AUTOMATYKA/ AUTOMATICS • 2012 • Vol. 16 • No. 2

http://dx.doi.org/10.7494/automat.2012.16.2.175

Piotr Szwed*, Grzegorz Rogus*, Paweł Skrzyński*, Michał Turek*, Jan Werewka*

Towards an Ontology Approach to ATAM Based Assessment of Service Oriented Architectures

1. Introduction

In this paper we describe SOAROAD (SOA Related Ontology of Architectural Decisions), which was developed to support the evaluation of architectures of information systems using SOA technologies. The main goal of the ontology is to provide constructs for documenting service-oriented architectures, however, it is designed to support future reasoning. Building the ontology we focused on the requirements of the Architecture Tradeoff Analysis Method (ATAM).

ATAM is a scenario-based method for architecture assessment defining a quality model and an organizational framework for an evaluation process. ATAM represents expected system qualities as a mapping between scenarios and quality attributes. System architecture being the input for ATAM is expressed in the form of views describing components and their connections. During the evaluation a team of experts analyzes selected properties of components and connections to detect sensitivity points, tradeoffs and assigns risks.

A limitation of the method is that it depends on expert knowledge, perception and previous experience. It may easily happen, that an inexperienced evaluator overlooks some implicit decisions and risks introduced by them.

We propose SOAROAD ontology as a tool supporting the ATAM based assessment of systems following a service-orientation paradigm and deploying web services.

The ontology gathering knowledge about various properties and decisions related to SOA may facilitate performing an assessment in a more exhaustive manner, helping to ask questions, revealing implicit design decisions and obtaining more reliable results.

1.1 Architecture evaluation – why SOA?

Nowadays, Service Oriented Architecture (SOA) might be treated as a state of the art approach to the design and implementation of enterprise software, which is driven by business requirements. In the SOA paradigm, the system is made up of loosely connected

* AGH University of Science and Technology, Krakow, Poland

services, which constitute components that can be reused many times and may occur in various compositions. The characteristic features of software developed in line with the SOA are as follows:

- The ability to call up services regardless of the technology and network location;
- One to one communications: at one moment in time, the service consumer can call up only one service, whereas the communication is two way;
- The flow is initiated by the service consumer – service call up;
- Synchronous communication.

Another paradigm is the EDA (Event Driven Architecture) in which applications and services exchange events in an asynchronous way one with another. EDA systems are often called event-oriented SOAs: this paradigm represents an extension of the SOA and not its competitor thus this is complementary to SOA approach. The characteristic features of the EDA architecture are as follows:

- Communication in which sources of events are not aware of the existence of event recipients;
- Many to many communication: one event can be consumed by many subscribers;
- Flow initiated by the buyer depending on the event type;
- Asynchronous communication.

Interestingly, from the implementation perspective, both architectures can use the same design pattern – the ESB (Enterprise Service Bus). This pattern combines both approaches, which makes integrating business modules and different platforms significantly easier. The ESB plays the role of an intermediate layer, which makes possible communication between different processes. A consumer of an event can initiate every service implemented within the bus. The ESB provides all the capabilities determined by both paradigms and for this reason it is the recommended [23] way of implementing the enterprise architecture within this approach. Hence if one aims in creating useful ontology the concept of ESB and related terms must be taken into consideration.

1.2. Related works

Ontology of architectural decisions was proposed in [16]. It distinguished by several types of decisions that can be applied to software architecture and its development process. The main categories included: Existence, Ban, Property and Executive decisions. The ontology defined also attributes, which were used to describe decisions, including states (Idea, Tentative, Decided, Rejected, etc.). In [9] an ontology supporting ATAM based evaluation was proposed. The ontology specified concepts covering the ATAM metamodel, quality attributes, architectural styles and decisions, as well as influence relations between elements of architectural style and quality attributes. The effort to structure the knowledge about architectural decisions, was accompanied by works aimed at the development of tools enabling edition and graphical visualization of design decisions, often in a collaborative mode, e.g. [4, 7, 17].

This short selection of works proves, that the problem of documenting and visualizing architectural decisions as a support for software development process and architecture evaluation remains a challenge. In contrast to approaches aimed at providing the classification of concepts and their relations (TBox) we attempt to gather in the proposed ontology also facts (ABox) constituting ready to use dictionaries of decisions (properties of architectural design) and the knowledge about their relations reflecting the current state of the art for SOA technologies.

