Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach

Claudia López [a,*], Víctor Codocedo [a], Hernán Astudillo [a], Luiz Marcio Cysneiros [b]

[a] Universidad Técnica Federico Santa María, Valparaíso, Chile
[b] York University, Toronto, Canada

ARTICLE INFO

Article history:
Received 27 February 2009
Received in revised form 26 May 2010
Accepted 22 June 2010
Available online 10 July 2010

Keywords:
Design rationale
Software architecture
Ontology
Semantic indexing
Rationale recovering

ABSTRACT

Documenting software architecture rationale is essential to reuse and evaluate architectures, and several modeling and documentation guidelines have been proposed in the literature. However, in practice creating and updating these documents rarely is a primary activity in most software projects, and rationale remains hidden in casual and semi-structured records, such as e-mails, meeting notes, wikis, and specialized documents. This paper describes the TREx (Toeska Rationale Extraction) approach to recover, represent and explore rationale information from text documents, combining: (1) pattern-based information extraction to recover rationale; (2) ontology-based representation of rationale and architectural concepts; and (3) facet-based interactive exploration of rationale. Initial results from TREx's application suggest that some kinds of architecture rationale can be semi-automatically extracted from a project's unstructured text documents, namely decisions, alternatives and requirements. The approach and some tools are illustrated with a case study of rationale recovery for a financial securities settlement system.

1. Introduction

Software architecture is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships between them [1]. In recent years, there has been growing awareness of the value of documenting not just the architecture, but also how and why architectural decisions are taken. "Design rationale" captures the knowledge and reasoning that justify the resulting design [2], and its primary goal is to support designers by providing means to record and communicate the argumentation and reasoning behind the design process [3]. Several guidelines to document software architecture decisions [4–7] and their design rationale [8–11] have been published, and also some templates exist to describe software architectures [12,13].

However, a recent industry survey [2] indicates that practitioners frequently do not document even key rationale information, such as tradeoffs and discarded solutions. The survey identified "no time," "no budget," "no standards" and "no suitable tool" as the most frequent reasons for not documenting architecture rationale. Kruchten et al. [14] argue that another adoption barrier for capturing design decisions is the intrusiveness of many of the processes involved, since they are not fully integrated into current software engineering practice.

Despite these problems, rationale information is still recorded, although casually and isolatedly, e.g. structured reports, wikis, e-mails and meeting notes, with some recording both old and updated discussions and decisions. Clearly, if existing

* Corresponding author.
E-mail addresses: clopez@inf.utfsm.cl, lopezmoncada@gmail.com (C. López), vcodocedo@inf.utfsm.cl (V. Codocedo), hernan@inf.utfsm.cl (H. Astudillo), cysneiro@yorku.ca (L.M. Cysneiros).

0167-6423/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2010.06.009

tools could be extended to make use of this extant design information, some key barriers to their wider use would be ameliorated. Perhaps even reuse of rationale would be in reach.

This article presents the TREx (Toeska Rationale Extraction) approach for architecture rationale recovery, which allows gathering architecture rationale knowledge from plain-text documents, synthesizing it in a centralized knowledge repository, and enabling its interactive exploration to investigate project rationale and its potential reuse. To this end, several tools have been developed, combining: (1) pattern-based extraction of architecture and rationale information from structured or free-form text documents; (2) semi-automatic, human-validated ontology-based representation of rationale extracted from these documents; and (3) faceted interactive exploration of the extracted rationale.

Section 2 introduces architecture design rationale; Section 3 reviews previous work to extract, represent and manipulate architecture rationale; Section 4 explains the TREx approach; Section 5 presents some tools developed to support it; Section 6 describes how TREx was used to recover rationale from a message-based financial clearinghouse system; Section 7 reviews some implications of this approach; and Section 8 summarizes and concludes.

2. Architecture rationale and its recovery

Design rationale explains how and why a device is designed as it is [15]. Tang et al. [2] recognized nine kinds of rationale about software architecture development that observe justification: (1) design constraints, (2) design assumptions, (3) weaknesses, (4) benefits, (5) costs, (6) complexity, (7) certainty in design correctness, (8) certainty in design implementability, and (9) tradeoffs among alternative designs.

Three kinds of design rationale (though not explicitly mentioned in previous work) have stood out clearly when examining some project documents: quality attributes, adopted decisions, and discarded decisions.

Several formalisms and annotations have been proposed to record these architecture design rationale aspects [4–13]. A major recent survey [2] indicates that practitioners usually neither record nor have access to information on most kinds of rationale, especially on design weaknesses, discarded solutions, tradeoffs, and certainty of correctness and implementability; it also reports a high prevalence of undocumented discarded design decisions. The survey authors suggest that reasons for not documenting design rationale include: (a) lack of standards and processes to guide why, how, what and when to document design rationale; (b) time and budget project constraints; and (c) questioning whether the costs and benefits of rationale documentation justify it.

Our experience suggests that, even when available, the reasoning behind design decisions is usually scattered across documents and haphazardly recorded (e-mails, meeting notes, wikis, diagrams, or out-of-date structured documents). Hence, there is a gap between formalisms (and their techniques and tools) and actual records. This work focuses on bridging this gap for some types of generic rationale: design constraints, tradeoffs among alternative designs, risks, quality attributes, and adopted and discarded decisions.

Clearly, addressing the rationale recovery problem as a mere model instantiating or document gathering problem, is just not powerful enough. We argue that this problem can be addressed if it is properly decomposed into three smaller problems:

(1) Recovery of implicit rationale information from design documents.
(2) Representing recovered information (with some formalism).
(3) Manipulation of formalized information (e.g. explore both the rationale description and the original documents).

Each of these problems has been traditionally dealt with by different approaches and communities, but recent work in all three areas converges into the notion of ontologies as enablers and integrators, and we adopt this perspective.

3. Related work

Several strands of work have separately addressed the three aspects of the rationale recovery problem.

3.1. Recovery of implicit rationale information from documents

In ideal conditions, we would be able to recover full rationale information from documents: texts and/or diagrams would be machine-read to generate a comprehensive description of the architecture and its derivation, and this description would allow humans or automated tools to process it. Unfortunately, at this time we are not aware of any project or system that extracts full architecture design rationale from text or diagrams, whether unstructured or formal.

A second and more modest approach is to index documents for retrieving and manipulation, using their own contents. The best known variant is adding metadata to documents, e.g."tagging" [16]. Many Web applications allow users to add metadata to documents by adding "tags" to categorize their contents [16] (e.g. del.icio.us[1] or Flickr[2]). This social categorization approach, dubbed folksonomy [17], has made people sensitive to information indexing [16]. Although this approach has

1 http://del.icio.us/.
2 http://www.flickr.com/.

proved to be a good and unexpensive solution for indexing Web documents [18], the resulting metadata is highly context- and domain-dependent and minimally structured, hence not too useful for automatic integration or knowledge reuse.

Work has been done to reduce problems in the tag universe: educating users on tagging [19], allowing only "good" tags through connection to an ontology [19], enhancing folksonomies with information retrieval technologies [16], and classifying users' tags within a controlled taxonomy connected to an ontology [20]. Although some of these approaches help to produce a controlled vocabulary, they still depend on users' ability (and good will) to describe concepts using metadata and not only as document content.

Other generic works related to automatic semantic metadata creation are based on text mining using neural networks to annotate texts and extend ontologies [22]; its execution requires test-case training, making it a less general process since training input varies with user's individual preferences and characteristics [23].

Recent work in "semantic indexing" includes the KYOTO project [24], which provides cross-linguistic indexation through the use of "Kybots" (automated text-mining processes) that relate documents to an ontology via multi-language Wordnet [25], enabling search using the relations among ontology concepts. This approach allows identifying meaningful concepts in documents for their categorization, but not extracting knowledge units for metadata creation.

