Towards an Architectural Element Recommender System

(March, 2015 version)

Balwinder Sodhi
Dept. of Computer Science and Engineering
Indian Institute of Technology Ropar, PB 140001, India
Email: sodhi@iitrpr.ac.in

***Abstract*—When building a software system, one of the critical steps is identifying suitable architectural *elements* which realize the *abstractions* comprising the system. In several scenarios there are several alternatives possible for realizing an abstraction, each of which has different functional and non-function properties. For instance, a data store (abstraction) can be realized using any of the several RDBMS and NoSQL database products (elements).**

As the size and complexity of a software system increases, the process of mapping abstractions onto elements becomes difficult and error prone if done manually. In this paper we propose semantic computing based *Architectural Element Recommender System* (AERS). Its goal is to assist an architect in analyses and mapping of abstractions to elements, and deducing additional facts about elements and abstractions. The AERS harvests unstructured information about architectural elements available from different sources and transforms it into useful semantic knowledge which can be queried via a suitable interface. Details of an implementation for the proposed AERS has also been discussed. In this paper design of an early prototype of the idea has been described. A limited verification of efficacy and correctness of the proposed AERS has been done by querying it for scenarios whose solutions are well known.

I. Introduction

Architecture design and finding solution for software design problems in general is a creative process. Skills and past experience of an individual has a significant influence on the quality of solutions that he/she produces for architecture/design tasks. A high level sketch of typical path adopted by many architects to design a solution is:

1) Search the individual, organization or community memory for similar problems handled in the past and recall their solution. One or more candidate solution outlines may come up.
2) From the candidate solution outlines obtained in previous step, choose suitable *abstractions*[1] and create structure of a solution to the problem at hand.
3) Identify suitable *elements*[2] to realize abstractions that comprise the solution determined in the preceding stage. For instance, Apache Tomcat™, MicroSoft IIS™or Red Hat JBoss™may be the alternatives to choose from when realizing an application server. Similarly, MySQL™(RDBMS engine) or MongoDB™(NoSQL engine) may viable options for realizing a data store abstraction.

Elements using which architectural abstractions can be realized are the basic building blocks usable in a variety of solutions. An *element* in our context could be a component, connector, module, framework or even an execution runtime. In practice, one has to analyse and assess several competing alternative elements (e.g. whether to use a Relational Data Base Management System (RDBMS) or a NoSQL data store) when mapping abstractions identified in the architectural solution onto suitable elements. Such analysis is done on functional and non-functional criteria. Key challenges in such analysis and assessment process are: i) ensuring that one has not missed out any important alternatives and ii) ensuring that suitable data points about all alternatives have been objectively considered. In several scenarios there is a large number of alternatives possible for realizing an abstraction. For instance, in Apache Software Foundation alone there are more than two dozen web frameworks [1] available. As the size and complexity of an architectural solution increases, the process of mapping abstractions onto elements becomes difficult and error prone if done manually.

Since the evaluation and selection of technology components to be used in any significant project is often a sophisticated process that involves the management of multiple stakeholder views, a plethora of quality requirements; the processes of technology acquisition and evaluating for "fitness for purpose" are expected to benefit from having suitable automation support for such tasks. In this paper we present (in Section II) the design of an Architectural Element Recommender System (AERS) which assists an architect in such tasks. One of its goals is to assist in analysing and recommending the abstraction to element mapping. This system harvests raw information about elements available from different sources such as open source repositories, software product catalogues and data sheets etc. The raw element information is transformed into an ontology by suitable semantic processing. This ontology is then used in an automated semantic inference

[1]In remainder of this paper we use the term *abstraction* or *architectural abstraction* to refer to an abstract entity having a well defined function and interface. Examples of abstractions are a data store, an application server and a computer network etc.

[2]We use the term *element* or *architectural element* to refer to a concrete entity which reifies an *abstraction*.

engine which forms a key part of the proposed AERS. Details of an implementation for the AERS is also discussed (in Section III).

