A Software Architecture for Ontology-Driven Situation Awareness

Norbert Baumgartner
team Communication
Technology Management Ltd.
Goethegasse 3/3
1010 Vienna, Austria
norbert.baumgartner at te-am.net

Werner Retschitzegger
Johannes Kepler University Linz
Altenberger Str. 69
4040 Linz, Austria
werner.retschitzegger at jku.at

Wieland Schwinger
Johannes Kepler University Linz
Altenberger Str. 69
4040 Linz, Austria
wieland.schwinger at jku.at

ABSTRACT

Human operators of large-scale control systems face the problem of information overload induced by the large amount of information provided by multiple heterogeneous and highly-dynamic information sources. Situation-aware information systems support operators by the aggregation of the available information to meaningful situations. Ontologies are a promising technology for realizing such systems, because of their semantically-rich kind of knowledge representation. The cross-cutting role of ontologies and the streaming character of situation awareness, however, challenge the design of an appropriate software architecture. In this paper, we propose a domain-independent software architecture based on a core ontology for situation awareness which leverages the reusability and the scalability of involved software components. This is achieved by the application of the well-known software architecture pattern pipes-and-filters. The proposed architecture is demonstrated by examples from the field of road traffic management. In addition, we contribute several lessons learned which should be helpful for developing ontology-driven information systems in general.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: domain-specific architectures, patterns

Keywords

Ontologies, Situation Awareness, Software Architecture

1. INTRODUCTION

Information overload is a severe problem for human operators of large-scale control systems. By large-scale control systems we mean systems in a heterogeneous and highly-dynamic environment, which deal with a large number of real-world objects as, e.g., in the field of road traffic management. An operator of such a system has to consider all *streams* of information about these objects—typically stemming from different information sources—in order to determine a complete and coherent view of the occurring situations and, thereupon, to take the right action (e.g., reduce the speed limit of a road using variable message signs). *Situation awareness* (SAW) supports operators by pointing their attention to relevant sets of interrelated objects thereby reducing information overload through the aggregation of the available information to the situations of interest (e.g., a traffic jam that causes an accident). Since the usage of *ontologies* has been regarded to be beneficial for SAW [9], we have developed a conceptual framework of an *ontology-driven information system* for SAW in our previous work (e.g., [3], [2]).

The first problem towards the implementation of such a conceptual framework is ascribable to the general characteristics of ontology-driven information systems (cf., Guarino et. al. [7]). That is, in ontology-driven information systems, the ontology is an integral and local part of the system which is used at run time. With respect to "traditional" multi-tier software architectures, this means that the ontology contributes to the persistence layer, the business logic layer, and the presentation layer across the whole architecture. To give an example, the individuals of the ontology represent the persistent information received by various information sources (e.g., sensor networks), they are aggregated to situations via mostly declarative business logic, and they are also used for the communication with the human operator on the presentation layer. Thus, the ontology has a *cross-cutting role* and induces a tight coupling of the layers of the architecture thereby endangering its *reusability*. Unfortunately, discussions how to overcome this problem in high-level software architectures for ontology-driven information systems are scarcely available.

The second problem is that current ontology-driven information systems typically focus on query answering. In contrast, SAW applications usually operate on *streams* of information, since they have to constantly monitor the environment under control. That is, the focus is on pushing dynamic individuals rather than pulling static individuals from an ontological knowledge base. Thus, ontology-driven information systems dealing with such streams of information have to be capable of processing highly-dynamic and volatile individuals (e.g., a continuously growing traffic jam)

SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil

which not least compromises the *scalability* of the software architecture.

In the scope of this paper, we contribute a software architecture for an ontology-driven, reusable framework for SAW applications which tackles the above problems and leverages reusability as well as scalability. Beforehand, we provide an overview of related work in section 2. Along with the introduction of the software architecture in Sect. 3, we demonstrate our approach by applying it to the domain of road traffic management[1]. In addition, we provide a number of lessons learned during the development of the proposed architecture which should be helpful for reconciling software architectures and ontologies in general (cf., Sect. 4). We conclude the paper with a summary of our contribution as well as an overview of further prospects in Sect. 5.

