SOAROAD: An Ontology of Architectural Decisions Supporting Assessment of Service Oriented Architectures

Piotr Szwed, Pawel Skrzyński, Grzegorz Rogus and Jan Werewka
Department of Applied Computer Science
AGH University of Science and Technology
al. Mickiewicza 30, 30-059 Kraków, Poland
E-mail: {pszwed, skrzynia, rogus, werewka}@agh.edu.pl

Keywords: software architecture, ontology, SOA, ATAM, architecture assessment, architecture evaluation, enterprise architecture

Received: November 23, 2013

Enterprise architecture (EA) management has become a widely discussed approach in both industry and academia due to the inefficiency of current IT architectures to cope with rapid changes in business environments. On the other hand Service Oriented Architecture (SOA) is widely accepted as a state of the art approach to the design and implementation of enterprise software. However software design and development according to SOA paradigm is a complex task, often integrating various platforms, technologies, products and design patterns. Hence, it arises a problem of early evaluation of a software architecture to detect design flaws that might compromise expected system qualities. Decisions related to software architecture have a great impact on the business value of a software product under development and influence the software company competitiveness. Usually, a software architecture is developed by a company team, whose experience is limited to narrow set of solutions and technologies. This is a motivation for developing a methodology for the assessment of architectural solution that can be performed in more independent way. Such assessment requires extensive knowledge gathering information on various types of architectural decisions, their relations and influences on quality attributes. In this paper SOAROAD (SOA Related Ontology of Architectural Decisions) is described, which was developed to support the evaluation of architectures of information systems based on SOA approach. The main goal of the ontology is to provide constructs for documenting architecture. However, it is designed to support future reasoning about architecture quality and fulfilling the non functional system requirements such as scalability, ease of maintenance, reuse of software components etc. The last important reason is building a common knowledge base. When building the ontology Architecture Tradeoff Analysis Method (ATAM) was adopted which was chosen as a reference methodology of architecture evaluation.

Povzetek: Opisana je ontologija, ki omogoča evaluacijo arhitektur za informacijske sisteme SOA.

1 Introduction

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. Within the last decade a number of concepts related to SOA have been developed, including ESB (Enterprise Service Bus), web services, design patterns, service orchestration and choreography and various security standards. Due to the fact that there are many technologies that cover the area of SOA and the fact that SOA is not related to any specific technology, the development and evaluation of SOA compliant architectures is especially interesting and problematic.

SOAROAD has been designed as a methodology for the assessment of software architectures developed according to SOA principles. During system development several stages and corresponding architecture evaluation goals can be identified. The first stage is related to the formulation of a strategy for a new system development or an integration of existing software. The next stage consists in proposing competitive architectural approaches and assessing them with respect to selected quality attributes. The third stage has as an input the assumed system architecture and aims at identifying requirements (usually expressed as scenarios) and determining risks and costs for achieving the assumed scenario responses. This step can be referred as *early* architecture evaluation. The last stage, that can be considered as a *late* architecture evaluation, is related to software verification and validation resulting in the specification of test

This paper is based on P. Szwed, P. Skrzyński, G. Rogus and J. Werewka *Ontology of architectural decisions supporting ATAM based assessment of SOA architectures* published in the proceedings of the 3rd International Workshop on Advances in Semantic Information Retrieval (part of the FedCSIS'2013 conference).

cases used for TDD (Test Driven Development) or BDD (Behaviour Driven Development) approach.

In this paper we focus on the third stage of architecture evaluation. SOAROAD has been designed as a methodology for the assessment of software architectures developed according to SOA principles. It is based on the Architecture Tradeoff Analysis Method (ATAM) [18, 7], which is a mature, scenario-based, early method for architecture assessment. ATAM defines a quality model and an organizational framework for evaluation process. Expected system qualities are represented as mappings between scenarios and quality attributes. System architecture being an input for ATAM is expressed in 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. In the evaluation process, the first information on expected system qualities, architectural approaches and decisions is collected from architecture documentation and interviews with stakeholders, then a team of experts analyze selected properties of components to identify sensitivity points and evaluate risks. A limitation of the ATAM method is that it depends on experts knowledge, perception and previous experience. It may easily happen that an inexperienced evaluator overlooks some implicit decisions and risks introduced by them.

In the SOAROAD approach the very basic set of ATAM terms used to describe architecture is enriched by including common terminology and relationships between concepts related to various aspects of service oriented architecture design and development. The gathered knowledge, formalized as an ontology, facilitates performing an assessment in more exhaustive manner, helping to ask questions, revealing implicit design decisions and obtaining more reliable results.

The contribution of the paper is a proposal of a SOAROAD ontology as a tool supporting scenario based assessment of systems following a service-orientation paradigm and service design, development and deployment.

The paper is organized as follows. In Section 2 related works are discussed. Section 3 gives an overview of ATAM methodology. Section 4 introduces a concept of ontology application in architecture evaluation. Section 5 provides the ontology description. Section 6 discusses an example of SOAROAD methodology application. Section 7 summarizes the paper and presents conclusions together with future works planned.

2 Related works

Architecture evaluation has attracted many researchers and practitioners during the last 20 years. A survey paper on this topic [26] lists 37 methods of architecture evaluation, classifying them according to two dimensions: location in the software lifecycle (early vs. late) and element being analyzed (system architecture, isolated architectural style or a design pattern). The paper suggests that scenario-based methods, including SAAM [20] and ATAM [18, 7] can be considered as a mature, reliable and easy to implement in practical situations.

