Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software xxx (2006) xxx–xxx
The Journal of Systems and Software
www.elsevier.com/locate/jss

Using Bayesian belief networks for change impact analysis in architecture design ☆

Antony Tang [a,*], Ann Nicholson [b], Yan Jin [a], Jun Han [a]

[a] Faculty of ICT, Swinburne University of Technology, John Street, Hawthorn, Melbourne, Vic. 3122, Australia
[b] Clayton School of Information Technology, Monash University, Melbourne, Australia

Received 16 December 2005; received in revised form 24 March 2006; accepted 4 April 2006

Abstract

Research into design rationale in the past has focused on argumentation-based design deliberations. These approaches cannot be used to support change impact analysis effectively because the dependency between design elements and decisions are not well represented and cannot be quantified. Without such knowledge, designers and architects cannot easily assess how changing requirements and design decisions may affect the system. In this article, we introduce the Architecture Rationale and Element Linkage (AREL) model to represent the causal relationships between architecture design elements and decisions. We apply Bayesian Belief Networks (BBN) to AREL, to capture the probabilistic causal relationships between design elements and decisions. We employ three different BBN-based reasoning methods to analyse design change impact: predictive reasoning, diagnostic reasoning and combined reasoning. We illustrate the application of the BBN modelling and change impact analysis methods by using a partial design of a real-world cheque image processing system. To support its implementation, we have developed a practical, integrated tool set for the architects to use.

Keywords: Design rationale; Software architecture; Bayesian belief networks

1. Introduction

Design rationale plays an important role in the design and subsequent maintenance of large and complex systems. It supports reasoning and it captures why design decisions are made (Moran and Carroll, 1996). Design rationale has many benefits. It can be used to verify and trace the design of a system as well as to support its enhancements. Despite its usefulness, design rationale is often not documented and the knowledge is evaporated or eroded after the design is completed (Bosch, 2004; Perry and Wolf, 1992). Without such knowledge, impacts of proposed changes to the system cannot be assessed accurately. This issue is accentuated when the design decisions that cater for the non-functional requirements cross-cut each other and are intertwined within the system. These types of requirements or design decisions are typically architectural in nature and have a profound impact on a project (Cysneiros and Leite, 2001).

Architecture design deals with the load-bearing issue (Perry and Wolf, 1992) or fundamental organisation of a system (IEEE, 2000). Changes in the architecture after its initial design can have a major impact on the system. Since rationales of architecture design decisions are often not documented, the knowledge about the design decisions such as assumptions, constraints and design alternatives are not communicated or retained in a systematic way. Problems may arise when the expertise is no longer available (Han, 1997) or the knowledge is forgotten (Tang et al., 2005a).

The approach taken in this paper is to record the tacit knowledge about the architecture decisions in a structured

☆ This article is an extended version of the WICSA'05 paper (Tang et al., 2005b).

* Corresponding author. Tel.: +61 3 92145198; fax: +61 3 98190823.
E-mail addresses: atang@ict.swin.edu.au (A. Tang), ann.nicholson@infotech.monash.edu.au (A. Nicholson), yjin@ict.swin.edu.au (Y. Jin), jhan@ict.swin.edu.au (J. Han).

0164-1212/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2006.04.004

way to enable these decisions to be traced. This can be achieved by employing a representation of the causal relationships between the decisions and their related architecture elements. The captured knowledge can be traced through traversing the relational representation. Thus, it helps software engineers who are not familiar with the architecture design to assess the potential impacts of the proposed enhancements. In order to predict and diagnose the extent of change impacts in the architecture design, we use probability calculus to quantify the dependencies between the cause and effect of the design decisions and architecture elements.

In our earlier work, we proposed the Architecture Rationalisation Method (ARM) (Tang and Han, 2005) for capturing architecture rationale so as to achieve the traceability and verifiability of architecture design decisions. The context of the architecture design we consider follows the IEEE 1471-2000 standard where multiple viewpoints are used (IEEE, 2000). They typically include viewpoints such as requirements, information, computation and engineering. Architecture decisions are the results of the multitude of influences across these views. When a decision is made, an *architecture rationale* (AR) captures the rationale of the design decision and links the rationale to the related *architecture elements* (AE) such as requirements, design components, data models or implementation components. This mechanism provides the traceability between design rationales and various parts of the architecture design.

In this article, we further enhance ARM to allow architects and designers to apply reasoning based on the design decisions. We use Bayesian Belief Networks (BBN) to quantify the relationship between design decisions and design elements. BBNs (Pearl, 1988) are graphical models for probabilistic reasoning, which are now widely accepted in the AI community as practical and intuitively appealing representations for reasoning under uncertainty. This article makes the following contributions:

- We provide a graphical model, Architecture Rationale and Element Linkage (AREL), to represent the causal relationships between architecture design decisions and architecture elements. This model simplifies the design rationale representation by encapsulating design rationale in a design decision node.
- We represent the AREL model as a BBN. The BBN graphical structure models and quantifies the causal relationship between architecture design decisions and architecture elements.
- Based on the AREL causal relationship in a BBN model, we employ three different probability-based reasoning methods to carry out change impact analysis:
  - *Predictive Reasoning* – predict what design elements might be affected if one or more architecture elements are to change (e.g. requirements).
  - *Diagnostic Reasoning* – if one or more architecture elements (e.g. data model) are to change, diagnose what might be the causes of such change.
  - *Combined Reasoning* – by combining the use of predictive reasoning and diagnostic reasoning, reason about the ripple effect of the likely changes to the system through diagnosing the causes and predicting the effects.
- We have provided an integrated tool set to support its practical application in an industry-setting.

Section 2 of this paper describes the background and related research. Section 3 introduces a definition of the AREL model. Section 4 describes how BBNs represents the AREL architecture decision network and discusses the three reasoning methods for change impacts analysis. Section 5 describes the tool implementation. We discuss the potentials and the limitations of this work in Section 6 and conclude in Section 7.

2. Background and related work

As evidenced by the industry standard (IEEE, 2000) and accepted by the software architecture community (Tyree and Akerman, 2005), design rationale is an important part of architecture design. In an empirical study (Bratthall et al., 2000), it was shown that the use of design rationale can help designers to understand the change impact of proposed software changes and to work faster and better. Despite its importance, the capture and the documentation of design rationale is often omitted in practice (Bosch, 2004).

Argumentation-based design rationale approaches attempt to tackle this issue by capturing the design deliberation, with a range of different methods (Conklin and Begeman, 1988; Maclean et al., 1996). An argumentation-based rationale system such as gIBIS (Conklin and Begeman, 1988) represents issues, positions and arguments in a graphical way to form a network of nodes capturing the design process. QOC (Maclean et al., 1996) uses questions, options and criteria to enforce a structure of how alternative design decisions are considered, i.e. capturing design choices. DRL (Lee and Lai, 1996) uses a language to represent the process of obtaining design rationale. These approaches facilitate issue-based requirements definition and refinement by exploring typed relationships between issues, positions and arguments. The requirements and design elements involved in the decisions are not linked to the decisions at a level of granularity where cross tracing between the decisions and the design elements can be performed easily (Shum and Hammond, 1994). It was argued that the missing relationships between the decisions and their related design elements prevent these models from being practical (Regli et al., 2000).

Other approaches such as the Architecture Frame (Rapanotti et al., 2004), the KAOS (Dardenne et al., 1993), the non-functional requirement refinement process (Mylopoulos et al., 1992) and the non-functional requirement framework (Chung et al., 2000) all provide ways to

decompose and refine requirements to develop the design. All these structured approaches have limited support for design rationale: (a) they do not record design rationale in their models; and (b) they do not provide a way to carry out trade-off analysis amongst alternative design decisions. Although Ramesh and Jarke (2001) identified different link types to capture the relationships between the design elements and design rationales to trace and explain design decisions, it does not support change impact analysis.

Bosch (2004) argues that since design decisions can be cross-cutting and intertwined, a first-class representation of the design decisions is therefore required to reduce the high maintenance costs. Dutoit and Paech (2000) identify that rationale knowledge needs to be cost-effective, complete, easily accessible and consistent. These qualities are essential for design rationale to be useful in the field. Burge (2005) and Kruchten et al. (2005) suggest that software system maintenance requires the use of design rationale to trace and analyse change impacts. However, current approaches (Burge, 2005; Conklin and Begeman, 1988; Lee and Lai, 1996; Maclean et al., 1996) lack a convenient way to support traceability and change impact analysis of architecture design rationales and design elements.

We developed the Architecture Rationalisation Method (ARM) (Tang and Han, 2005) to address the issue of architecture design rationalisation and its representation. ARM uses architecture viewpoints to organize the requirements, information model, computation model and deployment model of the architecture design (IEEE, 2000).

Fenton and Neil (2000) believe that management decision support tools must be able to handle causality, uncertainty and combining different (often subjective) evidence, and suggests a solution based on BBNs. BBNs have been used in many applications (see Korb and Nicholson (2004) for a recent survey) including a causal model to detect user interactions with user interface in safety-critical systems which are prone to human errors (Galliers et al., 1999).

In this article, we use BBN to model and quantify the probability of the causal relationships between design decisions and design elements. Given the BBN representation of AREL, we show how BBN reasoning algorithms can be used to reason about change impacts. More specifically, architects can use BBN to undertake *what-if* analysis, to predict and diagnose impacts given complex combinations of requirement and design changes.

3. Architecture rationale and causality

Design reasoning takes place in any system design process, but most of the time design rationale are not captured. A structured design rationalisation process would provide a systematic way to make this knowledge explicit and the retention of such knowledge can support system maintenance. This section describes a semi-formal design rationale model for capturing the qualitative and quantitative design reasoning.

3.1. Architecture design rationale

As mentioned earlier, the argumentation-based design rationale approaches use design reasoning and design deliberation to help designers clarify design questions. However, they can be improved in a number of ways. Firstly, it is a cognitive burden to capture the complete explanations initially and designers who want to make use of this knowledge at a later stage most likely need not to replay the deliberation process as it was captured (Gruber and Russell, 1996). A second issue of the argumentation-based approach is that the design artefacts being discussed do not appear in the representation itself and are not linked to it in a defined way (Potts, 1996). For designers who have to maintain a system, it is the design elements which are the focal point of investigation. For instance, a designer may ask "If a requirement is changed, which classes and data models might be affected and how?" A third issue is that decisions are often inter-linked and inter-dependent, the representation of such relationships in an argumentation-based model would be difficult to trace. In order to address these issues, we adopt a different approach to encapsulate and relate architecture rationale to design elements.

ARM uses the UML graphical notation for design rationale modelling (Tang and Han, 2005). There are two types of elements in ARM: architecture elements (AE) and architecture rationale (AR). The causal relationships between AEs and ARs are represented by the UML stereotyped associations that are directional arcs. A basic form of the ARM construct is $\{AE_1, AE_2, \ldots\} \rightarrow AR \rightarrow \{AE_a, AE_b, \ldots\}$ where AE_1, AE_2 etc. are the inputs or the causes of a decision AR and AE_a, AE_b etc. are the outcomes of the decision. The input set of AEs and the outcome set of AEs must be non-empty and are linked by the single decision AR.

An AE can be an input or an output or both when it is involved in two decisions. As an input, it can be a requirement, a use case, a class or an implementation artefact. As an outcome, it can be a refined requirement, a new design element or a modified class and so on.

ARM does not capture the deliberation process but instead it captures the result of the design reasoning in an AR element. An AR element represents a decision and contains the design rationale similar to that of Clements et al. (2002) and Tyree and Akerman (2005):

- the issue of the decision,
- the design constraints,
- the design assumptions,
- the strengths and weaknesses of the design decision,
- the tradeoffs made in a design decision,
- risks and non-risks of this design alternative,
- quantitative justifications of a design alternative by costs, benefits and risks,
- any alternative design which have been discarded and their considerations.

Since an AR encapsulates the justifications of a design decision, designers could use it to recall the decision reasoning. Designers can find out why certain input AEs are considered, why the outcome AEs are created or modified and what other alternatives have been discarded. During the design process, the decisions are incrementally made and they are connected via the AEs and ARs. The chain of dependency between AR and AE elements therefore represents the decision making process of the system architecture design. It also supports the traceability of the causal relationship.

