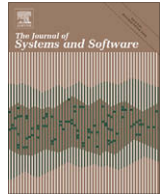




Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jssA comparative study of architecture knowledge management tools[☆]Antony Tang^{a,*}, Paris Avgeriou^b, Anton Jansen^b, Rafael Capilla^c, Muhammad Ali Babar^d^a Swinburne University of Technology, Melbourne, Australia^b University of Groningen, Groningen, The Netherlands^c Universidad Rey Juan Carlos, Madrid, Spain^d University of Limerick, Ireland

ARTICLE INFO

Article history:

Received 14 July 2008

Received in revised form 24 August 2009

Accepted 24 August 2009

Available online xxxx

Keywords:

Architectural knowledge management tool

Architectural design

Design rationale

ABSTRACT

Recent research suggests that architectural knowledge, such as design decisions, is important and should be recorded alongside the architecture description. Different approaches have emerged to support such architectural knowledge (AK) management activities. However, there are different notions of and emphasis on what and how architectural activities should be supported. This is reflected in the design and implementation of existing AK tools. To understand the current status of software architecture knowledge engineering and future research trends, this paper compares five architectural knowledge management tools and the support they provide in the architecture life-cycle. The comparison is based on an evaluation framework defined by a set of 10 criteria. The results of the comparison provide insights into the current focus of architectural knowledge management support, their advantages, deficiencies, and conformance to the current architectural description standard. Based on the outcome of this comparison a research agenda is proposed for future work on AK tools.

© 2009 Elsevier Inc. All rights reserved.

1. Introduction

Software architecture design is considered of paramount importance to the software development life-cycle (Bass et al., 2003). It is used to represent and communicate the system structure and behavior to all of a system's stakeholders. Additionally, architecture can facilitate stakeholders in understanding architecture design decisions and design rationale, further promoting a communication and understanding, reuse and efficient evolution. In the early '90s, (Perry and Wolf, 1992) used design rationale and principles to guide and justify the design of software architectures. More recently, the importance of recording design decisions is described in Clements et al. (2002), and also Bosch (2004) suggests that any architecture is the result of a set of design decisions which should be considered as first-class entities. Kruchten et al. define architecture knowledge (AK) with this equation: $AK = \text{design decisions} + \text{design}$ and they distinguish four types of design decisions according to the implicit and explicit knowledge that is documented or undocumented (Kruchten et al., 2006).

However, architecture knowledge (AK) encompasses not only decisions and rationale, but also other architecturally significant information. The CORE model suggests that AK is a set of relation-

ships between decisions, people, architectural design, and processes (de Boer et al., 2007). Hence, AK may contain alternative solutions, significant entities from the problem space (such as key stakeholders' concerns), technology constraints, business information, and general knowledge (such as design patterns) (Avgeriou et al., 2007). Presently, common architecting practices do not systematically capture or use AK such as design rationale. The evaporation of AK thus results in reduced stakeholder communication, increased system maintenance cost and limited reusability of architecturally significant entities.

Recently, modeling and management of AK has attracted much research interests in this area. A number of methods and models have been proposed, offering explicit support for managing AK. We assert that the research on AK management has progressed beyond the first stages of experimentation. It is therefore time to reflect upon the current state of the practice and draw some conclusions from past experiences. However, there is little consensus in the research community on what AK is, what processes are needed, and how these should be supported by existing tools. Hence, we provide a framework for analyzing these new AK activities and align current standards and well-known software architecture processes. More specifically, we provide an overview of the current tooling support for AK management and possible directions for future tools. First, we will align AK management activities with the architecting process, using a producer–consumer model of AK management in the architecture life-cycle. Second, we map these AK activities onto an evaluation framework to create a set

[☆] All authors have contributed equally to this work.

* Corresponding author. Tel.: +61 3 92145198; fax: +61 3 98190823.

E-mail addresses: atang@swin.edu.au (A. Tang), paris@cs.rug.nl (P. Avgeriou), anton@cs.rug.nl (A. Jansen), rafael.capilla@urjc.es (R. Capilla), Muhammad.AliBabar@lero.ie (M. Ali Babar).

of criteria for comparing AK management tools. Third, we compare the existing AK tools using the framework. In the comparison, we use the IEEE 1471-2000 standard, also known as the ISO/IEC 42010:2007 standard¹, for architectural description as a guideline. We also use a case study and a set of usage scenarios to compare other aspects of the studied AK tools. As a result, we have summarized the current state of the research on AK management tools. Through this comprehensive study, we have identified the strengths and limitations of the existing AK tools, thereby setting the agenda for future research in this area.

2. Managing architectural knowledge

Knowledge management codifies and reuses relevant knowledge that is considered valuable in a particular organization. Several techniques and tools have been proposed and used for this purpose. Recently, in software architecture, a new emerging trend is to store and document AK for reuse. AK comprises various artefacts such as requirements, design, architectural design decisions and their design rationale.

Recently, a number of initiatives have attempted to characterize and support AK implementation with specific processes, models, and tools. They assist software architects in their decision-making activities by capturing and characterizing architectural knowledge. Such tools have been founded upon their characterization and interpretation of what they mean by AK. (Tyree and Akerman, 2005) have proposed a template of attributes to represent architectural design decisions (Tyree and Akerman, 2005), which extends the documentation of design decisions described in Clements et al. (2002). Instead of providing a list of attributes to describe a design decision, the approach discussed in Capilla et al. (2007a) advocates the use of mandatory and optional attributes that can be tailored according to different needs for making more agile the efforts of capturing a decision. Moreover, specific attributes and relationships aimed to support the evolution of design decisions can be found in Capilla et al. (2007b).

Since AK comprises more than design decisions and design rationale, the general AK approaches also capture design decisions and their relationships with requirements and architecture design. An example of this is the Architecture-Centric Concern Analysis (ACCA) method (Wang et al., 2005). ACCA uses a meta-model for capturing architectural design decisions and linking them to software requirements and architectural concerns. As sharing AK is considered a key activity to communicate architecture to others, EAGLE tool (Farenhorst et al., 2008) is an AK sharing portal, which provides architects with a project specific document repository, discussion boards, yellow pages of people, and general reusable documents about software architecture. However, the actual format of documenting AK depends on several AK management approaches.

To support the capturing and use of AK, five AK management approaches and tools have recently been developed and they constitute the subject of comparison in this paper. The first tool is the Architecture Design Decision Support System (ADDSS), which is a web-based tool that provides traceability between requirements and design decisions (Capilla et al., 2006). It stores architectural design decisions following an iterative approach in the same way as architectures are developed and visualizes the evolution of the architecture over time. The second tool is Archium, which models design decisions and their relationships with resulting components (Jansen and Bosch, 2005). The third tool is based on the Architecture Rationale and Elements Linkage (AREL) approach, which models architecture design as causal relationships between design

concerns, decisions and outcomes (Tang et al., 2007). Fourth is the Knowledge architect tool suite, which is based on a common AK repository that is accessed by different clients (Word and Excel plug-ins, APIs) and provides a unified visualization and management interface for AK. The fifth tool is the Process-centric Architecture Knowledge Management Environment (PAKME), which is also a web-based tool that uses a data model for characterizing architectural constructs (such as design decisions, alternatives, rationale, and quality attributes), their attributes and relationships (Ali Babar et al., 2006). Each design decision is captured as a case along with rationale and contextual information using a template. More information about these tools can be found in Section 5.

Previously other researchers have evaluated other tools for their support and management of AK. For instance, Jansen and Bosch (Jansen and Bosch, 2004) analyzed tool support for architecture evolution. Only one of the tools mentioned in their work (i.e. Compendium) exhibit a limited ability to record design decisions, as all the aforementioned AK management tools were created after Jansen and Bosch had reported their work. More recently, the work by Farenhorst et al. (2007) compares five AK management tools from a knowledge sharing perspective. In this sense, our approach differs from these two approaches in three distinct points:

- we investigate tools aimed at managing AK while they have targeted general knowledge management tools;
- we look at a wide range of architecting activities that can be supported by such tools, while the other two focus only on a single aspect and;
- we analyze the AK management tool models with respect to the IEEE 1471/ISO-IEC 42010:2007 standards for software architectural description (IEEE, 2000; ISO/IEC, 2007).

Our approach also attempts to provide a broader perspective by analyzing and understanding how each tool works and the key similarities and differences among the studied tools in order to determine the suitable context of using each of them. Hence, the challenge for comparing AK management tools lies in the different perspectives from which they have been developed and used. To the best of our knowledge, there is no reference framework to evaluate AK management tools. We therefore establish a baseline in terms of the architecture life-cycle and its corresponding knowledge management activities to enable a meaningful comparison of the studied AK management tools.

3. Managing architectural knowledge in the architecture life-cycle

Architecting activities span the initial stages of architectural creation as well as the later stages of architecture evolution and maintenance in a system's life-cycle (Ali Babar et al., 2005). Hofmeister et al. have proposed a general model of software architecture design (Hofmeister et al., 2005). This model has three activities: *architectural analysis*, *architectural synthesis*, and *architectural evaluation*. Since one of the main goals of capturing AK is to support architecturally related activities such as implementation and maintenance of a system and its software, it is necessary to extend the Hofmeister et al. model to include the later stages of architectural design, that is: *implementation* and *maintenance* (see Fig. 1).

AK is mainly created during the architectural analysis, synthesis, and evaluation. During the implementation and maintenance stages, developers and maintainers can make use of the AK created and captured during the early three activities to support their work. They may also have to revisit the architectural analysis, synthesis, and evaluation activities if new design issues have to be addressed. To facilitate the understanding of various kinds of AK that

¹ Since both standards are identical, when we refer to IEEE 1471-2000, we also mean ISO/IEC 40210-2007.

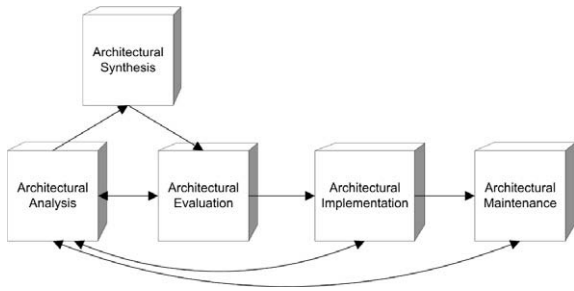


Fig. 1. Extended architecture life-cycle.

may be created and/or used at each stage of the life-cycle of an architecting process, we have classified AK into four general categories:

- *Context knowledge* is a collection of information about the problem space, for instance, architectural significant requirements (ASR) and the context of a project.
- *General knowledge* is a collection of knowledge that helps architects to design software and systems, for example, architectural styles and patterns, and tactics (Buschmann et al., 1996).
- *Reasoning knowledge* is a collection of reasoning information about a design, for example design decisions, design rationale, design alternatives, and trade-offs performed.
- *Design knowledge* is a collection of designs of a system such as components and architectural models.

While *reasoning knowledge* is at the centre of AK (see Fig. 2) *context*, *general* and *design knowledge* lie at the boundary. This is because some of the artifacts may be generated or consumed by other software development activities (such as requirements engineering and detailed design) besides the architecting activities.

We assert that these four categories encompass the overall cases that use AK as the baseline of the architecting processes in close relationship with other typical software engineering products like requirements. At each stage of the architecture life-cycle, the producers and consumers of AK would be involved in different knowledge management activities (Jackson, 1995), such as storing and retrieving AK (see Fig. 2). We map the different stages of the architecture life-cycle to the proposed AK categories to provide a comparison framework for AK management tools. The framework

can be used to evaluate the level of support provided by different tools for managing AK in the architecting life-cycle.

The *architectural analysis* stage serves to define the problems an architect must solve. An architect examines architectural concerns and context in order to come up with a set of architecturally significant requirements. At this stage, an architect acts as a producer who Integrates (B) general knowledge (e.g. the requirements, system context) into AK and as a consumer who Learns (E) and Searches/Retrieves (J) the existing AK to understand if there is any other relevant AK that may influence the analysis.