1.3. ATAM Overview

ATAM (Architecture Tradeoff Analysis Method) [15, 6] is an early scenario-based method for software architecture evaluation developed at SEI (Software Engineering Institute). Considering many different quality goals ATAM is capable of capturing the interaction between them, setting goal dependencies and assessing that a proposed system architecture can satisfy those goals. The ATAM process can work in a procedural loop. Each iteration should be ended with improvement suggestions, including better architectural decisions and new development scenarios.

Briefly speaking, the greatest benefit from the ATAM methodology is an early identification of risks, what in consequence can result in appropriate mitigation actions introduced before the software is fully designed and implemented, thus at lower cost. ATAM multi-step system analysis procedure includes detailed risks filtration, sorting and risk reducing processes. In the first place each risk is classified into particular risk themes. So-called impacts, extracted from those risk themes, are afterwards linked with business drivers or an architectural plans. If one of risk theme impacts requires a new business driver definition, a set of quality attributions is made. That also leads to a new scenario developed, which should be considered during the next iteration of the process: a scenario of interaction of one of the (so called) stakeholders with the system. On the other hand, when any risk theme impact requires a new architectural plan definition (instead of a business driver), an architecture approach (or approaches) is being created for subsequent software architecture change.

The ATAM process also includes development team interactions. The whole ATAM process usage typically includes the following steps:

- Steps 1–3: Presentation (ATAM introduction made by an evaluation leader, business drivers to be considered described by project spokesperson and present architecture to be evaluated – described by a system architect with a reference to current business drivers list),
- Step 4: Architectural approaches identification – currently user approaches, without analysis,
- Step 5: Quality attributes collecting. All those attributes should be gathered in an attribute utility tree. It will describe compromises between system utility parameters, such performance, security, usability, availability etc.,
- Step 6: Architectural approaches analysis – using knowledge established in step 5,
- Step 7: Scenarios prioritizing during a voting process,

– Step 8: Architectural approaches analysis – step 6 repetition, but with a scenario priorities knowledge used,
– Step 9: results resolving.

As shown above, ATAM provides the necessary techniques supporting the optimization of software architecture. There are also different types of scenarios in ATAM: classical use case scenarios are extended by growth scenarios (showing changes made to the system) and exploratory scenarios (for extreme changes possibly hazardous to the system). The ATAM approach provides a way to deal with multiple and strongly dependent parameters bound to a different system architecture requirements and also generating different risks. Making a risk dependency tree helps to identify sensitivity points and tradeoffs points (bounded multi-attribute points). It will also refine any descriptions of the architecture's driving quality attribute requirements and architectural design decisions.

A great advantage of ATAM would also be another mechanism. This one provides efficient business drivers translation into quality attribute scenarios. It is based on structures called utility trees. Those trees would allow determining important business issues and restrictions identifying high priority goals for future system architecture. The root of a tree (simple utility entry) leads into branches such performance, modifiability, availability, security etc. Each branch leads to detailed and decomposed information about all architecture features needed to be applied to meet the assumed requirements. Therefore (due to the graphical display of a tree) all the effort to meet enforced requirements could be balanced between business restrictions displayed on the utility tree branches.

2. A metamodel of software architectures

The basic model of software architecture used in ATAM [2] defines it after [20] as a set of components and linking them to connections. We extend this simplistic model by defining *Interfaces* and *Functions* of components. A connection links a component having the caller role with an interface (calee). *Components, connections and interfaces* can be attributed with: *ComponentProperties, ConnectionProperties* and *InterfaceProperties* respectively. Examples of such properties are: *platform, web service type, communication type, queueing, query granularity*.

A Composition is a coherent set of components and connectors. System architecture is itself a composition. For the purpose of analysis we may focus on a particular subset of components and connectors and describe their properties, e.g. the distribution of queries among a several databases building up a composition or realization of a design pattern.

During the ATAM based evaluation the overall system architecture and properties of its parts are analyzed to establish the scenario responses and achievements of corresponding quality attributes. It may be, however, observed that some architecture properties or their combinations have known influence on quality attributes, e.g. a use of asynchronous web services or applying MVC design pattern increases modifiability and a granularity of queries has an impact on performance.