Document categorization has also used data-mining techniques like associative classifiers [26], an automatable approach where associative rules can be quickly applied to many documents. Although it considers some text data complexities, context-awareness and metadata creation still remain problematic because the semantic relations among concepts within the text are not considered.

The Universal Parsing Agent (UPA) [27] allows users to create reusable transformation templates to convert text files to structured XML files with tags corresponding to domain concepts. Unfortunately, the approach itself is not ontology-driven and data can be tagged in different ways by different users, leading to potential lack of information coherency.

Generally speaking, existing approaches to semantically indexing project documents have two main problems:

(1) Metadata creation relies on users' perspective about document contents, and not in the contents themselves. It also relies on users' willingness to tag (in folksonomic systems) or to create transformer-templates for plain-text documents (e.g. UPA [27]).
(2) Since extraction tools are not ontology-driven, meaningful semantic relations cannot be extracted within plain-text documents to enrich a knowledge base or extend an ontology. These approaches may have problems with data coherence [26,22] or may fail to extract relevant part of available information [25].

For the specific problems of software architecture, Rambabu et al. [21] suggested annotating plain-text architecture documents with terms of a software architecture ontology. They focused on the ontology description and outlined its use, but did not describe any automated annotation process or a tool to implement it.

3.2. Representing recovered information with existing formalisms

Since the process by which software artifacts are built usually remains hidden in designers' memories and is difficult to recover and reuse, the challenge for design rationale research is to propose helpful, accessible representations for design reasoning [28]. Several design rationale notations have been already proposed.

Rittel proposed Issue-based Information Systems (IBIS) [8] to identify, record and deal with issues in wicked design problems. IBIS is an argumentative approach and its major elements are: (1) *Issues*, or questions to solve; (2) *Positions* that represent alternative answers; (3) *Arguments* for or against positions or other arguments; and (4) *Resolutions* or accepted positions. Different issue analysis or discussions can be related by different relationships such as: logical successor to, temporal successor to, among others. Later, Conklin adapted IBIS for its use in Software Engineering [29].

The Procedural Hierarchy of Issues (PHI) [30] is an IBIS extension for non-controversial issues. It uses answers instead of positions and adds sub-issue, sub-answer and sub-argument relationships.

Potts and Bruns [31] proposed another IBIS modification for its use in software design. It uses a single justification statement instead of multiple argumentation nodes and includes elements that represent "intermediate artifacts" such as models and documents. Thus, this approach offers a hybrid schema that allows designers to create design histories of linked intermediate artifacts and rationale nodes.

The Questions, Options and Criteria (QOC) notation was used in Design Space Analysis (DSA) [10] to explicitly represent design alternatives and their related argumentations. The major elements are (1) *Questions*, or issues; (2) *Options*, alternative answers; (3) *Criteria*, desirable properties of options or requirements; (4) *Assessments*, positive or negative links from options to criteria; (5) *Arguments* on Assessments; and (6) *Decisions*, final resolutions. The major differences between QOC and IBIS are: explicit statement of criteria used to evaluate options in QOC, and linking answers to criteria with positive and negative links.

DRL (Design Rationale Language) [9] is an extension of the Potts and Bruns model for decision rationale. It introduced into design rationale descriptions the notion of *softgoals* to describe evaluation criteria and *claims* to represent arguments. It also includes *decision problems* (questions) and *alternatives*. The main differences between DRL and QOC are that claims have attributes such as plausibility, degree and evaluation, and DRL allows goal-subgoal hierarchies, subdecision relationships between decision problems and presupposes relationships among claims.

Softgoal Interdependency Graphs (SIGs) [11] allow to represent tradeoffs in the decision making process. A SIG is a softgoal[3] network that describes quality attributes (also known as Non-Functional Requirements, NFRs), relationships among them, design solutions for them (both chosen and discarded), and tradeoffs.

Lately, several ontologies have been proposed to describe software architecture rationale. Kruchten [33] introduced an ontology to describe design decisions and relationships among them (i.e. rationale). Later this work was revised and applied to a study case that describe the SPAR Aerospace Dexterous Robotic Arm project [34]. It also described a tool to explore instances of their rationale ontology, which focused on cluster visualization to support decisions.

Akerman and Tyree [35] also proposed an ontology to record software architecture decisions, software assets, roadmaps, and architectural concerns. The approach was illustrated with a simplified version of the IBM Architecture Description Standard.

Sancho et al. [36] proposed an "ontological database" to describe SIGs, and exemplified its use with the Software Performance Engineering Body of Knowledge.

Finally, Babu et al. [37] introduced a software architecture ontology (ArchVoc) and a thesaurus of architectural concepts (including patterns and quality attributes); unfortunately, they did not illustrate their approach with an example, but only with information about the thesaurus itself.

Moreover, several design rationale approaches for software architecture [35,34,36,43] do describe architecture rationale in terms of decisions, concerns, goals, alternatives and software assets, but do not provide a more specific classification of decisions (such as application architectural patterns, or use of specific components and frameworks). On the other hand, although ArchVoc [37] has an operational view of software architecture and offers a thesaurus of architectural decision types, it does not tackle the issue of relating specific systems or projects to specific architectural decisions.

3.2.1. Design rationale tools

Several design rationale tools, such as Archium,[4] AREL,[5] PAKME,[6] ADDSS,[7] and Knowledge Architect[8] can be also used to describe rationale-related information, but they require software practitioners to represent their rationale information with ad hoc vocabularies and formats.

Archium [38] views software architecture as a set of design decisions. It proposes a metamodel to describe both software architecture and design decision elements, and a composition model that relates changes of the design decision to the elements of the architectural model. Archium provides a Java research prototype for validating these ideas.

AREL [39] introduces an UML rationale-based architecture model that describes design rationale, design objects and their relationships. This model also supports reasoning to explain why design objects exist, and what assumptions and constraints they are related to. The approach also provides a tool-set to support record and the automated tracing of elements of this model.

PAKME [40] is a web-based tool to support software architecture knowledge management. It aims to support a collaborative software architecture process that might involve geographically distributed stakeholders. PAKME presents its own software architecture and rationale data model; which also separates architectural knowledge into organizational (generic) and project-specific.

ADDSS [41] is a web-based tool to record and manage architecture design decisions. ADDSS proposes a meta-model to represent relations between architectural decisions, the architecture, stakeholders involved in different architectural views and the iterations performed in the design process.

Knowledge Architect [42] is a tool suite that provides features to create, use, translate, share and manage architectural knowledge. It aims to support collaboration among stakeholders in various activities of the architecture process. Knowledge Architect stores its own data model in RDF and provides interfaces for others tools to store and retrieve architectural knowledge. It also provides a plug-in to annotate and use explicit knowledge architecture inside Microsoft Word documents.

Unfortunately, software practitioners that do record rationale still seem more comfortable with plain-text, and rationale description notations, ontologies and tool have not yet won wide attraction among practitioners; indeed, lack of appropriate standards and tools to support the documentation process are precisely two key reasons cited [2] for not documenting rationale by actual architects.

3.3. Manipulation of formalized information

Software engineering has a long history of (intended) automation, and indeed tools for visualizing, editing and exporting software design artifacts (code, models and documents) are numerous [44]. Since software architecture is already part of

3 A softgoal is a goal without a clear-cut satisfaction criterion: it is *satisficed* (A term coined by Simon [32]).

4 www.archium.net.

5 www.ict.swin.edu.au/personal/atang/AREL-Tool.zip.

6 http://193.1.97.13:8080.

7 http://triana.escet.urjc.es/ADDSS.

8 http://search.cs.rug.nl/griffin.

several standard development processes [45], architecture descriptions are already subject to visualization, editing, analysis and exporting [46]. But architecture rationale information still lags behind regarding manipulation capabilities. Currently, its most evolved representation are ontology-based rationale graphs, but most tools still use closed formats and require that rationale information be input only through the tools themselves [35,34,38–42]. Moreover, existing tools allow only browsing [35,34,38,40,41], clustering [34], tracing [39] and manually annotation [42].