During the *architectural synthesis* stage, an architect designs architecture solutions for a set of architecturally significant requirements. This task requires an architect to take the role of a producer who Architects (A), thereby creating reasoning knowledge (i.e. the proposed solutions). For this purpose, the architect can Apply (I) general knowledge using existing solutions (e.g. patterns) to solve the problems at hand. The design is created and Synthesized (G) by the architect to capture the design knowledge. The architect also produces the necessary Traces (D) between reasoning knowledge, design knowledge, General and context knowledge. At this stage, an architect also consumes AK through Learning (E) and Searching/Retrieving (J).

Architectural evaluation ensures that the proposed architectural solutions are the right ones. The candidate architectural solutions are evaluated against the architecturally significant requirements. At this stage, an architect Shares (C) AK with one or more consumers (i.e. architecture evaluators). This allows the evaluators in the role of consumers to Learn (E), Search/Retrieve (J), and Evaluate (F) the reasoning knowledge and design knowledge. In order to perform an architecture evaluation, they often need to Trace (D) reasoning knowledge to context knowledge (i.e. the requirements), general knowledge and design knowledge. When an architecture design is evaluated and approved, architects and reviewers may Distill (H) the design as a general design pattern in general knowledge for future reuse.

After architecture evaluation, the architecture life-cycle continues with Architectural implementation. At this stage, the architecture is realized by designers who might add or modify (i.e. Synthesize (G)) the *design knowledge* by creating a detailed design. Designers and developers need to Learn (E), and Search/Retrieve (J) the available *reasoning knowledge* in order to understand the architecture design for implementation. Architects would Share (C) the AK with the implementers to facilitate their understanding.

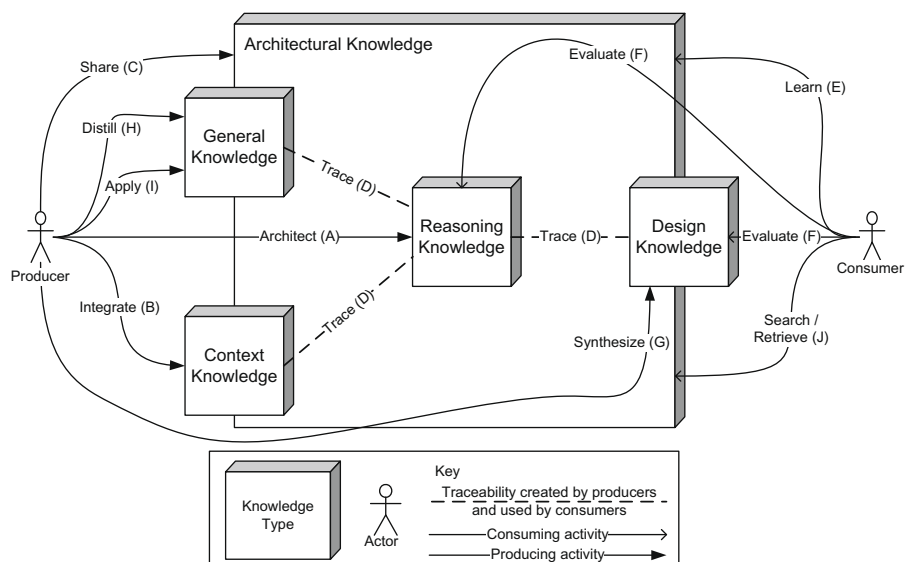


Fig. 2. Architectural knowledge activities.

Once the initial system is deployed, architectural changes may take place during the architectural maintenance stage. At this stage, *consumers* would trace the *design knowledge* to the *AK* to *Learn (E)* about design reasoning and *Evaluate (F)* the impact of certain architectural changes.

4. AK Tool comparison framework

The architectural life-cycle defines the knowledge activities that AK tools should support. On this basis, we have specified a set of criteria for comparing architecture knowledge management tools. The criteria are based on the following sources:

- The current IEEE 1471-2000 standard or ISO/IEC 42010:2007 standard as a reference meta-model
- Architectural knowledge research in existing literature;
- Architectural knowledge activities defined in Section 3.

(Table 1) presents a list of these criteria. The criteria column specifies the context for the comparison. For example, the Types and Representation of AK (row 1 of Table 1) determine what kind of reasoning knowledge can be captured. Research questions for each criterion are given in the description column. Each criterion is related to the AK activities (listed in the third column) of the architectural life-cycle because each of these activities provide usage contexts for the criterion.

The first two comparison criteria presented in Table 1 (i.e. AK representation and AK relationships) are based on the IEEE 1471-2000 standard, which acts as a baseline to evaluate the conceptual models of the AK tools. Evaluation criteria 3–7 are based on the architectural knowledge activities defined in Fig. 2, and they compare how AK tools can be used to support these architectural activities. In addition, based on the literature review, we have found that key supporting features are required in AK tools, and criteria 8–10 describe these requirements. We do not claim that the proposed criteria are the exhaustive list of features that a comparison framework for AK management tools should have. However, we are very confident that the proposed criteria include the majority of the features required by architecture life-cycle activities. In the following, we motivate each of the criteria for AK management tools:

1. *Types and representation of architecture knowledge (AK)*: The types of AK captured can vary depending on what a tool intends to do. AK representations also vary greatly using formal model (Shaw et al., 1995), textual documentation (Tyree and Akerman, 2005), graphs (Lee and Lai, 1996) to represent relationships or using defined links within a knowledge repository (Conklin and Begeman, 1988). In the comparison, we evaluate the underlying model (or meta-model) of the tools to compare what type of knowledge is captured and how they are represented. We have used the IEEE 1471-2000 reference model as a guideline to compare what AK is captured and how they are represented by the tools (IEEE, 2000).
2. *Relationships between AK elements*: As illustrated in Fig. 2, the consumer of architectural knowledge needs the ability to trace AK through different types of interrelated AK entities, for instance between requirements and design, or between design and implementation (Hughes and Martin, 1998; Ramesh and Jarke, 2001). An evaluation of such relationships allow a user to assess how AK tools support architects in the architecture life-cycle to perform functions such as tracing related architectural knowledge and performing change impact analysis (Bratthall et al., 2000).
3. *Architectural analysis support*: *Architectural analysis* is aimed at defining and refining the problems solved by architectural design decisions. This activity examines, filters, and/or reformulates architectural concerns and context to come up with a set of Architecturally Significant Requirements (ASR). AK plays an important role to support this iterative activity. It can provide appropriate templates to help elicit and structure scenarios, provide a repository of generic reusable definitions of ASRs (e.g. problem frames (Jackson, 1995), or record the ASRs and scenarios in the analysis (Clements et al., 2002; Bengtsson and Bosch, 1998). This criterion examines what facilities an AK tool provides to support architectural analysis.
4. *Architectural synthesis support*: *Architecture synthesis* is aimed at identifying candidate architectural solutions that can address the ASRs elicited during architectural analysis. This activity is carried out by architects to map the problem space to the solution space. AK tools can support architects

Table 1
A framework for comparing architecture knowledge management tools.

Criteria	Description	AK activities
1. Types and Representation of AK	What are the architectural knowledge types and representations captured by a tool for general, context, reasoning and design knowledge?	Architect (A), Synthesize (G), Distill (H), Apply (I), Integrate (B), Learn(E), Search/Retrieve(J)
2. Relationships between architectural knowledge	How does a tool support and manage dependencies between different types of architectural knowledge (such as reasoning and design knowledge)? What are the types of relationships and traceability supported by the tools? How do the tools support change impact analysis?	All AK activities
3. Architectural analysis support	How does a tool support the analysis of architecturally significant requirements?	Integrate (B), Learn (E), Search/Retrieve (J)
4. Architectural synthesis support	How does a tool support architects to use and produce AK during architecture synthesis stage?	Architect (A), Synthesize (G), Learn (E), Search/Retrieve (J)
5. Architectural evaluation support	How does a tool support evaluation?	Evaluate (F)
6. Architectural implementation support	How can a tool support architecture implementation activities? How does the knowledge captured in the early stages of the architecture activities be communicated?	Architect (A), Synthesize (G), Evaluate (F), Learn (E), Search/Retrieve (J)
7. Architectural maintenance support	How does a tool support architecture maintenance?	Evaluate (F), Learn (E), Search/Retrieve (J)
8. AK customization	How does a tool support the personalization of knowledge representation based on different user's preferences and profiles?	Architect (A), Learn (E), Search/Retrieve (J)
9. Integration with other tools	How does a tool support the integration with other software engineering and knowledge engineering tools and knowledge repositories?	Integrate (B), Share (C), Learn (E)
10. Collaborative environment	How does a tool support collaboration between distributed software development teams?	Share (C), Evaluate (F), Learn (E), Search/Retrieve (J)

by providing knowledge such as design patterns (Harrison et al., 2007), architectural tactics (Bass et al., 2003), previous architectural solutions and design rationale (Lee and Lai, 1996; Conklin and Burgess-Yakemovic, 1996), and alternative design solutions (Maclean et al., 1996). This criterion examines how an AK tool support design synthesis with its knowledge repository and facilities.

5. *Architectural evaluation support*: Architectural evaluation is aimed at ensuring that the architectural design decisions made are the right ones. During this activity the candidate architectural solutions are assessed with regard to the ASRs. This activity can be performed using architecture evaluation methods such as ATAM (Kazman et al., 1998) or SBAR (Bengtsson and Bosch, 1998), or review methods such as reported in Maranzano et al. (2005), Avritzer and Weyuker (1998). The types of evaluation or review that can be performed on a architecture design depends on what AK is captured and what kind of evaluation is desired (Dobrica and Niemela, 2002). This criterion evaluates an AK tool by examining how it supports this activity through the knowledge it can provide.
6. *Architectural implementation support*: A tool captures and represents AK not only during architecting but also during system implementation. It has been noted that one of the issues with some design rationale tools is its ineffective communication (Shipman and McCall, 1997). Evaluating how an AK tool enables the communication of knowledge to architects, designers, programmers, and testers would be an important aspect (Conklin and Burgess-Yakemovic, 1996).
7. *Architectural maintenance support*: One of the key purposes of capturing AK is to retain it for future use. AK tools can support maintenance activities in different ways, such as requirements traceability (Ramesh and Jarke, 2001), evaluation and impact analysis (Maclean et al., 1996). This criterion examines in what ways an AK tool can be used to support system maintenance.
8. *AK customization (8)*: A user of an AK management tool may like to customize different features of the tool for personal or organizational preferences such as reporting in a preferred format or tailored mechanism of capturing rationale. Such personalization allows the user to manage the knowledge to suit the objectives of the user (Ali Babar et al., 2007). This criterion evaluates the user customization facilities provided by an AK tool.
9. *Integration with other tools*: An AK tool would not be very useful as a standalone tool, it has to integrate with other software engineering tools to support the development life-cycle (Regli et al., 2000). For instance, it has to use and communicate the knowledge from a requirement repository, the knowledge entities may be exported to and imported from such repository. This criterion examines the types of knowledge that can be integrated with the AK tool and the facilities that the AK tool can provide to allow such integration.
10. *Collaborative environment*: A fundamental characteristic of software development is the collaboration between people and teams. AK tools aim to facilitate such collaborations by sharing relevant knowledge. An effective AK tool must provide a collaborative environment such as sharing, versioning, protected access (locking), and collaborative authoring of AK (Farenhorst et al., 2007). This criterion assesses an AK tool's capabilities of supporting collaboration.

5. AK management tools

In this section, we introduce the AK tools that are the products of recent research in this area. We believe that the tools are representative of the current research on tool support for the AK man-

agement. The next subsections briefly describe different features and the main concepts of each tool in this study.