Architectural decision is an assignment of a property to a component, interface, connector or a composition. In this context the terms property and architectural decision can be used to some extent interchangeably. However, it may happen that certain decisions or components are dependent on previously assigned properties. An example of such dependency is the composition type – a property assigned to a set (composition) of web service components. Selecting orchestration as the composition type requires that an orchestration component, e.g. BPEL capable module is be used. The *required* relation or its subproperties in the ontological model express this dependency.

The assumed metamodel adopts the reification strategy while modeling various properties of an architectural design. Properties are defined as classes, whose individuals can be linked by additional relations indicating specific roles. An example of such a property is the MVC design pattern – pattern, which requires identification of a components playing the roles of a Model (typically database), a Controller (e.g. an EJB) and a View (e.g. a set of HTML pages produced by JSP scripts).

Two types of components are distinguished: *ApplicationComponents* and *InfrastructureComponents*. Application components are developed and deployed software modules; infrastructure components provide such supporting functions, as message queuing or service registry.

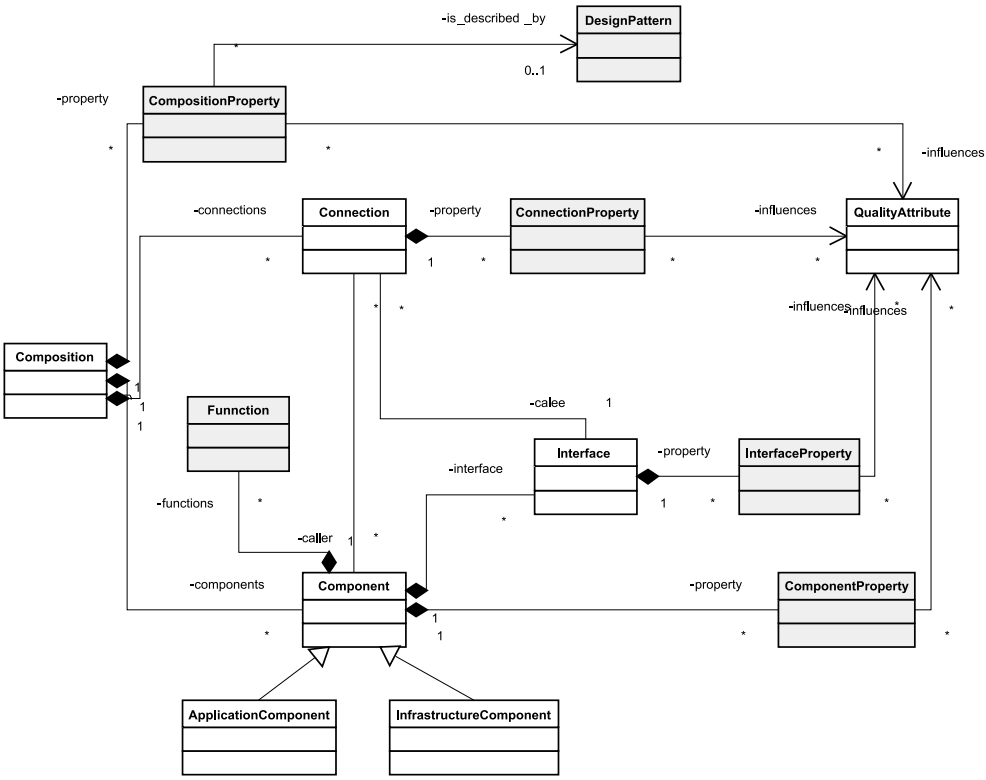

Fig. 1. Metamodel of software architecture and its properties

3. The ontology of the SOA architectural style

Ontology engineering methodologies [10, 10, 11, 21] usually distinguish the following common steps in the ontology development:

1) Specification, aimed at establishing the domain of the ontology, its scope, usage and competency questions (including preparing motivating examples);
2) Conceptualization – the goal of this step is to identify concepts, arrange them in hierarchies and establish relations;
3) Formalization consisting in coding ontology in a formal language, e.g. OWL;
4) Deployment – using the ontology in a software tool.

In this section we will briefly describe the assumptions determined in the specification phase and describe the content of the ontology obtained after formalization. The ontology has not been yet fully deployed: a tool supporting collection and an analysis of information about the properties of an architecture is under development.

