Short Paper: Using SOAR as a semantic support component for Sensor Web Enablement

Ehm Kannegieser , Sandro Leuchter

Fraunhofer IOSB, Department of Interoperability and Assistance Systems
{ehm.kannegieser, sandro.leuchter}@iosb.fraunhofer.de

Abstract. Semantic service discovery is necessary to facilitate the potential of service providers (many sensors, different characteristics) to change the sensor configuration in a generic surveillance application without modifications to the application's business logic. To combine efficiency and flexibility, semantic annotation of sensors and semantic aware match making components are needed. This short paper gives the reader an understandig of the SOAR component for semantic SWE support and rule based sensor selection.

Keywords: SOAR, SCA, rule based sensor selection, service discovery

1 Introduction

It is a common conception that semantic service discovery is necessary to facilitate the Internet of Services because the plethora of potential service providers has to be matched to the specific needs of a service consuming value chain. From our point of view this is also true for the Internet of Things – in our case a sensor web – because there are many sensors with different characteristics available. We want to be able to change the sensor asset configuration in a generic surveillance application without modifications to the application's business logic. We also envision a SOA-like, wide area enterprise architecture for sensor webs where different sensor service providers will be combined in a cost efficient and flexible way. To achieve this, semantic annotation of sensors and semantic aware match making components are needed. In the remainder of this paper we describe our solution to this problem definition, based upon Sensor Web Enablement (SWE), Service Component Architecture (SCA) and the SOAR rule engine as well as a prototype application in the surveillance domain.

2 Sensor Web Enablement

2.1 Semantic Support for SWE

SWE is a suite of OGC standards, e.g. Sensor Markup Language (SensorML), Sensor Planning Service (SPS) and Sensor Observation Service (SOS), which provides an open interface for sensor web applications as described in [1].

In a time of rapidly developing semantic web, sensors and sensor data have to be accessible in a feasible kind of way, thus have their capabilities described semantically. Ontological descriptions and annotations achieve this by adding helpful metadata to sensors and data, harnessing the massive amount of available information.

Originating from a semantic aware service discovery that fits into SOAs and is based upon the SOAR rule engine, we have developed a semantic component to aid the process of sensor tasking and sensor data retrieval in a SWE environment by finding the most feasible sensors and related data.

In a proof of concept we introduce a perimeter control application with simple service enabled sensors. The component uses an ontological representation of attributes and capabilities of deployed sensors and a custom rule set that uses context information to deduce constraints of the current situation and proposes sensor services best suited to current task and context.

3 System Architecture

3.1 SOAR

State, Operator, Action, Result (SOAR) describes the solution of a problem as a number of state transitions. A starting state (representing the problem) is changed into a final state (representing the solution) by changes to the systems memory, done by rules. A SOAR rule consists of two parts: condition and action. The condition describes a specific working memory (WM) pattern. If this pattern is matched by changes (e.g. input) to the working memory, the action is triggered, changing WM into a new pattern which may trigger other rules. This way SOAR is able to react to system context changes and transit towards the desired system state.

There are three different ways to store knowledge in SOAR: the WM is an acyclic directed graph which represents all known facts. Rules that are changing WM are stored in the production memory and last but not least, preference memory (PM) maintains a ranking of operator feasibility. In case that there is more than one feasible operator, an impasse arises and – if solved – one or more production rules (chunks) are created in the production memory to avoid same (or similar) WM constellation in the future. This mechanism enables SOAR with basic learning capabilities [2; 3].

To help with useful recommendations, SOAR needs to rely on facts. Thus, knowledge of the subject matter (e.g. sensors and their feasibility in specific environmental conditions) is modeled in an ontology. This allows for a reliable initialization of WM with facts from ontology classes and derivation of rules from relations (e.g. “is better/worse than”, “is part”), if specified.

3.2 Service-Oriented Architecture

For reasons of flexibility and re-usability we chose the Open SOA Collaboration (OSOA) Service Component Architecture (SCA) to encapsulate the SOAR engine. SCA applications consist of composites which again consist of components (Figure 1) which can be implemented in different programming languages (e.g. Java). Apart from the specific implementation used, the behavior of the components is described to combine them into complex applications [6].