There are several reports on successful applications of ATAM for assessment of a battlefield control system [19], wargame simulation [17], product line architecture [10], control of a transportation system [4], credit card transactions system[24] and a dynamic map system [32]. Recently, a few extensions of ATAM were proposed, including a combination with the Analytical Hierarchy Process [36] and APTIA [21].

Despite the fact that the area of enterprise architecture (EA) and service oriented architecture (SOA) has been gaining significant attention there have not been much research on SOA architecture assessment.

Song and Song [30] proposed EA institutionalization processes and its metric based assessment for implemented EA based on the currently available EA frameworks. In the EA processes, we define institutionalization strategies specific to an organization's goals, target architecture based on their baseline architecture, and transition plan for institutionalization. The assessment is based on changes made on existing architecture which describes the current or as-is state of an enterprise. To describe the baseline architecture they suggest organizing information structure according to the architectural views as in ANSI/IEEE Standard 1471-2000 [13].

Javanbakht, Pourkamali, Feizi [16] observed that in some enterprises, particularly in developing countries, baseline is not a suitable basis for creating target architecture and they proposed improvement and correction of organizational architecture by using enterprise architecture maturity. They used multifactor systems to provide a practical method for the assessment of any given organization and making accurate decisions on the improvement or redesign of its architecture based on missions, goals and restrictions of the organization. With the use of their method they claimed that the enterprise architectures can be assessed and an accurate decision about the development of the enterprises can be made based on its mission.

Jange and Medling [23] tried to address the problem of cost benefit ratio of EA with a qualitative research design. They conducted a series of semi structured interviews with industry experts on enterprise architecture in order to identify classes of EA goals, corresponding EA frameworks adoption to achieve those goals and employed EA benefit assessment approaches. Their findings point to, among others, a fairly stable set of EA goals that shift over time and EA frameworks that lack modularity and adjustment capabilities to easily customize towards these goals.

Zhou and Zhang [37] presented an architecture-centric assessment approach for model evaluation over reference architecture to quantitatively estimate architecture maturity and quality. They selected a nine-layer (S3) SOA solution stack as reference architecture, and introduced the neces-

sary mathematical definitions and formulation. The baseline for such an assessment is a model template composed of S3 solution patterns. A template is the starting point of creating a design model.

There has been interesting research performed on the analysis of the composition of services [6]. The authors observed and grouped common service composition techniques into six solution patterns with distinct characteristics of their integration intermediary. Their effort also can be used as a base to develop better solution templates to include architectural building blocks level interactive patterns into solution template creation.

The application of ontologies to provide a systematic and formal description of architectural decisions was first proposed by Kruchten in [22]. The ontology distinguished several types of decisions that can be applied to software architecture and its development process. 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 model of architecture, 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 a development of tools enabling the edition and graphical visualization of design decisions, often in a collaborative mode, e.g. [5, 8, 25].

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 classification of concepts and their relations (commonly referred as TBox), we attempt to gather in the proposed ontology also facts (commonly referred as ABox) constituting ready to use dictionaries of decisions (properties of architectural design) and the knowledge about their relations reflecting current state of the art for SOA technologies.

3 ATAM Overview

The goal of software architecture evaluation methods is to assess whether a system meets or will meet certain requirements concerning quality characterized as *quality attributes*. A standardized list of quality attributes is published in ISO/IEC 9126-1 norm [14], which enumerates six groups of quality attributes: Functionality, Reliability, Usability, Efficiency, Maintainability and Portability. This set was extended in the superseding norm ISO/IEC 25010[15] to 8 groups by adding Compatibility and Security. Many of the quality attributes were known elsewhere under different names, e.g. Efficiency as Performance, Changeability (sub-attribute of Maintainability) as Modifiability, etc.

Architecture evaluation methods may bring the greatest benefits to software development if applied early in the software lifecycle, as identified flaws in system design can be corrected at a lower cost [26]. Typically, an assessment is conducted based on the specification of the software architecture (architectural views) and use other sources of information, such as interviews with various stakeholders including owners, future users, architects and development teams. At an early development stage it is difficult to give the ultimate answer whether a particular quality attribute can or cannot be assured. Therefore, assessment methods aim at estimating such characteristics as a risk or cost.

Identified high risk to achieve a quality attribute can trigger mitigation actions which consist in revising the design and changing the design decisions. However, it should be emphasized that even after changes and corrections are applied some acceptable residual risks can still be present because the estimated effort required to remove them exceed expected losses.

ATAM (Architecture-based Tradeoff Analysis Method) was developed at the Software Engineering Institute (SEI) in 2000 [18], [7] as a successor of the SAAM method [20].

The method aims at evaluating architectural decisions against specific quality attributes and detecting:

- *risks* – architectural decisions that may cause problems to assure some quality attributes,
- *sensitivity points* – decisions related to components or their connections that are critical for achieving required level of quality attribute,
- *tradeoffs* – decisions of increasing one quality attribute with a negative impact on the others.

ATAM provides evaluations based on the requirements expressed as *scenarios* that are elicited and assessed in a formal process divided into phases and steps.

ATAM uses a quality model called the *utility tree*. At the root of the utility tree, an abstract concept *Utility* is placed. Its child nodes are annotated with general quality attributes, e.g. these specified in the ISO norm (performance, reliability, security, modifiability, etc.); at the next level they can be decomposed into more specific attributes, and finally, scenarios are placed at leaves. Both quality attributes present in the utility tree and scenarios are elicited from various stakeholders and represent their point of view on expected system qualities.

According to ATAM, the architecture assessment process is a group effort of various stakeholders involved in system development. It deploys typical group techniques such as brainstorming, assigning priorities and voting. The course of evaluation is divided logically into four phases including nine steps:

1. *Presentation*: (1) presentation of the ATAM method, (2) business drivers and (3) the assumed software architecture.