3.1. The idea of SOAROAD ontology application

The goal of the SOAROAD (SOA Related Ontology for Architectural Decisions) ontology is to gather the knowledge from the SOA domain and organize it in such manner that:

– it will help ask questions about various properties of architectural design and decisions;
– it will be capable of representing assignments of properties relevant to SOA technology to the elements of system architecture.

It was assumed that the ontology would follow the metamodel described in the previous section defining various properties corresponding to design decisions that can be attributed to components, connections, interfaces and compositions. If applicable, these design decisions can be supplemented by additional relations. The ontology would also specify design patterns.

Another assumption is related to a distribution of the knowledge between ontology TBox (classes and properties) and ABox (individuals and their relations). It was decided that properties would be represented by classes, whereas their values as individuals.

The concept of the ontology application is presented in the Figure 2. The process of building of an architecture description starts with eliciting architecture views ABox, i.e. a set of linked components, interfaces and connections. This model can be prepared either manually or with a support of dedicated import tools converting ArchiMate [18] models of Archi editor [1] or UML [3] from VisualParadigm.

A tool supporting architecture description uses the classes and individuals defined in the SOAROAD ontology TBox and basic ABox respectively to generate forms or

questionnaires in which software architects or members of development teams can make assignments of property values to elements of architecture views. Resulting Architecture ABox refers elements of Architecture views ABox and individuals defined in SOAROAD ontology (merging two input ontologies and asserting additional relations).

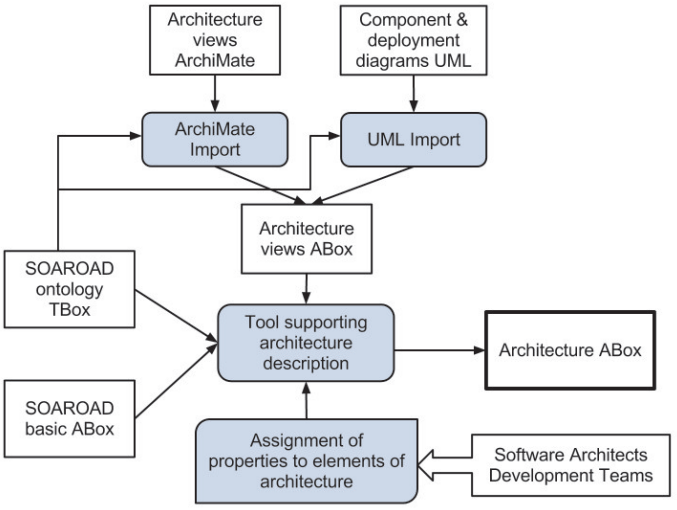

Fig. 2. A concept of application of SOAROAD ontology

Figure 3 illustrates this approach with an excerpt from a resulting Architecture ABox. *ComponentA* via *ConnectionAB* calls functions of *ComponentB* exposed by the *Interface B*. This information originates from architectural views and is contained in the Architecture Views ABox. In the next step each of these nodes is attributed with properties (architectural decisions) being instances of concepts gathered in the SOAROAD ontology to form full Architecture ABox.

In the presented example:

- Component A is deployed on Intel Xeon 2.13 GHz machine running Ubuntu 10.4 system and GlassFish application server.
- Connection AB is asynchronous, uses SSL based protection mechanism and a 10 Gb network.
- Interface B is a SOAP web service with low query granularity and exception handling based on soap faults.
- Component B is deployed on IIS a platform and is stateless.

The resulting graph of interconnected elements with assigned properties presented in a user-friendly browsable form can be input to ATAM analysis performed in the standard manner.