On the other side, some quality attribute knowledge catalogs [47] index their contents with "facets", thus enabling faceted browsing. Faceted classification was introduced by Ranganathan [48] as a concept-based classification; facets are disjoint orthogonal partitions, which allow to describe items in a knowledge store as points in a multi-dimensional space. Object descriptions can thus be synthesized by combinations of basic values, and conversely, can be assigned to multiple (partial) classifications.

Prieto–Diaz [49] employed faceted classification schemas to organize a software reuse library, and he concluded that faceted classification contributed to improve search and retrieval capabilities of the software library. Also, previous work by our research team [50] allows interactive exploration of rationale models by using ontologies to describe rationale and support a faceted search, but skipped the issue of how the rationale models would be populated.

3.4. Ontologies: A transversal issue

Interestingly, all three work streams reviewed in the previous sections (identification, representation and manipulation) are moving towards using ontologies as underlying technology:

- Semantic indexing is required for automatic integration or knowledge reuse, and ontology-based approaches have appeared as proper solutions.
- Recent approaches have proposed new ontologies to describe architecture rationale.
- Recent rationale manipulation tools are based on ontology-based descriptions of rationale.

An ontology is an explicit and formal specification of a shared conceptualization [51]. More precisely, an ontology is a formal model of a domain that describes its concepts and relationships among them. The power of ontologies resides not only in their formal description apparatus, but on their capacity to allow inference of new knowledge from existing one.

Ontologies can be represented with languages like OWL [52], which provides vocabulary to describe classes, their properties, relations among them and cardinalities, and supports description logic and inference rules to derive new information from input data. At an operational level, RDF (Resource Description Framework)[53] is widely used to describe ontologies (mainly at semantically enriched Web sites). RDF encodes information as triplets (resources) that relate a property to other resources or to plain literal data; thus, RDF models are directed labeled graph that allow representing meaningful contents.

The fact that both ontologies and their usual underlying data model are formal and structured ways of describing and indexing information makes them wonderfully appropriate for automated operation, but these same facts also hamper their adoption by architecture document creators. Needless to say, this same problem applies to software architecture description formalisms, but at least there are architecture recovery approaches, even commercial products (e.g. Lattix[9], Structure101[10]); no such sophisticated recovery approaches exist as yet for architecture rationale.

4. TREx: Extraction, representation and exploration of architecture rationale

Recovery of architecture rationale could be significantly helped if the existing rationale notations and tools could be coupled with powerful model-populating and manipulation tools. This requires to solve three major problems:

(1) Gathering rationale information from project documents.
(2) Expressing the recovered rationale information with the previously proposed formalisms (or others).
(3) Manipulating the resulting rationale models and the original documents in order to better understand the reasoning behind software architecture decisions.

To satisfy these processing requirements, TREx (the Toeska Rationale Extraction) combines four conceptual components:

(1) Two *ontologies* to represent *software architecture* (of systems) and *software architecture rationale* (of projects), respectively;
(2) *Ontology-based extraction* of segments of plain-texts documents (knowledge units) that are relevant according to a given ontology;
(3) *Expert-supervised validation and inconsistency management* of the connection ("anchoring") of automatically extracted knowledge units;
(4) *Facet-based exploration* of the knowledge store and the original project documents.

[9] www.lattix.com.
[10] www.structure101.com.

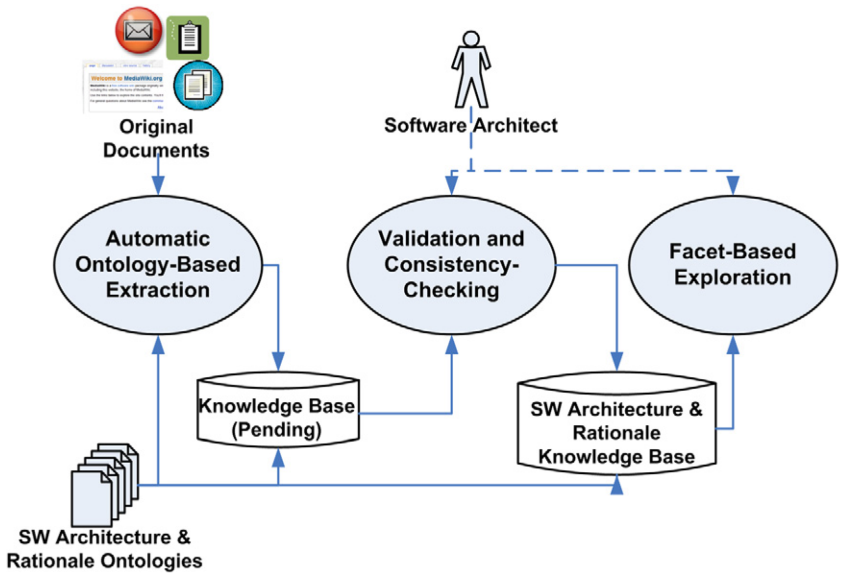

Original Documents
Software Architect
Automatic Ontology-Based Extraction
Validation and Consistency-Checking
Facet-Based Exploration
Knowledge Base (Pending)
SW Architecture & Rationale Knowledge Base
SW Architecture & Rationale Ontologies

Fig. 1. Conceptual components.

Fig. 1 illustrates the TREx architecture. Original project documents are selected to feed TREx; specialized information extraction tools automatically parse them and recover embedded rationale information. The extraction tools combine the ontologies for software architecture and rationale with well known thesauri to create grammars that allow finding useful information in the original documents. Since the information extraction tools may generate erroneous rationale information or inconsistencies when integrating new and old information, the recovered rationale is stored until additional human-validation. Software architects examine recovered to validate it and solve possible inconsistencies. Validated rationale information is finally stored in a Software Architecture and Rationale Knowledge Base to enable further automated manipulation of this data. A facet-based exploration has been already developed, which enables architects to interactively explore both software architecture rationale information and the original documents; they can use it to start a rationale recovery process, to look for specific information or to train a new colleague in the rationale of the system.

The reminder of this section explains in further details the TREx ontologies and tools.

4.1. *Ontologies for software architecture and rationale*

TREx has two ontologies:

- the Toeska Architecture Ontology, to represent the (software) architecture of a system, and
- the NDR (NFR Design Rationale) ontology, to describe the (software) architecture rationale of a project [43].

The Toeska Architecture Ontology (see Fig. 2) builds upon the ArchVoc thesaurus [37], which offers an operational thesaurus-based view of software architecture decisions. Software Architecture concepts (classes in OWL) are the nodes, and they represent key concepts to be recovered by the Toeska approach. The relationships among these concepts (properties in OWL) are represented as graph's edges.

Some highlights of the Toeska Architecture Ontology are:

- A Software Component may achieve Requirements and implement Architectural Alternatives;
- A Software Component may be a System, Subsystem and Component;
- Software Components can be related to Issues, Patterns, Drivers and Technologies;
- A Pattern may be Architectural or Design Pattern;
- A Software Component may support Business Processes and is built by a Project;
- A Requirement can be related to another Requirement through a Tradeoffs property;
- A Requirement may be Functional or Non-Functional Requirement;
- Non-Functional Requirements may be Design Restriction or Assumption; and
- An Architectural Alternative can indicate whether it was adopted or discarded, and additional properties can describe others types of rationale such as cost, benefit, weakness/strength, certainty of implementation, and certainty of design.