The ARM model has several advantages over argumentation-based approaches from the perspective of its application and support for impact analysis:

- *Simplification* – AR records the key design issues, argumentation and design alternatives without explicitly capturing the design deliberation relationships. We argue that in most cases the design deliberation relationships are not required by the designers and they are time-consuming to capture. Hence the ARM approach simplifies this process by capturing the justifications of the decisions without the overhead.
- *Encapsulation* – design rationale are encapsulated in an AR. This way the decisions can be incorporated into the design process naturally to show the causal relationships between design elements without over-complicating its representation. Should designers require more details about a decision, the details can be revealed by exploring the AR package. Our implementation in an UML tool supports this feature.
- *Causal Relationship* – since AR acts as a connector between the cause architecture elements and the effect architecture elements, we can construct a graphical representation showing direct dependencies between elements and decisions. Such relationships could then be analysed quantitatively and qualitatively to understand the design reasoning.

In the next section, we formalise the definition of the ARM design rationale model.

3.2. Architecture rationale and elements linkage

ARM uses the UML graphical notation. All AEs in the UML diagrams are stereotyped by ≪AE≫. All ARs in the diagram are stereotyped by ≪AR≫. The relationships between them are connected by a directional association stereotyped ≪ARtrace≫. The ≪ARtrace≫ stereotype represents the causal relationship between AEs and ARs. It is also referred to as *link* in the AREL definition below. The representation of the AREL model in UML should facilitate easy adoption by the software industry. An example AREL model is shown in Fig. 1.

The following definition of the AREL model formalises the relationship between ARs and AEs.

Definition 1. An Architecture Rationale and Element Linkage (AREL) model is a tuple (AE, AR, PL), where AE is a set of nodes representing architecture elements, AR is a set of nodes representing architecture rationales, and $PL \subseteq (AE \times AR) \cup (AR \times AE)$ is a set of directed links between the nodes, such that

(1) all rationale nodes must be associated by links with at least one cause and one effect: $\forall r \in AR$, $\exists e, e' \in AE$ such that (e, r), $(r, e') \in PL$;
(2) no subset of links in PL form a directed cycle.

According to Definition 1, an AR is connected to a minimum of two AE nodes through links, one from a cause AE and one to an effect AE. As such, the links to the AEs then represent the cause and effect of the design decision represented by AR.

Clause 2 of Definition 1 specifies that insertion of a link is not allowed if it would result in a directed cycle of links. Essentially, this means that the links maintain the integrity of the causal modelling whereby something cannot be the cause of itself, directly or indirectly. The links will be used in the causality analysis described in Section 4.

Fig. 1 shows the AREL diagram for the high-level design of a reporting sub-system. The functional requirement *Sales Report* and the non-functional requirement *Response Time* are represented by two ≪AE≫ nodes. *AR1* justifies the design by de-normalising the Sales Table so that external joins can be eliminated to improve efficiency, and use a generic reports to produce standard report formats. As a result, two design objects *De-normalised Sales Table* and *Generic Reporting Function* are created. The relationships between *AEs* and *AR* are connected by ≪ARtrace≫ associations or links. In another part of the AREL decision graph, three input AEs are inputs to decision *AR2*:

(1) *SQL Lookup* is a class to locate a SQL statement based on the required business function;
(2) *Generic Reporting Function* specifies the format of report; and
(3) *De-normalised Sales Table* supplies the data.

Together these AEs influence the decision *AR2* such that the optimal way to implement the design is to use embedded SQL statements in stored procedures. This design process continues as architects make design decisions by following design principles to satisfy requirements and creating data models, design objects or implementation artefacts.

3.3. Avoiding cyclic decisions in AREL Models

During the design process, architects may inadvertently cause a reasoning cycle to form as design decisions are incrementally made. For instance, this could happen when

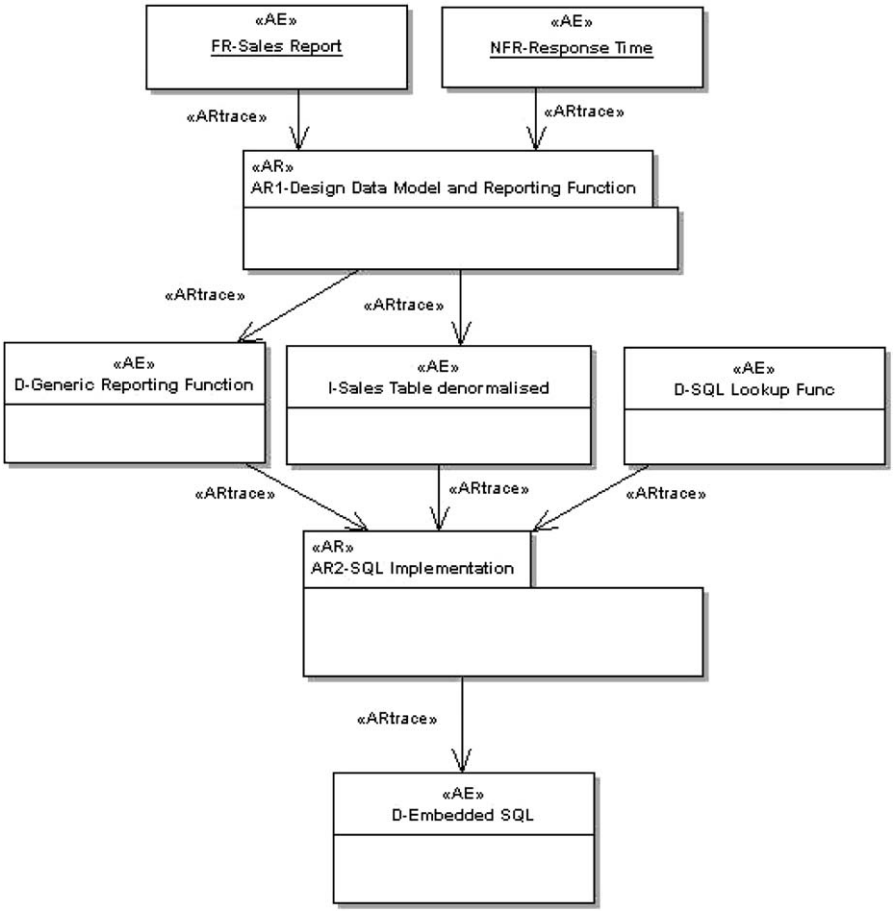


Fig. 1. An AREL diagram for a reporting sub-system, in a UML notation.

different teams of architects develop the design simultaneously. Fig. 2(a) and (b) shows two cyclic examples which violate the AREL definition above (Section 3.2). In Fig. 2(a), we can see that if *AR3* was to be created and added to the AREL model, with *AE3* as an input and *AE1* as an outcome, this would create a directed cycle in the graph. The dotted arrows denote the illegitimate ≪ARtrace≫ that would result in a cyclic graph, i.e. from *AE3* through to *AR3*, *AE1*, *AR1*, *AE2*, *AR2*, and back to *AE3*. Similarly, Fig. 2(b) depicts a cyclic graph created if *AR5* was modified to take *AE7* as an input, i.e. from *AR5* through to *AE6*, *AR6*, *AE7*, and back to *AR5*.

We do not allow cycles of design rationale links in AREL because they create ambiguity and inconsistency regarding the primary cause of decisions. Furthermore, a cyclic model would inhibit BBN-based change impact analysis. Using Fig. 1 as an example, we consider a new requirement *FR-Support Custom Made Report Format* in Fig. 3(a). We need to design a new element *D-Custom Report Design*. *D-Custom Report Design* would make use of the generic report template for customisation and some enhancements need to be made to *D-Generic Reporting Function* as well. As part of the decision, we use *D-Embedded SQL* as input to *AR3* because the design of *D-Custom Report Design* uses embedded SQL heavily. In this case, we have created a cyclic argument. Is *D-Generic Reporting Function* a primary cause or *D-Embedded SQL* a primary cause? The relationship is ambiguous.

Let us consider an alternative way to reason about the design. Fig. 3(b) shows that the decision making process should take into account all the relevant functional requirements in *AR1*. It adds to the justification for sharing generic report templates between the two reporting functions. At this stage, the *AR3-SQL Implementation* is not the issue in focus. It is *AR2-SQL Implementation* which decides how *D-Embedded SQL* should cater for both requirements. There are two implications in the elimination of cyclic decisions:

- *Identify the root goal of the design* – the important goals behind a decision are identified. In this case, relevant requirements such as *FR-Sales Report* and *NFR-Response Time* should be considered (as shown in Fig. 3(b)) because they affect the implementation of *D-Custom Report Design*. Fig. 3(a) does not have these relationships. Should there be a change in the performance requirements, it would not be able to trace this change to *D-Custom Report Design*.

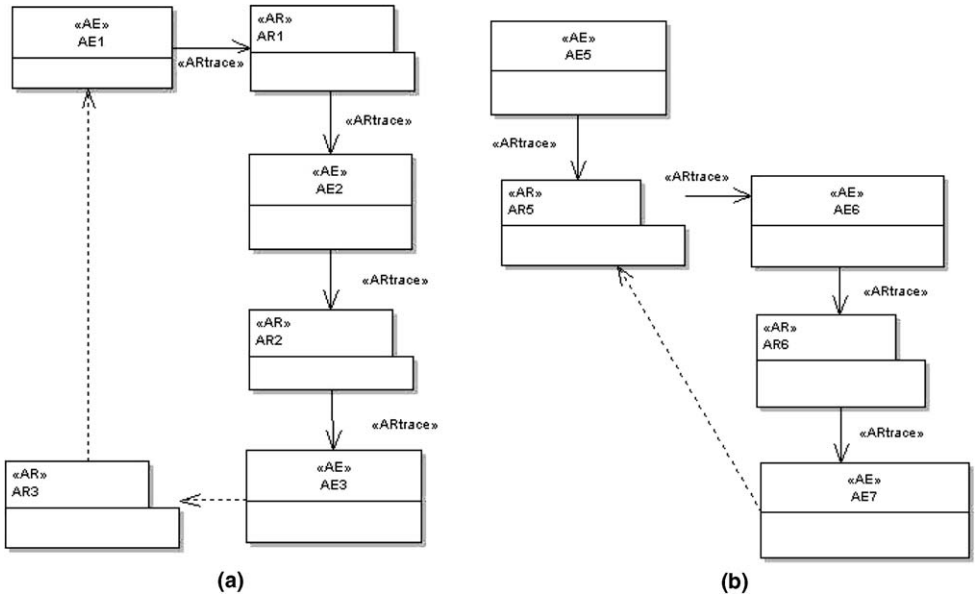


Fig. 2. Illegitimate cyclic graph: (a) AE-cyclic case and (b) AR-cyclic case.

- *Avoid ambiguity* – if we were to modify *D-Embedded SQL* to support remote database access (in Fig. 3(a)), it would be ambiguous and difficult to determine whether *D-Embedded SQL* or *D-Generic Reporting Function* is the cause of the decision relationship.

To prevent cycles from occurring in AREL, we provide an AREL consistency checking tool to detect the directed cycles. The tool would warn architects about the directed cycles in a model. Human reasoning would be required to modify the structure of the architecture design reasoning to resolve such reasoning anomalies.

4. Change impact analysis

AREL is a model of causes and effects between architecture elements (AEs) and architecture rationale (ARs) constructed as a result of the architecture design process potentially involving many design decisions. However, the AREL model does not provide any quantifiable means to indicate how each AE might influence a decision, nor does it quantify the impact of a decision on a design artefact. To predict and diagnose the impacts of a change in the architecture design in a quantitative manner, we utilise the probabilistic reasoning capability of Bayesian Belief Networks (BBNs) (Korb and Nicholson, 2004).

4.1. What are Bayesian belief networks?

Bayesian Belief Networks (BBNs) are a well established method in the Artificial Intelligence community for reasoning under uncertainty, using (i) a graphical structure to represent causal relationships and (ii) probability calculus to quantify these relationships and update beliefs given new information. Representing an AREL as a BBN enables prediction and diagnosis of change impact to the system when certain elements of the architecture have changed.

A BBN is a graph with edges connecting nodes with no directed cycles (i.e. a directed acyclic graph). Its nodes represent random variables and its edges represent direct dependencies between variables. In theory, variables can be discrete or continuous, but in this work we construct models with discrete variables only. Nodes representing discrete variables have two or more discrete states, representing the possible values the variable may take. For instance, a discrete Boolean node may represent whether or not a patient has lung cancer with the two states *True* and *False*. Nodes without parents are called *root nodes* and have an associated prior probability distribution. Each node with parents has a conditional probability table (CPT), which, for each combination of the values of the parents, gives the conditional probability of its taking these values. Thus the CPT can be considered to quantify the strength of the causal relationships.