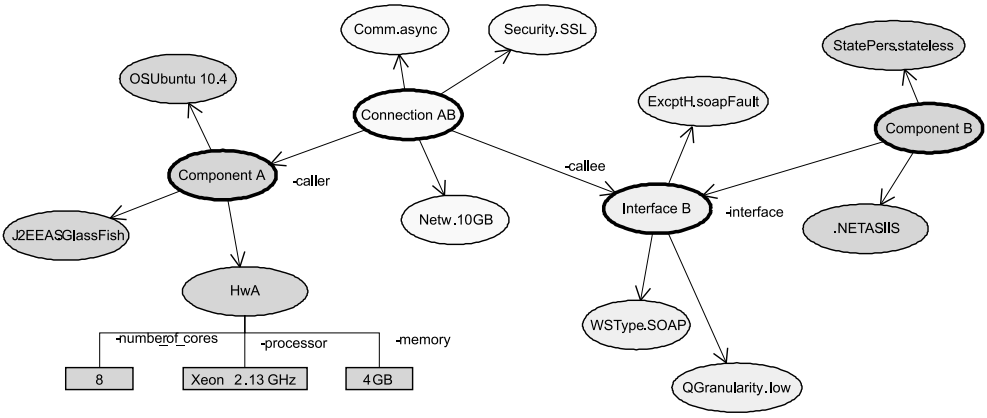

Fig. 3. An example of architecture ABox. Elements of an architectural view (marked with boldlines) are attributes with design decisions (individuals of classes defined in the ontology)

3.2. The ontology content

As mentioned above the model of SOAROAD ontology uses the notations proposed in [18] to represent the ontology. The ontology was formalized in the OWL language and can be accessed using standard ontology tools, e.g. Protégé or with Jena or OWL API software libraries.

Due to limited space, we will describe the SOAROAD ontology by presenting an overall model (Fig. 4) and next illustrate our concepts by choosing some specific examples of classes (ComponentProperty).

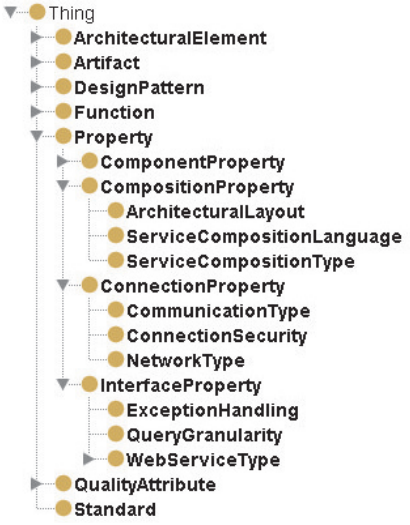

Fig. 4. SOAROAD ontology

The ontology specifies also functions of components. Their list is rather related to infrastructure components. The function is the implementation of a specific service that represents dynamic behaviors of components. Examples of such function are: *Routing, MessageMapping, ProtocolSwitch, MediationService*.

The ontology provides a taxonomy of quality attributes. Quality attributes (Fig. 5) are the overall factors that affect run-time behavior, system design, and user experience. They represent areas of concern that have the potential for application wide impact across layers and tiers. Some of these attributes are related to the overall system design, while others are specific to run time, design time, or user centric issues. The extent to which the application possesses a desired combination of quality attributes such as usability, performance, reliability, and security indicates the success of the design and the overall quality of the software application.

When designing applications to meet any of the quality attributes requirements, it is necessary to consider the potential impact on other requirements by analyzing the tradeoffs between multiple quality attributes. SOAROAD ontology defines *influences* in relation to describing the impact of architectural decisions on quality attributes.