2. RELATED WORK

Naturally, the closest field of related work are current ontology-driven approaches to SAW (cf., our comparison in [1]). SAWA (Situation Awareness Assistant) by Matheus et. al. [10], an ontology-driven SAW application originating from the military domain, provides an overview of the overall system architecture its SAW ontology is used in. However, Matheus et. al. merely depict the functional building blocks of their system, a discussion of the technical software architecture elaborating on the problems identified in the previous section is missing. In this respect, the work by Chen et. al. [5] is similar—CoBrA (Context Broker Architecture) provides an ontology-driven system architecture for pervasive context-aware environments. However, Chen et. al. do not discuss the aspects of reusability and scalability when developing such a highly-dynamic, ontology-driven system.

The second area of related work are software architectures for ontology-driven information systems. Protégé [6] is a framework or set of tools for building ontology-driven information systems. However, Protégé focuses on the persistence and presentation layers of a multi-tier architecture omitting the actual implementation of declarative business logic—which is admittedly not the goal of Protégé. Nevertheless, one can not deduce a generic software architecture for ontology-driven information systems by using the framework. In general, such middleware for using ontologies largely focus on specific layers. For example, Kalyanpur et. al. [8] provide, like many others, an approach to automatically map OWL ontologies into Java thereby providing an approach to tackle the persistence layer. In contrast to our work, all of them target software design—we are rather interested in high-level software architectures above concepts and classes. One of the scarcely available examples that detail the cross-cutting role of ontologies is the work by Tran et. al. [12] who describe an approach to use an ontology in a multi-tier web application based on the middleware KAON2[2]. However, their architecture is more about query answering and semantic mediation, whereas our SAW system involves a streaming-oriented processing with a highly complex business logic.

[1]Our work is supported by the Austrian highways agency ASFINAG Traffic Telematics Ltd.
[2]http://kaon2.semanticweb.org

3. AN ARCHITECTURE FOR ONTOLOGY-DRIVEN SITUATION AWARENESS

In this section, we introduce the application domain of road traffic management and detail the basic information flows. Subsequently, we give insight into the proposed software architecture for ontology-driven SAW applications.

3.1 From Road Traffic Management to Situation Assessment

A typical road traffic management system involves a number of heterogeneous information sources each providing information according to a Terminology (T)-box, i.e., the vocabulary of some domain ontology. The individuals constituting the domain ontologies' Assertions (A)-boxes form a continuous stream of information about real-world traffic objects (e.g., accidents, traffic jams) which have to be aggregated to traffic situations.

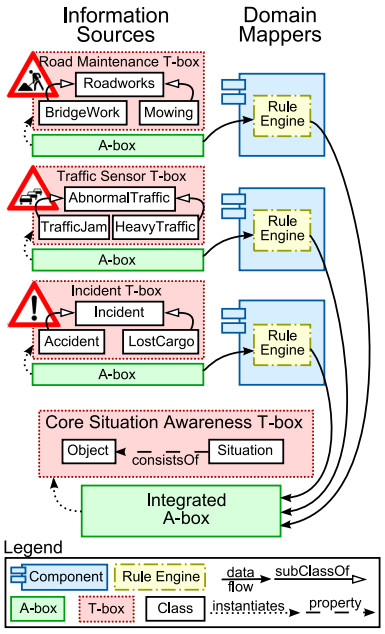

Figure 1: Integration of the domain ontologies

In our running example, we incorporate the following information sources with their corresponding domain ontologies (cf., Fig. 1 for *small extracts* of these ontologies based on our previous work [2]): A road maintenance system providing information about roadworks, an automatic traffic jam detection system based on traffic sensors providing information about current chunks of abnormal traffic, and an incident management system providing information about traffic-related incidents. The operator of such a road traffic management system has to be aware of all available information in an integrated way, in order to assess the occurring traffic situations (e.g., a traffic jam within a section of roadworks). Since these information are continuously and independently delivered by the different information sources (e.g., one information source reports an accident and another one independently reports a sensed traffic jam), an appropriate architecture for achieving SAW has to be capable of handling *streams* of A-box statements, depicted as