The NDR ontology [43] (see Fig. 3) describes software architecture rationale using Softgoal Interdependency Graphs (SIGs) [11]. SIGs explicitly describe Non-Functional Requirements (NFRs) as softgoals, which can be decomposed into more specific softgoals; this refinement process ends when softgoals can be satisfied by one or more solutions (OperSoftgoals). SIGs allow

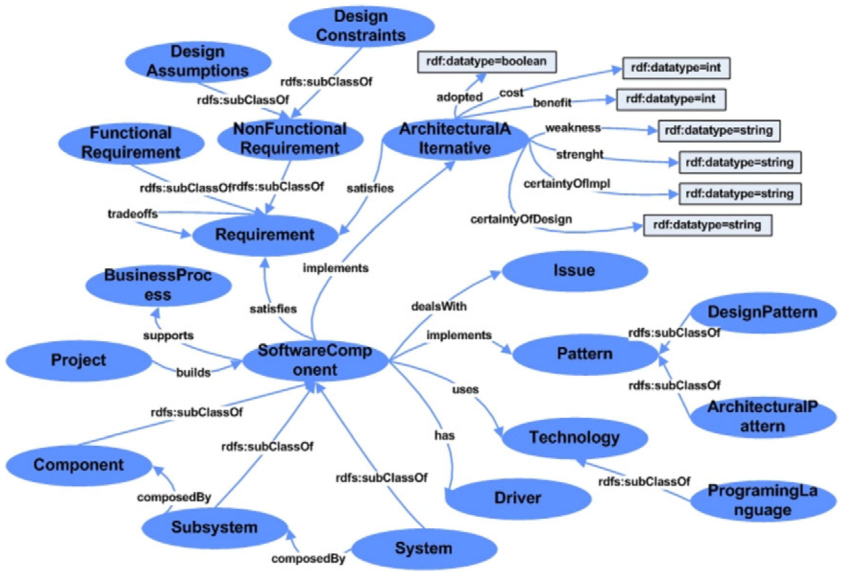


Fig. 2. The Toeska architecture ontology.

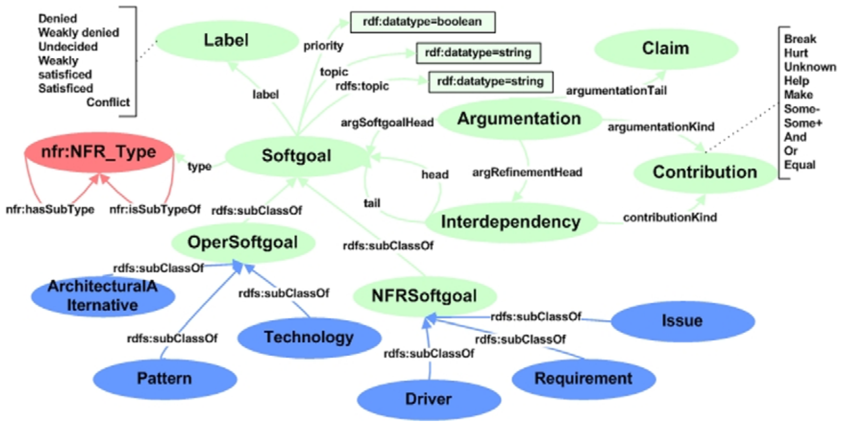


Fig. 3. The NDR (NFR Design Rationale) ontology.

to record interdependencies among softgoals, argumentation rationale, and evaluation of adopted and discarded design decisions.

The NDR Ontology reuses the NFR concept and its properties from the NFR Ontology. Additionally, some highlights of integrating the Toeska Architecture Ontology and NDR Ontology are:

- Architectural Decision, Technology, Architectural and Design Pattern are subclasses of OperSoftgoal;
- Driver, Requirement and Issue are subclasses of NFRSoftgoal.

4.2. Ontology-based extraction of software architecture rationale

To recover rationale information from plain-text documents, TREx applies text mining techniques [54] to discover knowledge units embedded in them. These knowledge units and their semantics are assembled to recompose the argument for system's construction based on TREx.

Information Extraction (IE) [55] is a Natural Language Processing (NLP) technology to extract information snippets from a text corpus and store them in a formal fixed format (knowledge units). IE[11] has five main tasks:

(1) Named Entity Recognition (NE): Finds and annotates nouns as entities within the text by using of large thesauri.
(2) Co-reference resolution (CO): Relates an entity with its references, such as like pronouns.

[11] Information Extraction (IE) differs from Information Retrieval (IR) because IR aims to find and retrieve whole texts for a query but IE only retrieves that information within a text that is relevant to a query.

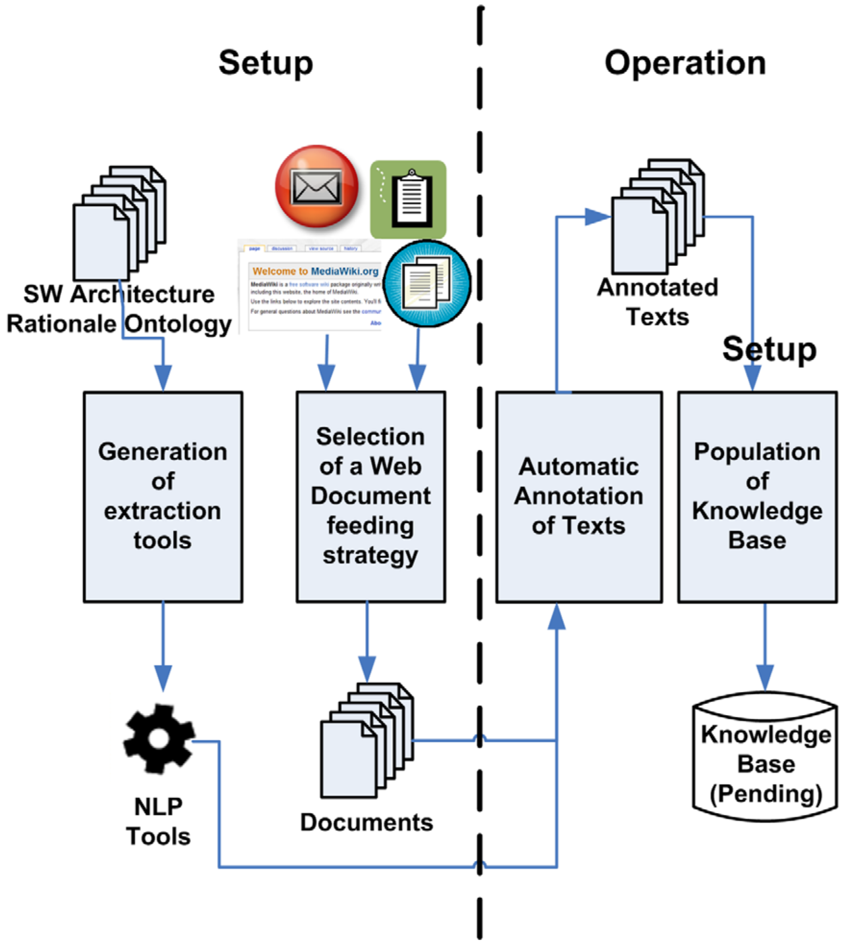

Setup
Operation
SW Architecture Rationale Ontology
Welcome to MediaWiki.org
Annotated Texts
Setup
Generation of extraction tools
Selection of a Web Document feeding strategy
Automatic Annotation of Texts
Population of Knowledge Base
NLP Tools
Documents
Knowledge Base (Pending)

Fig. 4. The TREx overall flow.

(3) Template Element Construction (TE): Finds attributes related to entities.
(4) Template Relation Construction (TR): Finds relations among entities.
(5) Scenario Template Production (ST): Finds events and their participating entities.

TREx uses IE techniques to discover entities (NE), relations among them (TR), and their attributes (TE). Rationale extraction has four steps (see Fig. 4):

(1) Setup (fully manual):
    (a) generation of extraction tools;
    (b) selection of a document feeding strategy;
(2) Execution (fully automatic):
    (a) automatic annotation of texts;
    (b) population of the Knowledge Base.

The ontologies described in Section 4.1 are used to create rationale-specific information extraction tools to recover software architecture and rationale information from documents. In addition, a feeding strategy should be chosen to select a set of input documents; these extraction tools for automated annotators that produce machine-readable texts describing software architecture and rationale concepts and relationships embedded in the original set of documents. The extracted annotated texts are integrated to a rationale knowledge base, related to old information, and enriched with thesauri. Since possible inconsistencies may be introduced by the integration, this knowledge base requires human validation.