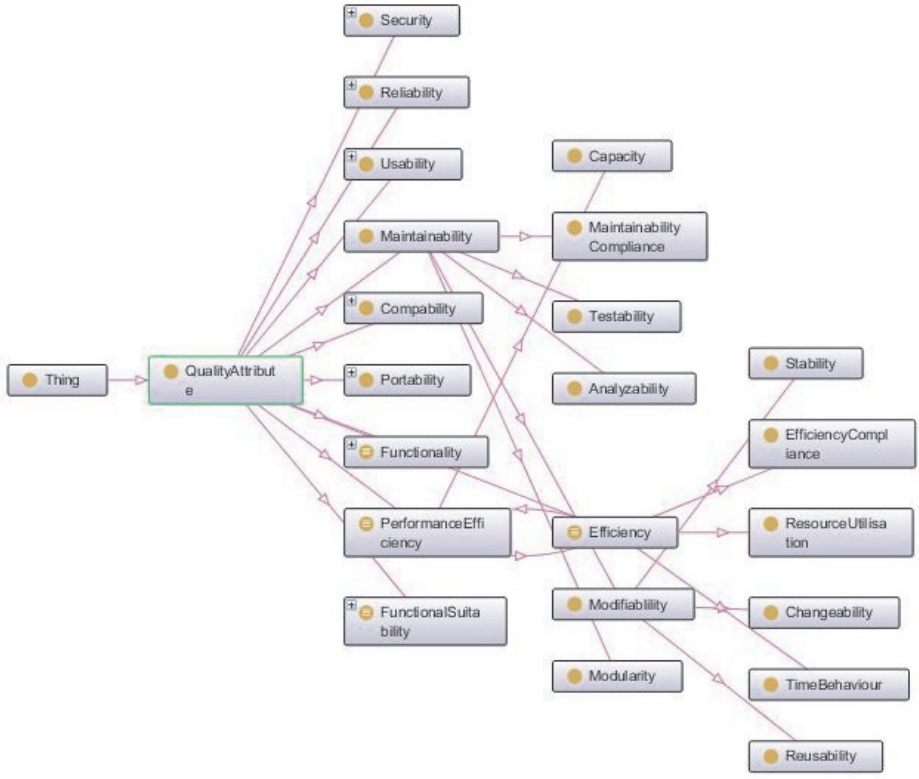

Fig. 5. A partial tree of quality attributes (according to ISO/IEC 9126 and ISO/IEC 25010)

3.3. Component properties

In this section we discuss component properties as an example illustrating the structure of the SOAROAD ontology. Subclasses of *ComponentProperty* (Fig. 6) class define various properties and design decisions, which can be assigned to components. Software architect preparing ATAM evaluation should consider them as an exhaustive list of questions related to important issues in SOA architectures. An example of such properties is *Platform* (*Hardware, OperatingSystem, ApplicationServer*), *PlatformTechnology, ProgrammingLanguage, ComponentLogic* and *ComponentSecurity*.

Most of the classes have predefined individuals, that can be selected in assignments, e.g. *JavaEECompliantAS* has several predefined individuals: *JBoss, Glassfish, WebLogic, WebSphere, ColdFusion*, etc.

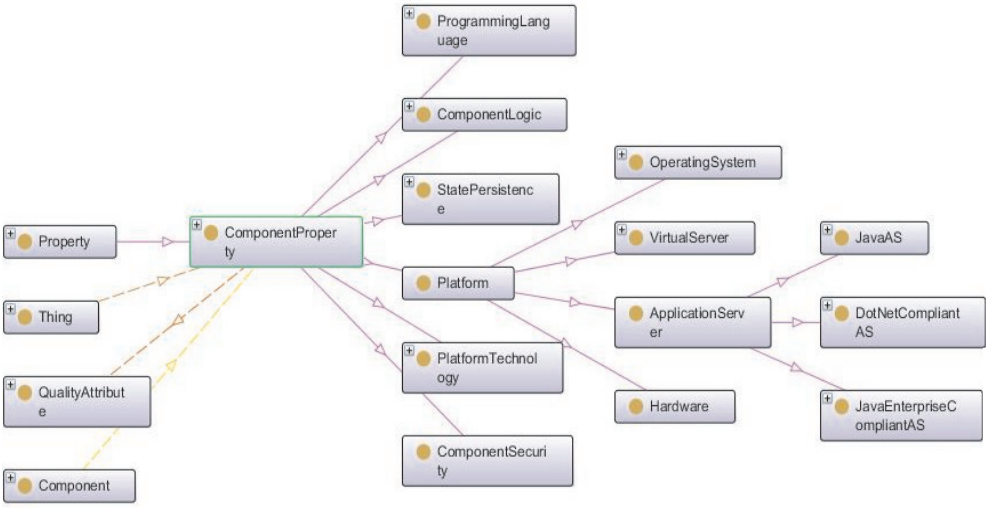

Fig. 6. Classes of component properties

3.4. ABox knowledge detailed

As indicated in the Figure 2 the specification process manages three sets of individuals and relations (ABoxes) using the common SOAROAD terminology: basic, views, and resulting Architecture ABox combining both previous and asserting additional relations concerning assignment of properties.