2. *Investigation and Analysis*: (4)identification of architectural approaches, (5) generation of quality attribute tree and (6) an analysis of the architectural approaches.

3. *Testing*: (7) brainstorming and the prioritization of scenarios, (8) repeated analysis of the architectural approaches with reference to high priority scenarios.

4. *Reporting*: (9) presenting the results of the analysis: risks, sensitivity points and tradeoffs.

4 The concept of SOAROAD ontology application

ATAM has many obvious benefits: it precisely defines the quality model based on a utility tree, enumerates the expected outcomes, indicates the participants and provides an organizational framework for conducting the evaluation. Nevertheless, due to its generic character, the method can cause problems related to collecting and representing information that can be used for an architecture assessment. The identification of key design decisions (properties) that should be considered is up to experts' knowledge and experience. In the case of inexperienced evaluators, some key architectural decisions strongly influencing the system qualities can be easily overlooked. Gathering knowledge related to leading technologies, e.g. web services, business process execution environments, databases, semantic web as a support to ATAM would be beneficial for the efficiency and reliability of the evaluation.

The proposed approach consists in collecting and formalizing this knowledge as an ontology. SOAROAD (SOA Related Ontology for Architectural Decisions) ontology has four main goals, it should:

1. provide a comprehensive description of architectural views, i.e. components and their connections;

2. gather a domain knowledge providing a unified vocabulary related to SOA and enterprise architecture;

3. help to ask question about various properties of architectural design and decisions;

4. be capable to represent assignments of properties relevant to SOA compliant technologies to elements of system architecture.

It was assumed that the ontology would follow a foundational model (ontology skeleton) described later in the section 5.1 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 (set of classes, their attributes and relations) and ABox (individuals, values of their attributes and relationships). The types of elements appearing in architectural views are classified in the TBox. Concrete elements, e.g. those appearing in the diagrams of architectural views, are represented as individuals in an ABox. The ontology describes types of design decisions (properties) as classes, whereas their values as individuals that can be directly assigned to elements of architectural views or linked to form trees. Such approach is more flexible, than e.g. a simplistic model of *key-value* pairs assigned to components, where a *key* would correspond to a decision type, and a *value* to a concrete decision.

The concept of the ontology application is presented in the Fig. 1 (thick lines indicate data flows and thin arrows describe import relations among ontologies). The process of building 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 the support of dedicated import tools converting ArchiMate [33, 34] models of Archi editor [2] or UML [27], e.g. from VisualParadigm. For clarity, the figure shows only one import tool that converts the ArchiMate model into *Architecture views ABox* encoded in OWL language.

A web based tool supporting architecture description uses the classes and individuals defined in the *SOAROAD ontology Domain Description TBox* and *SOAROAD Architectural decisions ABox* 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.

These questionnaires are dynamically generated from the ontology content by transforming relevant items to XML representation and then applying XSLT transforms to give them a visual appearance. Users selections in questionnaires after feeding them to a web server are converted into assertions in *Detailed Architecture ABox* ontology stored at the server side. For this purpose we use Jena library and TDB [1] as the storage system. The resulting *Detailed Architecture ABox* refers elements of *Architecture views ABox* and individuals defined in SOAROAD ontology (merging two input ontologies and asserting additional relations). This ontology serves as a detailed architecture documentation within a software development project. It can be examined either manually or with use of automated tools.

It should be mentioned that for large projects realized by multiple teams at distant locations, maintaining a centralized repository documenting software architecture and architectural decisions can be considered as a key factor for project success. In many cases, independent teams make many implicit decisions that may influence interoperability, performance, modifiability and other quality attributes. Collecting detailed information by the suggested in ATAM interviews is more time consuming and less exhaustive than filling in questionnaires driven by an ontology content.

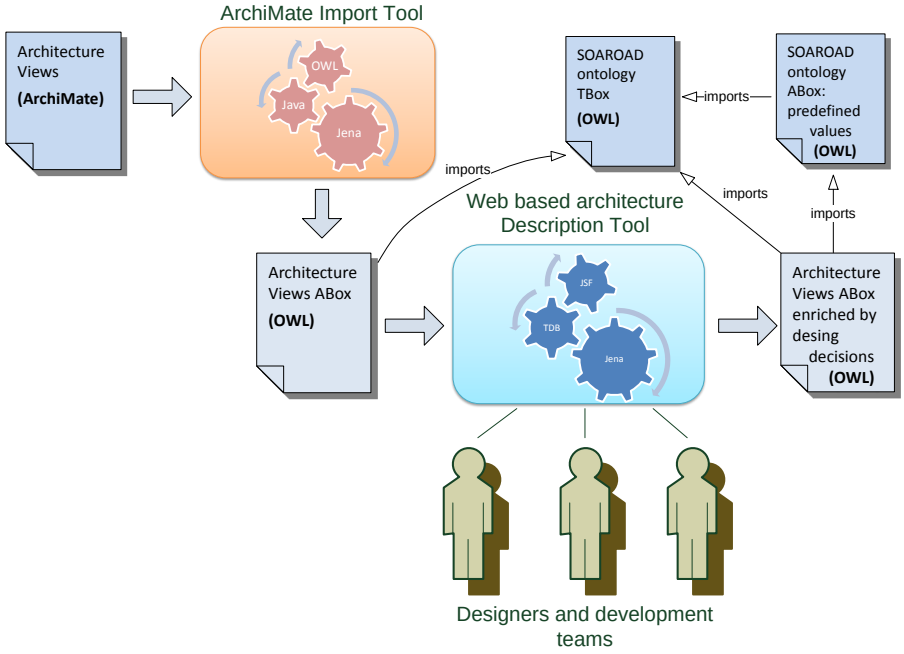

Figure 1: A concept of application of SOAROAD ontology

5 Ontology description

Ontology engineering methodologies [11, 12, 29] 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 which consists 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 results of formalization. The main outcome of the specification phase is the foundational model described in section 5.1. During the conceptualization step, we manually gathered and analyzed information related to service oriented architectures, technologies, architectural approaches, design patterns, etc. originating from various sources: books, technical papers, reference manuals and Internet resources.