Let us illustrate these basic elements of a BBN – the nodes, arcs and prior and conditional probabilities – with a very simple example from the medical domain. Suppose that a physician wants to reason about the chance that a patient presenting with a cough has lung cancer. The first relevant causal factor needing to be established would be whether the patient is a light, heavy or non-smoker. A possible diagnostic tool would be to take an Xray of the lungs. Fig. 4(a) shows a BBN model for this example.

Once a BBN model has been constructed, posterior probabilities – often referred to as *beliefs* – can be computed. Fig. 4(b) shows the physician's prior beliefs before any observations or evidence are taken into account. The marginal probability distribution is represented by the bars

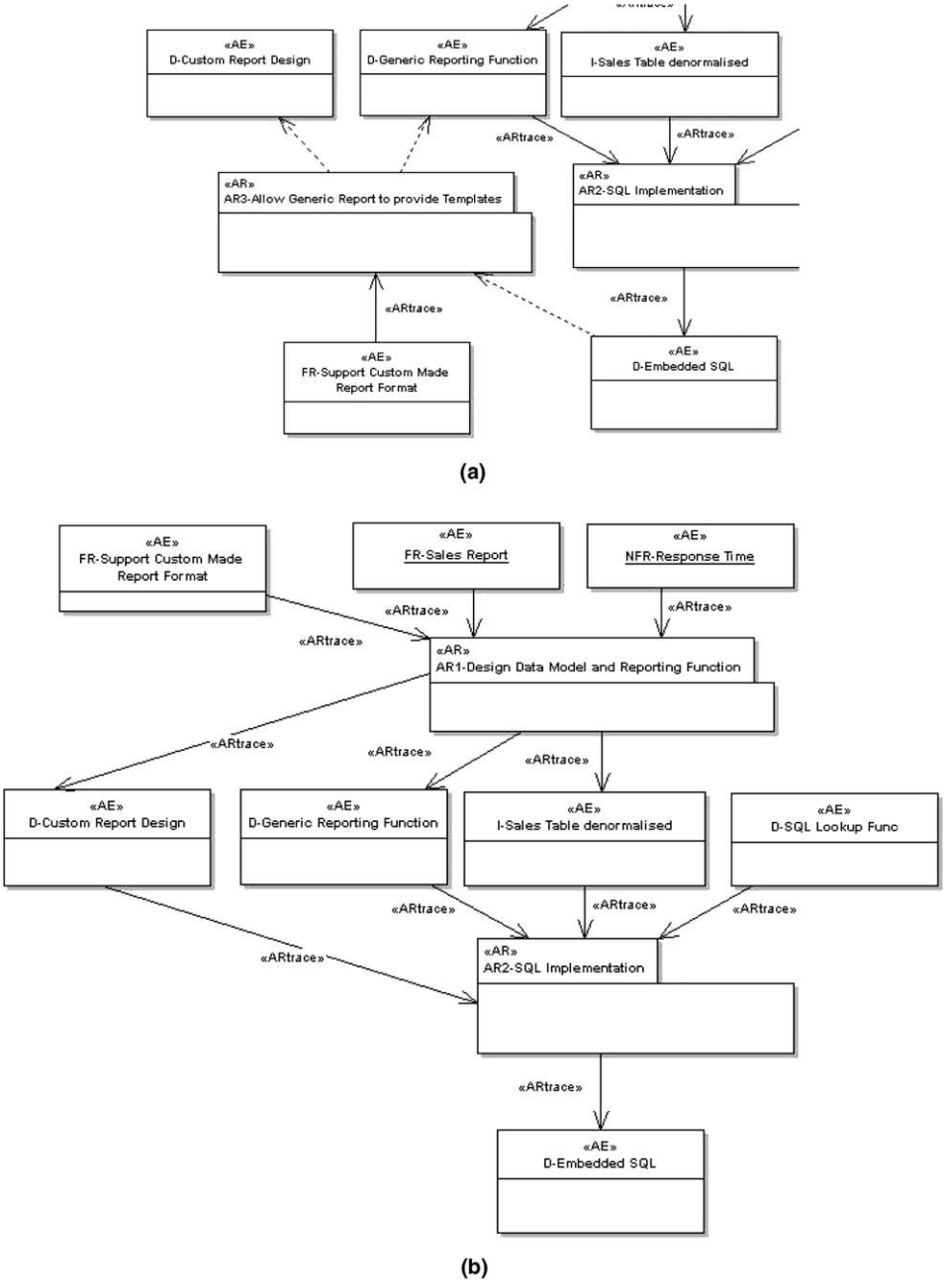


Fig. 3. (a) Cyclic design and (b) acyclic design.

in the diagram. Users can set the values of any combination of nodes in the network and this newly inserted evidence propagates through the network, producing a new probability distribution over all the variables in the network. There are a number of efficient exact and approximate inference algorithms for performing this probabilistic updating (Pearl, 2000), providing a powerful combination of predictive, diagnostic and explanatory reasoning. For example, Fig. 4(c) shows the physician's beliefs after diagnostic reasoning given that the patient has a cough, while Fig. 4(d) shows how the chance of the patient having lung cancer (and hence a positive lung X-ray) increases substantially if the patient is a heavy smoker (an example of predictive reasoning).

The reader is referred to Korb and Nicholson (2004) for a more detailed introduction to the fundamentals of BBNs. The following sections explore how AREL models can be represented by a BBN (Section 4.2) to provide causal reasoning under uncertainty for change impact analysis (Section 4.3).

4.2. Building a BBN to represent an AREL model

Representing an AREL as a BBN enables prediction and diagnosis of change impact to the system when certain elements of the architecture have changed. For our modelling purposes, we are interested in the causal relationships between variables representing architecture

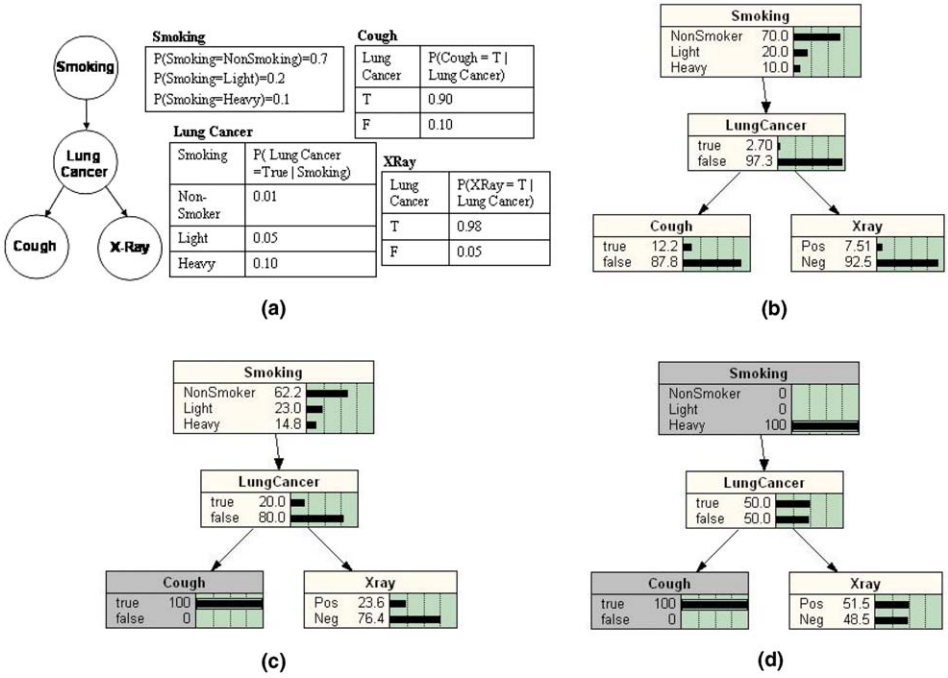


Fig. 4. A medical example: (a) BBN nodes, arcs and CPTs; (b) BBN graph without evidence; (c) patient has a cough (diagnostic reasoning); (d) patient has a cough and a heavy smoker (both predictive and diagnostic reasoning).

design rationales and elements. It is generally accepted that building a BBN for a particular application domain – commonly called ''knowledge engineering'' – involves three tasks (Korb and Nicholson, 2004):

(1) identification of the important variables, and their values, which become the nodes and their states in the BBN;
(2) identification and representation of the relationships between nodes, which means deciding which nodes should be connected by the directed edges; and
(3) parameterization of the network, that is determining
   (a) the prior probabilities for each root node in the network and
   (b) the conditional probability tables associated with each non-root node, which quantify the relationships between nodes.

We now show how to build a BBN to represent an AREL model by looking at each of these steps in turn.

4.2.1. Nodes: representing architecture elements and decisions

Recall that an AREL model contains two types of components, Architecture Elements (AEs) and Architecture Rationales (ARs). There is a direct mapping from these AREL components to the two types of nodes in the BBN, also called AE and AR nodes.

In an architecture design, we require initial inputs into the design. These inputs are functional requirements, non-functional requirements such as performance, basic design artefacts such as standard design templates or system constraints. These AE nodes are represented as root-nodes in AREL (see Section 4.2.3 for details). The AE nodes which are results of the design decisions are either the branches or the leaf nodes in AREL.

The BBN implementation for AREL uses probability to represent the possible change impact to the architecture design. If a design decision is to change, then it is probable that those design elements which depend on the validity of this design decision are likely to become unstable and subject to change as well. In order to represent dependency, first we have to represent the *states* of the AE and AR nodes. Both the AE and AR nodes in the BBN have two states each. The two states of an AE node represent whether or not the AE node is stable or likely to change. The two states of an AE node are:

- *Stable* – AE is not subject to change,
- *Volatile* – AE is subject to change.

If an AE is becoming volatile, then the decision which uses the AE as an input may not be as reliable because it is probable that a change in the underlying AE input could invalidate the design decision. As such, we need to represent the possible validity of the AR decision. The two states of an AR node are:

- *Valid* – the rationale of the decision is valid,
- *Invalid* – the rationale of the decision is invalid.

A detailed discussion on how to parameterise the probability tables of the AREL nodes is described in Section 4.2.3.

4.2.2. Edges: Representing causal relationships

Each link in an AREL model is represented by an edge in the BBN. As AREL links are defined to preclude directed cycles, this means that the corresponding BBN is valid, i.e. it is a directed acyclic graph. When an AREL model is formed, it is based on interconnected nodes with the fundamental structure of $AE_i \rightarrow AR_j \rightarrow AE_k$ This structure represents the directional causal relationship between the nodes starting from AE_i. AEs as requirements drive decisions to create high-level design. The high-level design are refined and decomposed into finer-grain architecture design objects through making further decisions for the different aspects of the architecture such as information models, computation models and deployment models. This process of making decisions, creating and modifying architecture elements is carried out in a progressive way until the architecture is complete (Tang and Han, 2005).

The three basic forms of relationships that are present in the BBN representation of the AREL model are shown in Fig. 5[1]:

A. AEs as causes of an AR: *AE1* and *AE2* are converging inputs of *AR1*;
B. AEs as effects of an AR: *AE4* and *AE5* are effects of *AR1*;
C. Multiple ARs as causes of an AE: *AE5* depends on both *AR1* and *AR2*.

The fundamental $AE_i \rightarrow AR_j \rightarrow AE_k$ structure reflects the fact that a decision (represented by an AR node) depends on its input AEs, and in turn can affect other AEs as the outcomes of the decision. At this stage, we can see intuitively that the edges in the BBN *could* represent the important causal relationships between AEs and ARs, such as

- while the input AE nodes are stable, a decision will probably remain valid;
- if the input AE nodes have changed, then it is likely that the conditions under which the decision was made are no longer valid and a reassessment of the inputs is necessary to validate the decision;
- the validity or otherwise of a AR node affects the stability of a consequent AEs.

However such relationships are defined and quantified through the specification of the CPTs; we describe this aspect of the BBN knowledge engineering process next, first for the root nodes, and then for non-root nodes in relationships A, B and C above.

[1] In the BBN literature, (A) and (C) are referred to a common effect, while (B) is a common cause structure. We distinguish between (A) and (C) here because of the difference for our purposes between the AE and AR nodes as a common effect.

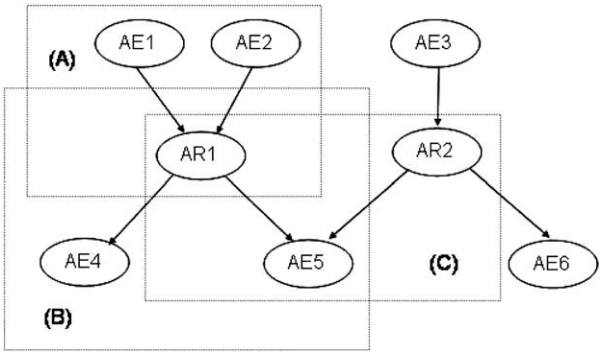


Fig. 5. Basic forms of AREL relationship in the BBN.