The basic SOAROAD ABox defines individuals of concepts belonging to particular classes to be presented as choices while specifying architectural decisions. Apart from providing a basic set of property values, it specifies relations (Fig. 7), that can be used in architecture assessment. The supports relation indicates that particular elements can be used together, e.g. JBoss (*ApplicationServer*) supports *Document.Literal* (SOAP web service

style). The *supports* property has two subproperties: *supports_fully* and *supports_partially*, that can be used to indicate possible incompatibility issues. Another way to define potentially conflicting architectural decisions is to use Conflict objects (reified multirole properties) that indicate sets of properties, which should not be used together, provide specification of conflict levels (e.g. *partially_compatible*, *incompatible*, *error_prone*) and textual description (*rationales*) . The required relation can be used to specify that one element requires another. Such assertions can be explored, while reasoning about implicit decisions, i.e. resulting from earlier assignments.

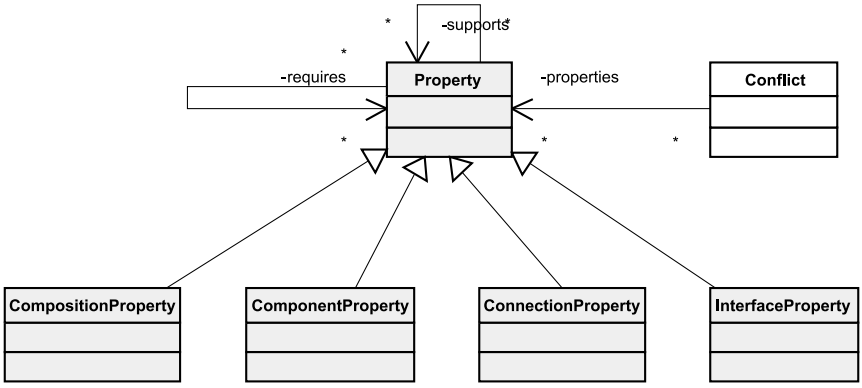

Fig. 7. Relations between properties

SOAROAD ontology is formalized in the OWL language. In consequence, the ontology should follow the Open World Assumption (OWA) to be compatible with OWL reasoners, e.g. Pellet, Fact+ or Racer.

According to OWA the following approach was adopted:

- A lack of the assertion on the property of a particular type, means that nothing is known about the assignment. For example in Figure 3 no information is provided about the hardware or operating system for the component B.
- A lack of decision is represented explicitly by an individual (constant) of a particular type, e.g. and individual OperatingSystem.not_decided can be assigned to the component B.
- Conflicting decisions of the same type can be attributed to a component, e.g. component B can be attributed with Windows and Linux properties. Such conflicts reflect, that in a certain step an alternative is envisaged. During an evaluation process (possibly supported by reasoning with the use of a separately developed set of SWRL rules) such an assertion can be indicated as non valid.
- Negative assertions about properties are represented by a special ban relation, whose object can be an anonymous individual of a selected type. For example an assertion (ComponentB, ban, IOS.anonymous) can be made, where IOS.anonymous belongs to the class IOS (operating system).

4. Conclusions

This paper describes the SOAROAD ontology and a concept of a tool that supports documenting architectures of SOA-based systems. The proposed approach addresses the problem that can be encountered during architecture assessment: to be reliable, a reasoning about architecture qualities, must have solid foundations in a knowledge related to a particular domain: architectural styles, design patterns, used technologies and products. The idea behind SOAROAD ontology is to gather expert knowledge to enable even inexperienced users performing ATAM-based architecture evaluation.

An advantage of the presented approach is that its result is a join representation of architecture views and properties attributed to design elements formalized in OWL language.

From a software engineering perspective, such centralized information resources may represent a valuable artifact, which, if maintained during the software lifecycle, can provide a reference to design decisions that can be examined later in the integration, testing and deployment phases.

On the other hand, the machine interpretable representation, constituting a graph of interconnected objects (individuals), can be processed automatically to check consistency, detect potential flaws and calculate metrics. An extensive list of metrics related to architectural design was defined in [22]. We plan to adapt them to match the structural relations in the SOAROAD ontology, as well to develop new ones.

Further plans are related to the extensions of the currently developed tool. At present its functionality is limited to building the architecture description. Our intention is to fully integrate it with the ATAM process allowing specifyed scenarios, describing sensitivity points, tradeoffs and risks.

References

[1] Archi, Archimate Modelling Tool, http://archi.cetis.ac.uk/download.html, 2011.

[2] Bianco P., Kotermanski R., Merson P., *Evaluating a Service-Oriented Architecture*. Engineering 1–91, http://repository.cmu.edu/sei/324/, 2007.

