REASONING ON DATA STREAMS FOR SITUATION AWARENESS*

Norbert Baumgartner[1], Wolfgang Gottesheim[2], Stefan Mitsch[2], Werner Retschitzegger[2] and Wieland Schwinger[2]

[1]*team Communication Tech. Mgt. GmbH, Goethegasse 3, 1010 Vienna, Austria*

[2]*Johannes Kepler University Linz, Altenbergerstr. 69, 4040 Linz, Austria*

norbert.baumgartner@te-am.at, {firstname.lastname}@jku.at

Keywords: Stream reasoning: situation awareness.

Abstract: Information overload is a severe problem for human operators of large-scale control systems, for instance, in road traffic management. In order to determine a complete and coherent view of the overall situation (i. e., gain situation awareness), an operator of such a system must consider various heterogeneous sources providing *streams* of information about a large number of real-world objects. Since the usage of ontologies has been regarded to be beneficial for achieving situation awareness, various ontology-driven situation awareness systems have been proposed. Coping with evolving and volatile individuals in ontologies, however, has not been their focus up to now. In this paper, we describe how concepts from data stream management systems and stream reasoning, such as sliding windows, continuous queries, and incremental reasoning, can be adjusted to support reasoning over highly dynamic ontologies for situation awareness. We conclude our paper with a prototypical implementation and a discussion of lessons learned, pointing to directions of future work.

1 INTRODUCTION

Gaining situation awareness in data streams. Information overload is a severe problem for operators of large-scale control systems, such as encountered in the domain of road traffic management (RTM). In order to determine a complete and coherent view of the overall (traffic) situation, an operator of such a system must consider heterogeneous sources, such as traffic jam detectors and traffic incident reports, providing information *streams* about a large number of objects. In order to reduce information overload, situation awareness (SAW) systems, according to Endsley (Endsley, 2000), support operators by pointing them to relevant information during (i) *perception* of objects (e. g., a traffic jam), (ii) *comprehension* of current situations (e. g., a wrong-way driver approaches a traffic jam), and (iii) *projection* of situation evolution (e. g., a wrong-way driver may cause an accident).

Ontology-driven situation awareness systems. Since the usage of ontologies is beneficial for SAW (Llinas et al., 2004), various ontology-driven SAW systems representing data streams in ontologies, for instance (Baumgartner et al., 2010), (Kokar et al., 2009) have been proposed. In contrast to traditional ontology-driven information systems focusing on query answering (cf. (Buccella et al., 2009) for a recent survey), SAW systems often operate on *data streams* resulting from constantly monitoring the environment under control. This entails, that reasoning components, such as a situation assessor determining the current situation, must process a highly dynamic ontology comprising evolving (e. g., a growing traffic jam) and volatile individuals (e. g., an accident). Although ontology-driven approaches in the area of situation awareness exist, coping with evolving and volatile individuals in ontologies has not been the focus up to now[1]. In this paper, we describe how concepts from data stream management systems and stream reasoning, such as sliding windows, continuous queries, and incremental reasoning, can be adjusted to support reasoning for SAW.

Challenges of highly dynamic ontologies. In order to establish a basis for describing the challenges of highly dynamic ontologies in situation awareness, in the following a brief overview of major reasoning components of an SAW framework (Baumgartner et al., 2010) is given. As part of applying this frame-

*This work has been funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under grant FIT-IT 829598.

[1]Note, that the research field of ontology evolution (Hartung et al., 2011) focuses on a different problem, namely the evolution of the *schema level* of ontologies.

work to a particular domain, *adaptors* and *domain mappers* resolve structural and semantic heterogeneity by representing information from a data stream first as individuals of a domain ontology, which is then mapped into a core ontology. This core ontology characterizes *objects* by *attributes* (e. g., location). The history of an object is described in terms of changes to their attributes, assigning to each attribute value a *temporal validity* and a direct predecessor. Due to frequent updates in the underlying data streams, the core ontology comprises highly dynamic individuals. Framework components must cope with these dynamic individuals as detailed below.