solid arrows in Fig. 1. These streams are aggregated to streams of A-box statements about situations (e.g., the situation "accident causes traffic jam"), and are finally communicated to the operator.

In our architecture, the actual assessment of situations as indicated above takes place based on an integrated A-box which adheres to a domain-independent core SAW ontology. We thereby follow the vision of Musen [11]—he anticipates that future software systems are actually generic problem solving methods that operate on specific ontologies; using such systems is just a matter of mapping a domain ontology to the ontology of the system. In our case, this mapping is achieved by means of `Domain Mapper` components which make use of a set of mapping rules per domain ontology (e.g., each instance of the class `Accident` in the incident management's domain ontology is an instance of the class `Object` in the core SAW ontology).

Apart from the possibility to reuse the situation assessment facilities, also the overall performance can be influenced via these mappings, because just the *relevant* individuals are mapped and transfered to the situation assessment system, thus reducing the size of the integrated A-box (e.g., mowing next to the road is negligible because it does not affect the traffic flow). To sum up, the integrated A-box just consists of the relevant A-box statements about individuals from the domain ontologies in terms of the core SAW ontology.

3.2 Leveraging Reusability and Scalability in Ontology-Driven Situation Assessment

Upon the update and mapping of domain individuals, the actual situation assessment is triggered. In detail, the whole situation assessment process is separated into several consecutive reasoning steps which themselves make intensive use of logic programming (LP) on the individuals of the ontology, the usual ontological inferences, and some procedural extensions. The reasoning steps depicted as software components in Fig. 3 are based on our previous work [3] and merely serve as examples to describe the architecture. In detail, the reasoning steps to be executed are the collection of A-box updates (`Change Collector`), the selection of situations to be reevaluated (`Situation Selector`), the projection of the evolution of the selected situations (`Evolution Tracker`), and the actual assessment of new situations (`Situation Assessor`). These reasoning steps should be regarded as functional building blocks which have to be implemented in coherent components. Then, the question arises how to actually reconcile them in an appropriate software architecture. Since all reasoning steps are triggered by incoming streams of A-box statements, the *individuals* again receive our attention when developing this architecture.

A first intuitive approach sketched in Fig 2 (a) would be that all components worked within a globally shared A-box representing the current state of all individuals received from all domain mappers and the statements inferred by a typical ontology inference engine (e.g., derived types). However, if all components worked within this A-box, they would have to be aware of each other in order to avoid inconsistencies. In fact, these inconsistencies would be due to the fact that the different components have a transactional character and should work in a given order—a requirement that can hardly be assured when working in a globally shared A-box. Of course, one could overcome this issue by adding control information to inference results indicating the results that are relevant for a component. Such an approach would however imply further unwanted dependences between the different components leading to an inflexible architecture reducing reusability. An alternative approach would be a *layered* architecture as depicted in Fig. 2 (b). With layering we mean that each component would take the result of the previous one and would thus work on the collected results so far. Although we would thereby avoid most of the unwanted influences between the components themselves, we still have the problem of performance or, hence, scalability. As we move up the layers, the number of individuals naturally increases, whereas it is not clear-cut that each higher level component needs every individual at all. Rather, the number of relevant individuals should decrease, because situation assessment targets at aggregating information about objects to situations. That is, by focusing on the relevant individuals, we could omit this burden on the reasoning engine.