[3] Booch G., Rumbaugh J., Jacobson I., *Unified Modeling Language User Guide*. 2nd Edition. Addison-Wesley Professional, 2005.

[4] Capilla R., Nava F., Pérez S. & Dueńas J.C., *A web-based tool for managing architectural design decisions*. SIGSOFT Softw Eng Notes, 31, 4, 2006.

[5] Clark, Parsia: Pellet: *OWL 2 Reasoner for Java*, http://clarkparsia.com/pellet/.

[6] Clements P., Kazman R., Klein M., *Evaluating Software Architectures: Methods and Case Studies*. Addison Wesley Longman SEI Series In Software Engineering 368 Addison-Wesley Professional, 2001, http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/020170482X

[7] De Boer R.C., Lago P., Telea A., Van Vliet H., *Ontology-driven visualization of architectural design decisions*. 2009 Joint Working IEEEIFIP Conference on Software Architecture European Conference on Software Architecture, 82, 51–60, 2009.

[8] De Nicola A., Missikoff M., Navagli R., *A proposal for Unified Process for Ontology*. Proceedings of 16th International Conference on Database and Expert Systems Applications, DEXA 2005, Copenhagen, Denmark, 2005.
[9] Erfanian A., Shams Aliee F., *An ontology-driven software architecture evaluation method*. Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge SHARK 08 79-86, 2008, http://portal.acm.org/citation.cfm?doid=1370062.1370081.
[10] Gomez-Perez A., Fernandez-Lopez M., Juristo N., *METHONTOLOGY: From Ontological Art Towards Ontological Engineering*. Proceedings of Symposium on Ontological Engineering of AAAI, Spring Symposium Series, Stanford, 1997, 33–40.
[11] Gruninger M., Fox M.S., *Methodology for the Design and Evaluation of Ontologies*. IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing, 1995.
[12] Horrocks I., Patel-Schneider P.F, Boley H., Tabet S., Grosof B., Dean M., *SWRL: A semantic web rule language combining OWL and RuleML*. 21, May 2004, 1–22, http://www.w3.org/Suctubmission/SWRL
[13] ISO/IEC 9126. Software engineering – Product quality (ISO/IEC), ISO/IEC, 2001.
[14] ISO/IEC CD 25010.3: Systems and software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Software product quality and system quality in use models. ISO 2009.
[15] Kazman R., *ATAM: Method for Architecture Evaluation*. CMUSEI2000TR004, 2000, http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr004.pdf.
[16] Kruchten P., *An Ontology of Architectural Design Decisions*. Proceedings of 2nd Groningen Workshop on Software Variability Management Groningen NL, 2004, 55-62.
[17] Lee L., Kruchten, P., *Visualizing Software Architectural Design Decisions*. Software Architecture 5292, 2008, 359–362.
[18] The Open Group, Archimate 1.0 Specificattion, http://www.opengroup.org, 2009, s. 122.
[19] OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, W3C Recommendation 27 October 2009 http://www.w3.org/TR/owl2-syntax/#Imports.
[20] Shaw M., Garlan D., *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 123, 242, Prentice Hall, 1996.
[21] Sliwa J., Gleba K., Chmiel W., Szwed P., Glowacz A., *IOEM – ontology engineering methodology for large systems*. Proceedings of the Third international conference on Computational collective intelligence: technologies and applications, Transactions on Computational Collective Intelligence, LNCS, Springer-Verlag Piotr Jedrzejowicz, Ngoc Thanh Nguyen, and Kiem Hoang (Eds.), Vol. Part I. Springer-Verlag, Berlin, Heidelberg, 2011.
[22] Vasconcelos A., Sousa P., Tribolet J., *Information System Architecture Metrics: an Enterprise Engineering Evaluation Approach*. Electronic Journal of Information Systems Evaluation, Vol. 10, Issue 1, 2007, 91–122.
[23] Vollmer K., Gilpin M., Rose S., *The Forrester Wave™: Enterprise Service Bus*. Q2 2011, www.forrester.comcture.
[24] Wongthongtham P., Chang E., Dillon T.S., *Ontology Modelling Notations for Software Engineering Knowledge Representation*. Inaugural IEEE International Conference on Digital Ecosystems and Technologies, Cairns, Australia, February 2007