International Journal of Software Engineering
and Knowledge Engineering
Vol. 15, No. 4 (2005) 695–717
© World Scientific Publishing Company

World Scientific
www.worldscientific.com

A BAYESIAN NETWORK APPROACH TO RATIONAL ARCHITECTURAL DESIGN

HONGYU ZHANG*
School of Computer Science and Information Technology,
RMIT University, Melbourne, Victoria 3001, Australia
hongyu@cs.rmit.edu.au

STAN JARZABEK
School of Computing, National University of Singapore,
Lower Kent Ridge Road, Singapore 117543
stan@comp.nus.edu.sg

Received 15 December 2003
Revised 29 November 2004
Accepted 23 December 2004

In software architecture design, we explore design alternatives and make decisions about adoption or rejection of a design from a web of complex and often uncertain information. Different architectural design decisions may lead to systems that satisfy the same set of functional requirements but differ in certain quality attributes. In this paper, we propose a Bayesian Network based approach to rational architectural design. Our Bayesian Network helps software architects record and make design decisions. We can perform both qualitative and quantitative analysis over the Bayesian Network to understand how the design decisions influence system quality attributes, and to reason about rational design decisions. We use the KWIC (Key Word In Context) example to illustrate the principles of our approach.

Keywords: Software architecture design; Bayesian network; automated design; software quality; knowledge capture; KWIC.

1. Introduction

In software architecture design, we explore design alternatives and make decisions about adoption or rejection of a design. Different architectural design decisions may lead to systems that satisfy the same set of functional requirements but differ in certain quality attributes (also referred to as non-functional requirements) such as reusability, modifiability, performance and security. Whether or not a final system will be able to exhibit its desired quality attributes is largely determined by the time

*Author for correspondence.

the architecture is designed. As the design decisions which a software architecture captures are early design decisions, correcting an architectural design at a later development stage is usually difficult and costly.

Although getting an architecture right is very important, achieving a rational architectural design is difficult. Some of the reasons are listed below:

1. One design decision may influence many quality attributes. For example, choosing an architectural style may influence system performance and reusability. One quality attribute may be influenced by many design decisions. For example, the system's time performance is influenced by the decisions on architectural style, data structure and algorithm. Furthermore, decisions that influence a quality attribute may be competing or synergic. For example, choosing a Shared Data architectural style [33] has a positive influence on the system's time performance, while an inefficient algorithm has a negative influence on it. It is important though difficult to analyze the tradeoffs among many competing and synergic design decisions.
2. There are also influential relationships among design decisions. One design decision may influence other decisions. For example, the decision made on computing platform may affect the decision on component model. Design decisions may also be mutually dependent — one decision depends on the other.
3. The uncertain nature of quality attributes. Functional requirements, such as withdrawing a correct amount of money from a bank account, are relatively easier to identify and validate. It is well known that software quality is "hard to define, impossible to measure, easy to recognize [21]". Quality requirements from customers are often vague and have no clear-cut criteria of satisfaction.
4. The impact of design decisions on quality attributes can be undeterministic. Very often, it is difficult to assert that one design decision will definitely (true or false) satisfy a quality requirement. It is difficult for us to emulate all possible factors and evaluate their influence on the quality attributes precisely, especially for quality attributes that are not observable at runtime, such as modifiability, reusability and testability. Thus, we will describe the influences between design decisions and quality attributes "probabilistically" (a degree from 0 to 1) instead of "deterministically" (0 or 1).
5. Objective data for many quality attributes are difficult to collect, especially at the architectural design stage when the actual system is not built. Subjective estimations based on domain experts' knowledge and experiences are often required during architectural design.

For all these reasons, without a proper approach it is difficult for a software architect to make rational designs consistently. Human mind is weak in reasoning with a complex web of inter-related information, especially with large volume and uncertain information. We need to develop methods and tools which can help the architects select among design alternatives and estimate the quality of a system before the system is built, so as to mitigate the potential risks associated with launching a design.

In this paper, we propose a Bayesian Network based approach to rational architectural design. Bayesian Networks [27] have been successfully applied to many domains that require reasoning under uncertainty. A Bayesian Network provides a graph-based model for representing uncertain knowledge in a domain, as well as an underlying probabilistic model for quantitative reasoning.