First, on the perception level a *chronologically consistent* ordering of attribute values in the face of arbitrary update sequences across multiple data streams must be established. For example, a traffic jam detector may report a traffic jam without delay, possibly resulting in a radio station report arriving later in time, but actually describing the jam's state that was valid *before*. During this ordering, a *change collector* avoids propagating undeterministic update frequencies of underlying data streams to components on higher levels by collecting updates into batches. Second, on the comprehension level attribute values determine relations between objects, which are further aggregated to situations (task of a *situation assessor*, cf. (Baumgartner et al., 2010) for details). Hence, the challenge in data streams is to handle reasoning complexity by appropriately chosing the size of sliding windows providing a view on data streams.

Structure of the paper. Section 2 discusses related work from data stream management and stream reasoning, as a basis for Sect. 3 and Sect. 4 applying concepts from these research communities to SAW. Section 5 describes a prototypical implementation, and Sect. 6 lessons learned and further research directions.

2 RELATED WORK

In this section, we discuss concepts from data stream management systems forming the basis for closely related work from the area of stream reasoning. In data stream management systems (DSMS), various concepts for handling the dynamic nature of data streams have been described (cf. (Golab and Ozsu, 2003) for a comprehensive overview of these concepts): *Fixed, landmark, and sliding windows*[2] constrain the size of an ever increasing data stream to those elements being relevant for query execution. Common practice is to define the size either in terms of *time* or *information item count* (Golab and Ozsu, 2003). DSMS either support *continuous queries* over such sliding windows in a *monotonic* fashion (i. e., assume that newly arrived information do not affect previous query results), or in a *non-monotonic* manner (i. e., may need to re-evaluate previous results). Such concepts are successfully applied, e. g., in context aware systems for continuous spatial queries (Farrell et al., 2011). Focusing, however, on information processing without considering rich background knowledge, DSMS are utilized for *querying rather than reasoning*. In constrast, existing semantic technologies supporting reasoning assume static knowledge (Stuckenschmidt et al., 2010). The exploitation of concepts from both worlds is the focus in stream reasoning (Stuckenschmidt et al., 2010).

In stream reasoning, as part of the LarKC project, various concepts have been proposed on the basis of DSMS (Barbieri et al., 2010), (Stuckenschmidt et al., 2010), (Valle et al., 2009). Della Valle et al. (Valle et al., 2009) describe two complementary stream reasoning frameworks: (i) combining data stream management systems with standard reasoners and (ii) extending existing query languages, such as SPARQL. In our work, we follow the first approach by using *adaptors* for resolving structural heterogeneity between data streams and applying standard reasoners in SAW *reasoning components*. In the terminology of Della Valle et al, these adaptors and SAW reasoning components are called transcoders and prereasoners, respectively. Besides the basic task of attaching timestamps to RDF triples (Barbieri et al., 2010), (Valle et al., 2009), these approaches, however, do not focus on sorting those triples describing object attributes into a chronologically consistent manner. Concerning comprehension-level challenges, Barbieri et al. (Barbieri et al., 2010) handle updates to materialized deductions whenever new information enters the sliding window. Exceeding their fixed window sizes, we additionally present concepts for dynamically adjusting windows. For non-monotonic continuous queries, which are necessary in the presence of deleted and changed facts, we utilize provenance information in our ontology for tracking deductions back to the facts they base upon and re-evaluate deductions only when necessary.

In summary, current stream reasoning approaches, although taking a first step by utilizing general DSMS concepts, have not yet focused on major challenges in situation awareness, in particular, (i) handling of arbitrary update sequences, and (ii) adjusting sliding window sizes. In the following sections, we describe *stream approaches* to SAW, focusing these challenges on the perception and comprehension level.

[2]Fixed windows have two fixed ends, landmark windows have one fixed and one sliding end, and sliding windows have two sliding ends (Golab and Ozsu, 2003).