5.1. ADDSS

The Architecture Design Decision Support System (ADDSS²) is a research web-based tool for storing, managing, and documenting architectural decisions (Capilla et al., 2006). ADDSS uses a flexible approach based on a set of mandatory and optional attributes for characterizing the design decisions. Hence, ADDSS provides a combined codification-personalization strategy that has flexibility in capturing knowledge. Moreover, ADDSS captures both architectures and decisions following an iterative process which clearly shows the evolution of AK over time, and simulates the way in which software architects build their architectures as a set of successive refinements. Depending on the specific phase of the software life-cycle (i.e. development, maintenance, testing), a *status* (e.g. pending, approved, rejected, obsolete) and a *category* (i.e. main, alternative, derived) attributes can be assigned to each decision indicating its current status. ADDSS provides links between requirements and architectures for forward and backward traceability. Also, a basic dependency model enables dependencies between decisions, which can be used to estimate the impact of adding, removing, or modifying a decision. During the reasoning process, ADDSS users can reuse general knowledge in the form of patterns and architectural styles. A query system provides valuable information to the architect of related requirements, decisions, and architectures stored in the tool. PDF documents can be generated containing the detailed description of the architectures and their decisions. Such documentation makes explicit the ideas of the decision view described in Dueñas and Capilla (2005) and provides a communication vehicle between the stakeholders. ADDSS has been evaluated with following case studies:

- A case study with undergraduate students of the Rey Juan Carlos University (Madrid, Spain) for evaluating the capabilities of ADDSS 1.0, estimating the effort capturing decisions of a subset of requirements of a real virtual reality system, and evaluating the usability of the tool;
- An internal case study with ADDSS 2.0 for capturing the design decisions of the overall set of requirements of the same virtual reality system;
- Use of ADDSS in combination with a reverse engineering tool called SAVE from the Fraunhofer IESE (Kaiserslautern, Germany) to capture the key design decisions of tool (i.e. DecisionModeler) for managing the variability of product line assets and store the architectures recovered with SAVE and;
- Use of ADDSS with Master students of the Rey Juan Carlos University to estimate the effort in capturing design decisions for the development, maintenance, and evolution of the architecture of the virtual reality system and compare this with the typical architecture modeling effort.

5.2. Archium

The Archium tool³ (Jansen and Bosch, 2005) aims at providing traceability among a wide range of concepts while maintaining this knowledge during the life-cycle of a system. Archium employs a range of concepts from requirements, decisions, architecture descriptions, to implementation. All these concepts can be expressed in the Archium language, which is an extension of the Java language. This makes a single language act as an architectural knowledge repository and implementation of a system. To support this

² <http://www.triana.escet.urjc.es/ADDSS>.

³ <http://www.archium.net>.

language, the Archium tool suite consists of a compiler, a run-time platform, and a visualization tool. The compiler turns the Archium source files into executable models for the run-time platform. The visualization tool in turn uses this run-time platform to visualize the AK, thereby making the knowledge easily accessible. The tool supports two types of traceability relationships: (a) *formal relationships*, which are traceability links defined in the Archium meta-model, which have well-defined semantics. (b) *informal links*, which are references made in the textual descriptions to model elements. The tool has been applied to several small case studies to demonstrate its application.

5.3. AREL

Architecture Rationale and Element Linkage (AREL⁴) aims to assist architects to create and document architectural design with a focus on architectural decisions and design rationale (Tang et al., 2007). AREL captures three type of AK: *design concerns*, *design decisions* and *design outcomes*. These knowledge entities are represented by standard Unified Modeling Language (UML) entities and they are linked to show the relationships between them. *Design concerns* are inputs that influence design decisions. It is an entity that encapsulates concepts such as functional requirements (e.g. scenarios and collaboration diagram), non-functional requirements (e.g. all quality attributes) and project contexts. It also captures information about design decisions and design rationale. *Design outcomes* comprise of the resulting designs. Examples are classes, components, interface, and use case. AREL has been implemented as a plug-in of a modeling tool called Enterprise Architect (EA). AREL plug-in uses standard UML notations to represent design and decision entities. Any customized AK is captured by applying pre-defined tagged template of a stereotype. Import tools are available to extract information from Word-based requirement specifications, and plug-in tools are available to enable graphical tracing and analysis of the information. This tool has been tested on an industrial electronic payment system specification (Tang et al., 2009). It is currently being used to develop a knowledge management system in an engineering firm.

5.4. The Knowledge architect

The Knowledge architect⁵ (Jansen et al., 2008) is a tool suite for capturing, managing, and sharing AK. The suite consists of an AK repository, the Knowledge architect server, which stores the knowledge entities, and of a number of Knowledge architect clients, which capture and manage AK in different formats and contexts. One client is the Word plug-in, which allows architects to capture and manage AK in MS Word documents. A second client, also a plug-in, captures and manages the AK of quantitative architectural analysis models expressed in MS Excel. A third client, the Knowledge architect explorer, is a visualization tool that supports the analysis of the captured AK. This tool allows for the exploration of the AK by searching and navigating through the web of traceability links among the knowledge entities. The Knowledge architect emphasizes on providing generic AK management without restricting itself to specific AK meta-models. The server is thus responsible of maintaining an accessible and generic AK repository while the different clients can implement specific AK meta-models and exchange AK with each other. Besides the standard Knowledge architect clients, external AK management tools can interoperate with the Knowledge architect server through standardized interfaces. The tool suite has been applied in two industrial case studies, one involving the Word plug-in, the other one the Excel plug-in.

5.5. PAKME

The Process-based Architecture Knowledge Management Environment (PAKME⁶) is a web-based tool that is aimed at providing knowledge management support for the software architecture process. PAKME has been built on top of an open source groupware platform, Hipergate (Hipergate, 2007), which provides features to manage AK for geographically distributed stakeholders involved in the software architecture process.

The knowledge repository of PAKME is logically divided into *knowledge-based artifacts*, *generic knowledge*, and *project-based artifacts*. PAKME consists of four components; web-based user interface component, knowledge management component, search component, and reporting component. Corresponding to these four components, PAKME's features can be categorized into four AK management services: knowledge acquisition, knowledge maintenance, knowledge retrieval, and knowledge presentation. A detailed description of the AK management framework support by PAKME can be found in Ali Babar et al. (2006) and Tool description can be found in Ali Babar and Gorton (2007). PAKME has been deployed and evaluated in an industrial context for providing AK management support for architecture evaluation of Avionics systems (Ali Babar et al., 2008).

6. Tool comparisons

The framework of 10 criteria for comparing AK tools represents the perspectives of AK management. They illustrate key focuses of the tools with respect to the architecting activities defined in Section 3. Since each criterion represents an aspect of how AK is created or used in the architectural life-cycle, we use specific methods in each criterion to enable the comparison using three general approaches:

- International standard – we compare the first two criteria based on the IEEE 1471-2000 reference model as a baseline to compare the entities and the relationships used in each tool (see Sections 6.1 and 6.2);
- Case study – criteria 3, 4 and 5 represent key architectural design activities. To illustrate how each AK tool is used in the AK development life-cycle, we use a case study to demonstrate what knowledge each tool captures and how the knowledge is represented and used (see Section 6.3) and;
- Scenario-based analysis – criteria 6–10 represent other perspectives on AK management. To evaluate and compare how the AK tools cater for these aspects of AK management, we have identified real-life scenarios to evaluate how each tool performs for the AK activities characterized by those scenarios (see Section 6.4).

6.1. Compare AK types and representation

To compare the different types and representations used by various AK tools, we have used the IEEE 1471-2000 standard as a basis for comparing the meta-models of each AK tool. With the reference model, we classify the modeling concepts. The resulting classification is presented in Appendix A.

As a result of the comparison using criteria 1, we observe that all the studied tools support the concepts described in the IEEE 1471-2000 standard. An exception is formed by the “mission” and “library viewpoint” concepts, not supported by any of the AK tools. In addition, the “environment”, and “viewpoint” concepts

⁴ <http://www.ict.swin.edu.au/personal/atang/AREL-Tool.zip>.

⁵ <http://www.search.cs.rug.nl/griffin>.

⁶ <http://193.1.97.13:8080/> (an online version of PAKME can be accessed through this URL).

are supported by some tools while a variety of new different concepts like: variability, decision making activity, design history, pattern, reviewer, or superseded design rationale are not addressed by the IEEE 1471-2000. Therefore, we can learn that the AK tools introduce new concepts, mostly associated to include the design rationale that is poorly supported or described in the reference model. As all the studied AK tools gather common architecture descriptions, the aforementioned standard lacks details to describe the underlying rationale and the decisions that belong to an architectural description.

Moreover, from Appendix A one could observe commonalities and differences in terminologies and concepts but there exists: (1) semantic differences for similar named concepts (2) semantic equalities for different named concepts. For example, the *Trade-off* concept of the Knowledge architect deals with trade-offs for various quality attributes of different alternatives, which are backed up by quantitative analysis results. This is a much more specialized version of the *Trade-off* concept as found in Archium, which covers both qualitative and quantitative trade-offs. Hence, additional information is needed of the relationships between the concepts of the various AK tools. To provide this information, we make these relationships explicit. The following notation is used in Table 2 for this purpose:

- *Same-as* (=) The concept is the same – as the concept(s) mentioned for the other AK tool.
- *Generalization* (G) The concept is a generalization of a concept from another AK tool.
- *Specialization* (S) The concept is a specialization of a concept from another AK tool.
- *No relationship* (-) No related concepts exists in the other AK tool.

Since every AK tool has many concepts (see Appendix A for an overview), five sets of comparisons are required to map them. Due to space constraints, not all these mappings are presented here. (Table 2) presents a small excerpt of these mappings. The table illustrates how the *Scenario*, *architectural design decision*, and *Trade-off*, concepts of the Archium tool relate to the other four tools. Similar tables have been created to carry out analysis for the other four tools.

Based on these mapping tables a number of interesting differences and commonalities become apparent among the five studied AK tools. The following is a list of the key differences:

- *Architecture design description versus no architectural design description.* Both Archium and AREL have an extensive support to describe the architecture design with the help of an architectural design model. Such a model is missing in the Knowledge architect, ADDSS, and PAKME. As such, Archium and AREL provide better support for the description of architectural models as described in 1471-2000 standard.

- *Requirement driven versus goal driven.* ADDSS and PAKME directly relate architectural decisions with scenarios and requirements. The Knowledge architect, AREL, and Archium instead use a goal driven approach, in which requirements and scenarios are explicitly scoped and transformed into a goal that an architectural decision tries to achieve.
- *Explicit architectural modification versus discrete evolution.* Archium is the only tool to model explicit changes to the architecture model by the design fragment and delta concepts. AREL uses an approach somewhere in between with the superseded design concept. This concept defines that architecture elements can be superseded by new (complete) definitions of them. Both tools model the expected consequences such a modification will give. ADDSS uses a discrete evolution model in which the modification itself is not modeled, but rather a picture of the new model is presented. It is the only tool, which makes such iterations explicit. On the other hand, the IEEE-1471-2000 standard does not prescribe any concepts to support architecture evolution.
- *Scenario support.* AREL, PAKME, and Archium have concepts for supporting architectural scenarios, as used in ATAM. Both the Knowledge architect and ADDSS do not support architectural scenarios.
- *Quantitative analysis support.* AREL, PAKME and the Knowledge architect support quantitative analysis. A difference between them is that AREL and PAKME only model the outcome of such an analysis, the Knowledge architect also models the analysis process itself. Archium and ADDSS do not support quantitative architectural analysis. This kind of quantitative analysis represents specialized design rationale analysis of the IEEE 1471-2000 standard on *rationale*.
- *Quality model.* The Knowledge architect and AREL explicitly model the quality attributes and relate them to the architectural decision-making process. Interesting enough, there are some noticeable differences in how these tools achieve this. In AREL, a quality attribute is seen as a specialized design concern. For ADDSS, a similar approach is taken, as quality attributes have explicit relationships with the non-functional requirements upon which the architectural design decisions operate. The Knowledge architect uses a different perspective on quality attributes, as it views this concept as a property of a design alternative. In addition, the Knowledge architect tool traces the refinement of abstract quality attributes into more concrete ones. The Archium and PAKME tools do not explicitly model attributes.
- *Architectural review.* The Knowledge architect is the only tool to provide concepts to capture architectural review knowledge. This includes facilities for different people to comment, approve, and disapprove the validity of some AK. Support for reviewing is missing in the Archium, AREL, ADDSS, and PAKME tools.
- *Patterns and style support.* ADDSS, Archium, and PAKME offer support for the notion of patterns and architectural styles. Of these three tools, patterns are the most visible in PAKME and

Table 2
Summary mapping of similar concepts of AK tools.