In our solution, we analyze a domain, explore quality requirements and architectural design alternatives, and then construct a DAG (Directed Acyclic Graph) based graphical model to represent the design space. Nodes in the graphical model correspond to design decisions and quality attributes. Edges model the influences of design decisions on quality attributes, and the influences between different decisions. We use subjective probabilities to quantify a graphical model. We can perform both qualitative and quantitative analysis over the Bayesian Network to understand how the design decisions influence quality attributes, and to make rational design decisions. Both modeling and analysis are supported by the Bayesian Network tools.

By constructing a Bayesian Network, we capture the design knowledge and experiences of domain experts. The graphical model intuitively records the relationships among design decisions and quality attributes. The use of subjective probability helps us make reasonable estimations at early design stage before the actual system is built. The constructed Bayesian Network can be stored in an organization's knowledge repository, and can be later retrieved during software maintenance and reuse. In this paper, we use the KWIC (Key Word In Context) example, a well-known example to the software engineering community, to illustrate the principles of our approach.

The remainder of this paper is organized as follows. In Sec. 2, we give a brief introduction to the Bayesian Network. Section 3 describes the construction of a Bayesian Network for the KWIC domain. Section 4 shows how we could use the constructed Bayesian Network to make rational design decisions. We discuss some issues related to our approach in Sec. 5 and compare our approach to related work in Sec. 6. Section 7 concludes the paper and describes the future work.

2. An Introduction to Bayesian Network

Over the last decade, Bayesian Networks (also called Bayesian belief networks or probabilistic networks) have dominated the field of reasoning under uncertainty. Bayesian Network is a technology of choice for expert systems that must deal with incomplete and uncertain information.

A Bayesian Network consists of two parts: a graphical model and an underlying conditional probability distribution. The graphical model is represented by a directed acyclic graph (DAG), whose nodes correspond to random variables in a domain and whose edges correspond, intuitively, to the causal/influential relationship. Figure 1 shows an example of a Bayesian Network adapted from [27].

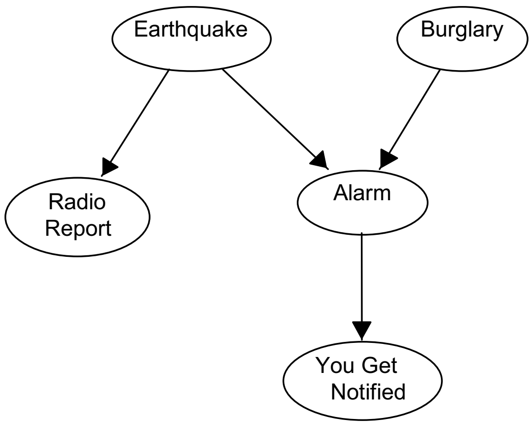


Fig. 1. Bayesian network — an example.

Table 1. The CPT of P(Alarm | Burglary, Earthquake).

		P(Alarm | Burglary, Earthquake)	
Burglary	Earthquake	True	False
TRUE	TRUE	0.9	0.1
TRUE	FALSE	0.8	0.2
FALSE	TRUE	0.3	0.7
FALSE	FALSE	0.1	0.9

Consider a domain where we are trying to reason about an alarm installed in a house. The occurrence of an earthquake or burglary directly influences whether or not an alarm goes off. If there is an alarm, then you may get notified. If there is an earthquake, the local radio station may report it. There are five variables in this domain, all of which are binary valued (true or false).

A Bayesian graphical model can be quantified by assigning a conditional probability to each node. A conditional probability is a probability conditioned by other known factors. A conditional probability statement has the following form:

Given the event B, the probability of event A is x.

The notation for the statement above is $\mathrm{P}(\mathrm{A}\,|\,\mathrm{B}) = x$. For example, in the earthquake alarm domain, the probability of alarming is mainly conditioned by two variables: Burglary and Earthquake. Thus the conditional probability for the Alarm node is given as P(Alarm | Burglary, Earthquake). Often a conditional probability is represented as a table (called Conditional Probability Table, or CPT), such as the one shown in Table 1.

The probability could be subjective (reflecting the domain experts' belief) or objective (theoretically or empirically determined). Studies show that the subjective probabilities from experts can be pretty close to the objective probabilities and most Bayesian Network practitioners subjectively assess the probabilities they need [4].