Since the extraction tools have been already generated by us, other teams using this approach only need to execute steps for organizations or projects that need to be newly parsed and indexed. The reminder of this section describes all four tasks in detail.

4.2.1. Generation of extraction tools from the ontology

An ontology describes relations among concepts through properties. Each relation instance represents a real world example of this relation formalization (machine-readable) and can be converted into a piece of text (human-readable) by adding proper syntactic elements around and between its elements. Thus, the ontology gives clues of what to look for in texts. To illustrate, consider the following property (taken from the Toeska Architecture Ontology):

```
<owl:ObjectProperty rdf:ID="addresses">
  <rdfs:domain rdf:resource="#ArchitecturalDecision"/>
  <rdfs:range rdf:resource="#Requirement"/>
  <owl:inverseOf rdf:resource="#IsAddressedBy"/>
  <rdfs:label>The Architecture Decision addresses a Requirement
  </rdfs:label>
  <rdfs:comment>Relate an Architecture Decision to a Requirement
  which is positively affected by the Architecture Decision
  </rdfs:comment>
</owl:ObjectProperty>
```

The property "addresses" relates an Architectural Decision to a Requirement that should be addressed by it. From this property, we can look for this kind of relationships within texts to fill our knowledge base. As an example, the revision of a sentence referring this fact within the description of Model-View-Controller (MVC) Pattern in Wikipedia[12] is "MVC helps to reduce the complexity in architectural design and to increase flexibility and reuse of code."

Applying NE (with a proper thesaurus) finds out that "MVC" is a "pattern", and that "complexity, flexibility and reuse of code" are "requirements". All these entities are in the same sentence and we know they are related by predicates, but we can not identify them yet; thus, TR must be used for this and TE helps to discover entity attributes. At the end of this step, we have IE tools specifically built to find rationale knowledge units using different relation descriptions of the ontology as starting points.

This task is fully manual, as rules are created by domain experts that have previously analyzed some example texts. Although some approaches to semi-automated creation of extraction tools [56] rely on extensive analysis of several Web documents, Rationale Architecture texts are not common and we created the tools by hand.

4.2.2. *Selection of a document feeding strategy*

The IE tools require selecting a set of input documents. The two strategies for document feeding are:

(1) Creating a list of URL's[13] audited by a domain expert: this yields more precise metadata but narrower information coverage.
(2) Creating a Web Crawler that finds URL's dynamically (e.g. in an intranet wiki): this yields wider information coverage but the extracted information is less reliable.

The feeding strategy should be selected in a rationale recovery project according to its requirements. The first approach can be used for settled and hard knowledge, since the information has been vetted by the domain expert and can be evaluated in a later stage; the second approach can be used for volatile information classification and dynamic domains, since a large amount of extracted information does not lend itself to careful human evaluation.

In either case, this step provides a mechanism to obtain documents that can be subjected to automatic semantic annotation. TREx supports both decisions though, by the reasons mentioned in the previous section, we recommend the first strategy.

4.2.3. *Automated semantic annotation of texts*

Semantic annotation proceeds by marking the selected documents with "tags", and then extracting the rationale knowledge units. Every annotation has to be "anchored" (linked) to a resource definition in the ontology instance, to describe the text meaning. To enhance the quality of this process, large and robust thesauri are required to identify entities and properties. Whereas some general good thesauri already exist (e.g. WordNet[14] to discover named entities), others were specifically developed by TREx.

Since manual annotation of texts is a quite expensive and time-consuming task [57], it must be automated to make rationale recovery practical. Applying this step for a rationale recovery project produces:

(1) A (single or set of) semantically annotated texts, and
(2) A transformation from these texts into a (single or set of) RDF files, which conform to the Toeska Architecture Ontology.

12 http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

13 A Uniform Resource Locator (URL) is a type of Uniform Resource Identifier (URI) that specifies where an identified resource is available and the mechanism to retrieve it.

14 http://wordnet.princeton.edu/.

4.2.4. *Population of the Knowledge Base*

TREx ingests[15] the extracted annotations to a Toeska Rationale repository. Integration of new rationale information can (must!) be enhanced by checking consistency with data already stored in the repository. Knowledge units could also be enriched with semantic information from external semantic repositories such as DBPedia,[16] Freebase,[17] or OpenCYC,[18] as this information could help in later stages of disambiguating or exploring data; we have done this in other tools but not here.

4.3. *Expert-supervised validation and inconsistencies management*

All recovered knowledge units are "Pending" before being stored in the Knowledge Base since our experience indicates that some (but not most) extracted knowledge units may introduce inconsistencies. Human-supervised validation is thus required.

There are some facts that an automated process cannot distinguish because it lacks necessary contextual information; for example, the word interface might refer to a user interface or a service interface. Obviously, this is not a machine-only problem, since a person without the necessary information would not known the difference either. Hence many extracted rationale knowledge units must be vetted by humans before becoming fully integrated into the repository.

TREx validation is interactive, and presumes the participation of an architect (i.e. a domain-expert for architecture rationale). It includes:

(1) Checking consistency of new information with the existing repository contents: it could lead to completing or changing some rationale knowledge units, or even discarding them.
(2) Checking existence of new information, i.e. whether the desired rationale knowledge units were successfully extracted. If not, the NE and TR tools should be modified and/or the ontology should be extended to cover the new information descriptions. These two actions fall outside the scope of most rationale recovery projects, though they could be tackled by teams with NLP expertise.

These tasks require domain knowledge that can not be integrated within IE tools because it is too complex, too expensive to implement, or too difficult to predict. The existence of a human-in-the-loop is key to ensure information quality.

At the end of this step (with some iterations of extraction and evaluation), a rationale recovery project has incorporated the extracted rationale knowledge units into the Toeska Rationale Repository.

4.4. *Interactive exploration of extracted rationale*

Once annotated rationale-related documents are available and integrated to existing information in a Rationale Repository. The Repository has terms to semantically describe a project, its rationale, and related documents anchored to these terms. It becomes feasible to build software tools to manipulate these descriptions and help architects to investigate rationale of a specific project, or some particular kind of requirement or constraint.

As a first tool, we have developed a faceted rationale browser (see Fig. 5) to support rationale reviews. A general ontology is hard to visualize effectively, given its general-graph structure and a faceted organization simplifies exploring rationale knowledge (see Section 3.3). TREx includes an ontology-based faceted classification to organize the ingested software architecture rationale information.

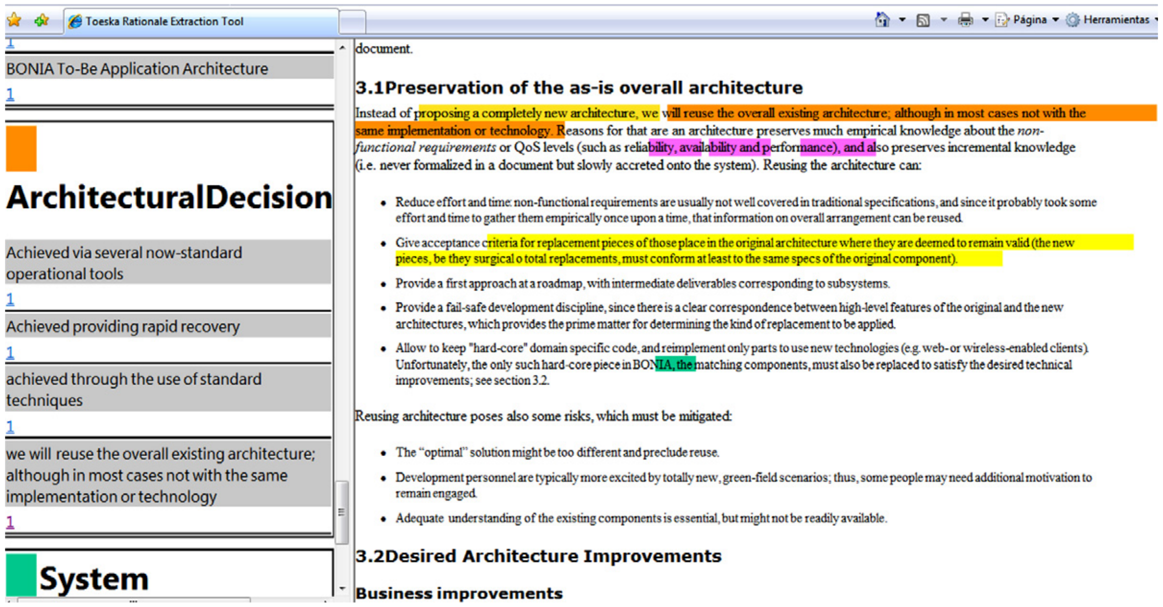
To exploit the semantic anchoring of documents and allow deep human investigation, instances of ontology concepts (i.e. graph nodes) are linked to the rationale-related documents from which they were extracted by TREx.