Archium	ADDSS	AREL	Knowledge architect	PAKME
Scenario	(S) Functional requirements	(S) Functional requirements	-	(=) Scenario
Architectural design decision	(=) Architectural Design Decision	(G) Design rationale	(=) Decision	(=) Architecture decision
Trade-off Solution	(G) Trade-off (G) Alternative design decision with design pattern if applicable	(=) Trade-off (G) Chosen design and alternative design solution	(G) Trade-off (=) Alternative	(=) Trade-off (=) Design option
Design rules	-	-	-	(=) Rule
Design constraints	(S) Constraints	(=) Design constraint	-	(=) Constraint
Consequences	-	-	-	(G) Benefit

ADDSS, as in these tools the patterns are the design options considered for an architecture decision. The difference between PAKME and ADDSS is that PAKME offers evaluation options for patterns and styles. The Knowledge architect and AREL do not have explicit concepts for patterns and styles. This concept is absent in the IEEE 1471-2000 standard.

- *Decision process tracking.* The Knowledge architect and ADDSS track the status, creators, and the people responsible for a design decision. Thus providing additional information to track the progress of the decision process. The AREL, PAKME, and Archium tools do not offer such facilities.
- *Classification/categorization of decisions.* ADDSS and PAKME are the only tools which categorize/classify design decisions. The categorization denotes whether a decision is a Main (selected), Derived (selected but specialized from a Main decision) or alternative (initially considered but afterwards is non-selected) decision. The classification of design decisions by these tools is usually based on whether the design decision involves a variation point, pattern, style, or something else. The other three tools do not offer such a classification or categorization for their decisions.
- *Risk identification support.* The AREL, Knowledge architect, and PAKME tools offer support to explicitly record risks. This feature is missing in the Archium and ADDSS tools. This concept is a specialization of design rationale in the IEEE 1471-2000 standard.
- *Multiple users/projects support.* Both ADDSS and PAKME offer support for different users for their tools on different projects. Hence, project management through multi-user support can be better controlled as well as permissions for accessing different types of knowledge. The Archium, Knowledge architect, and AREL tools do not offer such functionality; instead they use the model of a single project with a single type of user.

6.2. Compare AK relationships

The explicit relationships between different types of AK are important to understand how the AK can be used and what life-cycle activities they support. These relationships form parts of the architectural activities such as implementation and evaluation (Fig. 1). For instances, a requirement is *defined* by an end user as a stakeholder or a design decision is *explained* by its rationale. Hence, the relationships in AK models underpin how that knowledge can be used by architects effectively. In this section, we use 16 generic relationships defined in the IEEE 1471-2000 architectural model as a basis to analyze the relationships supported by the studied AK tools. Appendix B is a summary of the relevant relationships defined in each AK tool. Each relationship is specified in a

general form: $\langle Relationship \rangle$ between two entities, $\langle EntityA \rangle$ and $\langle EntityB \rangle$.

The diversity and types of relationships of the AK tools makes the comparison between them quite complex but the IEEE-1471-2000 provides a baseline for comparison (see Table 3). If an AK tool does not support the relationship in the reference model, the cell in Table 3 will show the symbol (-). If the AK tool models a relationship, then the number shown and the role in a cell is the relationship number which is defined for that tool in Appendix B. For instance, the Knowledge architect tool explicitly models *stakeholder has requirements* (in Appendix B), and this concept conforms to the IEEE 1471-2000 architectural description standard that *stakeholder has concerns* (Row 7 of Table 3).

IEEE 1471-2000 relationships can be broadly grouped into three categories:

- High-level relationships that relate key system concepts such as mission, systems, stakeholders, description and concerns (see relationships 1 to 7 in Table 3);
- Relationships that are central to the architectural description that involves concerns, views, model and rationale (see relationships 8–12 in Table 3) and;
- Relationships that highlight different perspectives or viewpoints of an architecture (see relationships 13–17 in Table 3).

The comparison of what and how IEEE 1471-2000 relationships are supported by AK tools in Table 3 have led to a number of observations:

- *AK tools are architectural description focused.* The AK tools commonly support relationships in modeling (relationship 11 in Table 3), reasoning (relationship 12) and architectural design concerns (relationship 8). It shows that the AK tools focus on the AK knowledge management activities surrounding architectural design analysis, synthesis and evaluation.
- *AK Tools are decision focused.* All of the AK tools use design decisions as one of the key concepts. However, this concept is not represented in the IEEE 1471-2000 standard. The concept of design decision as represented by these tools provides a broader meaning than the definition in the standard. For instance, the relationships in the AK tools represent a series of steps or connectivity to other architectural knowledge, whereas the standard defines the relationship as providing rationale to architecture description. From the analysis, we have found that ADDSS, AREL and Knowledge architect provide more specialized decision-based relationships than the standard.

Table 3

Mapping the relationships between the IEEE 1471-2000 and the AK tools.

IEEE 1471-2000 entities	IEEE 1471-2000 relationships	ADDSS	Archium	AREL	Knowledge architect	PAKME
Mission – system	1. Fulfills	–	–	–	–	–
Environment – system	2. Influences and	1	–	–	–	–
	3. Inhabits					
System – architecture	4. Has	8	–	–	–	–
System – stakeholder	5. Has	4	–	–	–	–
Architecture – architectural description	6. Described by	–	–	7	2, 3	–
Stakeholder – concerns	7. Has	9	1	7	1	1
Architectural description – concern	8. Identifies	1,2	2	1	4	2, 5
Architectural description – viewpoint	9. Selects	–	–	6	–	–
Architectural description – view	10. Organized by	10	–	6	5	4
Architectural description – model	11. Aggregates	6	3	2	9,10	7
Architectural description – rationale	12. Provides	6, 7	4	3	7,8	3, 11
View – viewpoint	13. Conforms to	–	–	–	–	–
View – model	14. Consists of	10	–	–	6	–
Viewpoint – model	15. Establishes methods	–	–	–	–	–
Concerns – viewpoints	16. Covers	–	–	6	–	–
Viewpoint – library viewpoint	17. Has source	–	–	–	–	–

- *Lack high-level architectural relationships.* There is none or little support for relationship types 1 to 5 from Table 3 by the AK tools except for ADDSS which emphasize these concepts. It indicates that the high-level relationships in a system are mostly omitted by most AK tools. The lack of relationships at that level implies that the architectural context of a system is probably not sufficiently represented by the studied AK tools.
- *Weak support for viewpoints and views.* In the standard, there are a number of relationship types that are related to views or viewpoints (relationship 9, 10, 13 to 17 in Table 3). However, in the AK tools, there is very limited support for these relationship types. ADDSS can visualize different architecture views, while AREL provides some viewpoint support similar to architectural frameworks (Open Group, 2003; Zachman, 1987).
- *AK tools provide evolution support.* IEEE 1471-2000 standard does not have relationship types that support architectural evolution. Two of the AK tools, AREL and ADDSS, provide some evolution relationships. Such relationships are defined in the tools to support activities such as architectural analysis and maintenance.
- *Support for architectural knowledge reuse.* IEEE 1471-2000 does not have relationships that support architectural reuse. These relationships are useful in the areas of architecture synthesis and evaluation. Three of the AK tools, PAKME, Archium, and ADDSS, support relationships that allow architects to trace a design to a general design pattern.

From the above comparisons, we have found that all the AK tools are design decision centric. All of the tools support relationships that are geared towards the architectural design activities and decision making. The relationships between key architectural concepts such as design concerns, design decisions, models and design rationale are supported. However, when it comes to the support of viewpoints, there is very limited support by the AK tools. Such lack of support for viewpoints may hinder the ability of an architect to see different perspectives of the architecture. Thus, there are limitations in performing AK activities presented in Fig. 2. The AK tools complemented the IEEE 1471-2000 standard by providing support for architectural reuse and evolution. On the other hand, the IEEE 1471-2000 standard does not prescribe design decision and design patterns that are key relationship concepts in some of the AK tools.

6.3. Comparing AK tool support for architectural analysis, synthesis and evaluation

Architecture analysis, synthesis and evaluation are major activities in the architectural life-cycle. In order to compare their support by AK tools in a meaningful way, we have selected to use a case based on a Cyber Video Company (CVC). Following the comparison of the type and relationship representation of the AK tools, an example case can help to illustrate development life-cycle support of architectural analysis, synthesis and evaluation (i.e. criteria 3, 4 and 5 respectively) of the AK tool.

The case is based on the CVC system, a home entertainment system that delivers to consumers the movies of their choice via satellite. The case involves the definition of software requirements, business rationale, use cases, assumptions, design decisions, design alternatives, workflows models and architecture products. In the case, a group of students have undertaken a project to build a video on demand system. In the architectural design, the design concerns, architectural model and design decisions have been documented. We compare the implementation of AK tools in terms of their support for analysis, synthesis and evaluation using a design decision (DD26) and its associated functional requirement (FR) and non-functional requirements (NFR):

FR01: The user is able to access CVC services through a GUI stored in the Brain-box (BB) and displayed on the TV. The GUI is controlled through the Set-top Box (STB) remote control.

FR06: Serving a request for a popular movie must not exceed 1 hour and 8 hours for non-popular movies in VCR quality and popular movies in DVD quality.

FR23: BB will not shut down due to a user request while it is receiving a movie that was requested.

NFR05: In the case of a failure of the BB during the receiving of an access key for an already paid movie, the key should be retransmitted.

NFR06: In the case of a failure of the BB during the receiving of a requested movie the movie should be retransmitted.

The BrainBox (BB) is a device developed by CVC in order to access and rent media content, a Set-top box, a satellite antenna, a TV and Internet connection. The requirements mentioned above influenced the selected decision (DD26) which was analyzed with its associated information and including the traces. From the analysis of points of failure in a system, a deadline of a request cannot be made. Therefore, when a new movie request makes scheduling infeasible, CVC cannot guarantee delivery within 8 hours or 1 hour. The movie will be delivered late. The solution for this situation implies that the system must inform the user that the deadline cannot be made. As a result, the design decision made was: *DD26: The Scheduler on CVC has a scheduling feasibility test before preceding a request.* This decision had two design options (A and B), which are based on giving the user the ability to cancel or accept the request when bandwidth allocation motivates the system to inform the user that the deadline of a request cannot be satisfied. We used this example to study how each tool would support an architect during the analysis, synthesis, and evaluation activities.

ADDSS – ADDSS models decision DD26 as a free text description with a set of attributes that characterize such decision. In *architectural analysis*, the architect stores the functional and non-functional requirements as the context knowledge that will be used to motivate decision DD26. In addition, ADDSS can store general knowledge in the form of design patterns and architectural styles, organized by different categories (e.g. structural, creational) that can be reused in the creation of design decisions. The ADDSS's meta-model supports design alternatives which are implemented in the tool by means of a category attribute that allows to classify decisions as Main, Derived (i.e. a more specialized decision from a Main one), and Alternative. This classification is also complemented using the Status attribute which indicates the current state of the decision in the life-cycle of the architecture project. The allowed values for the status are: pending, approved, rejected, and obsolete.

For *architectural synthesis*, we stored both alternatives for decision DD26 and we labeled these as alternative decisions with pending status. Each choice is characterized by the set of attributes that describe the decision itself and its rationale. Also, the responsible and a time-stamp are automatically filled by the tool. During the construction of the architecture, explicit links between requirements and decisions is defined. Those requirements used in previous decisions are marked, so the architect exactly knows which requirements are not yet implemented. In addition, ADDSS can define basic dependencies between decisions that are used to represent software constraints or dependencies between requirements. These dependencies form a traceable network of decisions that is used to estimate the impact of modifying, adding, or removing a decision, but validation facilities to check the integrity of the decision model are not yet implemented. Also, ADDSS does not describe the type of the link between two decisions. Because our example deals with an isolated design decision, we did not define dependencies to previous decisions. ADDSS supports different

architecture views as well as decisions with different granularity levels.

For *architectural evaluation*, the decisions are evaluated and the choice selected becomes a “main” decision which should be “approved”. Therefore, the status attribute allow ADDSS users to evaluate decisions after they have been stored. At the end of the evaluation process, a figure containing an image of the architecture which belongs to the result of the decisions made and considered as valid (i.e. approved) is uploaded and shown to the user who can easily browse the evolution of the architectures made along the construction process. Hence, an explicit link between decisions and architectures is defined. Such links provide a complete traceability between decisions and other products of the software life-cycle. Finally, the decisions, requirements, and architectures are documented in ADDSS as PDF documents that can be generated automatically from the tool. The integration to external tools for feeding the requirements from other sources and the integration with architecture modeling tools have not been implemented.