By assigning a conditional probability to each node, we quantify the graphical model of a Bayesian Network. We can then perform analysis over the constructed Bayesian Network such as diagnosis (e.g., if you got notified, infer the probability of Burglary), prediction (e.g., given Burglary, infer the probability of You Get Notified), or mixed inferences (given Alarm, infer the probability of Burglary and You Get Notified).

Bayesian Networks have been widely applied to many application domains, such as medical diagnosis [12], trouble-shooting [13], and image interpretation [19]. The most widely used Bayesian Network application is probably the Office Assistant that is shipped with Microsoft Office system family [15]. In software engineering, Bayesian Network was used for software defect prediction [8], software testing [36] and reliability assessment [9, 34].

We use Bayesian Network to represent knowledge of making architectural design decisions. The design decisions and system quality attributes have causal/influential relationships, analogous to the relationships shown in Fig. 1. Furthermore, the causal/influential relationships among decisions and quality attributes are not deterministic.

3. A Bayesian Network for KWIC Domain

We use the KWIC domain as an example to illustrate our approach. In this example, we firstly analyze the domain, and explore quality requirements, design alternatives and their inter-relationships. We then design a Bayesian Network to represent our knowledge of making architectural design decisions. To perform quantitative analysis, we evaluate the influence of design decisions on quality attributes and quantify the graphical model by assigning a conditional probability to each node. We implement the Bayesian Network for KWIC domain using Hugin$^{\text{TM}}$, a commercial tool from Hugin Expert A/S.[a]

3.1. *KWIC domain overview*

The KWIC (Key Word In Context) problem was first proposed by Parnas to contrast different criteria for decomposing a system into modules [25]. Since its introduction, the problem has become well known and has been used in several studies to illustrate the benefits of different architectural styles [11, 33].

Parnas formulated the KWIC problem as follows:

"*The KWIC [Key Word in Context] index system accepts an ordered set of lines; each line is an ordered set of words, and each word is an ordered set of characters. Any line may be "circularly shifted" by repeatedly removing the first*

[a]Hugin Expert A/S homepage: http://www.hugin.com

word and appending it at the end of the line. The KWIC index system outputs a list of all circular shifts of all lines in alphabetical order."

In addition to the above functional requirements, we have identified the following quality attributes that a KWIC system shall exhibit: *The KWIC system should be reusable, easy to change (modifiable), and has good performance in both response time and space.*

The KWIC example is widely used as a teaching device in software engineering. While KWIC can be implemented as a relatively small system, it is not simply of pedagogical interest. Practical instances of it are widely used by computer scientists. For example, the "permuted" [*sic*] index for the Unix Man pages is essentially such a system [33].

3.2. *KWIC architectural design decisions*

All KWIC systems are similar: they are all developed to satisfy the basic requirements described above. However, KWIC systems can be designed in many ways. After analyzing the KWIC domain, we understand that we should make the following decisions during architectural design:

- *Processing algorithm* (SHIFT_PROCESSING). Line shifting (including alphabetization) can be performed after all the lines are read (i.e., batch processing), or on each line as it is read from the input device (i.e., incremental processing). We refer to the two values of SHIFT_PROCESSING as Batch and Incremental. Batch processing improves system time performance, while Incremental processing hampers time performance.
- *Circular shifts data representation* (SHIFT_DATA). Circular shifts can be stored explicitly (as a set of strings), or implicitly (as pairs of index and offset). We refer to the two values of SHIFT_DATA as Explicit and Implicit. Explicit data representation improves time performance but may cause lower space performance, while Implicit data representation improves space performance but may cause lower time performance.
- *Internal representation of lines* (LINE_REPRESENTATION). The lines read from input device could be stored in compressed or uncompressed form. We refer to the two values of LINE_REPRESENTATION as Compressed and Uncompressed. Compressed representation improves space performance but may cause lower time performance, while Uncompressed representation improves time performance but demands more space.
- *Architectural Style* (ARCH_STYLE). As described by Shaw and Garlan, the architecture style for a KWIC system could be Shared Data (functional decomposition), ADT (Abstract Data Types, i.e., Object-Oriented), Implicit Invocation, and Pipe & Filters. Detailed discussions on these architectural styles and how they affect software quality can be found in [25, 33]. In this paper, for the sake of simplicity, we only discuss two architectural styles: ADT and Pipe & Filters.