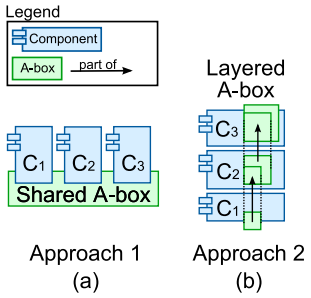

Figure 2: Two flawed approaches

Our alternative solution to the above problems of reusability and scalability proposed in this paper is the application of a well-known software architecture pattern: Pipes-and-filters [4]. The pipe enables the consecutive execution of several components, each being dependent on the previous one. Let us view these components as pluggable pipe elements[3], each containing its own volatile, local A-box and a reference to the local A-box(es) of the following pipe element(s) (cf. Fig. 3). That is, each pipe element constructs an A-box which is accessible from the outside world. Moreover, the pipe element listens for updates to this A-box (persistence layer), computes some inferences upon these updates using the local rule engine (business logic layer), presents the results to the user (presentation layer), and writes the results into the referenced A-box(es). Thereby, just the A-box statements that are relevant for the following pipe element(s) are handed over via the mutually shared A-box(es). By starting at the last element, one can construct a pipe for the whole situation assessment process backwards. Each pipe element is responsible for its local A-box, which references the core SAW T-box. Nevertheless, it may be necessary to incorporate some ground facts into situation assessment which are independent from the different pipe elements. Thus, an overall and persistent A-box outside the pipe exists which is accessible from the pipe elements. Apart from initial facts like, for example, the structure of the road

[3]Note that the pattern pipes-and-filters calls these pipe elements "filters".

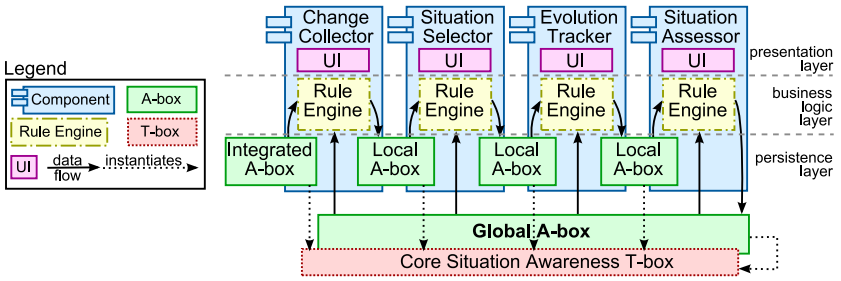

Figure 3: An architecture for ontology-driven situation awareness

network, this A-box contains the final and persistent results of situation assessment without inconsistencies.

Let us revisit this architecture with respect to the initial problem of ontology-driven information systems—the cross-cutting role of ontologies in a multi-tier architecture. Because this cross-cutting role is inevitable, we have separated concerns by introducing a fine-granular vertical slicing (collect changes, select situations, track evolution, assess situations). Each vertical component, i.e. pipe element, has its internal and—via the ontology—tightly coupled multi-tier architecture. Flexibility is introduced by a uniform internal architecture for each pipe element enabling customizable sequences of consecutive components. For example, we may simply eliminate the assessment of evolving situations by omitting the corresponding pipe element `Evolution Tracker`. In detail, each component has its own, but across components reusable interfaces to A- and T-boxes (persistence layer), the same ontological inference and logic programming engines (business logic layer), and a uniform user interface.

3.3 Implementation

We have implemented our architecture using the Jena Semantic Web framework[4]. We have chosen Jena, because it has a large developer community, a quite consistent API, and and a tightly integrated LP reasoning engine based on RDF triples. Regarding the actual setup, the domain ontologies as well as the core SAW ontology are formalized using OWL-DL. For the ontological inferences we need for our application we use the rule-based OWL reasoner provided by Jena. Moreover, most of the business logic is implemented with Jena's LP engine. Where needed, procedural extensions to rules are implemented.