Archium – Archium models the design decision as free text elements, which describe various knowledge entities. Formal code is used to express semantic relationships between these free text elements and the implementation of the solution. For the *architectural analysis*, the architect describes the problem, motivation, and cause for *DD26* first. Context knowledge in the form of software requirements can be stored. These requirements realize different scenarios in Archium. Next, the context of the decision is described, which includes the assumptions and software architecture upon which this decision is based.

For the *architectural synthesis*, two potential solutions are considered for this decision. Each of these is described in the solution part of the decision, as a separate formal element using the Archium language. For each decision, a free text description is provided in addition to the formal elements necessary to implement them, as well as any rules or constraints of a solution are also described using specific free text elements. When the architect adds a decision in Archium, the rationale is also stored with the description of the alternatives. Archium can check for the consistency of architectural design decisions. For instance, whether a functional dependency of a decision is satisfied before the decision is applied. Compared to ADDSS, Archium uses the concept of design fragment, which can include or apply patterns and styles in the architecture (i.e. general knowledge). These reusable fragments are organized in a library which contains at this moment with a few fragments. The reason for this, compared to ADDSS, is because Archium is much more code-oriented than the other tools.

The architectural solution belonging to a set of decisions is modeled in Archium using a component and connector view which allows the architect to perform the modeling tasks using the same tool interface (as opposite to ADDSS which uploads the architecture from an external modeling tool). Other architecture views like the deployment view, is not supported in Archium. The tool can check implementation against architectural decisions. Therefore, if a decision is ignored or disregarded, the tool warns and prohibits violations of the architectural decisions. Also, superfluous decisions can be checked when decisions overlap (i.e. are redundant) or when they are unnecessary (i.e. decisions that do not affect the architecture model at all). During the creation of architectural decisions, traceability between decisions and to other external artifacts is defined in Archium by means of formal and informal relationships. Formal relationships are those relationships defined in the Archium meta-model for which explicit language constructs are provided, and used to determine the impact of an architectural decision and relate it to the components in the architecture. Informal relationships are defined as textual descriptions in the characterization of a design decision and they work similar to hyperlinks to relate two model elements. This is called in Archium the design

decision dependency view, and it defines five different relationships between a decision and a requirement (i.e. creates, obsolete, uses, realizes, and threatens). In our example, the formal relationship “realizes” is used to relate *DD26* with *FR06* and the associated use case, as well as the uses relationship to *NFR05*, *NFR06*, and *FR06*.

For the *architectural evaluation*, the architect records the pros and cons using free text language elements for each solution. After which a trade-off is made between them. This forms the basis of the rationale (a free text element) that motivates the choice for one particular solution. This choice is formally expressed in the Archium language.

AREL – AREL is built with the Enterprise Architecture UML tool to create and visualize architecture entities AREL models which are mainly composed of architecture entities ($\langle\langle AE \rangle\rangle$) that are related to architecture rationale ($\langle\langle AR \rangle\rangle$) elements. AREL uses architecture design rationale as connectors to relate requirements, constraints, and assumptions to design objects. Traces between the different architecture elements and its rationale help explain architecture design, identify change impacts, trace root causes, relate architecture design objects, and verify architecture design among other uses. Design rationale is captured in AREL qualitatively and quantitatively within an $\langle\langle AR \rangle\rangle$ node. Qualitative design rationale provides the arguments for selecting a particular design alternative, whilst quantitative design rationale uses cost, benefit, and risk to quantify the merits of an alternative.

For the *architectural analysis*, AREL support different types of entities like requirements. Functional requirements *FR06*, *FR01* and *FR23* are UML entities that contain textual description of the requirements. AREL uses the $\langle\langle trace \rangle\rangle$ stereotype to link requirements to decisions. Hence, the functional requirements of the CVC product are linked to decision *DD26* using trace links. For the *architectural synthesis*, AREL models decision *DD26* and all associated architecture elements as UML entities. UML models of the architectures we need to build are well supported in AREL, and up to three viewpoints (i.e. business, information, and software design) are supported to model AREL architecture elements from different perspectives. Architects can create AREL model by dragging and dropping elements from a tool-box during design modeling, but not additional general knowledge as reusable patterns is defined. In the CVC example, decision *DD26* is modeled as a UML package and it contains two key elements: the design rationale of the decision and the discarded design option A. The design rationale for decision *DD26* comprises the reasons for choosing the design, the design issues and the assumptions (i.e. about *NFR05* and *NFR06*). Moreover, decision *DD26* is linked to the design artifacts of option B. In this way, the decision can be traced to its design outcomes. Forward and backward traceability is supported by AREL.

For the *architecture evaluation*, the rationale of the decisions being evaluated must contain the strengths and weaknesses of the design options. During the debate of the design alternatives, a number of factors are considered. For instance, the trade-offs of the alternatives using appropriate weights, the risks and non-risks, and an assessment which summarizes the decisions selected and non-selected and the justifications behind them.

The Knowledge architect – The Knowledge architect is a platform to create, manage, and share AK and consists of three related tools: the Knowledge architect Word and Excel plug-ins and the Knowledge architect explorer. The Knowledge architect uses the concept of a knowledge entity to represent different types of knowledge. The purpose of the Word plug-in is to create and use AK in which knowledge entities are special annotations (i.e. special kinds of Word comments) in a Word software architecture document that can be colored to discriminate the different types of knowledge entities.

Starting with the *architectural analysis*, the architect would first add the related requirements (*FR01*, *FR06*, *FR23*, *NFR05* and *NFR06*)

to the knowledge repository. Some may be marked as risks or concerns. This can be performed in the Word plug-in, where the architect selects the text and/or figure(s) representing the aforementioned concepts, fills in a few details in a form, and the tool automatically adds the knowledge to the knowledge repository. To start the analysis, the architect finds or defines a decision topic for DD26 in the document and adds this to the knowledge repository. While doing so, the architect defines the relationships with the requirements, thereby promoting some of them to architectural significant requirements.

For the *architectural synthesis*, the Word Client informs the architect, through textual coloring, about the open decision topic. This helps the architect to focus on the decision topics for which architectural synthesis is still needed. Knowledge entities like a decision can be related to other knowledge entities (e.g. requirements). The relationship to other entities depends on the type of the entity and on the domain model being used. The architect comes up with alternatives A and B to address the pending decision topic. Compared to the other tools, Knowledge architect captures design rationale in a different way with respect to the other tools because it is based on the existence of a software architecture word document in which design decisions and other knowledge entities are marked and colored in the document. Then, knowledge entities are characterized using a template like in previous approaches. In other tools such as ADDSS and PAKME, these templates are filled first, and the documentation containing the knowledge entities is generated afterwards. A refresh menu option has to be used to show the updates made in the word document. In addition, the Knowledge architect explorer tool is used to visualize the different types of knowledge entities and their relationships among them (e.g. the relations of a decision to other decisions or requirements). Also, this tool uses a pop-up menu to visualize the details of each knowledge entities. Compared to the visualization facilities of tools like ADDSS and AREL, is not so straightforward to visualize software architectures in the Knowledge architect explorer.

Finally, in *architectural evaluation*, the tool helps the architect, again with colored text, in spotting the alternatives that have no associated design decision. In this case, the tool highlights alternatives A and B encouraging the architect to choose between them. The choice is documented as DD26 in the repository. To assist an architect, the Word plug-in can assess about the completeness of architectural knowledge using a pop-up menu where red means problematic and green means perfect. Also, a status attribute is used to describe the current state of the knowledge entity (e.g. validated, reviewed), such as, for instance, when knowledge entities have to be validated in a review.

PAKME – Both ADDSS and PAKME are web-based tools for supporting the creation and use of architectural design decisions. Therefore, collaborative and sharing features are easier to implement than in the other studied tools. PAKME provides several features to support AK management for designing and maintaining software architecture of the case. For example, PAKME provides different templates to capture and maintain artifacts and their relationships that are characterized as architectural knowledge by PAKME's data model described in Ali Babar et al. (2006). PAKME uses more detailed templates than ADDSS for characterizing the design decisions. PAKME's model has three main activities: architecture analysis, architectural synthesis, and architectural evaluation.

For the *architectural analysis*, architectural significant requirements (ASRs) are elicited and characterized by concrete scenarios (i.e. context knowledge) that are captured using the knowledge acquisition service through specific templates. Similar to ADDSS, general knowledge can be retrieved from a repository which has been populated with architecture patterns, design primitives, and case studies as well. This generic knowledge can be instantiated

to project-specific knowledge. For example, instantiating abstract scenarios into concrete ones.

During *architectural synthesis*, the architect tries to identify candidate architectural solutions that address ASRs and uses specific templates to codify and capture the decisions that are motivated by the scenarios already captured. The same as in ADDSS, PAKME supports different granularity levels for the decisions being captured. The architect can describe the architecture decision DD26 along with the contextual information using design decision template provided by PAKME which also shows the traceability that has been established between different artifacts. The tool also helps establish and maintain traceability from DD26 to the concrete scenario and quality factor to be satisfied by this design decision. The architect can also establish several types of relationships (such as dependency and constraints) between different architecture design decisions. Compared to ADDSS, PAKME provides a complete list of dependency types between decisions. Also, artifacts created outside PAKME can be attached to each design decision. For example, the architect may want to model decision DD26 with a UML diagram using the AREL tool, and then export XMI or make a graphic file of the model and attached with DD26 captured using PAKME's template.

For the *architectural evaluation*, the architect ensures that the architectural decisions were the right ones. PAKME supports architecture evaluation to visualize the risks. Also, the rationale of design options is captured in PAKME in a separate template. By capturing design options as cases, PAKME enables architects to support a case-based reasoning. In the example of decision DD26, the rationale for each design option (A and B) is captured and viewed. This rationale describes the reasons underpinning the decisions. Finally, reporting facilities for architecture evaluation can be used to represent the relationships between artifacts and show the positive and negative effects between different architectural artifacts.

6.3.1. An analysis of the AK tool applications on the case study

Using the case study to compare AK tools for criteria 3, 4, and 5, we have observed the following. For *criterion 3* (i.e. *architectural analysis*), ADDSS, AREL, Archium, and the Knowledge architect focus on requirement analysis because the requirements are explicitly represented. PAKME, on the other hand, uses scenarios for the problem specification. Archium captures assumptions on which the decision is based to describe the problem, the motivation, and the causes. The Knowledge architect distinguishes between risks and concerns, but both PAKME and ADDSS include project information specific as an add-on to the requirements. PAKME provides more detailed templates for capturing information for the different development phases.

For *criterion 4* (i.e. *architectural synthesis*) ADDSS and PAKME support general knowledge in the form of patterns and architectural styles, while in the other tools this feature is less supported. Also, ADDSS provide a status and category attributes to discriminate between candidate decisions and assign a status depending on the moment the decision is being evaluated. Hence, users can replay the reasoning process that took place at a given moment. Also, the time-stamp and version fields can be used to track the evolution of decisions. PAKME supports different granularity levels for the decisions during the capturing process but ADDSS nicely relate the decisions to general knowledge. All of the studied tools provide traceability mechanisms from decisions to external artifacts as well as to other decisions. In particular, AREL provides a good support and PAKME can model a wide variety of dependencies. Archium provides an important capability as it warns about potential or incompatible violations when selecting the different alternatives. While ADDSS can relate decisions to general knowledge, this is not possible in AREL, but AREL relates a more refined

relationships depicted in UML whereas ADDSS associates the entire architecture with a set of decisions made. The Knowledge architect captures design rationale in a different way, because it is based on the existence of a word document in which design decisions and other knowledge entities are marked and colored in the document.