After careful analysis, we find that:

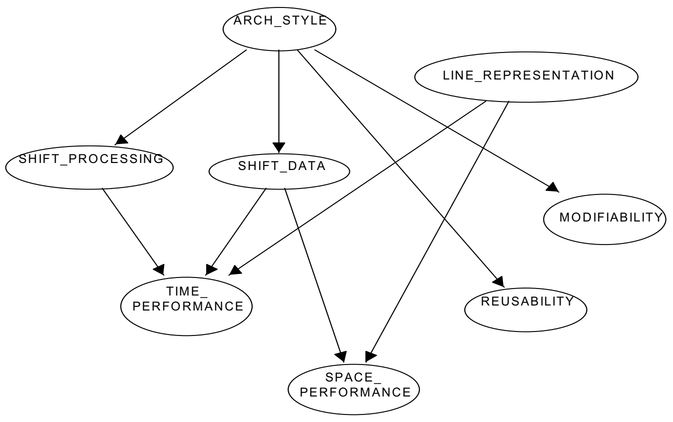
- One design decision may influence many quality attributes. For example, the SHIFT_DATA design decision has influence on both time performance and space performance.
- One quality attribute may be influenced by many design decisions. For example, the time performance is influenced by three design decisions: SHIFT_PROCESSING, SHIFT_DATA and LINE_REPRESENTATION.
- Design decisions may be competing or synergistic. Many design decisions may influence a quality attribute, with some contributing positively and others contributing negatively. For example, both SHIFT_DATA and LINE_PRESENTATION design decisions have impact on system space performance. If the SHIFT_DATA is Implicit, it will contribute positively towards good space performance. However, if at the same time the LINE_REPRESENTATION is Uncompressed, then it will have a negative influence on space performance. Thus, when "SHIFT_DATA = Implicit" and "LINE_REPRESENTATION = Uncompressed", these two design decisions have conflicting impact on space performance. We call them competing design decisions. Similarly, design decisions could also be synergistic, meaning that they have similar impact on a quality attribute.
- Some design decisions may be dependent on others. For example, the Pipe & Filter architectural style is limited to the sequential/batch processing, precluding the use of the incremental processing. Thus we say that the decision on SHIFT_PROCESSING depends on the decision on ARCH_STYLE. Furthermore, if the architectural style is Pipe & Filter, each filter (such as Input, Circular Shift and Alphabetizer) has to keep a copy of lines, thus the circular shifts can only be stored explicitly. Therefore the decision on SHIFT_DATA is also dependent on ARCH_STYLE.

3.3. *KWIC Bayesian network — The graphical model*

We use a DAG (Directed Acyclic Graph) to explicitly model quality attributes, design decisions and their inter-relationships. The DAG forms the graphical model of a Bayesian Network. Figure 2 shows the graphical model of the Bayesian Network for the KWIC domain.

In Fig. 2, nodes in DAG represent design decisions and quality attributes, and directed edges represent the influential directions between nodes (one has influence on the other). From Fig. 2, we can see that four quality attributes, TIME_PERFORMANCE, SPACE_PERFORMANCE, REUSABILITY and MODIFIABILITY, are identified. These quality attributes are influenced by design decisions SHIFT_PROCESSING, SHIFT_DATA, LINE_REPRESENTATION and ARCH_STYLE. Each quality attribute has two possible values: High or Low. It should be noted that the definition of "High" or "Low" depends on a specific domain's industrial norm and it varies from domain to domain.

The graphical model of Fig. 2 captures domain experts' knowledge and



Possible Values of Variables:

SHIFT_PROCESSING: Batch, Incremental	SHIFT_DATA: Explicit, Implicit
ARCH_STYLE: ADT, Pipe & Filter	LINE_REPRESENTATION: Compressed, Uncompressed
TIME_ PERFORMANCE: High, Low	SPACE_ PERFORMANCE: High, Low
REUSABILITY: High, Low	MODIFIABILITY: High, Low

Fig. 2. The graphical model of the Bayesian network for KWIC domain.

experiences. By studying the model, one can see the design alternatives and quality attributes that have been considered during architectural design. In this aspect, this model resembles the Softgoal Interdependency Graph (SIG) proposed by [5].