As described above, each component has its own reasoning engine and its own local A-box, which is kept in memory in order to provide a reasonable performance. Since components are triggered by changes to these A-boxes, A-box updates are performed in a transactional manner in order to avoid premature, i.e., too early, triggers and, hence, inconsistencies. The final results of situation assessment are persisted in the global A-box enabling the publication of the assessed situations.

4. LESSONS LEARNED

In this section, we describe several lessons we have learned during the development of the beforehand introduced architecture and which should be helpful for building ontology-driven information systems in general. We believe that in particular developers of streaming-oriented systems, i.e. those that push information instead of pulling it, should benefit from our experiences.

4.1 A-Boxes Drive Software Architecture

Most of current research on ontologies focuses on T-boxes, whereas actual applications have to cope with individuals in A-boxes. There are, however, several ontology middleware projects (e.g., Protégé, Jena, KAON2) that—partially together with external reasoners and triple stores—may handle thousands of A-box statements in a reasonable amount of time. Although these middleware mostly provide, for example, object-oriented abstraction layers, there are hardly any guidelines for how to consistently work with these individuals in a larger scope, especially when working in such a highly dynamic environment like SAW. In this context, the most severe problems we came across during the implementation is the processing of updates to A-boxes. Naturally, in a streaming-oriented system as induced by SAW, one is just informed of the *delta*, i.e. the changed statements regarding an individual which have to be sorted into the A-box. Therefore, we also had to introduce a mechanism to notify components about individuals that are created or cease to exist in a transactional manner in order to avoid inconsistencies. In a nutshell, we advise to not underestimate the problems involved with highly dynamic A-boxes and rather volatile individuals (e.g., traffic jams)—in fact, we believe A-boxes drive the architectural aspects of ontology-driven information systems.

4.2 Shared A-Boxes Enable Separation of Concerns

A tight coherence of components is one of the most important requirements in object-oriented design. Because of an ontology's cross-cutting role, this coherence can scarcely be reached along the vertical dimension of a multi-tier architecture in an ontology-driven information system. The horizontal, functional dimension may, nevertheless, be separated into coherent components—in case a separation of concerns regarding the different functional building blocks of the system is established. Once such a separation is found, the question is how coherent components actually interact. Examining approaches in traditional rule-based systems, such a separation of concerns corresponds to a combination of rule grouping and interface rules for bridging groups. However, such an approach involves a high level of complexity (cf., Musen [11]) which should be avoided. In our view, this complexity can be overcome if components interact via—at

[4]http://jena.sourceforge.net

best mutually—shared A-boxes thereby omitting rule dependencies. Whereas this approach is especially intuitive in streaming-oriented ontology-driven information systems, we believe that it should also be applicable to other applications. In the end, it seems to be natural to use A-boxes, which already inhere a cross-cutting role *within* a component, for the interaction with other components as well.

4.3 Sequenced A-Boxes Leverage Scalability

Another natural issue regarding logic programming and ontological inferences is the problem of reasoning performance and scalability. Although one may use the performance tweaks offered by the applied reasoning engine, a trivial way to achieve better performance is to decrease the size of the A-box which allows to keep as much statements as possible in memory. Our approach to achieve this decrease is to favor sequences of A-boxes to layered A-boxes. Though a bilateral dependency between two components that share an A-box is thereby established, this trade-off is acceptable, not least because a declarative description of the individuals a component operates with seems to be possible. A shortcoming of sequencing is that some statements are duplicated from one A-box to the other thereby allocating more memory than layering. Anyhow, the components become easier to distribute enabling better scalability. We believe that using sequences of components with local A-boxes for ontology-driven information systems involving consecutive reasoning steps is reasonable, because the application as a whole will be easier to distribute and to scale.