Since design rationale information usually grows to quite large amounts, identifying valuable information inside the metadata graphs can be daunting. To this end, TREx defines rationale knowledge fragment templates to represent relevant knowledge chunks to be recovered (and possibly reused) by architects [50].

5. TREx tools implementations

Several TREx tools have been built with the GATE (General Architecture for Text Engineering) [58] API,[19] a well-known development framework to build information extraction (IE) tools. GATE provides a basic implementation of IE process called ANNIE (A Nearly-New Information Extraction System), a modifiable serial process that applies several NLP techniques (including sentence splitter, English tokenizer, gazetteer, OrthoMatcher, etc.).

[15] Technical concept used in digital libraries to describe adding new objects and their relationships into an existing repository.
[16] http://dbpedia.org.
[17] http://www.freebase.com.
[18] http://www.cyc.com/opencyc.
[19] http://gate.ac.uk/.



1
BONIA To-Be Application Architecture
1

ArchitecturalDecision

Achieved via several now-standard operational tools
1
Achieved providing rapid recovery
1
achieved through the use of standard techniques
1
we will reuse the overall existing architecture; although in most cases not with the same implementation or technology
1

System

document.

3.1Preservation of the as-is overall architecture

Instead of proposing a completely new architecture, we will reuse the overall existing architecture; although in most cases not with the same implementation or technology. Reasons for that are an architecture preserves much empirical knowledge about the *non-functional requirements* or QoS levels (such as reliability, availability and performance), and also preserves incremental knowledge (i.e. never formalized in a document but slowly accreted onto the system). Reusing the architecture can:

- Reduce effort and time: non-functional requirements are usually not well covered in traditional specifications, and since it probably took some effort and time to gather them empirically once upon a time, that information on overall arrangement can be reused.
- Give acceptance criteria for replacement pieces of those place in the original architecture where they are deemed to remain valid (the new pieces, be they surgical o total replacements, must conform at least to the same specs of the original component).
- Provide a first approach at a roadmap, with intermediate deliverables corresponding to subsystems.
- Provide a fail-safe development discipline, since there is a clear correspondence between high-level features of the original and the new architectures, which provides the prime matter for determining the kind of replacement to be applied.
- Allow to keep "hard-core" domain specific code, and reimplement only parts to use new technologies (e.g. web- or wireless-enabled clients). Unfortunately, the only such hard-core piece in BONIA, the matching components, must also be replaced to satisfy the desired technical improvements; see section 3.2.

Reusing architecture poses also some risks, which must be mitigated:

- The “optimal” solution might be too different and preclude reuse.
- Development personnel are typically more excited by totally new, green-field scenarios; thus, some people may need additional motivation to remain engaged.
- Adequate understanding of the existing components is essential, but might not be readily available.

3.2Desired Architecture Improvements

Business improvements

Fig. 5. Toeska rationale browser.

To enable an agile and standardized text annotation, we have developed an OntoGazetteer as a plugin for GATE. An OntoGazetteer is a Simple Gazetteer[20] automatically constructed from an ontology definition; each word list correspond to an ontology concept and contains its labels. The gazetteer is used to identify ontology concepts within the texts and to annotate them[21].

Once texts are annotated, more sophisticated extraction tools can be created to identify relationships among terms and attributes. The TREx extraction tools are based on grammars[22] that are run as NLP task by ANNIE[23].

A final ad hoc task for ANNIE builds the document (as shown on Fig. 5) using HTML templates in Java and storing the extracted knowledge units in the Rationale Repository.

The Toeska Rationale Repository is built upon JENA,[24] a widely used semantic framework to build semantic applications, which includes complete handling of RDF graphs, RDFS and OWL. JENA manages persistence and integrity for semantic applications (knowledge bases).

6. A case study

TREx has been validated through a case study; this section describes its design, execution steps and analysis results.

6.1. Context

The objective of the case study was to measure the effects of using TREx. We compared effort and results quality of extracting software architecture and rationale information from plain-text documents manually versus automatically (i.e. using TREx).

We developed three types of rationale recovery projects:

(1) Subjects manually recovered software architecture and rationale information from a set of plain text documents.
(2) TREx was simply executed over the same documentation.
(3) Subjects used TREx and validated its results by hand.

These projects used a set of documents of Bonia System, a securities clearing system for which we obtained detailed architecture description but incomplete additional documentation. The documentation set was chosen because an experienced software architect who participated in the project was available for the study planning, allowing us to have a sound “ground truth” against which to compare the case study results.

20 A Gazetteer is a collection of one or more word lists related by a common concept used to annotate texts with tags.
21 For more information about OntoGazetteer, please contact the Toeska Group at UTFSM.
22 Written in JAPE (Java Annotation Pattern Engine), a language to write regular expressions to generate document annotations from existing annotations; see http://gate.ac.uk/sale/tao/index.html#x1-2020008 for details.
23 See http://gate.ac.uk/sale/tao/index.html#x1-2020008 for details.
24 JENA: http://jena.sourceforge.net/.

6.2. Hypothesis formulation

The main hypothesis (to be empirically refuted) was:

(1) H_0: There is no significant difference of effect and results quality between manually and TREx-supported extraction of software architecture and rationale information from plain-text documents.

The measured response variables were:

- Quality: The precision and recall of recovered software architecture and rationale information, measured against an expert-defined ground truth.
- Effort: The time spent by subjects (person-hours).

To evaluate precision and recall, we defined:

- A: The set of all valid software architecture and rationale items in the documents under analysis ("ground truth").
- B: The set of all items identified by a subject from the documents (whether valid or no).
- C: The intersection set between the items identified from the document and those actually present in it (i.e. valid).

Thus, *rationale recovery precision* is defined as:

$$RR_{precision} = \frac{C}{B} \tag{1}$$

and *rationale recovery recall* is defined as:

$$RR_{recall} = \frac{C}{A}. \tag{2}$$

6.3. Planning

The case study object was a set of architecture-related documents of the Bonia project: 26 pages of text, tables and figures that describe the final architecture and some reasons behind some specific architectural decisions.

The subjects were informatics graduate students with several years of industry experience in software development and architecture, but who had not participated in Bonia. Half of them executed a fully manual rationale recovery process (i.e. reading documents), and the other half used TREx.

All subjects used the same set of documents, and the response variables were collected both during and immediately after the recovery exercises.

6.4. Case study results

The case study results are summarized in Tables 1 and 2. Table 1 describes knowledge units recovered by the expert, considered as the ground truth of the study case (i.e. these are the valid knowledge units). This Table also depicts the total and valid knowledge units recovered by the different treatments of the case study:

- a totally manual recovery;
- the TREx extraction tool without any human validation;
- a human-validated TREx-based recovery.

The recovered knowledge units have been classified according to their corresponding concept in the ontology, and the kind of rationale they describe (following Tang et al. [2]).