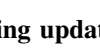
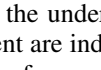
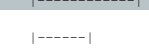
3 STREAM PERCEPTION

Adjusting update frequency. Since the underlying data streams in road traffic management are independently maintained and highly dynamic, frequent data updates at arbitrary times can occur. Each of these updates may cause subsequent processing steps of further components on the perception, comprehension, and projection level. In order to avoid high performance overhead caused by consecutive, temporally close updates, a *change collector* aggregrates incoming updates from the underlying data streams (in a similar manner as, for instance, context aggregators (Dey, 2000)). Thereby, the change collector moves forward a sliding window at configurable time intervals, allowing the execution of subsequent components on batch updates. As an additional task, the change collector marks updated objects, thus enabling subsequent components to work incrementally.

Chronologically sorting attribute value changes. As a result of arbitrary update sequences, the temporal intervals describing validity of attribute values may be in arbitrary order as well. To achieve a chronologically correct ordering of attribute values, the topological relationship between such temporal intervals is of major importance. We utilize Allen's interval algebra (Allen, 1983), being perhaps the most established topological temporal calculus, in order to describe topological relations between attribute valid time intervals. Following Allen's interval algebra, the data source update sequences of the valid time of a previous and a current attribute value result in 13 possible update cases (e. g., attribute valid times may overlap, or one may be contained in another). In case multiple attribute values are in conflict, strategies developed in the data integration community can be used to resolve such conflicts. A detailed discussion of possible strategies can be found in (Bleiholder and Naumann, 2008), two sample strategies are depicted in Table 1. In this example those cases making fusion necessary are shown (i. e., the valid time of previous information and that of an update *start* or *finish* at the same time, one is *during* the other, or both are *equal* to each other). By applying the strategy *Take the Information* (Bleiholder and Naumann, 2008), information newly added to the sliding window (top bar) is preferred over previously added information (bottom bar), whereas by applying the strategy *Trust your Friends* (Bleiholder and Naumann, 2008), information of a particular data stream (bottom bar) is preferred over information of another stream (top bar).

After consolidating incremental information from data streams into an integrated ontology, comprehension of current situations can take place.

Table 1: Comparison of sample fusion strategies.

Relation	Illustration	Take the Information	Trust your Friends
Starts			
During			
Finishes			
Equal			

4 STREAM COMPREHENSION

In order to support human operators in their comprehension of current situations, especially relevant is the description of relations between objects (e. g., close to) (Barwise and Perry, 1983). In SAW systems, this task is the responsibility of *situation assessment* algorithms, which derive relations between objects from object attributes and aggregate these relations to situations (cf. (Baumgartner et al., 2010) for details). In order to restrict situation assessment to the most relevant types of situations only, many SAW systems (e. g., (Kokar et al., 2009)) use rules for defining which types of relations must be derived between which types of objects. For example, the situation type `AccidentNearTrafficJam` could be defined as a rule stating that such a situation should be reported, if an object of type `Accident` is in a relation of type `Near` with an object of type `TrafficJam`, but only when the accident occurred `During` the lifetime of the traffic jam. Such rules in an SAW system represent reasoning goals for *continuous processing* (Barbieri et al., 2010). Since satisfying these rules involves pairwise comparison of objects for each possible relation (Baumgartner et al., 2010), sliding window size is of crucial importance to reduce computational effort. Sliding window size can be adjusted either *statically* on the basis of situation and object types or *dynamically* on the basis of current instances thereof, as described in the following paragraphs.

Statically adjusting sliding windows on the basis of situation rules. A central question in reasoning over data streams is concerned with finding the appropriate size of the sliding window. A large window size results in a large number of objects contained in the sliding window, thereby lowering situation assessment performance, whereas choosing a small window size results in the risk of missing relevant situations. The types of these situations, and in particular the rules defining how such a situation can be detected, however, provide valuable clues for defining an appropriate window size. As a first hint, types of relations describing *temporal distance* between objects define *upper bounds* for the size of sliding windows. Tempo-

Table 2: Relation types and relevant information about valid time intervals.