The ontology was populated with the information during the formalization phase by translating intermediate textual description into OWL constructs. For this purpose a small software tool using Jena [1] library was developed. The resulting ontology content is described in section 5.2.

5.1 Foundational model of software architectures

The basic model of software architecture used in ATAM [3] defines it after [28] as a set of components and linking them connections. We extend this simplistic model by defining *Interfaces* and *Functions* of components as presented in Fig. 2. 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 and query granularity.

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. a distribution of queries among 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 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, which increases modifiability and a granularity of queries, has an impact on performance. This kind of knowledge can be expressed as *influences* relations.

Architectural decision is an assignment of a property value to a component, interface, connection or a composition. In this context the terms *property* and *architectural*

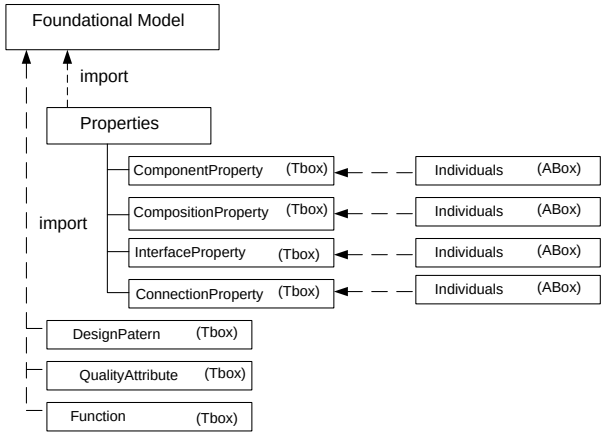

Figure 3: Structure of SOAROAD ontology

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 a 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 foundational model adopts a 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 MVC design pattern – pattern, which requires the identification of a components playing the roles of a Model (typically a 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 correspond to software developed modules; infrastructure components provide such supporting functions, as message queuing or service registry.

5.2 The ontology content

SOAROAD ontology, provides a knowledge about software architecture, its structure, components, connections and required properties in the context of the SOA paradigm. It consists of 110 classes, 9 object properties and 105 individuals.

The structure of SOAROAD ontology is shown in (Fig. 3). The Foundational Model presented earlier in Fig. 2 forms the ontology skeleton. In the conceptualization phase, the skeleton was extended by defining subclasses of classes marked in gray: various types of properties (*ComponentProperty*, *ConnectionProperty*, *CompositionProperty* and *InterfaceProperty*), functions, design patterns and quality attributes.

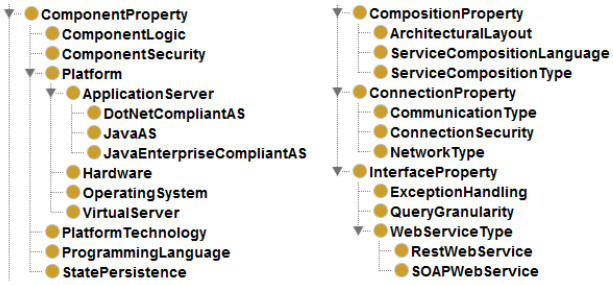

Figure 4: Classes of component properties

For each property, that can be treated as a class of design decision, a number of individuals (corresponding to decision values) is defined. They can be selected in assignments, e.g. *JavaEECompliantAS* (a subclass of *ComponentProperty*) has several predefined individuals: *JBoss*, *Glassfish*, *WebLogic*, *Web-Sphere*, *ColdFusion*, etc.

ComponentProperty class defines various properties and design decisions, which can be assigned to components (Fig. 4). Software architect preparing ATAM evaluation should consider them as an exhaustive list of questions related to important issues in SOA architectures. Examples of such properties are *Platform* (*Hardware*, *OperatingSystem*, *ApplicationServer*), *PlatformTechnology*, *ProgrammingLanguage*, *ComponentLogic* and *ComponentSecurity*.

Example ontology assertions related to component properties are presented in Table 1 and Table 2. A property (an ontology class) is followed by property values (individuals in the ontology) put in parentheses.

ConnectionProperty subsumes the *CommunicationType*, *ConnectionSecurity* and *NetworkType*. The class *ComunicationType* has two individuals: *ComunicationType.asynchronous* and *ComunicationType.synchronous*. *ConnectionSecurity* has individuals representing various security technologies SSL, VPN, WS_Security.

CompositionProperty is a superclass for *ArchitecturalLayout*, *ServiceCompositionLanguage*, *ServiceCompositionType*. *ArchitecturalLayout* defines types of application structure. Its individuals are: *LayeredArchitecture, P2P, ServiceComposition* and *SpokeAndHub*.

ServiceCompositionLanguage defines languages (*BPEL*, *CDL* or *not_defined*) and *ServiceCompositionType* with individuals: *choreography* and *orchestration*.

Class *InterfaceProperty* has subclasses *ExceptionHandling* (defining exception handling method), *QueryGranularity* (granularity level of of interface functions), *WebServiceType* (type of communication protocol: *SOAPWebService* or *RESTWebService*).