A. Related work

There has been significant interest in the community to leverage semantic computing in systems and software engineering contexts [2]. A framework that employs ontology-based semantic search to retrieve architectural properties and rationale behind those for existing software has been presented by [3]. They focus on searching such architecture knowledge from sources such as emails, meeting notes and wikis etc. Another similar framework that extracts architectural knowledge (e.g. design decision rational etc.) from documentation related to previous projects and domain specific literature is presented in [4]. This framework relies on natural language processing techniques for extracting such information. In this case the extracted knowledge needs to be manually post processed and fine-tuned by a knowledge engineer. Yet another piece of similar work to recover, represent and explore rationale information from text documents is presented by [5]; this work relies on semantic indexing.

A framework for supporting the architectural modeling phase has been described by [6]. It is mainly aimed at modeling the domain of design patterns and for tasks related to reasoning in the modeled domain.

Similarly, authors of [7] have employed a software ontology in a semantic wiki to address architecture documentation retrieval issues. An ontology driven method for supporting the software architecture design and evaluation is presented in [8] and [9] respectively. Similarly, semantic techniques have been applied in an ontology driven builder pattern [10].

In [11] static analysis is used for mining specifications which are later used in code search queries to find how an API should be used. Internet as a source of knowledge is used specifically by [12].

We observe that most automation effort such as discussed above has: i) either focused on gathering generic architecture design knowledge from existing software and then use it to answer similar questions in newer scenarios (works reported in [2]–[5]), or ii) aimed at some specific aspect of software architecture design (works reported in [6]–[12]). The problem of analysing and mapping architectural abstractions to suitable concrete elements finds very limited automation support. In this paper we propose the design of a system that can assist an architect in abstraction to element mapping and analysis.

II. ELEMENT SELECTION FOR ARCHITECTURAL ABSTRACTIONS

When faced with a decision choice scenario often our judgement is determined by relevant prior experience accessible to us from individual memory. An individual, such as an architect, can have only a limited amount of experiential memory. Further, such memory often tends to be organized around broader topics related to the domains of exposure. Collaborative sharing of knowledge helps to address the limitations of individual memory, skill and experience. We hypothesize that by applying semantic computing tools and techniques it is possible to turn vast amount of publicly available unstructured content on systems and software engineering topics into structured knowledge. A variety of decision support systems that are built using semantic technologies have already been reported in literature.

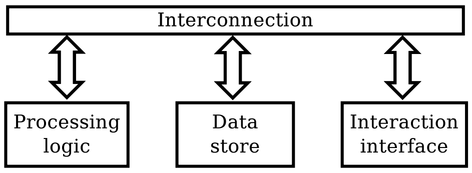

Fig. 1. Major abstractions of a software application

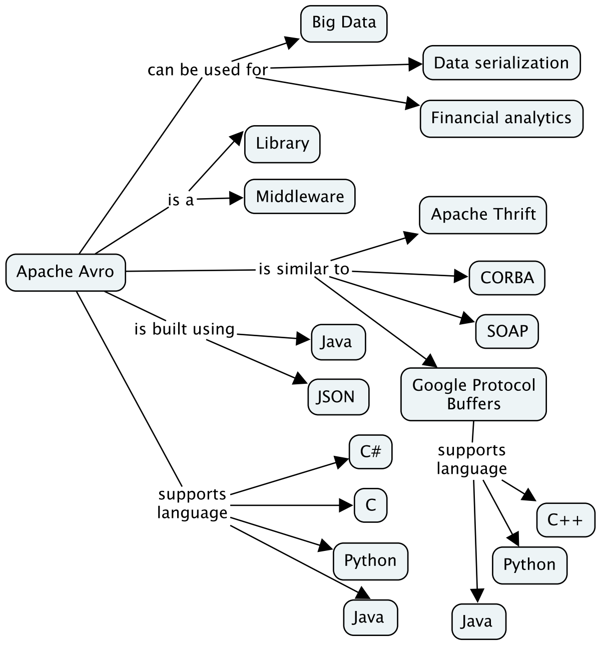

Fig. 2. Few semantic attributes of a basic building block

Considering one populous class of applications – data driven applications – it is well known that the structure of a large majority of those applications can fit into the general high-level structure depicted in Fig.1. We argue that a majority of applications comprise of basic building blocks (i.e. elements) that are chosen from a finite set, Ψ. Size and the elements of set Ψ change very slowly with time, and it may be populated by harvesting details of architectural elements available from various providers. Using suitable semantic processing (discussed later) it is possible to create an ontology of such a set of basic building blocks. Such an ontology will evolve over time as the set of elements Ψ evolves. Fig. 2 shows details of an example node from such ontology.