4.2.3. Probabilities: Quantifying the causal relationships

4.2.3.1. Root nodes. Each root node in a BBN (i.e. nodes without parents) has an associated prior probability. As mentioned above, in a BBN representation of an AREL model, all root nodes must be of type AE, and represent either functional or non-functional requirements, basic designs or systems constraints. The architect must therefore estimate the prior probabilities for the different types of AE nodes. Prior probabilities assigned to the AE root nodes indicate the likelihood of change of these primary requirements, design or constraints.

If the AE is a requirement, one could estimate whether this requirement is volatile or stable by assessing whether the requirement is likely to change over time. For instance, an industry standard or a regulation is not negotiable and therefore the probability of it being stable is 100% and the probability of it being volatile is 0%. Another example is the requirement to produce data through different interface file formats such as XMI, csv etc. The list of interface formats is likely to be extended as the system evolves, so the architect may estimate that the probability of this requirement being stable is 80% and the probability of it being volatile is 20%. Design elements which are root nodes in the BBN also need their volatility characteristics to be estimated by prior probabilities. For instance, general purpose library classes and routines are less dependent on changing requirements and so they tend to be more stable.

Obviously the volatility of different AEs will be highly domain dependent and hence the estimation of this volatility by providing prior probabilities for root nodes is part of the knowledge engineering task for the architect when building the AREL model.

4.2.3.2. Non-root nodes. Each non-root node in a BBN (i.e. nodes with incoming arcs) has an associated conditional probability table (CPT). These BBN conditional probabilities in the AREL context quantify the extent in which an architecture element influences the validity of a decision, or the extent in which a decision may change a resulting architecture element. All AR nodes and all non-root AE nodes in the BBN representation of AREL are assigned CPTs, which quantify the causal dependency on their

parents. These probabilities are assigned by the architects using their past organisational experience and their assessment of the current situation.

Given the fundamental structure of $AE_i \rightarrow AR_j \rightarrow AE_k$, AE_i is either a root node with an assigned prior probability $P(AE_i)$, as described above, or a non-root node with a CPT. The probability of AR_j is conditionally dependent on the event that has occurred to AE_i, represented by $P(AR_j|AE_i)$. The probability of AE_k is conditionally dependent on the event that has occurred to AR_j, represented by $P(AE_k|AR_j)$. The joint probability is therefore the product of multiplying probability tables of AE_k, AR_j and AE_i. This calculation is called marginalisation and the resulting probability is the marginal probability (Korb and Nicholson, 2004).

An example of the simplest $AE1 \rightarrow AR1 \rightarrow AE2$ substructure might be when a decision ($AR1$) to use a .Net framework (i.e. output $AE2$) depends on the basis that the Microsoft Windows is the operating system platform (i.e. input $AE1$). If this decision is based *only* on the single input, this is represented by the CPT entries

- $P(AR1 = valid|AE1 = stable) = 1$.
- $P(AR1 = invalid|AE1 = volatile) = 1$.
- $P(AE2 = stable|AR1 = valid) = 1$.
- $P(AE2 = volatile|AR1 = invalid) = 1$.

In this extreme case, a change in the input $AE1$ to a Linux operating system would completely invalidate the $AR1$ decision, which in turn renders the use of the .Net framework, represented by $AE2$, 100% volatile.

Of course, in real designs there are interactions between multiple AEs and ARs, and hence the BBN structures are more than chains. Also, the nature of the relationships may be more complex and less deterministic, which can be represented by probabilities other than just 0 and 1 in the CPTs. We will now look at how to quantify such more complex relationships, by considering in turn the three basic forms of relationships identified above in Section 4.2.2.

Relationship A

If an AE is an input (i.e. a cause) to a decision and an AR is dependent upon it, any changes in the AE will affect the probability of the AR's validity. Fig. 5 shows an example of a type A relationship where both $AE1$ and $AE2$ are architecture elements that influence the decision $AR1$. The inputs in this relationship can be generalised to one or more causes (AEs). If any of the input AEs of a decision (AR) changes, then there is a possibility that the decision will no longer be valid. In this example, the conditional probability of $AR1$ can be denoted by $P(AR1|AE1, AE2)$. As all nodes are binary in states, there are 4 possible combinations to consider for $AR1$ being valid.

- $P(AR1 = valid|AE1 = stable, AE2 = stable)$.
- $P(AR1 = valid|AE1 = volatile, AE2 = stable)$.
- $P(AR1 = valid|AE1 = stable, AE2 = volatile)$.
- $P(AR1 = valid|AE1 = volatile, AE2 = volatile)$.

Note that in each case $P(AR1 = invalid) = 1 - P(AR1 = valid)$. For the situation where all the AE causes are stable and not subject to change, the CPT entry for $P(AR1 = valid)$ reflects the confidence in the correctness of the original rationale for the architecture decision. We would expect in most cases it would be set to 1, or very close to 1. Conversely, the CPT entry for $P(AR1 = valid)$ would be lowest in the situation where all the AE causes are volatile.

For the intermediate situations where one or more, but not all, of the AE causes are subject to change, the CPT entries must be chosen carefully to reflect the relative weight of the individual causes on the architecture decision. For example, if there are two AE causes and both were of equal weight, say x, when making the decision, the CPT entries would be the same.

- $P(AR1 = invalid|AE1 = stable, AE2 = volatile) = x$.
- $P(AR1 = invalid|AE1 = volatile, AE2 = stable) = x$.

The actual value of x must then reflect the causal impact a single volatile AE is expected to have on the decision; x close to 1 means the decision will become invalid even with the single volatile cause, 0.5 means a fifty-fifty chance, while x close to zero means the decision is reasonably robust if only one AE cause is volatile. A more dominant ($AE2$) and less dominant ($AE1$) cause combination might be represented by

- $P(AR1 = invalid|AE1 = stable, AE2 = volatile) = 0.8$.
- $P(AR1 = invalid|AE1 = volatile, AE2 = stable) = 0.3$.

In this example, if $AE1$ is stable and $AE2$ is volatile, the probability of $AR1$ being invalid is higher than if $AE2$ is stable and $AE1$ is volatile.

Relationship B

When an architecture decision is made, an AR is created to record the rationale and linked with one or more resulting AEs. During architecture design, if a requirement changes then it is possible that the AR depending on it will be invalidated and so all resulting designs from that AR are more likely to be volatile and subject to change. Note that there is no direct relationship between AEs caused by the same AR, so we need only consider the effect of the AR on each AE individually. In Fig. 5, for example, even though $AE4$ and $AE5$ are both dependent on $AR1$, they are independent of each other given $AR1$. This means specifying the CPT $P(AE_i|AR)$ for each AE_i, that is, the values for:

- $P(AE_i = volatile|AR = valid)$.
- $P(AE_i = volatile|AR = invalid)$.

Note that in each case $P(AE_i = stable) = 1 - P(AE_i = volatile)$. In most cases, while the AR is valid, we would not expect the AE to change, hence $P(AE_i = volatile|AR = 1\ valid)$ would be close to 0. Conversely, if the AR is invalid, the AE is much more likely to change. The more an AE depends on an AR, the more likely it is subject to change when AR becomes invalid, and hence $P(AE_i = volatile|AR = invalid)$ becomes closer to 1. If an AE depends only very loosely on an AR, then $P(AE_i = volatile|AR = invalid)$ might be set much lower than 1. In Fig. 5, $AR1 \rightarrow AE4$ and $AR2 \rightarrow AE6$ are examples of relationship B.

Relationship C

It is possible that an AE is the result of multiple separate decisions (ARs). For example, in Fig. 5, *AE5* is the result of two separate decisions *AR1* and *AR2*. The conditional probability of *AE5* can then be defined as $P(AE5|AR1, AR2)$. Since *AE5* is dependent upon both decisions *AR1* and *AR2* simultaneously, estimates for the CPT can be expressed as

- $P(AE5 = volatile|AR1 = valid, AR2 = valid)$.
- $P(AE5 = volatile|AR1 = valid, AR2 = invalid)$.
- $P(AE5 = volatile|AR1 = invalid, AR2 = valid)$.
- $P(AE5 = volatile|AR1 = invalid, AR2 = invalid)$.

The causal relationship being modelled in this CPT is essentially an additive one: the more decisions that are invalid, the higher the probability that the common resultant AE will be volatile. While all of the decisions are still valid, the CPT entry for $P(AE5 = volatile)$ will be close to zero, and if all the decisions are invalid, it will be close to 1. For the intermediate situations where one or more, but not all, of the AR decisions are invalid, the CPT entries must be chosen carefully to reflect the relative weight of the individual decisions on the resultant AE. Assuming *AR1* has more influence on *AE5* than *AR2* in the example, the cause combination could be represented as

- $P(AE5 = volatile|AR1 = invalid, AR2 = valid) = 0.8$.
- $P(AE5 = volatile|AR1 = valid, AR2 = invalid) = 0.3$.

These CPT entries mean that the combination where *AR1* is invalid and *AR2* is valid is quite likely to cause *AE5* to be volatile (probability 0.8). Conversely, when *AR1* is valid and *AR2* is invalid, the probability that *AE5* is volatile is lower (0.3), signifying less influence by *AR2*.

4.3. Reasoning about change impact with AREL

4.3.1. An illustrative example

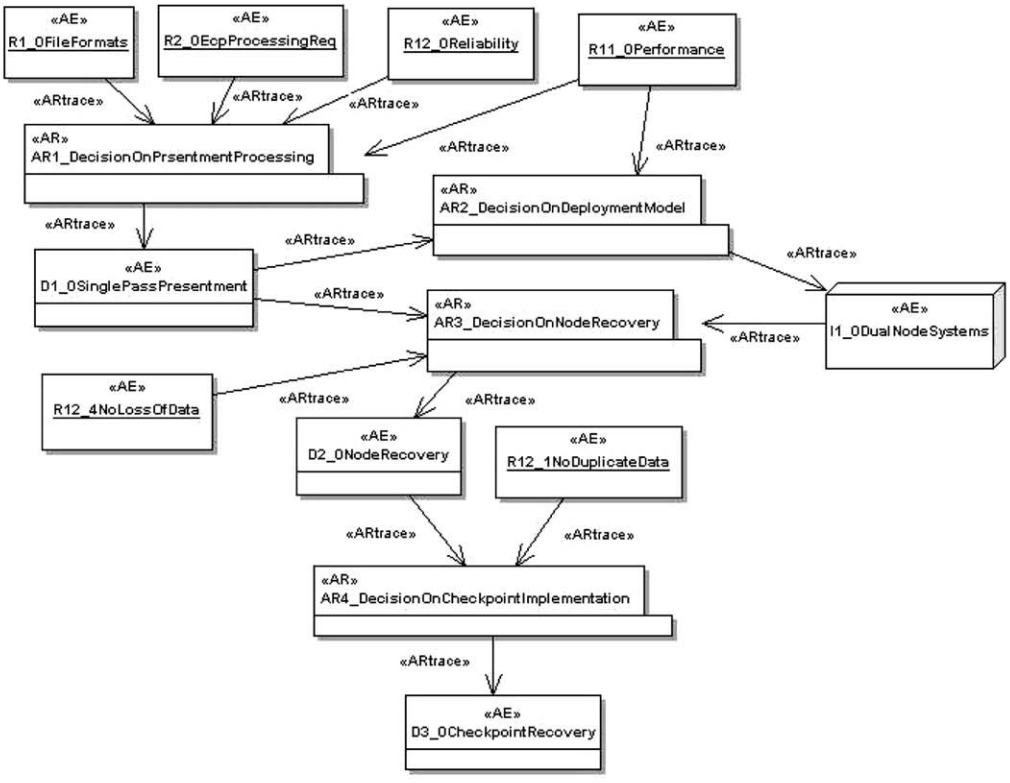
In this section, we use a system designed by the first author to illustrate the probabilistic causal relationships between requirements and the design, and their use in performing change impact analysis. The system was designed in 2000 according to the FSTC PACES standard (FSTC, 2000) to process cheque images transferred between banks and the clearing house. Presenting banks would send cheque presentment files containing cheque images and ECP data to the clearing house for settlement. This example illustrates the system reliability aspect of the cheque clearing system. There are extensive functional requirements and non-functional requirements in the system. The following are some of those requirements surrounding system reliability:

- R1.0 specifies the cheque presentment file formats using ANSI X9.37 and X9.46 standards.
- R2.0 specifies the cheque presentment processing requirements.
- R11.0 specifies the performance requirements.
- R12.0 specifies the reliability requirements.
  - R12.1 specifies that a cheque should only be processed once and once only.
  - R12.4 specifies that there could be no loss of cheque images or ECP data during processing.