Two interesting observations can be made about the case results summarized in Table 1.

First, and unexpectedly, more NFRs were found by the unsupervised tool than by the subject-validated tool. We attribute this to over-correcting by the subjects, who discarded some NFRs when they did not notice their importance as architectural requirements; this conjecture remains to be fully tested.

Second, the subjects were able to distinguish "generic" components (i.e. component kinds rather than proper components), but the unsupervised tool failed at this and lumped them all as "components". This points to a recognition of abstraction levels that humans seem to carry per se (or at least engineers) but that would need to be carefully factored into recognition devices.

As expected, the overall precision increased considerably when the recovery process was supported by the TREx tool, and results were even better when considering only rationale information (e.g. adopted and discarded alternatives, and weaknesses/strengths).

7. Discussion

The case study showed that architectural decisions, strengths/weaknesses and quality attributes (including design constraints) can be successfully mined, recovered and related automatically from free-form project documents. Some limitations were identified:

Table 1
Case study — detailed results.

Ontology concept	Ground truth	TREx tool results		Fully manual recovery		TREx-supported recovery		Rationale type
		Total	Valid	Total	Valid	Total	Valid	
Software components	22	4	0	17	14	13	13	
Design constraints	3	2	2	1	1	2	2	Design constraints
Non-functional requirements	7	10	8	2	2	4	4	Quality attributes
Requirements	10	12	10	3	3	6	6	
Technologies	4	1	1	4	4	7	4	
Architectural alternatives (adopted)	13	7	7	4	4	8	8	Adopted decisions
Architectural alternatives (discarded)	3	1	1	0	0	3	3	Discarded decisions
Strength	20	2	2	1	1	7	7	Weakness?
Other rationale types	0	0	0	0	0	0	0	Other rationale types

Table 2
Case study — quality and time results.

	Precision	Recall	Time (min)
Trex tool results	0.77	0.37	1
Totally manual recovery	0.91	0.35	75
Tool-supported recovery	0.94	0.57	75
Expert recovery	1	1	240

- The quality of the extracted metadata and global description of the software architecture and rationale is very dependent on the quality of the available records.
- The idiosyncrasies of documenting aspects of software architecture affect the extraction tools; for example, IE tools probably will require some modifications to examine texts of different teams or organizations than ours.
- Other kinds of manipulations beyond exploration may require more robust and richer ontologies, e.g. to support inferences.

The mere possibility of recovering and examining rationale, even when not explicitly preserved, open interesting avenues of research and practice:

- Reusing rationale: design problems could be recognized as previously dealt with, and the earlier solutions would be a good starting point (though probably never a perfect match due a changing context).
- Auditing architecture processes: decision trails of projects could be explored to determine compliance to corporate or professional standards, or even for legal argumentation.
- Training of architects: newer or less experienced team members could pore over projects rationale to understand better how and why architecture decisions are taken.

Ongoing work aims to build a Semantic Annotation Platform (SAP) [57] using pattern-based IE to find and store rationale, and provide additional services:

(1) Integration of architectural documents from several projects (probably from the same organization) into a single repository;
(2) Integration of architectural documents from several types of documents to obtain a fuller rationale picture for a project; and
(3) Dynamic creation of rationale information from existing architecture and/or rationale information.

8. Conclusions

This paper presented the TREx (Toeska Rationale Extraction) approach and toolset to extract, represent and explore software rationale information from plain-text documents. TREx uses information extraction techniques to extract meaningful rationale knowledge units from plain-text documents; to integrate this information into a rationale repository; and to enable its manipulation by other tools to facilitate human investigation of projects' rationale. This approach will hopefully enable rationale reuse, rationale auditing and architects training.

The entire process is ontology-driven, allowing identification and representation of meaningful relations that describe the content of several rationale documents. A key aspect is human-in-the-loop review to provide expert knowledge, validate extracted information, and enhance ontologies and thesauri.

A case study compared the performance (precision and recall) of the unassisted TREx tool versus human-reviewed TREx execution, using project description documents from an actual projects. Preliminary results suggest that humans are poorer than TREX at identifying rationale, but are better at dealing with "generic" components.

Further progress down this line of research will hopefully lead to treating software architecture rationale with the same kind of sophisticated tools that the software architecture community uses to understand and manipulate architecture descriptions. Architecture rationale deserves to be a project first-class citizen too.

Acknowledgements

This work was partially funded by FONDEF (project CONTEXTA D05I10286), UTFSM (project DGIP 24.08.58), STIC-Amsud (project SECODE - Semantic Based Support for Collaborative Design), and NSERC (grant number 262148-05).