A. Proposed Design

Various architectural views (e.g. logical, deployment etc.) of a software application depict the constituent elements of

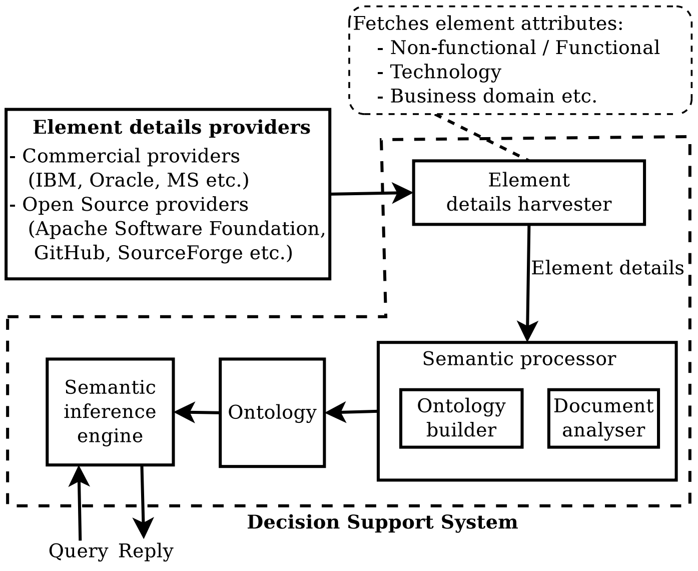
Fetches element attributes:
- Non-functional / Functional
- Technology
- Business domain etc.
Element details providers
- Commercial providers (IBM, Oracle, MS etc.)
- Open Source providers (Apache Software Foundation, GitHub, SourceForge etc.)
Element details harvester
Element details
Semantic processor
Ontology builder
Document analyser
Ontology
Semantic inference engine
Decision Support System
Query
Reply

Fig. 3. Structure of proposed AERS

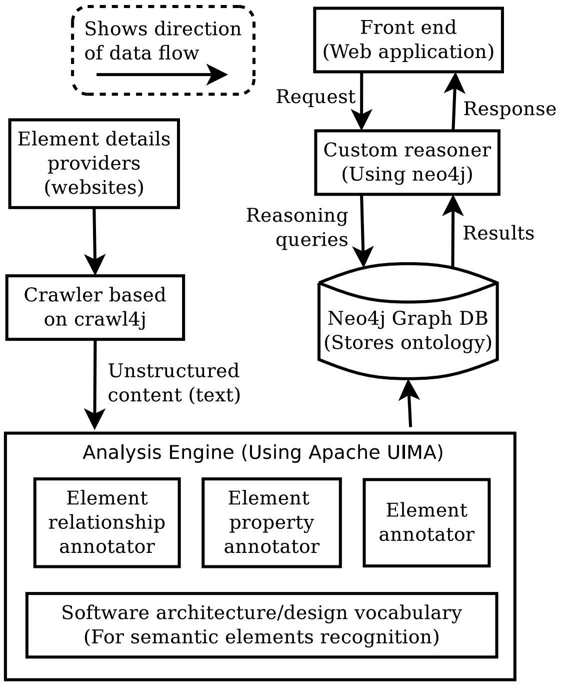
Shows direction of data flow
Front end (Web application)
Request
Response
Custom reasoner (Using neo4j)
Reasoning queries
Results
Neo4j Graph DB (Stores ontology)
Element details providers (websites)
Crawler based on crawl4j
Unstructured content (text)
Analysis Engine (Using Apache UIMA)
Element relationship annotator
Element property annotator
Element annotator
Software architecture/design vocabulary (For semantic elements recognition)