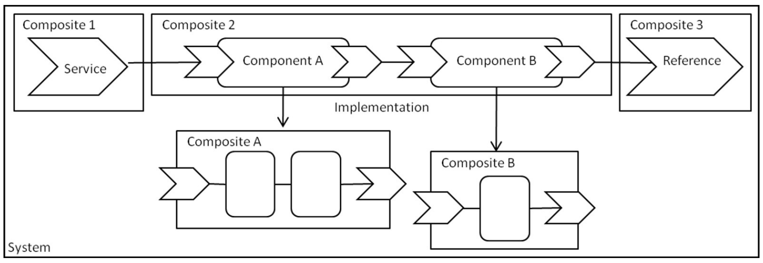

Figure 1. SCA-architecture

The component provides an interface for operation and communication with the encapsulating system. Through this interface the SOAR-kernel and its memories can be accessed (e.g. updating WM with new facts). The interface provided by the component is implemented in Java and features a number of methods for communication and control of the SOAR-kernel (see Table 1).

Table 1. SOAR-component methods

Method name	Performed action
start()/void stop()	Starts/Stops the SOAR-kernel
query(IDs)	Sends a query to the SOAR-kernel. A list of sensor ids (IDs) is passed to the method. Returns a list of sensor ids as result.
setContext(KVPs)	Writes updates/changes to the WM. A list of key-value-pairs is passed to the method. This is the most important method because it allows the WM to be changed. Changes to the WM may trigger rules that affect the recommendation of sensors.
setOntologyFile(String,String)	Sets the ontology and the corresponding XSLT file for transformation of semantic information into the SOAR-kernel. Two filenames are passed to the method.
setProductionsFile(String)	Sets the productions file which contains the production and preference rules needed for

sensor recommendation.

3.3 Ontology

While SensorML features description of physical sensor capabilities and quality, more general properties like cost or access restrictions plus the system context (e.g. weather conditions) which the sensors are matched with, needs to be described semantically. To model this, we chose the W3C standard OWL. XML encoded files can easily be transformed into arbitrary output formats using XSLT and XPATH [5] therefore easily be transferred into SOAR WM.

3.4 Case Study Evaluation

Objective of this study is to create a surveillance system that operates on an exchangeable set of sensors and adapt its recommendations to the changing environment conditions. A specific scenario is planned as follows: A defined area (e.g. small room, hallway, laboratory) is equipped with different sensors (e.g. photoelectric barriers, pressure contacts, acoustic, ultrasonic, video, and luminosity sensors) and actuators (e.g. horn, warning light) to detect movement and then sound an alarm. Therefore the SOAR-component is integrated into an Open Geospatial Consortium (OGC) SPS and SOS architecture which are described in length at [7; 8].

In the following, the workflow of a recommendation process is described: Firstly if there is a request, the SOAR-component is initialized with the current system context and sensor description by its setContext()-method. This method inserts a key value pair (attribute, value) which describes the sensors properties and relations into the WM structure. A list of (available) sensor ids corresponding to those sensor descriptions used for initialization is fetched from the SPS using the query()-method. The SOAR-component then elaborates a list of recommended sensors according to a set of rules specifying the optimization goal (e.g. cost, time, access) provided by the setProductionsFile()-method. The recommended sensor Ids are passed to the SOS for further use (e.g. for inserting observations from this sensors only).

4 Conclusion & Future Work

In this paper we have presented a semantic component capable of rule driven sensor selection. In a proof of concept a perimeter control application which uses a semantic representation of sensors to propose sensor services best suited to current task and context, was set up. Future work will focus on additional semantic capabilities of the SOAR-component such as inference and mapping ontological description of sensors, relations and system context, using additional semantic features of SOAR (e.g. semantic memory) and extending SWE support (e.g. sensor data interpretation). Further fields of activity will be semantic service discovery in the context of SOA, Semantic Grid and Cloud Computing.

5 References

1 Open Geospatial Consortium Inc.: Sensor Web Enablement, http://www.opengeospatial.org/ogc/markets-technologies/swe, 2009.
2 John E. Laird and Clare Bates Congdon: The Soar User's Manual, 2011.
3 The Soar Group (2009): Soar Website. University of Michigan. http://sitemaker.umich.edu/soar/home, 2009.
4 W3C (2004): OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/, 2004.
5 W3C (o.J.): Extensible Markup Language (XML). http://www.w3.org/XML/, 2009.
6 Chappell, David: Introducing SCA. http://www.davidchappell.com/articles/Introducing_SCA.pdf, 2007.
7 Arthur Na; Mark Priest: Sensor Observation Service, 2008.
8 Johannes Echterhoff; Ingo Simonis: OGC 09-000. Sensor Planning Service, 2011.