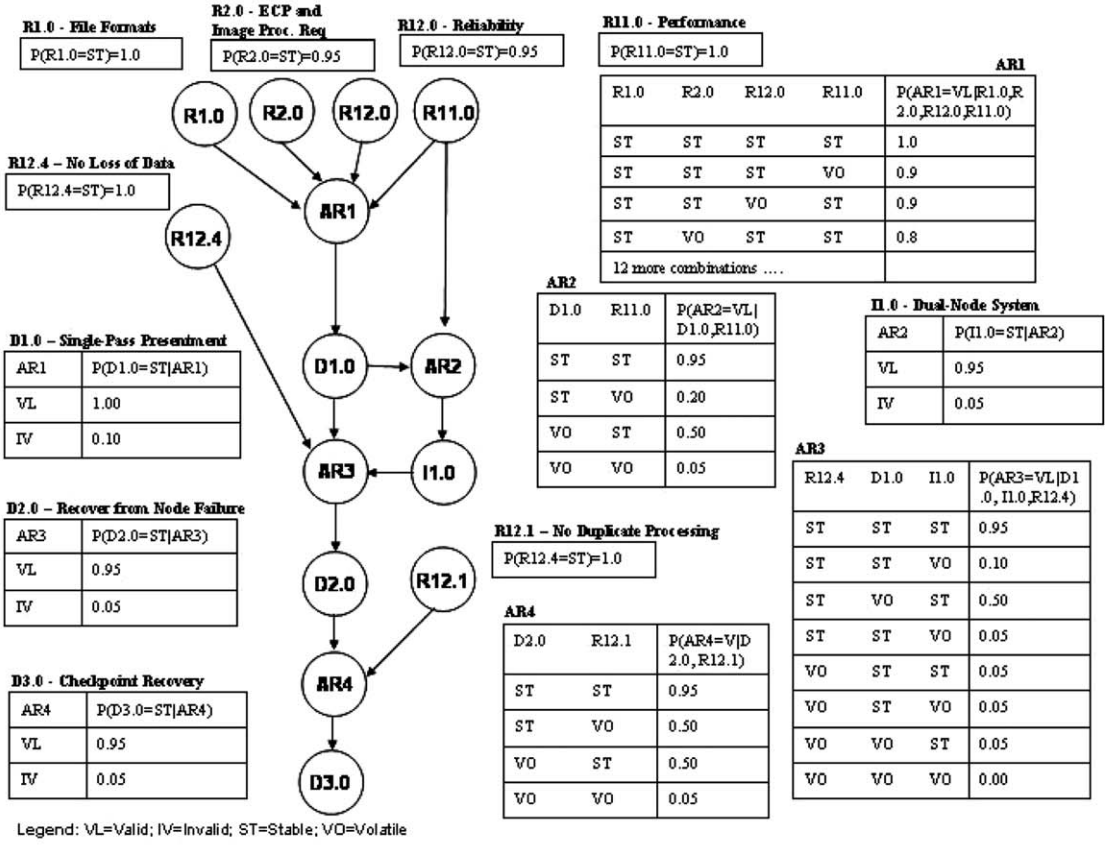
A part of this design is shown in Fig. 6(a) using the UML representation. We use the UML element *package* to implement ≪AR≫ to encapsulate the design reasoning and any design alternatives. *AR1* captures the rationale for designing the Single-Pass Presentment Process (denoted by design *D1.0*) to satisfy four functional and non-functional requirements. A single pass strategy would improve the performance by eliminating multiple reads of a file. In a single pass of the presentment file, it carries out decryption, authentication and decoding of the data. The design alternative is to read the presentment and temporary files several times during processing which would require more disk access and therefore results in slower performance.

AR2 represents the decision of selecting a platform to satisfy the performance requirements of processing 25,000 cheques every minute. The design alternative is to use a single-node computer or a dual-node computer system. Since a high-end single node system would cost much more than a dual-node system, it is decided that a dual-node system provides a better price performance. As a result, we choose a dual node system denoted by *I1.0*.

The next decision is to determine how to support fail-over when one of the two computer nodes fails. The decision in *AR3* examines the fail-over mechanism. Two alternatives are available, use the NCR LifeKeeper software to monitor system health through a heartbeat mechanism or custom-develop failure detection scripts. It is decided that the NCR Lifekeeper is a far superior solution because it can switch over all applications as well as the Oracle database instance. This design involves using a common disk storage that is accessible by both nodes and employing the fail-over software to detect failure of a node and activate the switch over processing functions.



(a)



(b)

Fig. 6. Cheque image clearing system: (a) an UML representation and (b) a BBN representation with priors/CPTs.

In order to ensure that all cheques are processed once and once only (requirement *R12.1*), a decision is made in *AR4* to justify the checkpoint recovery mechanism. Another alternative to *AR4* is to rollback all transactions and reprocess them. This is not a viable option. A checkpoint recovery mechanism ensures that all the processed cheques are committed to permanent storage and any partial transactions that have not been committed when the system fails are rolled back.

Fig. 6(b) shows the BBN representation of an AREL model for the system with probability tables showing the prior probabilities and CPTs of some selected nodes. Prior probabilities for root node *R1.0*, *R11.0*, *R12.1* and *R12.4* are set to 100% because industry standards and reliability requirements dictate that they cannot be compromised. *I1.0* Dual-node System has a high dependency on *AR2* because if the decision based on performance and the fundamental design change, there is a high probability (95% or 0.95 in CPT) that the platform decision may become volatile. In decision *AR4*, if either *D2.0* or *R12.1* changes, then *AR4*'s decision validity reduces to 50% (0.5 in CPT), it means that either of those inputs exert the same influence on the decision. The influence of *I1.0* on *AR3* is relatively higher than that of *D1.0*. When *I1.0* is volatile and the other two inputs are stable, *AR3*'s validity is 10% (or 0.1 in CPT) whereas when *D1.0* is volatile and the other inputs are stable, the validity of *AR3* is at 50% (or 0.5 in CPT).

4.3.2. Original beliefs modelled by AREL

The AREL model in Fig. 7 is another representation of the same model in Fig. 6(b) except it shows the marginal probability distribution when no evidence (i.e. change) has been added. The specific visualisation format is that provided by Netica (Norsys Software Corp, 1997) BBN software. The visualisation includes the name of each node across the top, the state names down the left (*valid/invalid* for AR nodes, *stable/volatile* for AE nodes), marginal probability shown as percentages (e.g. 88.7% and 11.3% for *I1.0 Dual Node System*) by two horizontal bars. The marginal probabilities, for instance, of a leaf-node such as *D3.0 Checkpoint Recovery*, are calculated based on its own CPT and the CPTs and prior probabilities distributions of its ancestors. For instance, the marginal probability of *D3.0* indicates that based on the current requirements and design, there is a probability of 0.828 (shown as 82.8% in Fig. 7 by Netica) that the design is stable.

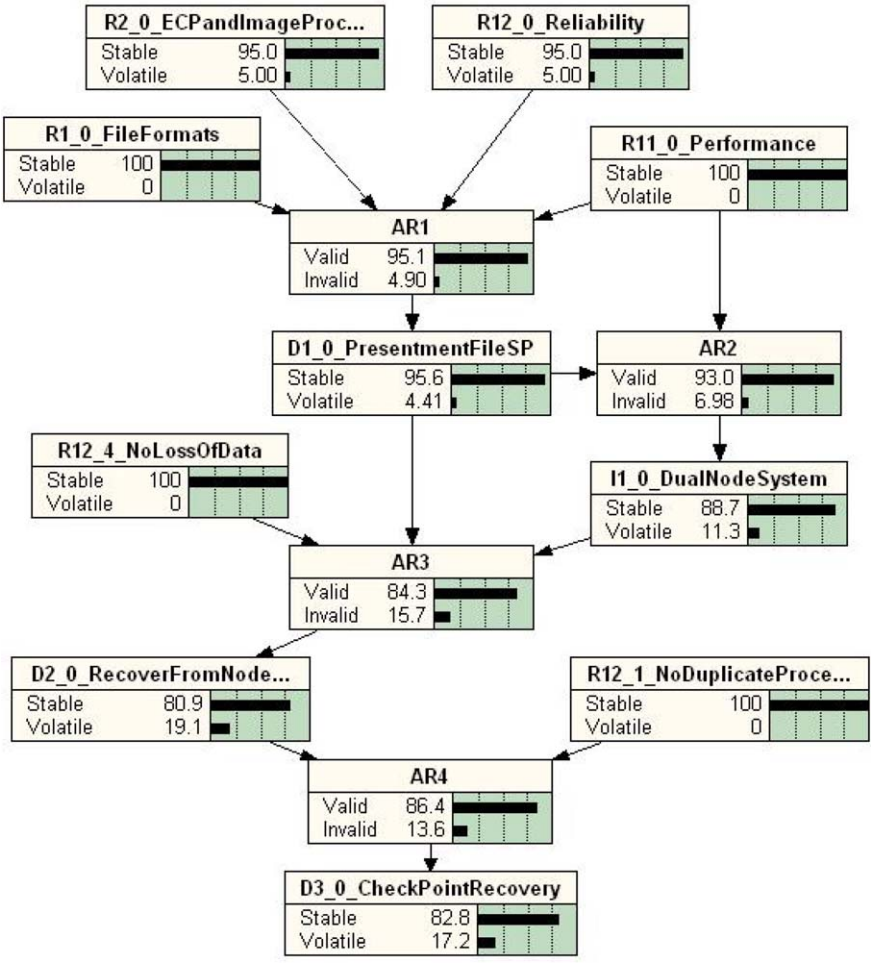


Fig. 7. An example BNN shown with beliefs before any evidence is entered.

4.3.3. Predictive reasoning

As requirement or environmental changes are introduced to the system, we want to predict their effect in terms of possible architecture design changes. Being able to identify the areas that are likely to change would improve both the accuracy and efficiency of architecture enhancements. Recall from Section 4.2 that reasoning in a BBN means users first set the values of certain nodes in the network to indicate evidence that has been gathered about the system. For the AREL BBN model, setting values means that change takes place in the system. These changes are then propagated through the network, producing a new probability distribution over the remaining variables in the network (Korb and Nicholson, 2004), showing the what-if scenarios of the impact of change. Multiple changes could be introduced simultaneously. Let us consider how we can do this type of predictive reasoning in the BBN representation of AREL.

Architects could add evidence of one or more likely changes to the AREL model. That is, instantiate the BBN with evidence that an AE takes a new value "volatile", i.e. it is in the volatile state. The BBN belief updating algorithms would then compute the posterior probabilities for AR and AE nodes which are affected by the additional evidence in the network. The architects will then be able to assess the change impacts based on the posterior probabilities. In particular, which decisions (AR nodes) are more likely to be invalid and which architecture elements (AE nodes) are more likely to be volatile. We provide an illustration of this predictive reasoning using the cheque clearing system example in Fig. 8.

Let us assume that there is a new hardware platform that has the processing capacity to satisfy the *R11.0* performance criteria and it uses a single-node computer system to replace the dual-node system. This change is inserted as evidence to set the volatile state of *I1.0* to 100% (see highlighted node in Fig. 8). Given this evidence, we can do a what-if analysis and predict that there is a 91% chance that the decision *AR3* will be invalid leading to a 86.9% chance that design decision *D2.0 Recovery from Node Failure* will be volatile. This is due to the fact that the need to switch-over between nodes is no longer required and therefore *D2.0* requires some modifications to support recovery of the single node.