Fig. 4. POC implementation

the application. Such elements have specific properties and interact with each other in a well defined manner. With this in mind, the design of our AERS is centred on the following tenets:

- At a given time, t, there are only a finite number of elements, Ψ, choosing from which one can build any possible software application. The set Ψ is expected to evolve slowly over time. Examples of such elements include: i) a data store element, ii) a programming runtime such as Python or Java etc., iii) an input/output interface to allow interacting with application.
- All elements present in Ψ can be organized as an ontology. For each element, the ontology expresses its properties and possible relationships which it may have with other elements. Such ontology can be represented as a graph $G(V, E)$, where V is the set of vertices and E is the set of edges in the graph. V is a set containing elements and their properties, and E is the set of relationships that can exist among elements of V. This ontology is expected to evolve slowly over time.
- An inference machinery can be built on top of the above ontology to answer queries about and deduce additional facts about architectural abstractions and the elements which can realize those abstractions.

We construct an *Architectural Element Recommender System* (AERS) as an ontology driven application. High level structure of the proposed AERS is shown in Fig. 3. This AERS uses an automated inference engine to, for instance, identify additional alternatives to decisions concerning the elements. Use of inference engine allows objective and methodical comparison of decision choices.

III. Implementation details for AERS

We implemented a limited version of the AERS in order to validate our approach and design. Fig. 4 shows the logical structure of our implementation. A web crawler fetches elements details from various sources and stores it locally. These element detail documents, which typically contain unstructured information, are then consumed by an Analysis Engine (AE). The AE, built using the Apache UIMA [13] framework, extracts *useful semantic information* from element detail documents. The AE internally uses *annotator*s for extracting semantic information such as element types, properties and interrelationships of elements. Annotators make use of suitable vocabulary of software architecture/design terms to accomplish its task. The rich semantic information thus extracted by AE is represented as a graph and is stored in a graph DB (we used neo4j [14]). A web application front end is implemented for submitting queries such as for abstraction to element mapping. The front end application invokes a reasoning engine which uses neo4j's Cypher query language to derive useful truths about/from the concepts stored in graph DB.

A. Harvested elements

For initial experiments we focused on harvesting element data from repositories such as Apache Software Foundation (ASF) and GitHub. We also gathered information from product catalogues of some leading commercial software vendors. From ASF we considered approximately 100 projects (i.e. elements), from GitHub about 200 projects were considered. About 50 elements were considered from various commercial vendors. Projects were selected such that they provide necessary elements as typically used in most data driven enterprise application. The list included, but not limited to, elements such as data store implementations (e.g. MySQL, MongoDB etc.), middleware such as Java Message Service implementations, web application development frameworks and encryption and data compression libraries.

B. Example verification scenarios considered

In order to check the correctness of the recommendations suggested by this AERS we selected three existing systems whose features, design and implementation related documentation was available to us. We derived the abstraction to element mappings that were present in the selected systems. These mappings constituted the reference data for our verification. The proposed AERS was queried for providing recommendations for the abstractions selected from the derived reference. We then compared the AERS recommended elements with those present in reference data. For example, in one scenario we queried the AERS for elements corresponding to abstraction *middleware* subject to the constraints: {`supports language` *java*, `can be used for` *data serialization*}. The AERS output included Apache Avro and Google Protocol Buffers. Relevant part of the ontology that was used by AERS is shown in Fig. 2. The results closely matched with the expected results from our reference data. However, we also observed that the correctness of results strongly depends on "richness" of ontology used by AERS. That is, how much complete and accurate are the relationships and attributes of various elements and abstractions captured by the ontology. This is not a surprise because our implementation infers the recommendations by navigating the ontology.