1st Valid Time	2nd Valid Time	Relation	Illustration
?------|	|-----?	Before	|------| |------|
|------?	?-----|	After	|------| |------|
?------|	|-----?	Meets	|------| |------|
?------|	|-----?	Meets Inverse	|------| |------|
|------|	|-----|	Starts	|------| |------------|
|------|	|-----|	During	|------| |------------|
|------|	|-----|	Finishes	|------| |------------|
|------|	|-----|	Equal	|-------| |-------|

ral relation types concerning the topology of intervals (e. g., the relation type `During` used above), however, do not define such clear-cut boundaries. Rather, they depend on valid times of the involved objects. For adjusting sliding windows to fit such relation types, we can only provide *experience-based* boundaries observed from real-world object evolution (e. g., accidents are typically cleared within two hours). Such experience-based boundaries, however, bear the risk of missing relevant situations. In order to find all relevant situations, we must *dynamically* adjust sliding windows on the basis of both, situation rules and current objects, as discussed below.

Dynamically adjusting sliding windows depending on current objects. Temporal relation types without clear boundaries, such as topological relations between time intervals, make it necessary to define sliding windows not in terms of statically fixed boundaries, but in terms of the objects contained in them. Table 2 lists the information about valid time intervals being relevant for defining sliding window sizes that are sufficiently large to satisfy Allen's interval algebra (Allen, 1983).

From this table showing Allen's interval relations, we see that the relation `After` should only be used in combination with temporal distance relations. Otherwise, objects can never be removed from the sliding window, since at any time other objects occurring after them may be added to the sliding window.

5 Prototypical Implementation

In this section, we describe the software infrastructure of our prototypical implementation and its application in RTM on real-world information from the Austrian highways agency[3].

[3] `http://www.asfinag.at`

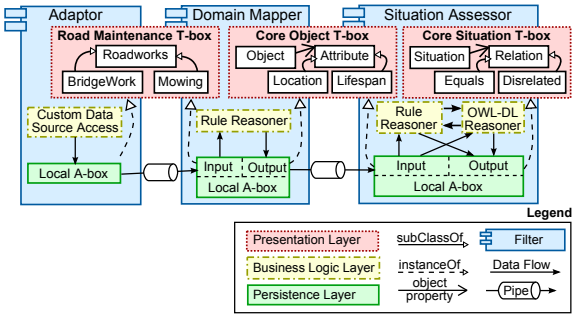


Figure 1: Components in a pipes-and-filters architectural style.

Software infrastructure. Our infrastructure is largely based on Java and the Jena Semantic Web Framework[4]. For OWL reasoning within Jena, we use the Pellet OWL Reasoner[5] offering reasonable performance. For situation assessment, which accesses object information extensively, we additionally use the RDF triple store AllegroGraph[6] for its in-server reasoning capabilities (Baumgartner et al., 2010). AllegroGraph is a client-server system that integrates with Jena and Pellet on the client side. On the server side, AllegroGraph provides a Prolog and Common Lisp environment which both are employed for developing the algorithms of our SAW framework.

Arrange reasoning components in a pipeline. Employing the described software infrastructure, in this paragraph we discuss the application of the well-known software architectural style *pipes-and-filters* (Buschmann et al., 1998) with respect to data stream processing. In this architectural style (cf. Fig. 1), processing components—filters implementing parts of the three levels of SAW, e. g., a situation assessor supporting comprehension—are arranged in a consecutive manner by connecting them with pipes (e. g., implemented using plain method calls or Java RMI communication), forming altogether a processing pipeline. In such a pipeline, filters depend only on the results of their direct predecessors, and hence, processing of elements in a data stream can be parallelized. Let us discuss the design of these filters making them pluggable components in detail. Each filter contains its own volatile, A-box and a pipe connecting the filter to the A-box(es) of the succeeding filters(s) (cf. (Baumgartner et al., 2008)). A filter listens for updates to this input A-box, computes inferences upon these updates using a local OWL DL and rule reasoner, and writes the results into its output A-box and the referenced succeeding A-box(es). Thereby,

[4] `http://jena.sourceforge.net`
[5] `http://pellet.owldl.com`
[6] `http://agraph.franz.com`

just the A-box statements that are relevant for the following filter(s) are handed over via pipes, resulting in mutually shared A-box(es). The individuals in these shared A-boxes are consistent with the concepts in a shared T-box: Each filter defines the expected input vocabulary (i. e., individuals written to its input A-box must be consistent with respect to the input T-box), as well as the output vocabulary it complies with (i. e., each filter guarantees to produce only individuals that are consistent with respect to the output T-box). As a result, filters in a processing pipeline can be arranged in an arbitrary manner, as long as neighboring filters agree on a shared T-box. In case such an agreement cannot be established, additional *domain mappers* must be inserted between them, which mediate between filters by mapping the concepts of a preceding filter to concepts understood by a succeeding one.