References

[1] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 2nd ed., Addison-Wesley Professional, 2003.
[2] A. Tang, M.A. Babar, I. Gorton, J. Han, A survey of architecture design rationale, J. Syst. Softw. 79 (12) (2006) 1792–1804.
[3] J. Horner, M.E. Atwood, Effective design rationale: Understanding the barriers, in: I.M. Allen, H. Dutoit, Raymond McCall, B. Paech (Eds.), Rationale Management in Software Engineering, Springer, Berlin Heidelberg, 2006, pp. 73–90.
[4] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford, Documenting Software Architectures: Views and Beyond, Addison-Wesley Professional, 2002.
[5] F. Bachmann, L. Bass, J. Carriere, P. Clements, D. Garlan, J. Ivers, R. Nord, R. Little, Software architecture documentation in practice: Documenting architectural layers, Tech. rep., CMU/SEI-2000-SR-004, 2000.
[6] Institute of Electrical, Electronics Engineers, IEEE Std 1471-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, October 2000.
[7] P. Kruchten, The 4 + 1 view model of architecture, IEEE Softw. 12 (6) (1995) 42–50.
[8] H. Rittel, W. Kunz, Issues as elements of information systems, Tech. Rep. Working Paper 131, Inst. Urban and Regional Devt., Univ. Calif. at Berkeley, 1970.
[9] J. Lee, K.-Y. Lai, What's in design rationale? Hum.-Comput. Interact. 6 (1991) 251–280.
[10] A. MacLean, R.M. Young, V.M.E. Bellotti, T.P. Moran, Questions, options, and criteria: elements of design space analysis, Hum.-Comput. Interact. 6 (1991) 201–250.
[11] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos, Non-Functional Requirements in Software Engineering, Springer, 1999.
[12] J. Tyree, A. Akerman, Architecture decisions: Demystifying architecture, IEEE Softw. 22 (2) (2005) 19–27.
[13] Carnegie Mellon - Software Engineering Institute, SEI software architecture documentation template, www.sei.cmu.edu/architecture/SAD_template2.dot.
[14] P. Kruchten, R. Capilla, J.C. Dueñas, The decision view's role in software architecture practice, IEEE Softw. 26 (2) (2009) 36–42.
[15] T.R. Gruber, D.M. Russell, Design knowledge and design rationale: a framework for representation, capture, and use, Tech. Rep., Laboratory, Stanford University, 1991.
[16] I. Peters, W. Stock, Folksonomy and information retrieval, in: Proceedings of the 70th Annual Meeting of the American Society for Information Science and Technology, 2007.
[17] I.V. Yakovlev, Web 2.0: Is it evolutionary or revolutionary? IT Prof. 9 (6) (2007) 43–45.
[18] H. Wu, M. Zubair, K. Maly, Harvesting social knowledge from folksonomies, in: HYPERTEXT '06: Proceedings of the Seventeenth Conference on Hypertext and Hypermedia, ACM, New York, NY, USA, 2006, pp. 111–114.
[19] M. Guy, E. Tonkin, Folksonomies: Tidying up tags? D-Lib Magazine 12:1 (Folksonomies).
[20] S. Christiaens, Metadata mechanisms: From ontology to folksonomy... and back, in: On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, 2006, pp. 199–207.
[21] D. Rambabu, T. Prabhakar, Helping architects in retrieving architecture documents: a semantic based approach, in: Semantic Matchmaking and Resource Retrieval: Issues and perspectives, Vldb Workshop, 2006.
[22] H.-C. Yang, C.-H. Lee, Automatic metadata generation for web pages using a text mining approach, in: WIRI '05: Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration, IEEE Computer Society, Washington, DC, USA, 2005, pp. 186–194.
[23] M.G. Noll, C. Meinel, Authors vs. readers: a comparative study of document metadata and content in the WWW, in: DocEng '07: Proceedings of the 2007 ACM symposium on Document Engineering, ACM, New York, NY, USA, 2007, pp. 177–186.
[24] P. Vossen, E. Agirre, N. Calzolari, C. Fellbaum, S. Hsieh, C.-R. Huang, H. Isahara, K. Kanzaki, A. Marchetti, M. Monachini, F. Neri, R. Raffaelli, G. Rigau, M. Tescon, J. VanGent, KYOTO: a system for mining, structuring and distributing knowledge across languages and cultures, in: LREC'08: Proceedings of the Sixth International Language Resources and Evaluation, Marrakech, Morocco, 2008.
[25] H. Kornilakis, M. Grigoriadou, K.A. Papanikolaou, E. Gouli, Using WordNet to support interactive concept map construction, in: ICALT'04: Proceedings of the IEEE International Conference on Advanced Learning Technologies, IEEE Computer Society, Washington, DC, USA, 2004, pp. 600–604.
[26] B. Li, N. Sugandh, E.V. Garcia, A. Ram, Adapting associative classification to text categorization, in: DocEng '07: Proceedings of the 2007 ACM Symposium on Document Engineering, ACM, New York, NY, USA, 2007, pp. 205–208.
[27] M.A. Whiting, W. Cowley, N. Cramer, A. Gibson, R. Hohimer, R. Scott, S. Tratz, Enabling massive scale document transformation for the semantic web: the universal parsing agent, in: DocEng '05: Proceedings of the 2005 ACM symposium on Document Engineering, ACM, New York, NY, USA, 2005, pp. 23–25.
[28] S. Buckingham Shum, N. Hammond, Argumentation-based design rationale: what use at what cost? Int. J. Hum.-Comput. Stud. 40 (4) (1994) 603–652.
[29] J. Conklin, M.L. Begeman, gIBIS: a hypertext tool for team design deliberation, in: HYPERTEXT '87: Proceedings of the ACM conference on Hypertext, ACM, New York, NY, USA, 1987, pp. 247–251.
[30] R.J. McCall, Phi: a conceptual foundation for design hypermedia, Des. Stud. 12 (1) (1991) 30–41.
[31] C. Potts, G. Bruns, Recording the reasons for design decisions, in: ICSE '88: Proceedings of the 10th International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, USA, 1988, pp. 418–427.
[32] H.A. Simon, The Sciences of the Artificial, 3rd edition, The MIT Press, 1996.
[33] P. Kruchten, An ontology of architectural design decisions in software intensive systems, in: Second Groningen Workshop on Software Variability, 2004, pp. 54–61.
[34] P. Kruchten, P. Lago, H. van Vliet, Building up and exploiting architectural knowledge, in: QoSA'05: Second International Conference on Quality of Software Architectures, Springer, Berlin/Heidelberg, 2006, pp. 43–58.
[35] A. Akerman, J. Tyree, Using ontology to support development of software architectures, IBM Syst. J. 45 (4) (2006) 813–825.
[36] P. Sancho, C. Juiz, R. Puigjaner, L. Chung, N. Subramanian, An approach to ontology-aided performance engineering through NFR framework, in: WOSP '07: Proceedings of the 6th International Workshop on Software and Performance, ACM, New York, NY, USA, 2007, pp. 125–128.
[37] L. Babu, M.S. Ramaiah, T. Prabhakar, D. Rambabu, ArchVoc: Towards an ontology for software architecture, in: SHARK-ADI '07: Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, IEEE Computer Society, Washington, DC, USA, 2007, p. 5.

[38] A. Jansen, J. Bosch, Software architecture as a set of architectural design decisions, in: WICSA '05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture, IEEE Computer Society, Washington, DC, USA, 2005, pp. 109–120.
[39] A. Tang, Y. Jin, J. Han, A rationale-based architecture model for design traceability and reasoning, J. Syst. Softw. 80 (6) (2007) 918–934.
[40] M.A. Babar, I. Gorton, A tool for managing software architecture knowledge, in: SHARK-ADI '07: Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, IEEE Computer Society, Washington, DC, USA, 2007, p. 11.
[41] R. Capilla, F. Nava, S. Pérez, J.C. Dueñas, A web-based tool for managing architectural design decisions, SIGSOFT Softw. Eng. Notes 31 (5) (2006) 4.
[42] P. Liang, A. Jansen, P. Avgeriou, Knowledge architect: a tool suite for managing software architecture knowledge, Tech. Rep. RUG-SEARCH-09-L01, Software Engineering and Architecture (SEARCH) Group, Department of Mathematics and Computing Science, University of Groningen. 2009.
[43] C. López, L.M. Cysneiros, H. Astudillo, NDR ontology: Sharing and reusing NFR and design rationale knowledge, in: MARK 2008: First International Workshop on Managing Requirements Knowledge, 2008.
[44] Carnegie Mellon - Software Engineering Institute, Computer-aided Software Engineering (CASE) Environments, http://www.sei.cmu.edu/legacy/case/case_whatis.html.
[45] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, in: Addison-Wesley Object Technology Series, Addison-Wesley Professional, 1999.
[46] I. Gorton, Essential Software Architecture, Springer, 2006.
[47] L.M. Cysneiros, E. Yu, J.C.S. Leite, Cataloguing non-functional requirements as softgoals networks, in: Proceedings of the Workshop on Requirements Engineering for Adaptable Architectures at the 11th IEEE International Requirements Engineering Conference, 2003, pp. 13–20.
[48] S. Ranganathan, Prolegomena to Library Classification, Asian Publishing House, Bombay, India, 1967.
[49] R. Prieto-Díaz, Implementing faceted classification for software reuse, Commun. ACM 34 (5) (1991) 88–97.
[50] C. López, H. Astudillo, L.M. Cysneiros, Semantic-aided interactive identification of reusable NFR knowledge fragments, in: On the Move to Meaningful Internet Systems: OTM 2008 Workshops, in: LNCS, vol. 5333/2008, 2008, pp. 324–333.
[51] T.R. Gruber, A translation approach to portable ontology specifications, Knowl. Acquis. 5 (2) (1993) 199–220.
[52] D.L. McGuinness, F. van Harmelen, OWL web ontology language overview, W3C recommendation, 2004, http://www.w3.org/TR/owl-features/.
[53] G. Klyne, J.J. Carroll, Resource description framework (RDF): Concepts and abstract syntax, W3C recommendation, 2004, http://www.w3.org/TR/rdf-concepts/.
[54] A. Kao, S. Poteet, Text mining and natural language processing: introduction for the special issue, SIGKDD Explor. Newsl. 7 (1) (2005) 1–2.
[55] H. Cunningham, Information Extraction, Automatic. Encyclopedia of Language and Linguistics, 2nd edn, 2005, p. 665–677.
[56] M. Ruiz-Casado, E. Alfonseca, P. Castells, From wikipedia to semantic relationships: a semi-automated annotation approach, in: M.Völkel, S.Schaffert (eds.), Proceedings of the First Workshop on Semantic Wikis – From Wiki To Semantics, Workshop on Semantic Wikis, ESWC2006, 2006.
[57] L. Reeve, H. Han, Survey of semantic annotation platforms, in: SAC '05: Proceedings of the 2005 ACM Symposium on Applied Computing, ACM, New York, NY, USA, 2005, pp. 1634–1638.
[58] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, GATE: a framework and graphical development environment for robust NLP tools and applications, in: 40th Anniversary Meeting of the Association for Computational Linguistics, 2002.