Figure 2 also reveals additional dependencies among design decisions. To identify these dependencies, we can perform qualitative reasoning over the graphical model as follows:

(1) We know that SHIFT_DATA and LINE_REPRESENTATION have influence on the SPACE_PERFORMANCE. Thus, if all these two design decisions contribute positively, the probability of having high space performance is high (here, we use probability because there may be other factors that affect the space performance, we cannot explicitly and precisely emulate all possible factors that could affect a quality attribute). If for some reasons, one design decision has a negative influence on SPACE_PERFORMANCE (e.g., when SHIFT_DATA is Explicit), then in order to get high space performance, we should choose the appropriate value of the other (e.g., LINE_REPRESENTATION to be Compressed) so that it can contribute positively. Note that if we do not care about the value of SPACE_PERFORMANCE and TIME_PERFORMANCE, SHIFT_DATA and LINE_REPRESENTATION are mutually independent: a decision made on one node does not affect the decision on the other. However,

if we want the SPACE_PERFORMANCE (or TIME_PERFORMANCE) to have certain value (e.g., SPACE_PERFORMANCE to be high), then the SHIFT_DATA design decision and the LINE_REPRESENTATION design decision are "connected" (the selection of the SHIFT_DATA may have an impact on the selection of LINE_REPRESENTATION). We say that SHIFT_DATA and LINE_REPRESENTATION are conditionally dependent given SPACE_PERFORMANCE (or TIME_PERFORMANCE). Similarly, SHIFT_PROCESSING and SHIFT_DATA are conditionally dependent given TIME_PERFORMANCE.

(2) If we want TIME_PERFORMANCE to be high, then the most probable value of SHIFT_DATA is Explicit, and we can then reason that the SPACE_PERFORMANCE could be lower. Similarly, if we want SPACE_PERFORMANCE to be high, the most probable value of SHIFT_DATA is Implicit, and we can then reason that the TIME_PERFORMANCE could be lower. However, if we have made a decision on SHIFT_DATA, then there will be no information flow between TIME_PERFORMANCE and SPACE_PERFORMANCE. These two nodes become independent of each other. We say that TIME_PERFORMANCE and SPACE_PERFORMANCE are conditionally independent given SHIFT_DATA. Similarly, TIME_PERFORMANCE and SPACE_PERFORMANCE are conditionally independent given LINE_REPRESENTATION.

(3) If the decision on SHIFT_DATA is not made, the decision on ARCH_STYLE will have influence on SHIFT_DATA, which in turn has influence on SPACE_PERFORMANCE. However, if we have made a decision on SHIFT_DATA, then the "information path" between ARCH_STYLE and SPACE_PERFORMANCE is blocked. We say that ARCH_STYLE and SPACE_PERFORMANCE are also conditionally independent given SHIFT_DATA.

In general, as shown in Fig. 3, conditional independence appears in the cases of linear and diverging connections in a DAG, where nodes a and c are conditional independent given b. Conditional dependence appears in the case of a converging connection, where a and c are conditional dependent given b. Identifying conditional independence/dependence that is encoded in a Bayesian Network is important, as it reveals complex relationships among variables in a domain. Knowing conditional independence/dependence also reduces the time and space required for an inference engine. For more information about qualitative reasoning over Bayesian Network, we refer the reader to [20].

3.4. *KWIC Bayesian network — The quantitative part*

Based on the graphical model, we can perform simple qualitative reasoning as described in the previous section. One advantage of Bayesian Network is that it can capture not only the qualitative relationships among variables but also quantify the

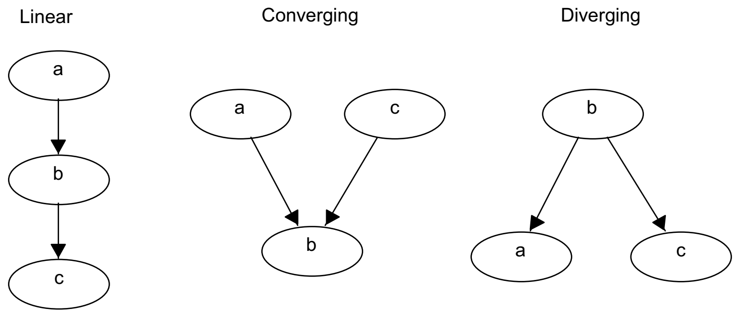


Fig. 3. The general cases of conditional independence/dependence.

conceptual relationships. By quantifying a graphical model, the reasoning process could be automated.