Apart from defining design decisions, the ontology specifies functions of components. Their list is rather related to infrastructure components. Class Function contains classes of entities such as: *Routing*, *MessageMapping*, *ProtocolSwitch*, *MediationService*, *MessageValidation*, *AuditFunction*, *DatbaseIntegration*, etc.

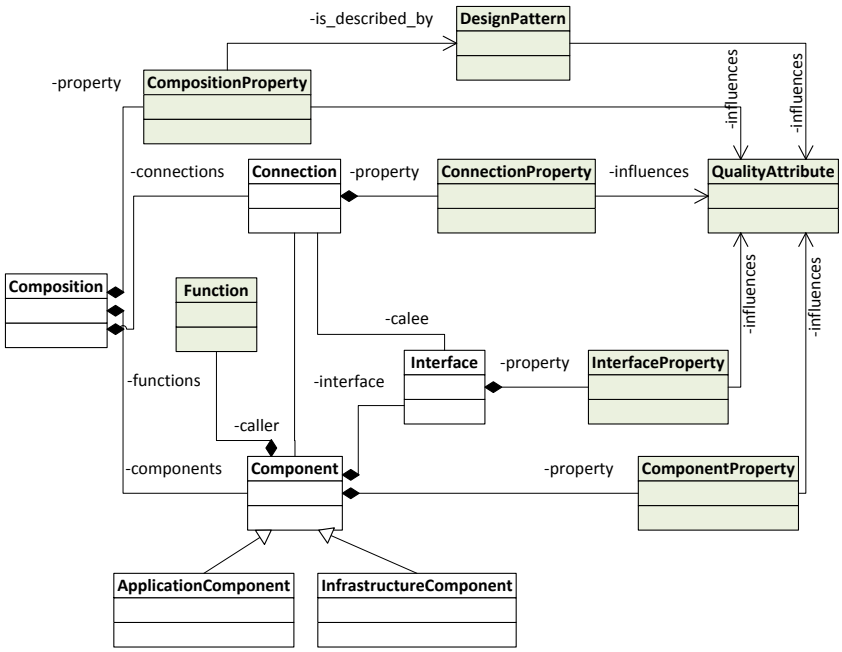

Figure 2: Foundational model of software architecture and its properties

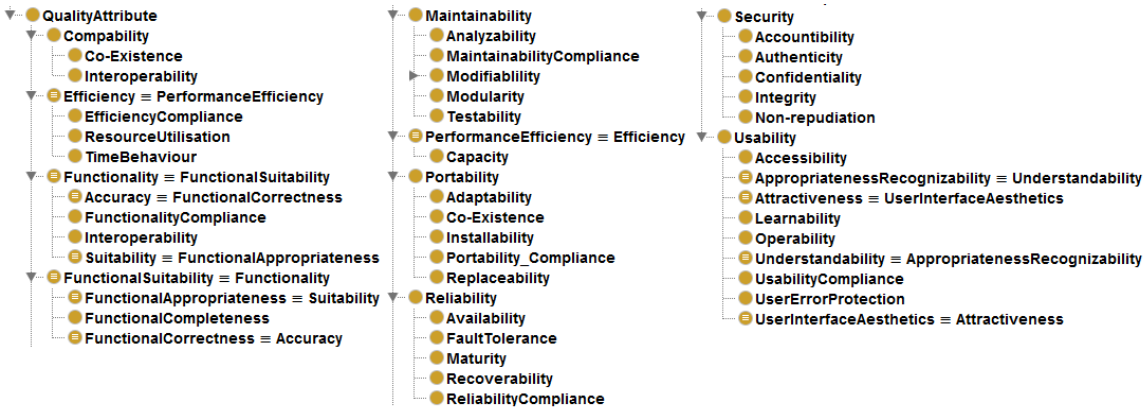

Figure 5: The tree of quality attributes (according to ISO/IEC 9126 and ISO/IEC 25010)

Table 1: Component properties

Property (values)	Description
PlatformTechnology (CORBA, EJB, JINI, RMI)	Set of technologies used on the platform.
ComponentLogic (flexible, fixed, rulebased)	Specifies an approach the component logic implementation.
Platform	Defines the component platform. Has several subclasses: ApplicationServer, Hardware, OperatingSystem and VirtualServer
ProgrammingLanguage (Cpp, Java, Ruby, PHP, Erlang, Python, C, C_sharp)	Define programming language used to implement a component.
StatePersistence (Stateless, Statefull)	Specifies whether a component saves internal data during and in between calls of operations on the client's behalf.

The ontology provides a taxonomy of quality attributes. Quality attribute is a nonfunctional characteristic of a component or a system. It represents the degree to which software possesses a desired combination of properties, which are defined by means of externally observable features of software systems. Some of the attributes are related to the overall system design, while others are specific to run-time or design time. Quality attributes can be categorized into two broad groups: attributes that can be directly measured (e.g. performance) and attributes that can be indirectly measured (e.g., usability or maintainability). In the latter category, attributes are divided into subcharacteristics.

SOAROAD ontology defines 30 quality attributes including both terms defined in software quality model by the ISO/IEC 9126-1 norm [14] and those arising directly from requirements to architectures formulated in the SOA man-

Table 2: Properties describing platform (subclasses of *Platform*).

