vision object has been created.

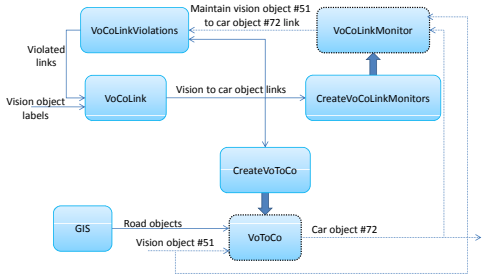


Fig. 3. The set of processes after VisionObject#51 has been linked to CarObject#72.

one or more stream generators producing refined, abstracted or otherwise processed values. Several examples can be found in the traffic monitoring application, such as:

- VoCoLink – Manages the set of links between vision objects and car objects, each link being represented as a pair of labels. When a previously unseen vision object label is received, create a new car object label and a link between them. When a link is received from the VoCoLinkViolations process, the maintain constraint of the link has been violated and the link is removed. The output is a stream of sets of links. A suitable policy may request notification only when the set of links changes.
- VoToCo – Refines a single vision object to a car object by adding qualitative information such as which road segment the object is on and whether the road segment is a crossing or a road. Because quantitative data is still present in a car object, a suitable policy may request new information to be sent with a fixed sample frequency. Using a separate process for each car object yields a fine-grained processing network where different cars may be processed at different frequencies depending on the current focus of attention.
- VoCoLinkMonitor – An instantiation of the formula progressor. Monitors the maintain constraint of a vision object to car object link, using the stream of car object states generated by the associated VoToCo. The output is false iff the maintain constraint has been violated.

The second type of process, the *configuration process*, takes a set of streams as input but produces no new streams. Instead, it enables dynamic reconfiguration by adding or removing streams and processes. The configuration processes used in our example are:

- CreateVoCoLinkMonitors – Takes a stream of sets of links and ensures VoCoLinkMonitor refinement processes are created and removed as necessary.
- CreateVoToCos – Takes a stream of vision to car object links and ensures VoToCo refinement processes are created and removed as necessary.

Finally, a *mediation process* generates streams by selecting or collecting information from other streams. Here, one or more of the inputs can be a stream of labels identifying other streams to which the mediation process may subscribe. This allows a different type of dynamic reconfiguration in the case where not all potential inputs to a process are known in advance or where one does not want to simultaneously subscribe to all potential inputs due to processing cost. One mediation process is used in our example:

- VoCoLinkViolations – Takes a stream of sets of links identifying all current connections between vision objects and car objects. Dynamically subscribes to and unsubscribes from monitor information from the associated VoCoLinkMonitors as necessary. If a monitor signals a violation (sending the value "false"), the corresponding link becomes part of the output, a stream of sets of violated links.

In Figure 2 the VoCoLinkViolations mediation process subscribes to no streams, since there are no VoCoLinkMonitor streams. In Figure 3 it subscribes to the stream of monitor results of the maintain constraint of the new VisionObject#51 to CarObject#72 link.

This example shows how stream-based knowledge processing middleware can be applied in a very fine-grained manner, even at the level of individual objects being tracked in an image processing context. At a higher level, the entire anchoring process can be viewed as a composite knowledge process with a small number of inputs and outputs, as originally visualized in Figure 1. Thus, one can switch between different abstraction levels while remaining within the same unifying framework.

### B. Timing

Any realistic knowledge processing architecture must take into account the fact that both processing and communication takes time, and that delays may vary, especially in a distributed setting. As an example, suppose one knowledge process is responsible for determining whether two cars are too close to each other. This test could be performed by subscribing to two car position streams and measuring the distance between the cars every time a new position sample arrives. Should one input stream be delayed by one sample period, distance calculations would be off by the distance traveled during that period, possibly triggering a false alarm. Thus, the fact that two pieces of information arrive simultaneously must not be taken to mean that they refer to the same time.

For this reason, stream-based knowledge processing middleware should support tagging each piece of information in a stream with its *valid time*, the time at which the information was valid in the physical environment. For example, an image taken at time $t$ has the valid time $t$. If an image processing system extracts vision objects from this image, each created

vision object should have the same valid time even though some time will have passed during processing. One can then ensure that only samples with the same valid time are compared. Valid time is also used in temporal databases [15].

Note that nothing prevents the creation of multiple samples with the same valid time. For example, a knowledge process could very quickly provide a first rough estimate of some property, after which it would run a more complex algorithm and eventually provide a better estimate with identical valid time.

The *available time*, the time when a piece of information became available through a stream, is also relevant. If each value is tagged with its available time, a knowledge process can easily determine the total aggregated processing and communication delay associated with the value, which is useful in dynamic reconfiguration. Note that the available time is not the same as the time when the value was retrieved from the stream, as retrieval may be delayed by other processing.

The available time is also essential when determining whether a system behaves according to specification, which depends on the information actually available at any time as opposed to information that has not yet arrived.