For *criterion 5* (i.e. *architectural evaluation*) ADDSS provides support to specify if a decision has been pending, approved, rejected, or obsolete but no support for other external evaluation based on quality attributed is available. Further review or assessment procedures can be done on the PDF reports generated by the tool. PAKME can visualize the risks to ensure the decisions taken are the right ones and reporting facilities can be used, like in ADDSS, to represent the relationships between artifacts and show the positive and negative effects between different architectural artifacts. Archium users can review the pros and the cons of the decisions and check the existence of overlapping or incompatible decisions. AREL uses the strengths and weaknesses of the design options contained in the rationale and appropriate weightings of risks to assess about the best or optimal decisions. The Knowledge architect uses the colors to highlight the alternatives and the same as ADDSS, a status attribute to describe the current state of the knowledge entity (e.g. validated, reviewed). All the tools except the Knowledge architect uses the rationale for further trade-off analysis, but ADDSS provides only partial support for this. As a summary of the discussion, we can say that all the tools support the three criteria mentioned before but some of them put more emphasis on some of the activities than the others.

6.4. Comparing AK tool support for implementation, maintenance and others

The comparisons of the AK tools on the ground of implementation, maintenance, customization, integration and collaboration provide insights into different aspects on the use of AK tools. To enable such comparisons, we have provided scenarios to illustrate the criteria. Appendix C describes a scenario for each criterion, and we then analyze how each of the studied AK tools deals with the scenarios. As a result of the evaluation of use cases for criteria 6–10, we have found the following:

Criteria 6 (i.e. *implementation*) all the tools provide some traceability support to help implementation. However, only AREL and Archium offer traceability between implementation artifacts and the design rationale and vice-versa. In Archium, implementation adherence is available through constraint and rule implementation, which is something AREL lacks.

Criteria 7 (i.e. *maintenance*) all the tools provide traceability support to assist users in maintenance tasks when decisions are modified; and analyze the impact of changes both in decisions as well as in the related elements. It is interesting to note that AK tools aim at retaining and reusing AK, therefore traceability support for maintenance purpose is naturally a key feature of such tools.

Criteria 8 (i.e. *customization*) only ADDSS and PAKME offer support for different users with different levels of permissions as they provide project management features. In addition, ADDSS defines optional attributes that can be used by different user profiles to characterize different amount of information of the decisions stored.

Criteria 9 (i.e. *integration with other tools*) ADDSS and PAKME provides integration with PDF documentation but not with external modeling tools. AREL and the Knowledge architect can interact with Word documents to describe architecture models. In addition, the Knowledge architect interacts with

Excel models. Archium lacks any kind of automatic or semi-automatic integration.

Criteria 10 (i.e. *collaborative environment*) supports different users with access privileges but lacks locking mechanisms for concurrent users accessing the same information. Archium is a standalone tool and depends on a Source Code Management (SCM) system. AREL has multi-user support and users can lock and secure the information contained in Word documents to prevent concurrent edits. The Knowledge architect has a versioning system integrated in its knowledge repository to prevent conflicts in case of shared access. PAKME support concurrent access from multiple users and has also collaborative features for distributed teams. The AK tools generally require improved integration with external tools, support more collaborative features, and provide better multi-user features for project management in distributed teams.

7. Findings

In this work, we have compared five different AK management tools in terms of how they support the activities in the architecture life-cycle. It is noted that the authors of this paper are also the researchers behind the creation of the tools that are being compared. This may introduce bias in the comparison. To address this potential issue, we define a set of ten criteria that represent the architectural knowledge activities. The foundation of these criteria is formed by using current research literature as well as an international standard as guidelines. The following are the findings of comparing the AK tools with the criteria for comparison.

7.1. Architectural knowledge representation

From Appendix A, we have found that all the AK tools comprehensively support the IEEE 1471-2000 concepts of rationale and concern. The studied AK tools also provide support for AK *reasoning knowledge* and *context knowledge* that are illustrated in Fig. 2. This shows that researchers generally consider the importance of design context and design reasoning. On the other hand, the level of support for *design knowledge* varies, and it is evident from Appendix A that *architectural models* are not well supported. Archium and AREL support design knowledge representation explicitly because the tools are tightly integrated with the architecture development process. The emphasis of these tools is to *support software architectural design*. However, ADDSS, Archium and PAKME support *general knowledge* representation (e.g. design patterns or tactics) to facilitate *AK reuse*. From the analysis, we notice that there are different knowledge focuses amongst the AK tools and the IEEE standard. It indicates that AK can vary even though the architectural activities are similar. For the future, AK tools should provide a comprehensive and tailorable knowledge representation to allow organizations to support and evolve their AK. Representation and conceptual models should be opened to allow integration with external knowledge sources.

7.2. Architectural knowledge relationships

The AK relationships offered by the studied tools allow knowledge consumers to relate different types of AK. From Table 3, we see that all the tools can trace the design rationale to architectural descriptions. Additionally, these tools have specialized relationships that are not defined in the standard. It indicates that the AK tools emphasize the concept of design reasoning which is largely missing from the IEEE 1471-2000 standard. The AK tools also provide support for architectural evolution support and knowledge reuse support. These key features are missing from the standard.

On the other hand, the AK tools generally lack the relationships to support architectural view and viewpoints. This could limit the capabilities of the knowledge users to retrieve different perspectives of the AK.

7.3. Architectural life-cycle activities

Architecture analysis, synthesis, evaluation, implementation, and maintenance are the key activities in the architecture life-cycle. In the comparison, we have found that all the studied tools are capable of capturing requirements and architectural concerns to support solution design. Furthermore, they all support architectural implementation by providing traceability between artifacts created in different development phases. However, each of them has been designed with a different purpose in mind, and therefore some provides better support for certain stages of the architecture life-cycle than the others. For instance, the goal of Archium was to investigate the relationship between reasoning knowledge and design knowledge. It focuses on the architecture implementation and maintenance stage, not just through traceability but actual round-trip engineering and impact analysis. The goal of the Knowledge architect is to offer a general AK platform, which can be extended for specific purposes, such as software architecture documentation (i.e. the Word Client) or quantitative architectural analysis models (i.e. the Excel Client). Due to these specific clients, the current Knowledge architect tool suite is focused on architectural synthesis and evaluation.

AREL focuses on the synthesis, implementation, and maintenance stages, as it captures design reasoning alongside UML models for use in the implementation and maintenance stages. ADDSS focuses on the analysis and synthesis stages. It captures, manages, and documents design decisions and relates these to architecture products. It also supports evaluation by explicitly documenting the status and categories of alternative solutions. The forward and backward traceability support is also useful for maintenance activities. PAKME supports architecture analysis by providing hundreds of general scenarios. It provides patterns and design decision cases to support architecture synthesis. It captures rationale and supports various architecture evaluation methods. With the different degree of support of the tools for the different phases of the software life-cycle, we observe a need to generalize and enhance the conceptual models of existing AK tools to provide a more comprehensive AK tool that can support all the activities defined in Fig. 2. For instance, combine the concepts of design patterns emphasized in ADDSS and PAKME with the design support offered by Archium and AREL in a single AK tool.

7.4. Supporting features in architectural knowledge management

Architecture customization, integration, collaboration and knowledge retrieval support are important parts of AK tools. Current AK tools offer very limited customization, integration capabilities and collaborative features. The web interface used in ADDSS and PAKME provide better opportunities to enhance collaborative and distributed features. ADDSS include optional attributes that can be tailored to different users that are interested in different types of information items used to characterize the design decisions. The Knowledge architect is integrated with Microsoft Word and Excel to capture and manages AK from different sources and in different formats while it interoperates with other systems through web services. We expect that the customization of AK to serve individual user's needs, tool interoperation, as well as computer-supported collaborative AK management will receive greater attention in the future. In the case study, we have identified the following issues with respect to the supporting features of the studied AK tools:

- Low degree of integration with each other or other knowledge repositories – all the tools have been created as stand-alone tools with little capabilities to export or import AK, they therefore lack the capabilities for collaboration and sharing;
- Limited assessment and evaluation facilities – none of the tools provide any architecture evaluation although they can capture the reasoning if an evaluation method is used and;
- Knowledge retrieval capabilities are simplistic and reactive, all the tools provide search functions that require the architects to take initiative and have a certain level of knowledge about the domain and the design.

8. Conclusions and future work

Research on architecture knowledge management and architectural design decisions recently have led to the development of several architectural knowledge management tools. In this paper, we have compared five architectural knowledge management tools to analyze the current states of research in this area.

In order to carry out the comparison in a uniform way, we have defined a comparison framework based on the AK management activities that take place during the software architecture life-cycle. In the comparison framework, we have defined ten distinct criteria to compare the AK management tool support. These criteria represent how AK management is used in the architecture life-cycle. We have used IEEE 1471-2000 standard for architectural description to compare the entity and relationship representations of the studied AK tools. We have also used a case study and a set of scenarios to illustrate and compare how these tools serve the architectural activities.

The results of the comparison have shown that all AK tools focus on reasoning and contextual knowledge, some tools emphasize on design knowledge whilst others emphasize on general knowledge. The difference of emphasis highlights the two key aspects of knowledge: to support *architectural design activities* and to support *knowledge reuse*. Traceability is well supported by all the tools but the support on knowledge integration and customization are limited. The results of the study have shown that views and viewpoints defined in IEEE 1471-2000 are not well supported by the AK tools, except ADDSS, which emphasizes more than the others on describing architecture views. Conversely, architectural knowledge management aspects such as design evolution and design patterns are the key features of the AK tools, but are neglected in the IEEE standard. In addition, knowledge sharing features are not yet implemented in the tools analyzed, and it is only mentioned as a future capability. Such feature is closely related to multi-user management and collaborative development.

Through this study, we have provided an in-depth description of AK tools and their underlying support of architectural activities. Such knowledge will guide future enhancements to the studied tool. This research has enabled us to identify some of the future research directions for AK management tools:

- tools interoperability to enable AK exchange;
- intelligent AK management tools to providing automated support to architectural activities, e.g. notifications, suggestions, etc and;
- define an encompassing AK representation model that combines the features from different AK models.

This work has already begun, for instances, the Knowledge architect is moving towards a universal knowledge repository on the server and different implemented meta-models on the clients; and we are working on the extraction of concrete AK design models from AREL into a generic design decision pattern in PAKME. PAKME and ADDSS are being analyzing for integrating their data

Appendix A. Mapping IEEE 1471 to AK tool concepts

IEEE 1471-2000 Entities	Archium	ADDSS	AREL	Knowledge architect	PAKME
Mission Environment	N/A N/A	N/A Project	N/A Business env., information systems env. technology env.	N/A N/A	N/A Project
System Architecture Stakeholder Architectural description Rationale	N/A Design fragment Stakeholder Design fragment Architectural design decision, trade-off, decision, motivation, Cause, Solution, Design rules, Design constraints, consequences, pros, cons	System Architecture Stakeholders Architecture description Architectural design decision Open description for design decision (rationale), decision types (category), constraints, dependencies, status	UML models UML models Stakeholder UML models Design rationale, qualitative design rationale, design issue, design assumption, design constraint, design strengths and weaknesses, trade-offs, design decision supporting info., alternative design solution and rationale, quantitative design rationale, cost and benefit of design, implementation and outcome certainty risk	N/A Knowledge entity Stakeholder Artifact Artifact fragment Decision topic, alternative, design concept, quick decision, design decision, specification, decision, trade-off, ranking	N/A Stakeholder Architecture description Support information, architecture decision, design option, design rationale, constraint Assumption, strength, weakness, costs, benefit, complexity, justification, rule, context, trade-offs
Concern	Requirement category, requirement, actor, scenario, problem	Quality attributes Functional and non-functional requirements	Motivational reason/design concerns, functional requirements, non-functional requirements	Risk, requirement, concern, quality attribute	Architectural significant requirement Quality factor Viewtype
Viewpoint	N/A	N/A	Business viewpoint Data viewpoint Application viewpoint Technology viewpoint Viewpoint-based	N/A	
View	N/A	Decision view Architecture view		Topology/view	Architecture view
Library viewpoint Model	N/A Component entity, port, interface, connector, abs. connector, arch modification, delta composition technique, composition config., design fragment composition	N/A Variation points Patterns Architecture styles	N/A Design models/design outcome Data model Application model Technology model	N/A Component	N/A N/A
Concepts not in IEEE 1471 standard	Architectural design patterns	Iterations Decision making activity Evolution support	Evolution support, superseded design concern, superseded design rationale, superseded design	Scenario, analysis result analysis output, number, value, system parameter, quick sys parameter, analysis function/analysis model, mapping, confidence, variability, author, reviewer, review state, reviewable, person	Analysis model, effect of a pattern, pattern, design tactic, user group, user, log, design history, parameters