Property (values)	Description
ApplicationServer	Subclass of Platform. Defines an application server on which a component is deployed, can have such attributes, as: version (string), vendor (string)
JEECompliantAS (TomEE, Glassfish, JBoss, Interstage, JOnAS, Geronimo, SAPNeatWeaver, WebSphere, Resin, ColdFusion, WebLogic)	Subclass of ApplicationServer dedicated to JEE compliant components.
DotNetCompliant-AS (AppFabric, IIS, TNAPS, Base4, Mono)	Subclass of ApplicationServer; its individuals define products for .NET enviroment
JavaAS (Jetty, Enhydra, iPlanet)	Application servers for Java environment
Hardware	Subclass of Platform. Used to specify a hardware configuration on which the component is deployed. Attributes: memory (double), processor (string), number_of_cores (int)
OperatingSystem (Windows, Unix, Linux, iOS, Android, Bada, Blackberry)	Subclass of Platform. Defines types of operating systems on which a component is executed. Attributes: version (string), vendor (string), product (string)
VirtualServer (no, yes)	Subclass of Platform. Specifies whether a component is deployed on a virtual server

ifesto . Examples of classes belonging to the first group (see Fig. 5) are: *Functionality*, *Reliability*, *Usability*, *Efficiency*, *Maintainability* and *Portability*. The example of classes originating from SOA manifesto are *ServiceAutonomy*, *PlatformIndependency*, *LooseCoupling*, *Modularity*, *OpenStandardAdoptation*, *BusinessAgility* etc.

When designing an applications to meet quality requirements, it is necessary to consider a potential impact of design properties on various quality attributes. SOAROAD ontology defines *influences* object property to this kind of relation.

A design pattern can be seen as a structure build of components of particular types, defining their roles and relations among them together with a set of restrictions on their usage. Design patterns do not change the functionalities of a system but only the organization or structure of those functionalities. One of the most important benefits of using design patterns is that they constitute standardized software building blocks with a well defined influence on quality attributes. In SOAROAD ontology the class *DesignPattern* has 56 subclasses representing patterns dedicated to SOA architecture. The examples of subclasses are: *EnterpriseServiceBus*, *EventDrivenMessaging*, *Orchestration*. The relation *is_described_by* links a particular *CompositionProperty* to one of the defined design patterns.

6 Example

We illustrate the proposed approach on an example of a small system aimed at publishing and browsing of free of charge announces. The diagram in Fig. 6 gives the system architecture specified in ArchiMate language. As it can be noticed, two layers: application and technology are presented. In the application layer several system components are distinguished: *Data Base* with the *SQL interface*, three Java beans: *Announcement JPA* (Java Persistence API), *Announcement Business Logic* and a *Facade* providing *Announcement WS* – web service based interface. The last component of the application layer visible on the diagram is *Announcement JSF Presenter* being responsible for presentation and interaction with end users. It plays here the role of web service consumer. The components are packaged as three artifacts: *ANN_DB* (PostgreSQL), *ann.ear* and *ann_pres.war* and deployed at three separate servers (technology layer nodes) linked with two connections: *JDBC* and *WS Presenter*.

The above specification is an input for *ArchiMate Import* tool (indicated in Fig. 1) that transforms it into *Architecture Views ABox*. Fig. 7 gives and excerpt of this ontology (node marked with boldlines). We focus on three elements application layer: *Announcement Facade*, *WS* interface and accessing it *JSF presenter*. Following the foundational model that encompasses connections and their properties, the *WS Presenter Connection* was also included. The remaining elements of of ArchiMate specification are converted into properties.

The tool supporting architecture description allows to assign various properties (architectural decisions) to ontology individuals corresponding to components and connections of the software architecture. In the presented example:

- *Announcement Facade* is deployed on Intel Xeon 2.13 GHz machine running Ubuntu 10.4 system and GlassFish application server.
- *Announcement WS* is a SOAP web service with low query granularity and exception handling based on soap faults.
- *WS Presenter Connection* is asynchronous, uses SSL based protection mechanism and 10Gb network.
- *Announcement JSF Presnter* is deployed on JBoss application server and is stateless.

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

The SOAROAD ontology specifies additional relations (Fig. 8) 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*,

http://www.soa-manifesto.org/

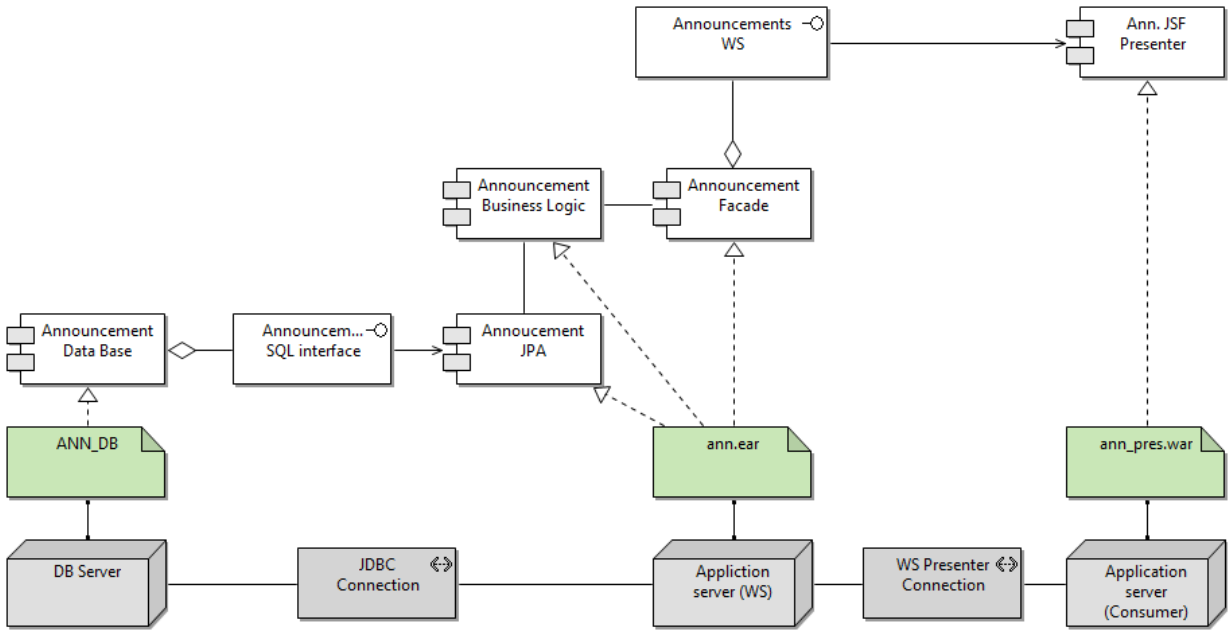

Figure 6: Announcement system expressed in Archimate language.