Once *D2.0* is identified as likely to change, the design rationale that follows would identify any scripts that deal with dual node data sharing and switch-over would have

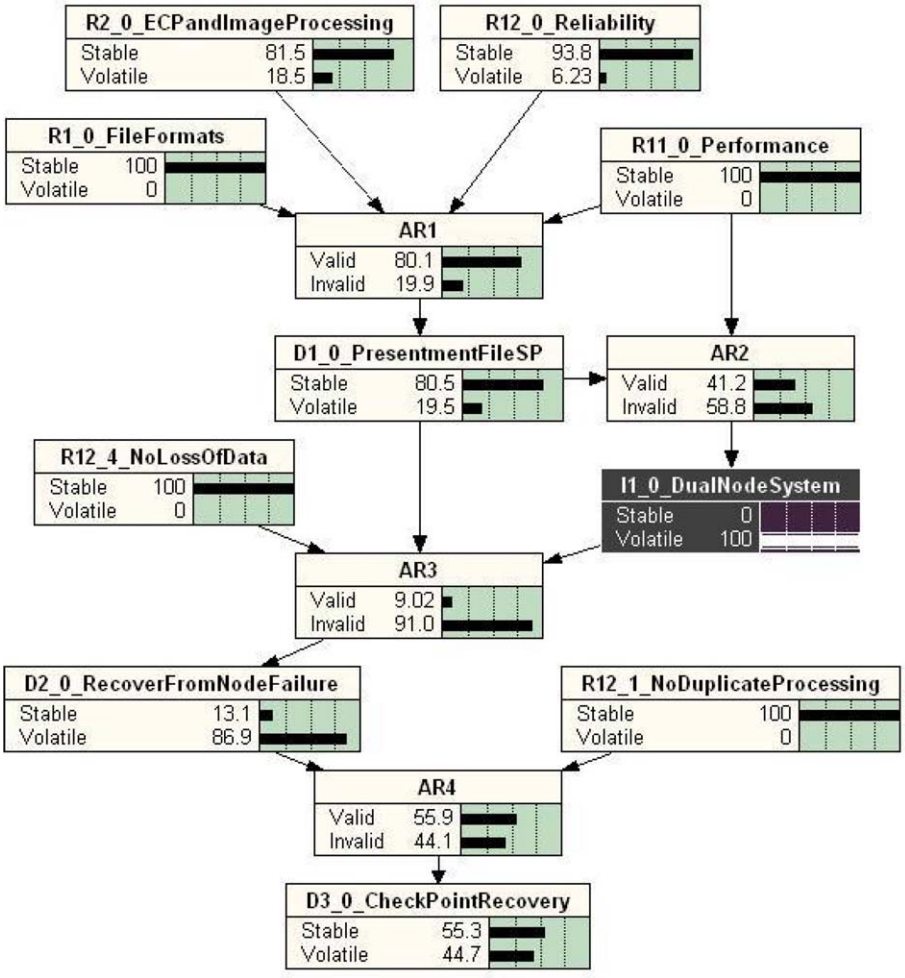


Fig. 8. Predictive model.

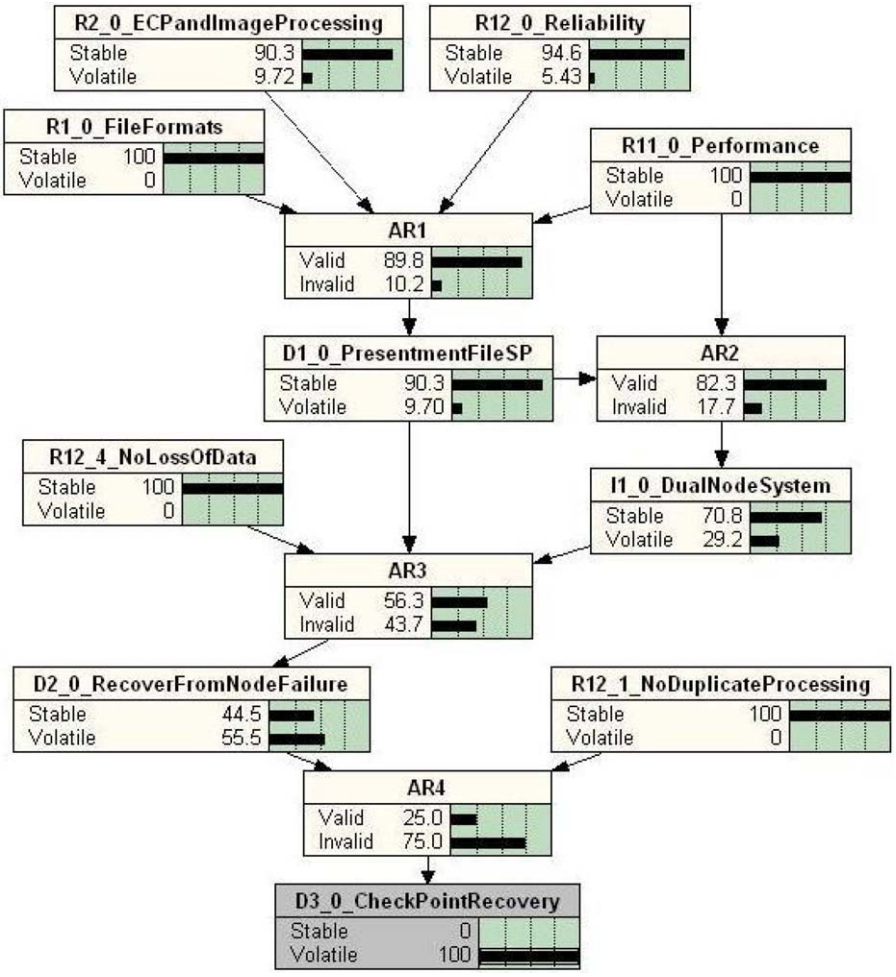


Fig. 9. A diagnostic model.

to be re-designed. The impact flows further down to decision *AR4* and design *D3.0* but with a lesser effect. The impact of *I1.0* on *AR4* is less because *R12.1* is equally influential as *D2.0*. The non-functional requirement *R12.1* specifies the integrity of the transaction processing irrespective of platform implementation. Therefore, the effect of changing platform design in *I1.0* is balanced by the stable requirement *R12.1*, so *D3.0* is less volatile than *D2.0* with a marginal probability of volatility of 44.7%. As such, *D3.0 Checkpoint Recovery* changes come from its indirect dependency on *D2.0 Recovery from Node Failure*.

4.3.4. Diagnostic reasoning

Another use of BBN reasoning in AREL is to diagnose possible causes and influences in architecture design. As shown in the preceding section on predictive reasoning, changes in the high-level requirements or design objects might affect the decisions and architecture elements which depend on them. Conversely, given an architecture element, an architect might want to reason about the architecture elements and decisions which influence it. When a dependent design object changes, the reasons of the change might be due to changes in its ancestor objects such as requirements.[2] The BBN can be used to diagnose these possible causes or influences by identifying changes in the posterior probabilities of the ancestor nodes. Note that such diagnostic reasoning propagates "backwards" against the direction of the edges, which represent the causal relationships.

In AREL modelling, such diagnostic reasoning might take place if evidence is inserted into a non-root AE. For instance, if we change design *D3.0 Checkpoint Recovery* (an AE node) to set its volatile state to 100% (Fig. 9), we observe that the posterior probabilities of decisions *AR4*, *AR3*, *AR2* and *AR1* will change. Also, design objects *D2.0*, *I1.0*, *D1.0* and requirements *R2.0* and *R12.0* will also change. In this case, *D3.0* is modified to implement a recovery mechanism based on a batch-level instead of a file-level recovery. The trade-off is that the batch-level recovery approach would involve many more commit cycles in normal processing but a much shorter recovery time when performing recovery. *D3.0* depends on the design of *D2.0* in

[2] Changes might also be caused by new external elements, i.e. external causal intervention. As such, new root-nodes can be used to model the change.

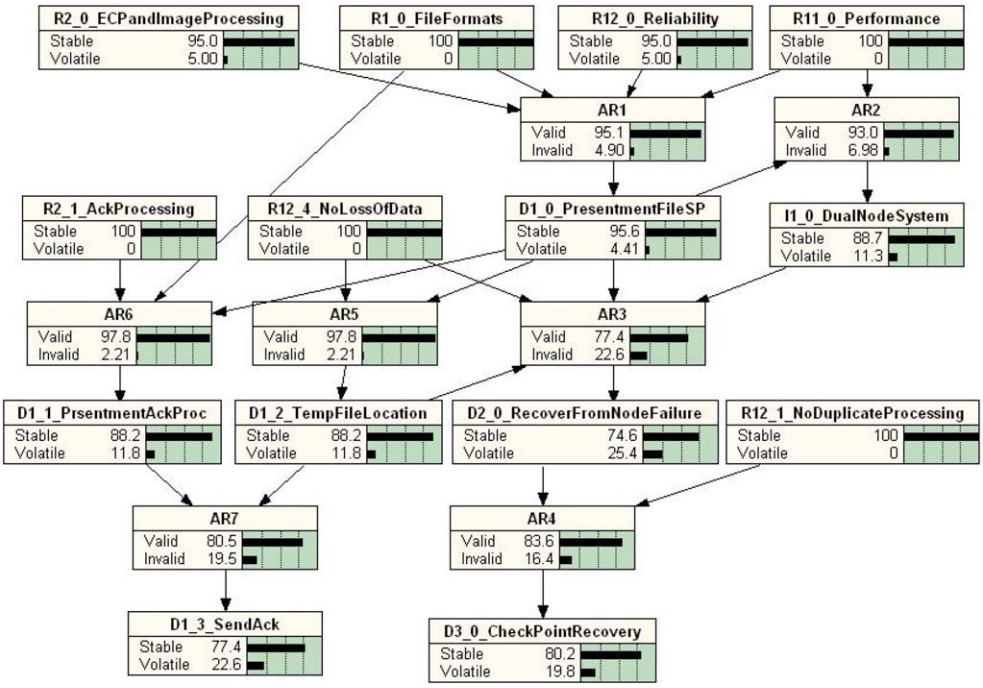


Fig. 10. A BBN model of cheque clearing system with acknowledgement processing.

that when a computer node fails, all presentment files would need to be recovered from the failed file system before any transaction processing recovery can proceed. Its decision rationale is captured in *AR4*. As such, when *D3.0* is to change, it is necessary for an architect to assess *AR4* and *D2.0* to ensure that changing checkpoint recovery strategy has not violated the node recovery strategy.

Architects could use BBN diagnosis to continue the analysis of all the ancestor architecture decisions and elements of *D3.0*. In this case, changes in *D3.0* can be traced to *D2.0* with a volatile probability of 55.5%. The dependency as represented by the posterior probabilities carries to all the ancestor nodes of *D3.0*. Such diagnostic reasoning is useful because, before any changes are introduced to the system, architects can investigate the possible causes (i.e. both architecture elements and decisions) which might have led to the change. Further considerations and possible modifications to these causes could then be made to ensure the consistency of the design decisions and the design elements.

4.3.5. Combining diagnostic and predictive reasoning

When an architecture is subject to impact analysis in a real-life project, it is often necessary to understand the relationships between the design object which is undergoing modification and related requirements which might be affected by such modification. The side-effects of a modification may come from a few areas: (a) design objects which are directly affected by the modification; (b) requirements, constraints or assumptions which are in conflict with the modification; and (c) other design objects that may be indirectly affected when the requirements, constraints or assumptions are compromised.

Using BBN's diagnostic and predictive reasoning, architects may traverse the AREL network to investigate the elements that might be affected by the changes in the network. Fig. 10 shows the same Cheque Clearing System with additional nodes for acknowledgment processing. Acknowledgment is a protocol between the presentment bank and the clearing house such that all presentment files are acknowledged by the clearing house after they have been received and validated. The acknowledgment can be positive or negative depending on whether the presentment file satisfies all the necessary conditions such as authentication and reconciliation.

D1.1 is a module which prepares the acknowledgment file after *D1.0* has parsed the presentment file. *D1.3* is the module responsible for sending the acknowledgment file to the presenting bank to notify it whether the presentment file has been accepted or rejected.

Let us assume that there has been no fault resilience built-in to *D1.3* and the acknowledgment file could be lost during its processing or transmission due to network or computer failures. It would cause the presenting bank to resend the presentment file after a time-out period. So it is decided that the *D1.3* needs to be modified to support recovery.