Appendix B. Relationships supported by the AK tools

Tool	Relationships (RelationshipName:: EntityA: EntityB)
ADDSS	<ol style="list-style-type: none"> 1. Meets::System:Requirements – This relationship links the requirements that have to be satisfied by a particular system 2. ImpactsOn::Requirements:ArchitecturalDesignDecision – This relationship directly links the motivation for making a particular design decisions to one or more requirements 3. Grows::Iteration:ArchitecturalDesignDecision – This relationship describes that decisions are made on the basis on successive refinements (iterations) of the architecture to show the evolution of decisions and architecture products 4. IsInterested::Stakeholders:ArchitecturalViews – This relationship links the interest of the different stakeholders to different architecture views which includes explicitly the decision view 5. AffectedBy::ArchitecturalDesignDecision:Constraints – This relationship models the restrictions affecting a decision which can be motivated by the requirements and causes to model a dependency to other decision 6. DesignOutcome::ArchitecturalDesignDecision:Architecture – This relationships links directly a set of design decisions with its outcome (i.e. a design object) 7. DescribedBy::ArchitecturalDesignDecision:Status/Category/Open description (rationale)/Pattern/Architectural Style/Variation Point – This relationship outlines how a design decision is modeled in terms of the elements mentioned 8. IsSupported::System:Architecture – This relationship provides a direct link between a system and its corresponding architecture 9. Examines::Stakeholder:Requirements – This relationships describes that one or more stakeholders are interested on one or more requirements 10. IsComposed::Architecture:TraditionalViews: This links defines that one architecture can be described by one or several architecture views
Archium	<ol style="list-style-type: none"> 1. DesiredBy::Requirement:Stakeholder – This relationship links the explicit concerns to the stakeholders, which require them 2. Uses::ArchitecturalDesignDecision:Requirement/Scenario – This relationship links the architectural design decisions, which are part of design fragments with the requirements and scenarios they address 3. PartOf::ArchitecturalModification:ArchitecturalDesignDecision – This relationship links the modification of an architecture with an architectural design decision 4. PartOf::Trade-off/Decision/Motivation/Cause/Solution/Design rules/Design constraints/Consequences/Pros Cons:ArchitecturalDesignDecision – This relationship makes the rationale part of the more general architectural design decision concept 5. Composes::DesignFragmentComposition:DesignFragment – This relationship models how a solution of a design decision or a pattern is applied to the design
AREL	<ol style="list-style-type: none"> 1. DesignCause::DesignConcern:DesignDecision – This relationship links the design concerns with the decisions that they influence 2. DesignOutcome::DesignDecision:DesignOutcome – This relationship links a design decision to the results of the decision, i.e. design models 3. DecisionContainment::DesignDecision:DesignRationale – This relationship represents the design rationale that are contained by a design decision 4. DesignEvolution::NewDesignElement:OldDesignElement – This relationship enables architects to trace superseded design elements 5. DecisionEvolution::NewDecision:OldDecision – This relationship enables architects to trace superseded design decisions 6. Viewpoint::Element:Viewpoint – An architecture element and decision is identified by a viewpoint 7. PartOf::ArchitectureElement:Attribute – Information is contained in the architecture element entity
Knowledge architect	<ol style="list-style-type: none"> 1. Has::Stakeholder::Requirement – This relationship links stakeholders with their explicit concerns 2. DescribedBy::KnowledgeEntity:ArtifactFragment – This relationships relates a AK element with a specific part of an Artifact (e.g. a paragraph in a Word document or a particular range of Excel cells) 3. ContainedIn::KnowledgeEntity:Artifact – This relationship links an AK element with the artifact in which it is described 4. Creates::Alternative:Concern – This relationship links the concerns that come up from an alternative 5. PartOf::SystemParameter:Topology/View – This relationship links a system parameter with a specific view 6. PartOf::Component:Topology/View – This relationship links components with one or more views 7. DescribedBy::DecisionTopic/Alternative/DesignConcept/QuickDecision/DesignDecision/Specification/Decision/Trade-off/Ranking:ArtifactFragment – This relationship links rationale elements with parts of artifacts 8. ContainedIn::DecisionTopic/Alternative/DesignConcept/QuickDecision/DesignDecision/Specification / Decision/Trade-off/Ranking:Artifact – This relationship links rationale elements with the artifacts in which they are described 9. DescribedBy::Component:ArtifactFragment – This relationship links an abstraction of one or more system parameters with a part of an artifact 10. ContainedIn::Component:Artifact – This relationship links rationale elements with the artifacts in which they are described

(continued on next page)

Appendix B (continued)

Tool	Relationships (RelationshipName:: EntityA: EntityB)
PAKME	<ol style="list-style-type: none"> 1. Proposes::Stakeholder:Scenario – The relationship links the stakeholder with scenarios, which are proposed by stakeholders. This is a many-to-many relationship that means each stakeholder can propose many scenarios and each scenario can be proposed by many stakeholders 2. Characterises::scenario:Architecturally Significant Requirement (ASR) – This relationship links each scenario with the ASR that has been characterized using that scenario 3. CapturedBy::Architectural Design Decision:Architectural description – This relationship links each architectural design decision with its architectural description 4. DocumentedBy::Architectural description:Architectural views – This relationship links architectural description of each architectural decision with appropriate architectural views used for architectural description 5. Identified::scenarios:analysis model – This relationship relates scenarios with suitable analysis model 6. SatisfiedBy::Scenario:Design tactics – This relationship links scenarios with design tactics that are used to satisfy the required scenarios 7. DevelopedFor::Analysis model:ASR – This relationship relates analysis model with suitable ASRs for which an analysis model has been developed for 8. ImpactOf::Pattern:ASR – This relationship links the patterns with ASRs on which a pattern has negative or positive impact 9. FoundIn:: Tactics:Patterns – This relationship relates design tactics with corresponding design patterns 10. Has:: Architectural decision:design options - This relationship links design options that have been considered with architectural design decision 11. Attachedto::architectural design rationale:Architectural design decision – This relationship relates architectural design rationale with architectural design decisions 12. ExtractedFrom::Scenario:Pattern – This relationship links scenarios with the patterns from which those scenarios have been extracted. A scenario is usually attached with one pattern, however, a pattern can be the source of many scenarios DependsUpon::Architectural Design Decision:Architectural Design Decision – This is a circular relationship that relates an architectural design decision with other architectural design decisions to represent the dependency relationship

Appendix C. A comparison of AK tools with criteria 6–10

Scenarios	ADDSS	Archium	AREL	Knowledge architect	PAKME
Criterion 6 – A developer is implementing a design, which must be enforced so during implementation the design is not violated. The rules, constraints and context need to be adhered to during implementation	ADDSS does not support modeling. However, it allows check the decisions and the requirements linked to the architectures uploaded into the tool. The architect can visualize the architectures and their underpinning decisions along the development process. In addition, patterns can be reused as proven design choices	Archium enables a user to express part of the architecture in a Java-based ADL. Archium's compiler checks for constraints such as communication integrity constraints, i.e. constraints on what and how the pieces of code communicate with each other. Some rules and constraints can also be textually expressed with different levels of scope. The run-time visualization uses these scopes to inform an architect about the specific rules and constraints	AREL provides traceability support between design elements and design decisions. An architect can select one or more design concerns and have the system automatically trace to all interrelated design components, design decisions and requirements. The results of the search are displayed in a UML diagram. It can help programmers to adhere to the architectural design. Design evolution is also supported. AREL cannot enforce architectural design adherence	Knowledge architect does not support implementation tools. However, it helps implementers by providing traceability among architectural decisions, their context, and constraints	PAKME helps describe architecture in terms of design decisions, captured as cases along with contextual information, constraints, and rationale behind a design decision. Availability of the contextual information and rationale provided by each design decision case can help understand the context and reasoning for each architectural design decisions as well as their constraints, and rules

<p>Criterion 7 – An architect needs to modify a system in production which is new to her. She needs to analyze the impact of the changes to the existing architecture</p>	<p>During maintenance, an architect can revisit and the decisions modified and visualize which architectures are impacted by changes. The trace links also help track the root causes of changes and know which requirements can be affected by displaying the architectures and requirements affected by a decision. ADDSS can only display entire architecture products affected by decisions, not single design artifacts</p>	<p>The Archium's run-time platform helps visually inspect the AK of a running system. An architect any use the visualization feature to trace the dependencies among the architectural design decisions underlying the architecture design</p>	<p>When a requirement or a design artifact is modified, the impact of the modification can be traced forward to a detailed design or backwards to the requirements. For this kind of scenarios, AREL provides traceability support and probability estimation of the level of impacts a change may cause</p>	<p>Knowledge architect's Word plug-in enables a reader to follow traceability links among software architecture documents. The Explorer enables one to view and filter relationships between various types of knowledge entities. These features helps perform impact analysis</p>	<p>PAKME enables an architect to retrieve the architecture decisions taken, design options considered, rationale for choosing a certain design option, trade-offs made, and findings of architecture evaluation. Such information can help the architect to understand original design decision as well as the potential effects of any modifications in the architecture</p>
<p>Criterion 8 – An organizational policy might require members with different privileges to access AK. The users may like to customize the tool for personalized features</p>	<p>The current version of ADDSS supports different user roles and enables privileges for accessing the information of projects and architectures. ADDSS provides also some personalization mechanism, as users can select certain optional attributes for capturing and documenting their design decisions</p>	<p>Archium does not support different levels of viewing and editing permissions. This responsibility has been left to Source Code Management systems (e.g. SVN, CVS, Visual Source Safe, etc.) for maintaining the Archium source code containing the codified AK</p>	<p>AREL does not support different levels of viewing privileges. It also does not support customized preferences</p>	<p>The Knowledge architect suite does not offer support for making differences in privileges</p>	<p>PAKME allows an organization to implement a security policy with different levels of access to different architectural artefacts. A user's access privileges are attached to login account and any request to retrieve/modify an artefact is accommodated based on a user's privilege</p>
<p>Criterion 9 – An architect is describing architecture in a text document, which needs to be stored a tool's knowledge repository</p>	<p>ADDSS only provides integration with a graphical library for generating thumbnail images of the architectures uploaded to the tool as well with PDF documents containing AK stored in the tool. These documents are generated automatically</p>	<p>The architect has to manually convert the concepts in the architecture documents into Archium code, thereby formalizing the software architecture</p>	<p>AREL is built on a UML tool Enterprise Architect. Users may enter different AK into the UML repository. If the knowledge is in a document form such as Word, a document link can be created in an AK entity to that external document</p>	<p>Knowledge architect provides a repository accessible through APIs and a DB interface. It provides plug-ins for MS Word and Excel that can annotate AK and export it to the repository</p>	<p>PAKME is not fully integrated with any other tool. An architect can use templates to capture decisions and rationale. Any associated documents can be uploaded as attachments to each decision. PAKME can also generate PDF reports based on the artefacts in the repository</p>
<p>Criterion 10 – The chief architect and 2 architects responsible for different subsystems are creating knowledge entities in their respective areas. They do not want to overwrite each other's work</p>	<p>ADDSS supports concurrent users but not locking mechanisms. Users can access their own information on behalf of system privileges and enable access to other users as a way to share the decisions for a given architecture or project</p>	<p>Archium is dependant on the use of an external Source Code Management system (SCM) to support this scenario. Using such a system, Archium supports isolated creation and editing of design decisions and their associated architectural modifications</p>	<p>AREL has multi-user support. Users may lock and secure information to stop concurrent edits</p>	<p>The Knowledge architect has a versioning system integrated in its knowledge repository. Shared access of knowledge is thus guaranteed to prevent conflicts</p>	<p>PAKME supports concurrent access to the knowledge base for multiple users. It doesn't have locking/unlocking mechanism. However, it has several collaborative features to support distributed software development</p>

models in order to provide a unified web-based tool which can also support decisions specific to product line architectures.