## V. DyKnow

A concrete example of a stream-based knowledge processing middleware framework called DyKnow has been developed as part of our effort to build UAVs capable of carrying out complex missions [6, 10, 12]. Most of the functionality provided by DyKnow has already been presented in the previous section, but one important decision for each concrete instantiation is the type of entities it can process. For modeling purposes, DyKnow views the world as consisting of *objects* and *features*.

Since we are interested in dynamic worlds, a feature may change values over time. Due to the dynamic nature of the value of a feature a *fluent* is introduced to model the value of a feature. A fluent is a total function from time to value, representing the value of a feature at every time-point. Example features are the speed of a car, the distance between two cars, and the number of cars in the world.

Since the world is continuous and the sensors are imperfect the fluent of a feature will in most cases never be completely known and it has to be approximated. In DyKnow, an approximation of a fluent is represented by a *fluent stream*. A fluent stream is a totally ordered sequence of *samples*, where each sample represents an observation or an estimation of the value of the feature at a particular time-point.

To satisfy the sixth requirement of having a declarative specification of the information being generated, DyKnow introduces a formal language to describe knowledge processing applications. An application declaration describes what knowledge processes and streams exists and the constraints on them. To model the processing of a dependent knowledge process a *computational unit* is introduced. A computational unit takes one or more samples as inputs and computes zero or more samples as output. A computational unit is used by a dependent knowledge process to create a new fluent generator. A *fluent generator declaration* is used to specify the fluent generators of a knowledge process. It can either be primitive or dependent. To specify a stream a *policy* is used.

The DyKnow implementation sets up the stream processing according to an application specification and processes the streams to satisfy their policies. Using DyKnow an instance of the traffic monitoring scenario has successfully been implemented and tested [13].

## VI. Related Work

There is a large body of work on hybrid architectures which integrate reactive and deliberative decision making [2–4, 18, 19]. This work has mainly focused on integrating actions on different levels of abstraction, from control laws to reactive behaviors to deliberative planning. It is often mentioned that there is a parallel hierarchy of more and more abstract information extraction processes or that the deliberative layer uses symbolic knowledge, but few are described in detail [1, 16, 17].

The rest of this section focuses on some approaches claiming to provide general support for integrating sensing and reasoning as opposed to approaches limited to particular subproblems such as symbol grounding, simultaneous localization and mapping or transforming signals to symbols.

4D/RCS is a general cognitive architecture which claims to be able to combine different knowledge representation techniques in a unified architecture [20]. It consists of a multi-layered hierarchy of computational nodes each containing sensory processing, world modeling, value judgment, behavior generation, and a knowledge database. The idea of the design is that the lowest levels have short-range and high-resolution representations of space and time appropriate for the sensor level while higher levels have long-range and low-resolution representations appropriate to deliberative services. Each level thus provides an abstract view of the previous levels. Each node may use its own knowledge representation and thereby support multiple different representation techniques. But the architecture does not, to our knowledge, provide any support for the transformation of information in one node at one abstraction level to information in another node on another abstraction level.

SCENIC [21] performs model-based behavior recognition, distributing tasks such as spatial reasoning and object recognition, classification and tracking into three processing stages: Low-level analysis, middle layer mediation and high-level interpretation. From an integration perspective the middle layer, which tries to match top-down hypotheses with bottom-up evidence and computes temporal and spatial relations, is clearly the most interesting. However, it is also quite specific to this particular task as opposed to being a general processing framework.

Gunderson and Gunderson (2006) claim to bridge the gap between sensors and symbolic levels for a cognitive system using a *Reification Engine* [8]. While other approaches mainly focus on grounding for the purpose of reasoning

about the world, the authors claim that a system should also be able to use a symbol to affect the world, citing this bidirectionality as a critical insight missing in other work on symbol grounding. The major weakness with this approach is the focus on a single step approach to connecting a symbol to sensor data, a process we believe will require several steps where the intermediate structures will be useful as well.

The CoSy Architecture Schema Toolkit (CAST) is another general cognitive architecture [9]. It consists of a collection of interconnected subarchitectures (SAs). Each SA contains a set of processing components that can be connected to sensors and effectors and a working memory which acts like a blackboard within the SA. One special SA is the *binder* which creates a high-level shared representation that relates back to low-level subsystem-specific representations [14]. It binds together content from separate information processing subsystems to provide symbols that can be used for deliberation and then action. By deliberation the authors mean processes that explicitly represent and reason about hypothetical world states. Each SA provides a *binding proxy* which contains a set of attribute-value pairs called *binding features* corresponding to the internal data in the SA. The binder will try to bind proxies together to form *binding unions* which fuse the information from several proxies to a single representation. The set of unions represent the best system wide hypothesis of the current state. A weakness is that the binder is mainly interested in finding matching *binding proxies* which are then merged into *binding unions* representing the best hypothesis about the current system state. The system provides no support for other types of refinement or fusion.

## VII. Summary

As autonomous physical systems become more sophisticated and are expected to handle increasingly complex and challenging tasks and missions, there is a growing need to integrate a variety of functionalities developed in the field of artificial intelligence. A great deal of research in this field has been performed in a purely symbolic setting, where one assumes the necessary knowledge is already available in a suitable high-level representation. There is a wide gap between such representations and the noisy sensor data provided by a physical platform, a gap that must somehow be bridged in order to ground the symbols that the system reasons about in the physical environment in which the system should act.