Similarly, from this system (used ontology shown in Fig. 2) we were able to deduce facts such as:

- `CORBA, SOAP and Google Protocol Buffers are possible alternatives to Apache Avro`
- `An alternative to Apache Avro which supports Python is Google Protocol Buffers.`

IV. Conclusion

For many abstractions that comprise a software solution there can be several alternatives available to realize them. Often the alternatives differ based on their functional and non-functional properties. Realizing such abstractions typically involves identifying suitable architectural elements (e.g. components, connectors, modules or programming runtimes etc.) by careful analysis of the available alternatives. When complexity of the software system being built, and the number of available alternatives that can realize abstractions present in such a system increases, an objective analysis of those alternatives becomes quite difficult.

In this paper we have proposed an Architectural Element Recommender System (AERS) which uses semantic techniques to assist an architect in tasks such as analyses and assessment of abstraction to element mapping. The AERS harvests raw information about elements available from different sources (e.g. public websites and repositories). Through suitable semantic processing we transform the raw information about elements into an ontology. An automated semantic inference engine, which forms a key part of the proposed AERS, then uses this ontology to infer useful truths about various elements.

A limited version of AERS has been implemented based on the design highlighted in preceding paragraph. The semantic analysis engine to extract (semi-automatically) the elements and abstractions' ontology from unstructured data was built using Apache UIMA [13] framework. A graph database (Neo4j [14]) was used to store the ontology as a graph. Neo4j also provides a powerful graph query language – Cypher – using which an automated reasoning engine is built. Efficacy and correctness of the proposed AERS has been verified by querying it for scenarios whose expected output was known. These scenarios were derived/synthesized by examining existing software applications. As we had expected, the accuracy of AERS output was influenced by the "richness" of ontology used.

References

[1] www.apache.org, "The apache software foundation projects," http://projects.apache.org/indexes/category.html#web-framework, The Apache Software Foundation, retrieved: January 2015.

[2] P. Tetlow, J. Z. Pan, D. Oberle, E. Wallace, M. Uschold, and E. Kendall, "Ontology driven architectures and potential uses of the semantic web in systems and software engineering," *W3C Working Draft*, 2005.

[3] A. M. Figueiredo, J. C. dos Reis, and M. A. Rodrigues, "Improving access to software architecture knowledge an ontology-based search approach," *International Journal Multimedia and Image Processing (IJMIP)*, vol. 2, no. 1/2, 2012.

[4] M. Anvaari and O. Zimmermann, "Semi-automated design guidance enhancer (sadge): A framework for architectural guidance development," in *Software Architecture*. Springer, 2014, pp. 41–49.

[5] C. Lpez, V. Codocedo, H. Astudillo, and L. M. Cysneiros, "Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach," *Science of Computer Programming*, vol. 77, no. 1, pp. 66 – 80, 2012.

[6] T. Di Noia, M. Mongiello, and E. Di Sciascio, "Ontology-driven pattern selection and matching in software design," in *Software Architecture*. Springer, 2014, pp. 82–89.

[7] K. de Graaf, A. Tang, P. Liang, and H. van Vliet, "Ontology-based software architecture documentation," in *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, Aug 2012, pp. 121–130.

[8] J. Sun, H. H. Wang, and T. Hu, "Design software architecture models using ontology." in *SEKE*, 2011, pp. 191–196.

[9] M. N. Omidvar and R. Vaziri, "Provide a method for evaluation of software architecture using ontology," *International Journal of Computer Applications*, vol. 64, no. 16, 2013.

[10] A. Chaturvedi and T. V. Prabhakar, "Ontology driven builder pattern: a plug and play component," in *ACM SAC 2014, Gyeongju, Korea - March 24-28, 2014*, 2014, pp. 1055–1057.

[11] A. Mishne, S. Shoham, and E. Yahav, "Typestate-based semantic code search over partial programs," in *ACM SIGPLAN Notices*, vol. 47, no. 10. ACM, 2012, pp. 997–1016.

[12] M. Allamanis and C. Sutton, "Mining source code repositories at massive scale using language modeling," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 2013, pp. 207–216.

[13] www.apache.org, "Apache uima," https://uima.apache.org/, The Apache Software Foundation, retrieved: January 2015.

[14] neo4j.org, "Neo4j, the world's leading graph database," http://www.neo4j.org/, neo4j.org, retrieved: January 2015.