Table 1 illustrates a sequence of reasoning steps to find out how changing *D1.3* and other architecture elements could affect related functionalities of the system, through changing the validity of architecture rationales. It involves using both the BBN diagnostic reasoning and predictive reasoning. The leftmost column in Table 1 shows the name of the BBN node (grouped by AE type in upper set, with AR type in lower set). Each of the remaining columns shows the change that is being investigated at each step, corresponding to the evidence being entered into the

Table 1
Volatility of architecture elements and validity of architecture rationales over a sequence of changes

BBN node	Step 1 Modify *D1.3* (%)	Step 2 Modify *R12.0* (%)	Step 3 Modify *D1.0* (%)	Step 4 Modify *D1.1* (%)	Step 5 Modify *D3.0* (%)
AE nodes – Volatile State					
D1.3 Send Ack	100				
R2.0 ECP Image Processing	12.9	8.93	24	24	24
R12.0 Reliability	5.72	100			
D1.0 Presentment File SP	13.3	29.6	100		
D1.1 Presentment Ack Processing	41.8	45.8	63.2	100	
D2.0 Recover from Node Failure	50.1	56.3	82.8	80.2	95.5
D3.0 Checkpoint Recovery	29.8	32.3	43	42	100
AR nodes – Invalid State					
AR1	13.7	30.6	100		
AR6	8.05	17.9	60.6	90	
AR3	50.1	57	86.5	83.5	96.5
AR4	27.6	30.3	42.3	41.1	93

BBN, by setting the appropriate AE node volatility to 100%.

- Step 1 – evidence is inserted in *D1.3* (i.e. its volatility is set to 100%) to reflect that the module has to change. The change triggers probabilities to be updated in the network as shown in the column (step 1) of Table 1. Through diagnostic reasoning, two requirements *R2.0* and *R12.0* are shown to be the possible causes of this change (i.e. inserting evidence in *D1.3* increases *R2.0*'s volatility from 5.0% to 12.9% and *R12.0* changes from 5.0% to 5.72%). By examining the two requirements, we reason that the requirement on application reliability *R12.0* is the cause which induces the change to *D1.3*. As such, this requirement needs to be enhanced to cater for presenting bank acknowledgment.
- Step 2 – evidence is inserted in *R12.0* (i.e. its volatility is set to 100%) to reflect that it is being updated. This change induces volatility and validity changes in *R12.0*'s descendent. *AR1*'s invalidity is increased to 30.6% and the probabilities for *AR6*, *AR3* and *AR4* have all increased. The immediate design object *D1.0 Presentment File SP* in the BBN graph has a volatility of 29.6%. In fact this module needs to store information in the database to indicate that a presentment file has been received. This information enables the application to check and carry out recovery if necessary.
- Step 3 – evidence is inserted in *D1.0* (i.e. its volatility is set to 100%) to reflect that the design object has to change. This change influences *D1.1*, *D2.0* and *D3.0*. Furthermore, *AR1*'s invalidity is now 100% because of the change in *D1.0*. *AR3* is also affected (83% invalid) because it depends on the design that is now subject to change. Although *D2.0* is the next design object in the decision chain which has a volatility of 82.8%, we reason that *D2.0*'s recovery mechanism is transparent to the acknowledgment file and therefore it can be left alone. However, *D1.1 acknowledgment processing* which now has a volatility of 63.2% needs to change to be able to recover from failure.
- Step 4 – evidence is inserted in *D1.1* to indicate changes in the acknowledgment processing. The design change invalidates decisions *AR6*, *AR3* and *AR4*, which are represented by their respective probabilities being 90%, 83.5% and 41.1%. Since we have already visited decisions *AR6* and *AR3*, we can focus on *AR4* and its descendent. At this stage, *D3.0 Checkpoint Recovery* has a volatility of 42%.
- Step 5 – evidence is inserted in *D3.0 checkpoint recovery* to enhance this design module. Its new function has to detect any errors to trigger the recovery process of unsuccessful acknowledgement processing. Once an error is detected, it would notify *D1.1* and *D1.3* to resume any unfinished processing when the system is back on-line.

Fig. 11 shows the posterior probabilities for the full network after all predictive and diagnostic reasoning have been performed, i.e. combining all the evidence for nodes *D1.3*, *R12.0*, *D1.0*, *D1.1* and *D3.0*. Note that the posterior probabilities would be the same regardless of the order in which the evidence was added, or if evidence for all five nodes was added together. However the change in the probabilities over the *sequence* of reasoning steps is clearly crucial to supporting the human reasoning of the architect performing the what-if analysis.

This example demonstrates that in impact analysis, it is necessary to use both diagnostic and predictive reasoning in combination to help determine the change impacts. One modification to the architecture often triggers a chain reaction where multiple modifications are required.

5. Tool support

There are two usability criteria for design rationale systems. Capturing design rationale must have minimal interference with the design process (Regli et al., 2000) and the users of design rationale must be able to retrieve the reasons to answer their *why* questions (Herbsleb and Kuwana, 1993). Therefore, the tool set should be able to

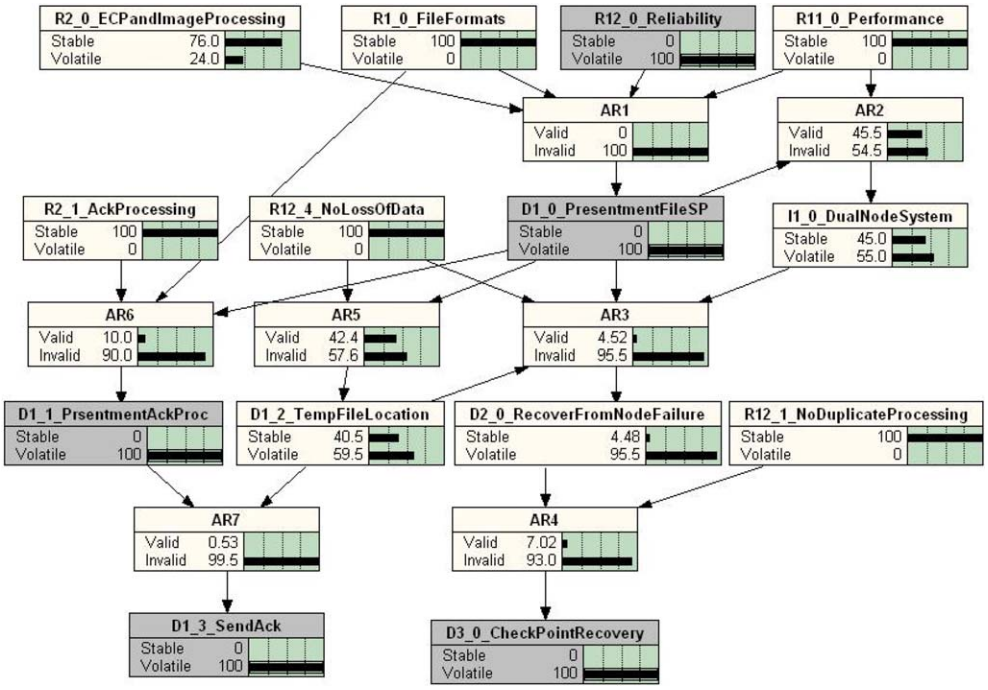


Fig. 11. A BBN model showing a combination of diagnostic and predictive reasoning.

support both the architecture design process and the design reasoning process. We address these issues in AREL by integrating tools which are commercially available and we streamline the implementation to support design reasoning capture and retrieval processes. The tool set comprises of three programs. An UML tool, Enterprise Architect (version 5.00.767), is used to capture system design and design rationale (Sparx Systems, 2005). Netica (version 2.17) captures BBN models and computes their probabilities (Norsys Software Corp, 1997). We built the AREL Tool in .Net to perform several integration functions (Tang, 2006). It checks the correctness of the AREL model, provides traceability support and integrates Enterprise Architect UML graphs with the Netica BBN representation.

Existing and new design artefacts created in UML can be modelled as AEs to serve the dual-purpose of being a design specification and part of a design rationale representation. Enterprise Architect provides a convenient way to input design rationale using the design rationale capture templates. The implementation is by way of creating Enterprise Architect profiles for ≪AR≫ and ≪AE≫ stereotypes (Tang, 2005).

The AREL Tool supports the AREL model consistency checking. It takes an Enterprise Architect (EA) repository (i.e. extension EAP) as input and performs the following:

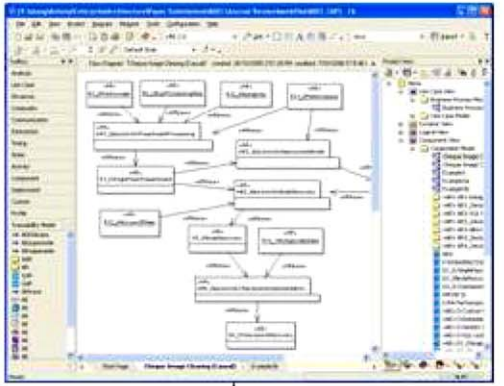
- Detect any directed cycles in the AREL model.
- Detect any improper model construct which violates the AREL definition. For instance, two AEs linked directly by ≪ARtrace≫ without an AR.
- Detect any isolated items such as AE or AR which are not connected to any other elements. The tool reports these as warnings.

Once a consistent AREL model is established (in UML format), we need to capture the probabilistic design reasoning relationship between the ≪AE≫ and ≪AR≫ nodes using Netica. If designers have enter data into two separate systems (i.e. Enterprise Architect and Netica), it would double the data entry workload and increase the cost of design rationale capture. So we extract the AREL model from Enterprise Architect into a format that Netica can use.

The steps to extract the AREL model are illustrated in Fig. 12. Step 1 involves constructing the design objects and capturing the design decisions using Enterprise Architect. In Step 2, the AREL Tool converts the UML data contained in Enterprise Architect into the Netica repository format (i.e. Netica *dne* extension). In Step 3, Netica opens the exported file and designers can assign prior and conditional probabilities to AEs and ARs.

As designers continue to make decisions and modify the decision structure over the life-span of a system, the AREL model (in UML) would change accordingly. These changes need to be reflected in the BBN model (within Netica) without having to re-enter all the probabilities previously assigned. The following are the possible changes:

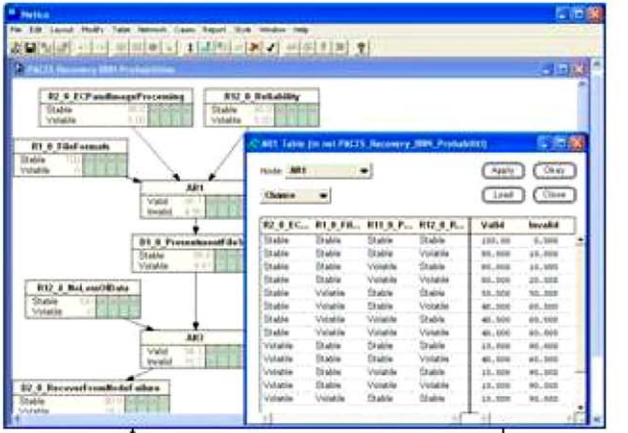
- AE and AR nodes are added – prior or conditional probabilities need to be captured for the new nodes.
- AE and AR nodes are removed – those nodes which depend on the deleted nodes should have their CPTs updated.
- new links are added or removed – the CPTs of affected nodes need to be updated.