Preliminary performance results. We obtained preliminary performance test results describing the performance of the *situation assessor* component, since this component is the slowest filter in our processing pipe due to pairwise object comparisons being performed between all objects in its sliding window. Note, that in our current prototypical implementation, sliding windows are only adjusted statically, while the implementation and evaluation of dynamically adjusting sliding windows is part of our future work. The performance tests were run upon real-world traffic data recorded centrally for Austrian highways over a period of four weeks. These data are reported by multiple heterogeneous sources, comprising (i) a *roadworks management system* that provides information about scheduled roadworks, traffic restrictions, and expected traffic jams, (ii) a *traffic jam detection system* that reports traffic jams which are automatically detected, (iii) an *incident management system* that provides manually entered traffic-related incident data, and (iv) a nation-wide broadcasting station that provides diverse traffic information, ranging from incidents and traffic jams to poor driving conditions. The recorded data set used for this evaluation consists of 28,616 distinct traffic objects, comprising 25,269 traffic jams, 820 road works, 1,803 other obstructions, 614 accidents, 46 wrong-way drivers, and 64 severe environmental conditions, such as snow or ice on the road. Currently, a rather small number of approximately 250 traffic objects and their associated information are active at the same time (i. e., should be part of the statically adjusted sliding window).

The test case design is influenced mainly by (i) the *number of objects* to be compared, i. e., the size of the sliding window, ranging from 10 to 500 objects (*N10*, *N50*, *N100*, *N500*), and (ii) the *focus* of situation types: a low focus means that a situation type is rather general and therefore results in many matches, whereas a high focus means that a situation type is very specific (*F1*: low focus, *F2*: high focus). We measured two performance indicators, being independent of the execution environment: (i) the number of direct object comparisons (CMP), and (ii) the number of derived relations (REL). Fig. 2 summarizes the test results. Unsurprisingly, the size of the sliding window

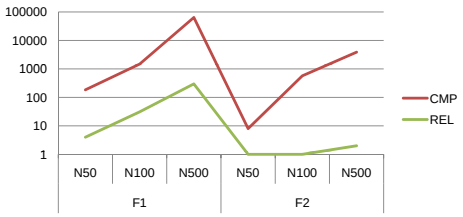


Figure 2: Evaluation test results.

dramatically influences the performance of the situation assessor, since it compares objects within its sliding window in a pairwise manner. However, the effect on runtime performance can be counteracted with correctly focused situation types (i. e., by configuration of the situation assessor): low-focused situation types (cf. F1) lead to a more than tenfold increase of pairwise object comparisons compared to F2 (just to give a runtime estimation, the 184 comparisons of N50 in F1 take up 16.8 seconds on our test machine—AMD Athlon 64 3500+ with 1.8 GB RAM).

6 CONCLUSION

In this section, we present lessons learned from implementing and evaluating the framework in RTM and indicate directions for future work.

Knowledge about window size boundaries makes issuing warnings possible. In order to comply with real-time constraints for situation assessment (e. g., report critical situations to human operators within one minute), during intervals of extensive traffic, such as beginnings of holidays or periods of heavy snow fall, it may be necessary to reduce the sliding window size. With knowledge from relation types used in particular situation rules, however, in such a case we are at least able to issue warnings to human operators, bringing to their attentation that particular types of situations will potentially be missed.

Pipes mediate between filters with different push-pull processing characteristics. Reasoning components in an SAW system may exhibit different data stream processing characteristics, such as pro-actively *pulling* information from a pipe (e. g., upon execution

of a user query) or re-actively waiting for the pipe to *push* new information into the component (e. g., when new information is reported by an automated system). Different kinds of pipes, such as pipes buffering the results of a preceding filter for being pulled from it later, are therefore necessary to bridge between adjacent reasoning components with different processing characteristics. Such buffering pipes may also flatten data stream peaks, allowing a slower filter to catch up with a faster one during intervals of reduced load.