References

- Avgeriou, P., Kruchten, P., Lago, P., Grisham, P., Perry, D., 2007. Architectural knowledge and rationale – issues, trends, challenges. *ACM SIGSOFT Software Engineering Notes*, 41–46.
- Avritzer, A., Weyuker, E.J., 1998. Investigating metrics for architectural assessment. In: *Proceedings of the Fifth International Software Metrics Symposium*, pp. 4–10.
- Ali Babar, M., Gorton, I., 2007. A tool for managing software architecture knowledge. In: *Proceedings of the Second Workshop on Sharing and Reusing Architectural Knowledge (ICSE Workshops)*.
- Ali Babar, M., Gorton, I., Jeffery, D.R., 2005. Capturing and using software architecture knowledge for architecture-based software development. In: *Proceedings of the Quality Software International Conference (QSIC '05)*, pp. 169–176.
- Ali Babar, M., Gorton, I., Kitchenham, B., 2006. A framework for supporting architecture knowledge and rationale management. In: Dutoit, A.H., McCall, R., Mistrik, I., Paech, B. (Eds.), *Rationale Management in Software Engineering*. Springer, pp. 237–254.
- Ali Babar, M., de Boer, R.C., Dingsøyr, T., Farenhorst, R., 2007. Architectural knowledge management strategies: approaches in research and industry. In: *Second Workshop on SHaring and Reusing Architectural Knowledge – Architecture, Rationale, and Design Intent (SHARK/ADI 2007)*.
- Ali Babar, M., Northway, A., Gorton, I., Heuer, P., Nguyen, T., 2008. Introducing tool support for managing architectural knowledge: an experience report. In: *Proceedings of the 15th IEEE International Conference on Engineering Computer-Based Systems (ECBS '08)*, pp. 105–113.
- Bass, L., Clements, P., Kazman, R., 2003. *Software Architecture in Practice*. Addison Wesley, Boston.
- Bengtsson, P., Bosch, J., 1998. Scenario-based software architecture reengineering. In: *Proceedings of Fifth International Conference on Software Reuse*, pp. 308–317.
- Bosch, J., 2004. Software architecture: the next step. In: *Software Architecture: First European Workshop, EWSA 2004*, St Andrews, UK, pp. 194–199.
- Bratthall, L., Johansson, E., Regnell, B., 2000. Is a design rationale vital when predicting change impact? – a controlled experiment on software architecture evolution. In: *Second International Conference on Product Focused Software Process Improvement*, pp. 126–139.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., 1996. *Pattern-Oriented Software Architecture – A System of Patterns*. Wiley.
- Capilla, R., Nava, F., Pérez, S., Dueñas, J.C., 2006. A web-based tool for managing architectural design decisions. In: *Proceedings of the First Workshop on Sharing and Reusing Architectural Knowledge*.
- Capilla, R., Nava, F., Dueñas, J.C., 2007a. Modeling and documenting the evolution of architectural design decisions. In: *Proceedings of the Second Workshop on Sharing and Reusing Architectural Knowledge*.
- Capilla, R., Nava, F., Tang, A., 2007b. Attributes for characterizing the evolution of architectural design decisions. In: *Proceedings of the Third International IEEE Workshop on Software Evolvability, IEEE CS*, pp. 15–22.
- Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J., 2002. *Documenting Software Architectures: Views and Beyond*. Addison Wesley.
- Conklin, J., Begeman, M., 1988. gIBIS: a hypertext tool for exploratory policy discussion. In: *Proceedings of the 1988 ACM conference on Computer-Supported Cooperative Work*, pp. 140–152.
- Conklin, E., Burgess-Yakemovic, K.C., 1996. A process-oriented approach to design rationale. In: Moran, T., Carroll, J. (Eds.), *Design Rationale: Concepts, Techniques and Use*. Lawrence Erlbaum Associates, pp. 393–427.
- de Boer, R.C., Farenhorst, R., Lago, P., van Vliet, H., Clerc, V., Jansen, A., 2007. Architectural knowledge: getting to the core. In: *Third International Conference on the Quality of Software Architectures (QoSA)*.
- Dobrica, L., Niemela, E., 2002. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering* 28, 638–653.
- Dueñas, J.C., Capilla, R., 2005. The decision view of software architecture. In: *Proceedings of the Second European Workshop on Software Architecture (EWSA 2005)*. Springer-Verlag, pp. 222–230.
- Farenhorst, R., Lago, P., Vliet, H.V., 2007. Effective tool support for architectural knowledge sharing. In: *First European Conference on Software Architecture (ECSA'07)*, pp. 123–138.
- Farenhorst, R., Izaks, R., Lago, P., Vliet, H.V., 2008. A just-in-time architectural knowledge sharing portal. In: *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pp. 125–134.
- Harrison, N., Avgeriou, P., Zdun, U., 2007. Architecture patterns as mechanisms for capturing architectural decisions. *IEEE Software*.
- Hipergate, 2007. An open source CRM and groupware system.
- Hofmeister, C., Kruchten, P., Nord, R.L., Obbink, H., Ran, A., America, P., 2005. Generalizing a model of software architecture design from five industrial approaches. In: *Fifth Working IEEE/IFIP Conference on Software Architecture (WICSA 2005)*, pp. 77–88.
- Hughes, T., Martin, C., 1998. Design traceability of complex systems. In: *Fourth Annual Symposium on Human Interaction with Complex Systems*, pp. 37–41.
- IEEE, 2000. *IEEE Recommended practice for architecture description of software-intensive system (IEEE Std 1471-2000)*. IEEE Computer Society.
- ISO/IEC, 2007. *ISO/IEC 42010:2007 Systems and software engineering – Recommended practice for architectural description and software-intensive systems*, p. 23.
- Jackson, M., 1995. *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*. ACM Press/Addison-Wesley Publishing Co.
- Jansen, A., Bosch, J., 2004. Evaluation of tool support for architectural evolution. In: *Nineteenth International Conference on Automated Software Engineering (ASE'04)*, pp. 375–378.
- Jansen, A., Bosch, J., 2005. Software architecture as a set of architectural design decisions. In: *Proceedings Fifth IEEE/IFIP Working Conference on Software Architecture*, pp. 109–120.
- Jansen, A., Vries, T.D., Avgeriou, P., Veelen, M.V., 2008. Sharing the architectural knowledge of quantitative analysis. In: *Proceedings of the Quality of Software Architectures (QoSA 2008)*.
- Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere, J., 1998. The architecture tradeoff analysis method. In: *Proceedings of the Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '98)*, pp. 68–78.
- Kruchten, P., Lago, P., Vliet, H.v., 2006. *Building up and Reasoning about Architectural Knowledge, Quality of Software Architecture (QoSA)*. Springer-Verlag.
- Lee, J., Lai, K., 1996. What is design rationale? In: Moran, T., Carroll, J. (Eds.), *Design Rationale – Concepts, Techniques, and Use*. Lawrence Erlbaum, New Jersey, pp. 21–51.
- Maclean, A., Young, R., Bellotti, V., Moran, T., 1996. Questions, options and criteria: elements of design space analysis. In: Moran, T., Carroll, J. (Eds.), *Design Rationale – Concepts, Techniques and Use*. Lawrence Erlbaum, New Jersey, pp. 53–105.
- Maranzano, J.F., Rozsypal, S.A., Zimmerman, G.H., Warnken, G.W., Wirth, P.E., Weiss, D.M., 2005. Architecture reviews: practice and experience. *IEEE Software* 22, 34–43.
- Perry, D.E., Wolf, A.L., 1992. *Foundation for the study of software architecture*. ACM SIGSOFT Software Engineering Notes 17, 40–52.
- Ramesh, B., Jarke, M., 2001. Towards reference models for requirements traceability. *IEEE Transactions on Software Engineering* 27, 58–93.
- Regli, W.C., Hu, X., Atwood, M., Sun, W., 2000. A survey of design rationale systems: approaches, representation, capture and retrieval. *Engineering with Computers*, 209–235.
- Shaw, M., DeLine, R., Klein, D.V., Ross, T.L., Young, D.M., Zelesnik, G., 1995. Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering* 21, 314–335.
- Shipman III, F., McCall, R., 1997. Integrating different perspectives on design rationale: supporting the emergence of design rationale from design communication. *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing* 11, 141–154.
- Tang, A., Jin, Y., Han, J., 2007. A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software* 80, 918–934.
- Tang, A., Han, J., Vasa, R., 2009. Software architecture design reasoning: a case for improved methodology support. *IEEE Software*, 43–49.
- The Open Group, 2003. *The Open Group Architecture Framework (v8.1 Enterprise Edition)*. The Open Group.
- Tyree, J., Akerman, A., 2005. Architecture decisions: demystifying architecture. *IEEE Software* 22, 19–27.
- Wang, Z.Y., Sherdil, K., Madhavji, N.H., 2005. ACCA: an architecture-centric concern analysis method. In: *Fifth IEEE/IFIP Working Conference on Software Architecture*, pp. 99–108.
- Zachman, J., 1987. A framework for information architecture. *IBM Systems Journal* 38.

Antony Tang is a senior lecturer at Swinburne University of Technology. His research interests are software architecture design reasoning, software engineering techniques and vehicle software systems. He has been working in the IT industry since 1984. He received his PhD in IT from Swinburne University of Technology in 2007. He's a member of the IEEE Computer Society and ACM. Contact him at atang@swin.edu.au.

Paris Avgeriou is Professor of Software Engineering at the Department of Mathematics and Computing Science, University of Groningen, the Netherlands. He heads the Software Engineering research group since September 2006. He has participated in a number of national and European research projects on software engineering, that are directly related to the European industry of Software-intensive systems. He has been co-organizing international workshops in conferences such as ICSE, ECOOP, ICSR, UML, ACM SAC and editing special issues for journals like IEEE Software. He is in the editorial board of Springer TPLOP. He is a member of IEEE, ERCIM, Hillside Europe and acts as a PC member and reviewer for several conferences and journals. He has received awards and distinctions for both teaching and research and has published more than 80 articles in peer-reviewed international journals, conference proceedings and books. His research interests concern the area of software architecture, with a strong emphasis on architecture modeling, knowledge, evolution and patterns. Contact him at paris@cs.rug.nl.

Anton Jansen is a scientist at ABB corporate research in the software architecture & usability (SARU) group in Västerås, Sweden since 2009. Between 2002 and 2009, he was a member of the Software Engineering and Architecture (SEARCH) research group at the University of Groningen, the Netherlands. He received a master of science degree in computing science in 2002, as well as a Ph.D. in Software Architecture in 2008 from the University of Groningen, the Netherlands. He has worked as a Ph.D. associate (2002–2006) on architectural decisions under supervision of Jan Bosch, and as a postdoc (2006–2008) on the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) project GRIFFIN. His research interests are software architecture and architectural knowledge. In his spare time, he likes to play multiplayer computer games, cycling, and reading books. Contact him at antony@ca.rug.nl.

Rafael Capilla is an assistant professor of software engineering in the Universidad Rey Juan Carlos of Madrid (Spain) since 2000. Previously, he worked as system analyst and a Unix system administrator for several years. He holds a PhD in Computer Science, and his research interests are software architectures, particularly decision-making processes, product line engineering, software variability, and

Internet technologies. Capilla is a member of IEEE CS. Contact him at rafael.capilla@urjc.es.

Muhammad Ali Babar is a Senior Researcher with Lero, University of Limerick, Ireland, where he leads projects on software architecture and empirical assessment of software development technologies. Previously, he worked with National ICT Australia (NICTA). He has authored/co-authored more than 90 peer-reviewed publications. He has co-edited a book, software architecture knowledge management: theory and practice. Dr. Ali Babar has also been a co-guest editor of special issues of IEEE Software, JSS, IST, and ESEJ. He has presented tutorials in the areas of software architecture and empirical approaches at various international conferences including ICGSE09, XP08, ICSE07 and WICSA07. Apart from being on the program committees of several international conferences such as WICSA/ECSA, ESEM, SPLC, ICGSE, and ICSP, Dr. Ali Babar is program chair of ECSA2010 and program co-chair of PROFES2010. Prior to joining R&D field, he worked as a software engineer and an IT consultant for several years in Australia. His current research interests include software architecture, software product lines, and evidence-based software engineering. Contact him at Muhammad.AliBabar@lero.ie.