5. SUMMARY AND FUTURE WORK

In this paper, we have proposed a software architecture for an ontology-driven information system which should support SAW of human operators in large-scale control systems like the field of road traffic management. By the incorporation of coherent software components, which we have separated along the functional dimension of a "traditional" multi-tier architecture, into the well-established software architecture pattern pipes-and-filters, we leveraged reusability and scalability despite the cross-cutting role of ontologies. In addition to this reconciliation of software architectures and ontologies for SAW, we have described our experiences gained during the implementation of the proposed architecture in form of several lessons learned. Although these lessons learned clearly focus on streaming-oriented systems, we believe that they should also be beneficial in a more general context.

An open issue is to verify the claimed reusability and scalability of our approach by concrete evaluation metrics. We will perform this evaluation based on BeAware!, a software framework for SAW applications we are currently implementing based on the architecture proposed in this paper. Potential metrics could be the throughput of the architecture or the lines of code one has to implement for adopting the framework in a concrete application domain. In the near future, we plan to demonstrate BeAware!'s real-world scalability by deploying a prototypical implementation for a road traffic management system, in order to support operators achieving situation awareness in complex road traffic management scenarios. Finally, we are eager to develop prototypes for further application domains, like the field of air traffic control, thereby verifying the applicability of our approach with other large-scale organizations.

6. REFERENCES

[1] N. Baumgartner and W. Retschitzegger. A survey of upper ontologies for situation awareness. In *Proc. of the 4th IASTED International Conference on Knowledge Sharing and Collaborative Engineering, St. Thomas, U.S. VI*, pages 1–9, Nov. 2006.

[2] N. Baumgartner, W. Retschitzegger, and W. Schwinger. Lost in time, space, and meaning—an ontology-based approach to road traffic situation awareness. In *Proc. of the 3rd Workshop on Context Awareness for Proactive Systems (CAPS), Guildford, UK*, June 2007.

[3] N. Baumgartner, W. Retschitzegger, W. Schwinger, G. Kotsis, and C. Schwietering. Of situations and their neighbors—evolution and similarity in ontology-based approaches to situation awareness. In *Proc. of the 6th International and Interdisciplinary Conference on Modeling and Using Context, Roskilde, Denmark*, Aug. 2007.

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

[5] H. Chen, T. W. Finin, and A. Joshi. Using OWL in a pervasive computing broker. In *Proc. of the Workshop on Ontologies in Agent Systems (OAS), Melbourne, Australia*, pages 9–16, July 2003.

[6] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of Protégé: An environment for knowledge-based systems development. *International on Human-Computer Studies*, 58(1):89–123, 2003.

[7] N. Guarino. Formal ontology and information systems. In *Proc. of the 1st International Conference on Formal Ontologies in Information Systems (FOIS), Trento, Italy*, pages 3–15, June 1998.

[8] A. Kalyanpur, D. J. Pastor, S. Battle, and J. A. Padget. Automatic mapping of owl ontologies into java. In *Proc. of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE), Banff, Canada*, pages 98–103, June 2004.

[9] J. Llinas, C. Bowman, G. Rogova, and A. Steinberg. Revisiting the JDL data fusion model II. In *Proc. of the 7th Intern. Conference on Information Fusion, Stockholm, Sweden*, pages 1218–1230, June 2004.

[10] C. Matheus, M. Kokar, K. Baclawski, J. Letkowski, C. Call, M. Hinman, J. Salerno, and D. Boulware. SAWA: An assistant for higher-level fusion and situation awareness. In *Proc. of SPIE Conference on Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications, Orlando, USA*, pages 75–85, March 2005.

[11] M. Musen. *Knowledge Engineering and Agent Technology*, chapter Ontology-Oriented Design and Programming, pages 3–16. Frontiers in Artificial Intelligence and Applications. IOS Press, 2004.

[12] D. T. Tran, H. Lewen, and P. Haase. On the role and application of ontologies in information systems. In *Proc. of 5th IEEE International Conference on Research, Innovation & Vision for the Future (RIVF), Hanoi, Vietnam*, pages 14–21, March 2007.