REFERENCES

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.

Almeida, A., de Ipia, D. L., Aguilera, U., Larizgoitia, I., Laiseca, X., Ordua, P., and Barbier, A. (2008). An approach to dynamic knowledge extension and semantic reasoning in highly-mutable environments. In *Proc. of 3rd Symp. of Ubiquitous Computing and Ambient Intelligence*. Springer.

Babcock, B., Babu, S., Data, M., Motwani, R., and Widom, J. (2002). Models and issues in data stream systems. In *Proc. of 21st Symp. on Principles of Database Systems*. ACM.

Barbieri, D., Braga, D., Ceri, S., Valle, E. D., and Grossniklaus, M. (2010). Stream reasoning: Where we got so far. In *Proc. of 4th Int. Workshop on New Forms of Reasoning for the Semantic Web: Scalable and Dynamic*.

Barwise, J. and Perry, J. (1983). *Situations and Attitudes*. MIT Press.

Baumgartner, N., Gottesheim, W., Mitsch, S., Retschitzegger, W., and Schwinger, W. (2010). BeAware!—situation awareness, the ontology-driven way. *International Journal of Data and Knowledge Engineering*, 69(11):1181–1193.

Baumgartner, N., Retschitzegger, W., and Schwinger, W. (2008). A software architecture for ontology-driven situation awareness. In *Proc. of 23rd Annual Symp. on Applied Computing*. ACM.

Bleiholder, J. and Naumann, F. (2008). Data fusion. *ACM Comp. Surveys*, 41(1).

Buccella, A., Cechich, A., and Fillottrani, P. (2009). Ontology-driven geographic information integration: A survey of current approaches. *Computers and Geosciences*, 35(4):710–723.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1998). *Pattern-Oriented Software Architecture—A System of Patterns*. Addison-Wesley.

Dey, A. K. (2000). *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology.

Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Ma, L., Schonberg, E., and Srinivas, K. (2007). Scalable semantic retrieval through summarization and refinement. In *Proc. of the Conf. of the Association for the Advancement of Artificial Intelligence*. AAAI.

Endsley, M. (2000). *Situation Awareness Analysis and Measurement*, chapter Theoretical Underpinnings of Situation Awareness: A Critical Review, pages 3–33. Lawrence Erlbaum Associates, New Jersey, USA.

Farrell, T., Rothermel, K., and Cheng, R. (2011). Processing Continuous Range Queries with Spatiotemporal Tolerance. *IEEE Transactions on Mobile Computing*, 10(3):320–334.

Golab, L. and Ozsu, M. (2003). Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14.

Gupta, A., Mumick, I. S., and Subrahmanian, V. S. (1993). Maintaining views incrementally. In *Proc. of the SIGMOD Conf.* ACM.

Hartung, M., Terwilliger, J., and Rahm, E. (2011). *Schema Matching and Mapping*, chapter Recent advances in schema and ontology evolution. Springer.

Kokar, M. M., Matheusb, C. J., and Baclawski, K. (2009). Ontology-based situation awareness. *International Journal of Information Fusion*, 10(1):83–98.

Krämer, J. and Seeger, B. (2009). Semantics and impl. of continuous sliding window queries over data streams. *ACM Transactions on Database Systems*, 34(1).

Llinas, J., Bowman, C., Rogova, G., and Steinberg, A. (2004). Revisiting the JDL data fusion model II. In *Proc. of 7th Int. Conf. on Information Fusion*.

Stuckenschmidt, H., Ceri, S., Valle, E. D., and van Harmelen, F. (2010). Towards Expressive Stream Reasoning. In *Semantic Challenges in Sensor Networks*, Dagstuhl Seminar Proceedings.

Valle, E., Ceri, S., Barbieri, D. F., Braga, D., and Campi, A. (2009). A first step towards stream reasoning. In Domingue, J., Fensel, D., and Traverso, P., editors, *Future Internet—FIS 2008: Revised Selected Papers*. Springer.