As physical autonomous systems grow in scope and complexity, bridging the gap in an ad-hoc manner becomes impractical and inefficient. At the same time, a systematic solution has to be sufficiently flexible to accommodate a wide range of components with highly varying demands. Therefore, we began by discussing the requirements that we believe should be placed on any principled approach to bridging the gap. As the next step, we proposed a specific class of approaches, which we call stream-based knowledge processing middleware and which is appropriate for a large class of autonomous systems. This step provides a considerable amount of structure for the integration of the necessary functionalities, but still leaves certain decisions open in order to avoid unnecessarily limiting the class of systems to which it is applicable. Finally, DyKnow was presented to give an example of an existing implementation of such middleware.

## References


[1] Virgil Andronache and Matthias Scheutz. APOC - a framework for complex agents. In *Proceedings of the AAAI Spring Symposium*, pages 18–25. AAAI Press, 2003.

[2] R. C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[3] Marc S. Atkin, Gary W. King, David L. Westbrook, Brent Heeringa, and Paul R. Cohen. Hierarchical agent control: a framework for defining agent behavior. In *Proc. AGENTS '01*, pages 425–432, 2001.

[4] P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *J. Experimental and Theoretical AI*, 9, April 1997.

[5] S. Coradeschi and A. Saffiotti. An introduction to the anchoring problem. *Robotics and Autonomous Systems*, 43(2-3):85–96, 2003.

[6] Patrick Doherty, Patrik Haslum, Fredrik Heintz, Torsten Merz, Per Nyblom, Tommy Persson, and Björn Wingman. A distributed architecture for autonomous unmanned aerial vehicle experimentation. In *Proc. DARS'04*, 2004.

[7] Malik Ghallab. On chronicles: Representation, on-line recognition and learning. In *Proc. KR'96*, pages 597–607, November 5–8 1996.

[8] J. P. Gunderson and L. F. Gunderson. Reification: What is it, and why should i care? In *Proceedings of Performance Metrics for Intelligent Systems Workshop*, pages 39–46, 2006.

[9] Nick Hawes, Michael Zillich, and Jeremy Wyatt. BALT & CAST: Middleware for cognitive robotics. In *Proceedings of IEEE RO-MAN 2007*, pages 998–1003, August 2007.

[10] Fredrik Heintz and Patrick Doherty. DyKnow: An approach to middleware for knowledge processing. *J. Intelligent and Fuzzy Systems*, 15(1):3–13, nov 2004.

[11] Fredrik Heintz and Patrick Doherty. Managing dynamic object structures using hypothesis generation and validation. In *Proc. Workshop on Anchoring Symbols to Sensor Data*, 2004.

[12] Fredrik Heintz and Patrick Doherty. A knowledge processing middleware framework and its relation to the JDL data fusion model. *J. Intelligent and Fuzzy Systems*, 17(4), 2006.

[13] Fredrik Heintz, Piotr Rudol, and Patrick Doherty. From images to traffic behavior – a UAV tracking and monitoring application. In *Proc. Fusion'07*, Quebec, Canada, July 2007.

[14] Henrik Jacobsson, Nick Hawes, Geert-Jan Kruijff, and Jeremy Wyatt. Crossmodal content binding in information-processing architectures. In *Proc. HRI'08*, Amsterdam, The Netherlands, March 12–15 2008.

[15] Christian Jensen and Curtis Dyreson (eds). The consensus glossary of temporal database concepts - february 1998 version. In *Temporal Databases: Research and Practice*. 1998.

[16] Kurt Konolige, Karen Myers, Enrique Ruspini, and Alessandro Saffiotti. The Saphira architecture: a design for autonomy. *J. Experimental and Theoretical AI*, 9(2–3):215–235, April 1997.

[17] D.M. Lyons and M.A. Arbib. A formal model of computation for sensory-based robotics. *Robotics and Automation, IEEE Transactions on*, 5(3):280–293, 1989.

[18] Barney Pell, Edward B. Gamble, Erann Gat, Ron Keesing, James Kurien, William Millar, Pandurang P. Nayak, Christian Plaunt, and Brian C. Williams. A hybrid procedural/deductive executive for autonomous spacecraft. In *Proc. AGENTS '98*, pages 369–376, 1998.

[19] M. Scheutz and J. Kramer. RADIC – a generic component for the integration of existing reactive and deliberative layers for autonomous robots. In *Proc. AAMAS'06*, 2006.

[20] Craig Schlenoff, Jim Albus, Elena Messina, Anthony J. Barbera, Raj Madhavan, and Stephen Balakrisky. Using 4D/RCS to address AI knowledge integration. *AI Mag.*, 27(2):71–82, 2006.

[21] Kasim Terzić, Lothar Hotz, and Bernd Neumann. Division of work during behaviour recognition – the SCENIC approach. In *Workshop on Behaviour Modelling and Interpretation, KI'07*, 2007.