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

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

In the past several years, attempts have been made to broaden the traditional definition of data fusion as state estimation via aggregation of multiple sensor streams. One of the more successful proposals for providing a framework and model for this broadened notion of data fusion is the U.S. Joint Directors of Laboratories (JDL) data fusion model [1, 3] shown in Figure 1. The gap between such models, which describe a set of functions which should be components of a deployed system, and the actual instantiation of data fusion in a software architecture is very much an open and unsolved problem.

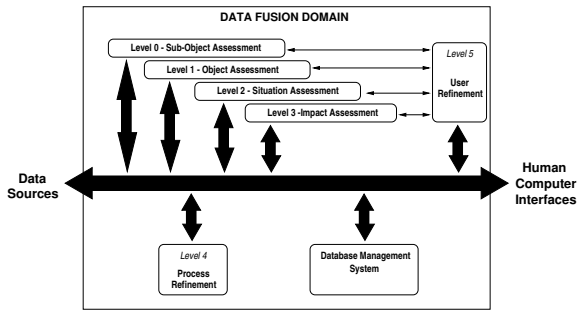

Figure 1: Revised JDL data fusion model from [1].

We have developed a knowledge processing middleware framework called DyKnow in an attempt to bridge the gap. DyKnow was designed, implemented, and tested in a prototype deliberative/reactive software architecture for a deployed unmanned aerial vehicle [2].

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

Ontologically, we view the external and internal environment of the agent as consisting of physical and non-physical *entities*, *properties* associated with these entities, and *relations* between these entities. The properties and relations associated with entities will be called *features*. Due to the potentially dynamic nature of a feature, that is, its ability to change values through time, a *fluent* is associated with each feature. A fluent is a function of time whose range is the feature's type.

Grounding and anchoring internal representations of external entities in the world is one of the great open problems in robotics. Consequently, middleware systems for knowledge processing must provide suitable support for the management of representations and their relation to the external entities they represent. In DyKnow an *object identifier* refers to a specific entity and a *link* between two object identifiers represents that they are hypothesized as refering to the same entity. The collection of object identifiers linked toghether represents the current knowledge about the identity of the entity. The object structures are one of the main concepts used

on the object assessment level of the JDL model where they are used to reason about objects and object types.

The fluents associated with features are the target of the representation in DyKnow. A feature has exactly one fluent in the world which is its true value over time. The actual fluent will almost never be known due to uncertain and incomplete information, instead we create approximations. Therefore, the primitive unit of knowledge is the *fluent approximation*. There are two types of fluent approximations, primitive and computed. A primitive fluent approximation acquires its values from an external source, such as a sensor or human input, while a computed fluent approximation is a function of other fluent approximations. To do the actual computation a *computational unit* is used. The computational unit is a function taking a number of fluent approximations as input and generating a new fluent approximation as output. The fluent approximation is a very general concept and are used on all the JDL levels to represent dynamic information while computational units are used to encapsulate existing data fusion algorithms.

Since a fluent may be approximated in many different ways each feature may have many approximated fluents associated with it. Each fluent approximation is specified by a declarative *policy* which represents how it is created and what properties it has. The policy is the context in which the observations of the feature are interpreted. The policy can also be used by other services to reason about the fluent approximations. This is very useful at the process and user refinement levels where the system or the user should be able to adapt the current data fusion process as the world evolves.

Two important concepts in many applications are states and events. In DyKnow a *state* is a composite feature which is a coherent representation of a collection of features. A state synchronizes a set of fluent approximations, one for each component feature, into a single fluent approximation for the state. The need for states is obvious if we consider that we might have several sensors each providing a part of the knowledge about an object, but whose fluent approximations have different sample rates or varying delays. The state concept is the other important concept on the object assessment level since it is used to fuse data streams from different sensors related to a single object into a coherent representation of that object.

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

The event concept is important on the situation assessment level of the JDL model where relations between objects should be recognized. Situations can be represented by computed events such as temporal logic formulas or chronicles describing temporal relations between events. This way different features are fused together over time in order to extract more abstract situations. The computed events are also well suited to extract higher level events that can be reported to the user as an indication that something important has happend. This abstraction can be used on the user refinement level to reduce the cognitive load on the user.

To summarize, we believe that DyKnow provides suitable concepts to integrate existing software and algorithms related to data fusion and world modelling in general. Observe that the focus is not on individual data fusion techniques but on the infrastructure which permits the use of many different data fusion techniques in a unified framework.

References

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

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

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