Step 1

Step 2

Netica Repository
(.dne file)



Step 3

Netica Repository
(.dne) with
probabilities assigned
To nodes

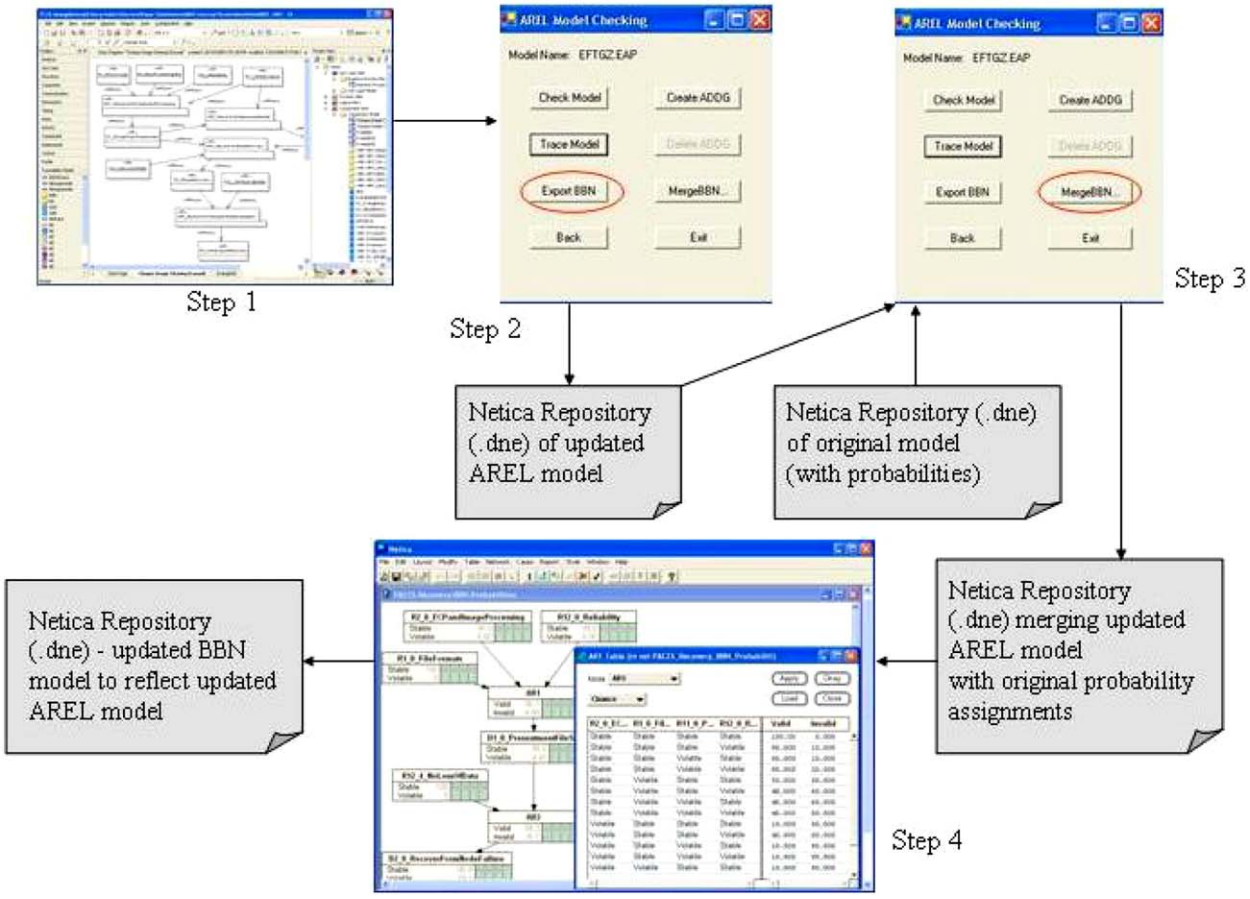


Fig. 13. A process to synchronise change between UML and BBN.

AREL overcomes the issue of relating design decision to design elements by joining them in a causal relationship. This is an improvement over argumentation-based design rationale systems. Tool support is key to the success of design rationale systems. We choose to use industry standard notation and tools to support AREL. This means that the designers can design the architecture and capture the design rationale using the same tool. The BBN tool for analysis is integrated with the design repository without redundant data re-entry. It reduces the efforts required to capture design rationale for analysing change impacts. AREL and its BBN application have the following benefits: (a) tacit knowledge can be captured and represented explicitly for decision tracing; (b) the graphical representation help architects detect design inconsistencies such as cyclical decisions; and (c) it provides a quantitative approach to architecture change impact analysis.

The probabilities for the AREL nodes are provided by architects and are estimates based on their experience and intuition. These estimates are semi-objective in nature where different architects may provide different estimates given the same design scenario. However, in the longer-term architects should be able to supply consistent and accurate estimates because feedbacks of any inaccuracy would help to refine them. Providing a probability value to a fine degree of granularity, say one percentage point, would be difficult and probably meaningless. One possible way to overcome this granularity issue is to specify the probabilities with a qualitative verbal scale. For instance, the volatility of an AE node could be specified as very high, probable, fifty-fifty, improbable and impossible. These ordinal categories can then be mapped into probabilities, and tests can be undertaken to determine the calibration of that mapping to obtain the desired outcomes of such modelling. A Netica-compatible software tool (Hope et al., 2002) for such qualitative elicitation of probabilities and verbal maps already exists. Interested readers are referred to Korb and Nicholson (2004) for a comprehensive discussion of the subject.

It is obvious that not all system designs need to use AREL, smaller systems that are easy to trace and understand will gain very little from it. We suggest that large, complex and long life-span systems that contain intricate design decisions to cater for extensive non-functional requirements are candidates to use AREL. In most cases, the issues that architecture design deals with are more significant, complex and intricate compared with detailed software design. Therefore AREL focuses on capturing the design rationale for architecture design instead of detailed design. As for the level of architecture design details

required to be explored, we suggest that architecture design is delineated from detailed design by the level of risks involved in the design. If an architect assess that the risks of realising the design outcome is acceptable and the risk of implementing an architecture design is acceptable, then the architecture design is complete (Tang and Han, 2005).

The application of BBN to AREL provides a foundation for some further research opportunities in architecture design and software engineering: (a) to predict the cost of change impact to architecture enhancements; (b) to examine the complexity of the system through analysing the dependency between architecture elements and decisions; (c) to consider using Dynamic Bayesian Networks for temporal analysis as systems evolve over time; (d) to derive from AREL a more abstract and concise model that represents the associations between decisions (this is currently being investigated and is called Architecture Decision Dependency Graph – ADDG); and (e) to enhance traceability support in multi-viewpoints architecture frameworks using design rationale and causality.

7. Conclusion

Previous work in the area of design rationale has mostly focused on design rationale argumentation for design deliberation but has ignored the dependency between design elements and design rationale. In this article, we introduce the AREL model to capture design decisions and their relationships with design elements. AREL represents design reasoning by the inter-dependency of design rationale and design elements. This approach has three advantages: (a) integrated design reasoning process – the design decision and reasoning process are an integral part of the architecture design process; (b) simplified representation – design rationale is encapsulated into a design decision; and (c) quantifiable reasoning structure – the BBN model is applied to AREL to quantify change impact analysis.

We apply Bayesian Belief Networks to quantify the likelihood of change impact based on the dependency information between architecture decisions and elements. We have identified three different reasoning methods in which change impact can be analysed: (a) predictive reasoning; (b) diagnostic reasoning; and (c) combined reasoning. A key consideration in the usability of the AREL model is how easy it is to support this method using commercially available tools. As such, we have selected a tool set which comprises of an UML tool called Enterprise Architect, a BBN tool called Netica, and our custom-developed tool, the AREL Tool, which performs consistency checking and integrates the UML and BBN models.

References

Bosch, J., 2004. Software architecture: the next step. In: Proceedings of the 1st European Workshop on Software Architecture (EWSA). 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.

Burge, J., 2005. Software Engineering Using design RATionale. Ph.D. thesis, Worcester Polytechnic Institute.

Chung, L., Nixon, B., Yu, E., Mylopoulos, J., 2000. Non-Functional Requirements in Software Engineering. Kluwer Academic, Boston.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J., 2002. Documenting Software Architectures: Views and Beyond, first ed. Addison Wesley.

Conklin, J., Begeman, M., 1988. gIBIS: a hypertext tool for exploratory policy discussion. In: Proceedings of the ACM Conference on Computer-Supported Cooperative Work, pp. 140–152.

Cysneiros, L., Leite, J., 2001. Using UML to reflect non-functional requirements. In: Proceedings of the 11 CASCON, Toronto, pp. 202–216.

Dardenne, A., Van-Lamsweerde, A., Fickas, S., 1993. Goal-directed Requirements Acquisition. Science of Computer Programming 20, 1–36.

Dutoit, A., Paech, B., 2000. Rationale management in software engineering. In: Handbook of Software Engineering and Knowledge.

Fenton, N., Neil, M., 2000. Software metrics: roadmap. In: Proceedings of the 25th International Conference on Software Engineering (ICSE), May 2000, pp. 357–370.

FSTC, 2000. Paperless Automated Check Exchange and Settlement (PACES). Available from: <http://www.fstc.org/projects/paces/index.cfm>.

Galliers, J., Sutcliffe, A., Minocha, S., 1999. An impact analysis method for safety-critical user interface design. ACM Transactions on Computer–Human Interaction 6 (4), 341–369.

Gruber, T., Russell, D., 1992. Derivation and use of design rationale information as expressed by designers. Tech. Rep. KSL-92-64, Stanford University.

Gruber, T., Russell, D., 1996. Generative design rationale: beyond the record and replay paradigm. In: Design Rationale: Concepts, Techniques and Use. Lawrence Erlbaum Associates, pp. 323–350 (Chapter 11).

Han, J., 1997. Designing for increased software maintainability. In: Proceedings of the International Conference on Software Maintenance. IEEE Computer Society Press, pp. 278–286.

Herbsleb, J.D., Kuwana, E., 1993. Preserving knowledge in design projects: what designers need to know. In: Proceedings of the Conference on Human Factors in computing systems, 24–29 April 1993. ACM Press, New York, pp. 7–14.

Hope, L., Nicholson, A., Korb, K., 2002. Knowledge engineering tools for probability elicitation. Technical Report, School of Computer Science and Software Engineering, Monash University.

IEEE, 2000. IEEE Recommended Practice for Architecture Description of Software-Intensive System (IEEE Std 1471-2000).

Korb, K.B., Nicholson, A.E., 2004. Bayesian Artificial Intelligence. Chapman and Hall/CRC.

Kruchten, P., Lago, P., Vliet, H.v., Wolf, T., 2005. Building up and exploiting architectural knowledge. In: Proceedings of the 5th IEEE/IFIP Working Conference on Software Architecture.

Lee, J., Lai, K., 1996. What's in design rationale. In: Design Rationale: Concepts, Techniques and Use. Lawrence Erlbaum Associates, pp. 21–52 (Chapter 2).

Maclean, A., Young, R., Bellotti, V., Moran, T., 1996. Questions, options and criteria: elements of design space analysis. In: Design Rationale: Concepts, Techniques and Use. Lawrence Erlbaum Associates, pp. 53–106 (Chapter 3).

Moran, T., Carroll, J., 1996. Design Rationale: Concepts, Techniques, and Use. Lawrence Erlbaum Associates.

Mylopoulos, J., Chung, L., Nixon, B., 1992. Representing and using non-functional requirements: a process-oriented approach. IEEE Transactions on Software Engineering 18 (6), 483–497.

Norsys Software Corp, 1997. Netica – Application for BBN and Influence Diagrams User's Guide. Available from: <http://www.norsys.com/dl/NeticaMan_Win.pef.zip>.

Pearl, J., 1988. Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, San Mateo, CA.

Pearl, J., 2000. Causality: Models, Reasoning, and Inference. Cambridge University Press.

Perry, D.E., Wolf, A.L., 1992. Foundations for the Study of Software. ACM SIGSOFT 17 (4), 40–52.

Potts, C., 1996. Supporting software design: integrating design methods and design rationale. In: Design Rationale: Concepts, Techniques and Use. Lawrence Erlbaum Associates, pp. 295–322 (Chapter 10).

Ramesh, B., Jarke, M., 2001. Towards reference models for requirements traceability. IEEE Transactions on Software Engineering 27 (1), 58–93.

Rapanotti, L., Hall, J., Jackson, M., Nuseibeh, B., 2004. Architecture-driven problem decomposition. In: 12th IEEE International Conference on Requirements Engineering, pp. 73–82.

Regli, W.C., Hu, X., Atwood, M., Sun, W., 2000. A survey of design rationale systems: approaches, representation, capture and retrieval. Engineering with Computers (16), 209235.

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 (2).

Shum, B.S., Hammond, N., 1994. Argumentation-based design rationale: what use at what cost? International Journal of Human–Computer Studies 40 (4), 603–652.

Sparx Systems, 2005. Enterprise Architect V5.00.767. Available from: <http://www.sparxsystems.com/>.

Tang, A., 2005. An UML Profile Extension to Support Architecture Decision Traceability using Enterprise Architect. Available from: <http://www.ict.swin.edu.au/personal/atang/ArelStereotypePackage.zip>.

Tang, A., 2006. An AREL Model Checking Tool. Available from: <http://www.ict.swin.edu.au/personal/atang/AREL-Tool.zip>.

Tang, A., Han, J., 2005. Architecture rationalization: a methodology for architecture verifiability, traceability and completeness. In: Proceedings of the 12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'05). IEEE, USA, pp. 135–144.

Tang, A., Babar, M., Gorton, I., Han, J., 2005a. A survey of the use and documentation of architecture design rationale. In: Proceedings of the 5th IEEE/IFIP Working Conference on Software Architecture. IEEE, USA.

Tang, A., Jin, Y., Han, J., Nicholson, A., 2005b. Predicting change impact in architecture design with Bayesian belief networks. In: Proceedings of the 5th IEEE/IFIP Working Conference on Software Architecture. IEEE, USA.

Tyree, J., Akerman, A., 2005. Architecture Decisions: Demystifying Architecture. IEEE Software 22 (2), 19–27.