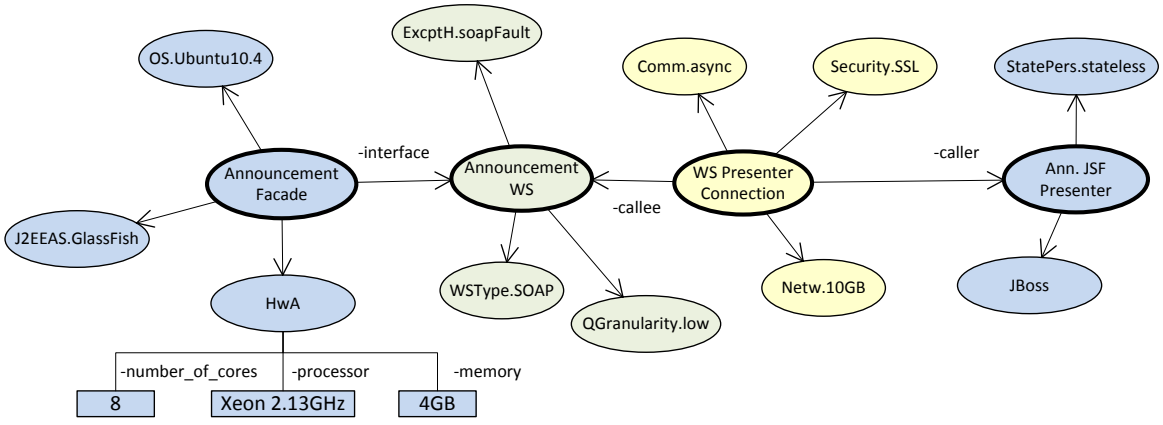

Figure 7: ABox describing the architecture of the announcement system. Elements of an architectural view (marked with boldlines) are assigned with design decisions (individuals of classes defined in the ontology)

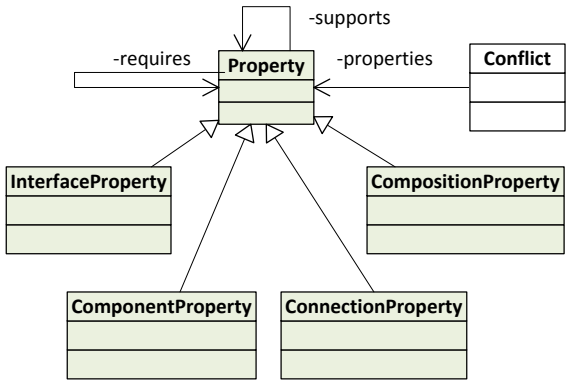

Figure 8: Relations between properties

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.

The SOAROAD ontology is formalized in the OWL language. In consequence, it 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 property of a particular type, means that nothing is known about the assignment. For example in Fig. 7 no information is provided about the hardware or operating system for *Annotations JSF Presenter*.

- 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 *Annotations JSF Presenter*.

- Conflicting decisions of the same type can be attributed to a component, e.g. *Annotations JSF Presenter* 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 (*Annotations JSF Presenter*, *ban*, *IOS.anonymous*) can be made, where *IOS.anonymous* belongs to the class *IOS* (operating system).

7 Conclusion

This paper describes the SOAROAD ontology and a concept of a tool that supports the 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 experts knowledge to enable even inexperienced users performing ATAM-based architecture evaluation. An advantage of the presented approach is that its result is a joint representation of architecture views and properties attributed to design elements formalized in OWL language.

From a software engineering perspective, such centralized information resource may represent a valuable artifact, which, if maintained during the software lifecycle, can provide 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 [35]. We plan to adapt them to match the structural relations in the SOAROAD ontology, as well to develop new ones.

Another direction that is at present researched is an application of fuzzy reasoning to evaluate quality attributes. We use fuzzy Mamdani rules encoded in SWRL language defining influence of selected design decisions on quality. The approach taken follows the idea presented in [31].

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 ATAM process allowing specifying scenarios, describing sensitivity points, tradeoffs and risks.

References

[1] Jena - a semantic web framework for java.

[2] Archi, archimate modelling tool, 2011. [Online; accessed 23-June-2012].

[3] P. Bianco, R. Kotermanski, and P. Merson. Evaluating a service-oriented architecture. Technical Report CMU/SEI-2007-TR-015, Carnegie Mellon, September 2007.

[4] N. Bouck'e, D. Weyns, K. Schelfthout, and T. Holvoet. *Applying the ATAM to an Architecture for Decentralized Control of a Transportation System*, volume 4214, pages 180–198. Springer, 2006.

[5] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas. A web-based tool for managing architectural design decisions. *ACM SIGSOFT Software Engineering Notes*, 31(5), 2006.

[6] Y.-C. Chang, P. Mazzoleni, G. A. Mihaila, and D. Cohn. Solving the service composition puzzle. *IEEE SCC*, 2:387–394, 2008.

[7] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, 2001.

[8] R. C. de Boer, P. Lago, A. Telea, and H. van Vliet. Ontology-driven visualization of architectural design decisions. In *WICSA/ECSA*, pages 51–60. IEEE, 2009.

[9] A. Erfanian and F. S. Aliee. An ontology-driven software architecture evaluation method. In *Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge*, SHARK '08, pages 79–86, New York, NY, USA, 2008. ACM.

[10] S. Ferber, P. Heidl, and P. Lutz. *Reviewing product line architectures: Experience report of ATAM in an automotive context*, volume 2290, pages 364–382. Springer, 2001.

[11] M. Fernandez-Lopez, A. Gomez-Perez, and N. Juristo. Methontology: from ontological art towards ontological engineering. In *Proceedings of the AAAI97 Spring Symposium*, pages 33–40, Stanford, USA, March 1997.

[12] M. Gruninger and M. S. Fox. Methodology for the design and evaluation of ontologies. In *International Joint Conference on Artificial Inteligence (IJCAI95), Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[13] IEEE. IEEE standard 1471-2000, ieee recommended practice for architectural description of software-intensive systems, 2000.

[14] ISO/IEC. Software engineering – product quality, ISO/IEC 9126-1. Technical report, International Organization for Standardization, 2001.

[15] ISO/IEC. 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. Technical report, International Organization for Standardization, 2009.

[16] M. Javanbakht, M. Pourkamali, and F. M. Derakhshi. A new method for enterprise architecture assessment and decision-making about improvement or redesign. *Proceedings of the Fourth International Multi-Conference on Computing in the Global Information Technology*, pages 69–76, 2009.

[17] L. G. Jones and A. J. Lattanze. Using the architecture tradeoff analysis method to evaluate a wargame simulation system: A case study. *Technical Report CMUSEI2001TN022 Software Engineering Institute Carnegie Mellon University Pittsburgh PA*, (December):33, 2001.

[18] Kazman. Atam:method for architecture evaluation. *CMUSEI2000TR004*, 2000.

[19] R. Kazman, M. Barbacci, M. Klein, J. Carriere, and S. G. Woods. Experience with performing architecture tradeoff analysis. *Proceedings of the 21st international conference on Software engineering ICSE 99*, pages 54–63, 1999.

[20] R. Kazman, L. Bass, G. Abowd, and M. Webb. *SAAM: a method for analyzing the properties of software architectures*, volume 16pp, pages 81–90. IEEE Comput. Soc. Press, 1994.

[21] R. Kazman, L. Bass, and M. Klein. The essential components of software architecture design and analysis. *Journal of Systems and Software*, 79(8):1207–1216, 2006.

[22] P. Kruchten. *An ontology of architectural design decisions in software intensive systems*, pages 54–61. Citeseer, 2004.

[23] M. Lange and M. Jan. An experts' perspective on enterprise architecture goals, framework adoption and benefit assessment. *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference Workshops*, pages 304–313, 2011.

[24] J. Lee, S. Kang, H. Chun, B. Park, and C. Lim. Analysis of VAN-core system architecture- a case study of applying the ATAM. In *Proceedings of the 2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, SNPD '09, pages 358–363, Washington, DC, USA, 2009. IEEE Computer Society.

[25] L. Lee and P. Kruchten. *Visualizing Software Architectural Design Decisions*, volume 5292, pages 359–362. Springer-Verlag, 2008.

[26] B. Roy and T. C. N. Graham. Methods for evaluating software architecture : A survey. *Computing*, 545(2008-545):82, 2008.

[27] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[28] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*, volume 123. Prentice Hall, 1996.

[29] J. Sliwa, K. Gleba, W. Chmiel, P. Szwed, and A. Glowacz. IOEM - ontology engineering methodology for large systems. In P. Jedrzejowicz, N. T. Nguyen, and K. Hoang, editors, *ICCCI (1)*, volume 6922 of *Lecture Notes in Computer Science*, pages 602–611. Springer, 2011.

[30] H. Song and Y.-T. Song. Enterprise architecture institutionalization and assessment. *Proceedings of the 9th IEEE/ACIS International Conference on Computer and Information Science*, pages 870–875, 2010.

[31] P. Szwed. Application of fuzzy ontological reasoning in an implementation of medical guidelines. In *Human System Interaction (HSI), 2013 The 6th International Conference on*, pages 342–349, 2013.

[32] P. Szwed, I. Wojnicki, S. Ernst, and A. Glowacz. Application of new ATAM tools to evaluation of the dynamic map architecture. In A. Dziech and A. Czyżewski, editors, *Multimedia Communications, Services and Security*, volume 368 of *Communications in Computer and Information Science*, pages 248–261. Springer Berlin Heidelberg, 2013.

[33] The Open Group. Archimate 1.0 specificattion, 2009.

[34] H. Van Den Berg, H. Bosma, G. Dijk, H. Van Drunen, J. Van Gijsen, F. Langeveld, J. Luijpers, T. Nguyen, R. Oosting, Gerand Slagter, and et al. ArchiMate made practical. *Work*, 2007.

[35] A. Vasconcelos, P. Sousa, and J. Tribolet. Information system architecture metrics: an enterprise engineering evaluation approach. *The Electronic Journal Information Systems Evaluation*, 10(1):91–122, 2007.

[36] P. Wallin, J. Froberg, and J. Axelsson. Making decisions in integration of automotive software and electronics: A method based on ATAM and AHP. *Fourth International Workshop on Software Engineering for Automotive Systems SEAS 07*, pages 5–5, 2007.

[37] N. Zhou and L.-J. Zhang. Analytic architecture assessment in soa solution design and its engineering application. *Proceedings of the IEEE International Conference on Web Services*, pages 807–814, 2009.