To quantify a graphical model, we assign a conditional probability to each node (nodes that have no parents are assigned prior probabilities). For example, in KWIC domain, the probability of having high or low space performance system is mainly conditioned by two design decisions: SHIFT_DATA and LINE_REPRESENTATION. Thus the conditional probability is given as:

$$P(\text{SPACE\_PERFORMANCE} \,|\, \text{LINE\_REPRESENTATION, SHIFT\_DATA})$$

A probability number reflects a domain expert's belief in how much a given design decision influences a quality attribute. The process of acquiring probabilities from experts is the process of knowledge engineering, in which knowledge engineers conduct surveys or interviews with experts. To assess the probability numbers, one can adapt Boehm's method used in software risk management [3]. In this method, one uses a probability-range estimation to represent the high (0.7 to 1.0), medium (0.4 to 0.6) and low (0.0 to 0.3) degree of influence. Value 1.0 is logic True — if $P(A = a \,|\, B = b)$ is 1.0, it means that given $B = b$, A must be a. Value 0.0 is logic False — if $P(A = a \,|\, B = b)$ is 0.0, it means that given $B = b$, A should not be a. Other values between 0.0 and 1.0 are in proportion to the degree of influence. The use of probability reflects the uncertain and undeterministic natures of the quality attributes and design decisions.

For example, for the SPACE_PERFORMANCE node, we have the following statements:

- When LINE_REPRESENTATION is Compressed and SHIFT_DATA is Implicit, the SPACE_PERFORMANCE has the highest probability of being high.
- When LINE_REPRESENTATION is Uncompressed and SHIFT_DATA is Explicit, the SPACE_PERFORMANCE has the lowest probability of being high.
- If none of the above, the SPACE_PERFORMANCE has medium probability of being high.

Table 2. The CPT of P(SPACE_PERFORMANCE | LINE_REPRESENTATION, SHIFT_DATA).

		P(SPACE_PERFORMANCE | LINE_REPRESENTATION, SHIFT_DATA)	
LINE_REPRESENTATION	SHIFT_DATA	High	Low
Compressed	Implicit	0.8	0.2
Compressed	Explicit	0.6	0.4
Uncompressed	Implicit	0.6	0.4
Uncompressed	Explicit	0.2	0.8

Table 3. The CPT of P(SHIFT_DATA |ARCH_STYLE).

	P(SHIFT_DATA | ARCH_STYLE)	
ARCH_STYLE	Explicit	Implicit
ADT	0.5	0.5
Pipe & Filter	1	0

Table 4. The prior probability P(ARCH_STYLE).

ARCH_STYLE	P(ARCH_STYLE)
ADT	0.5
Pipe & Filter	0.5

Using Boehm's method, we obtain the conditional probability table (CPT) for the SPACE_PERFORMANCE node as shown in Table 2.

In this experiment, the probabilities we obtained are subjective probabilities, which reflect our beliefs and analysis. The assessments can be refined by the group-consensus technique [14, 7], which assess the probability from a group of experts.

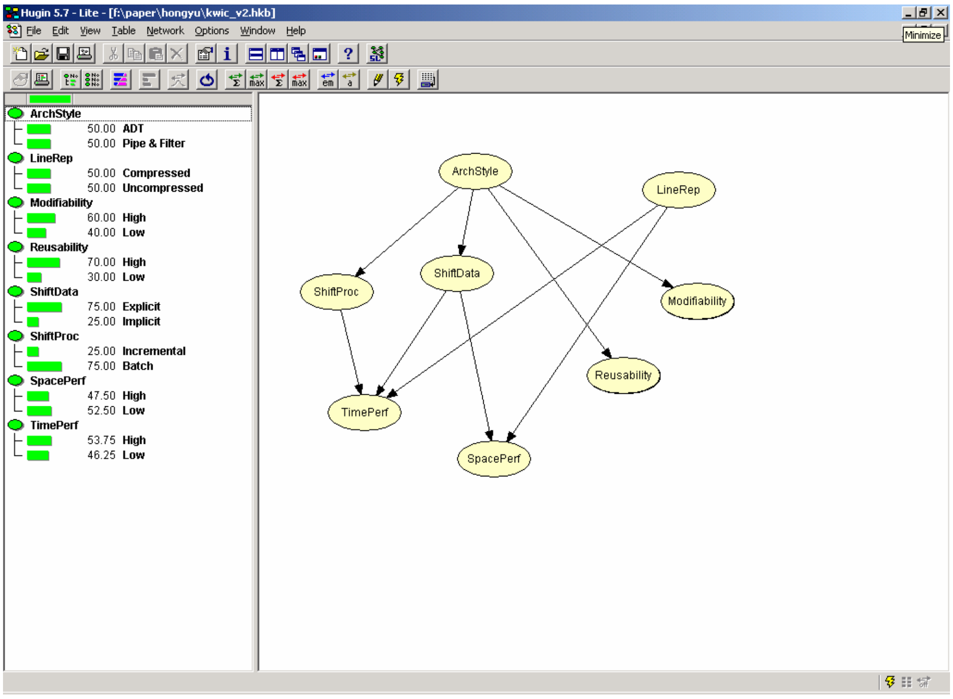
For the SHIFT_DATA node, we have the following statements:

- When ARCH_STYLE is Pipe & Filter, the SHIFT_DATA can only be Explicit.
- When ARCH_STYLE is ADT, the SHIFT_DATA can be either Explicit or Implicit with equal probability.

We obtain the conditional probability table for the SHIFT_DATA node as shown in Table 3.

If no conditions (i.e., no parents) are given, nodes are assigned prior probabilities. For example, the prior probability table for the node ARCH_STYLE is shown in Table 4. The probability of 0.5 shows that if no information is given, the probability of having ADT architectural style and the probability of having Pipe & Filter architectural style are the same (we have no preference for one over the other as no information for decision making is known).

Similarly, we can assign probabilities to other nodes in Fig. 2. Due to space limitation, here we will not list all the probability tables.



```
ArchStyle
   50.00  ADT
   50.00  Pipe & Filter
LineRep
   50.00  Compressed
   50.00  Uncompressed
Modifiability
   60.00  High
   40.00  Low
Reusability
   70.00  High
   30.00  Low
ShiftData
   75.00  Explicit
   25.00  Implicit
ShiftProc
   25.00  Incremental
   75.00  Batch
SpacePerf
   47.50  High
   52.50  Low
TimePerf
   53.75  High
   46.25  Low
```

Fig. 4. The KWIC Bayesian Network constructed using Hugin tool.

3.5. *KWIC Bayesian network — The implementation*

Many software tools have been developed to support the construction, maintenance and usage of Bayesian Network based knowledge bases. Some examples include the Microsoft Bayesian Network toolkit[b] and the Bayes Net Toolbox for Matlab.[c] We implement the Bayesian Network for KWIC domain using the Hugin$^{\text{TM}}$ tool.

Figure 4 shows a screenshot of the KWIC Bayesian Network constructed in Hugin$^{\text{TM}}$ (the nodes' names are displayed in a shorter form). The probability of each node is shown in the left frame of the GUI.

In this experiment, our Bayesian Network only contains eight variables. But it is possible to construct very large Bayesian Networks. For example, Bayesian Networks containing several thousand nodes and links have been used successfully to represent medical knowledge and to achieve high levels of diagnostic accuracy [12]. In the software defect prediction domain, Fenton and Neil reported that they have built Bayesian Networks with several hundred variables in three separate industrial applications [8].

[b]Microsoft Bayesian Network toolkit is available at: http://www.research.microsoft.com/adapt/MSBNx
[c]Bayes Net Toolbox for Matlab is available at: http://www.ai.mit.edu/~murphyk/Software/BNT/bnt.html

The Bayesian Network constructed for a domain is an asset of an organization, it can be reused when designing similar systems in the domain. The efforts on constructing a Bayesian Network will pay off when the model is reused for future projects. The model is also useful during software maintenance and evolution as we often need to go back to review the early design decisions.

4. Rationalizing KWIC Architectural Design

4.1. *Making rational design decisions*

During architectural design, we can examine the Bayesian Network to understand the design alternatives and their relationships with quality attributes, and to make decisions on adopting or rejecting a specific design. Not all decisions are "good" with respect to the quality requirements. Different decisions may lead to systems that satisfy the same set of functional requirements, but differ in certain quality attributes. Very often, we need to make tradeoffs among many "competing" decisions. Thus we should make careful decisions so that the resulting system could satisfy required functional requirements and exhibit desired quality attributes.

To better understand the overall influence of design decisions on quality attributes, we can predict the quality of the system by performing quantitative analysis over the Bayesian Network. For example, if an architect designs a specific KWIC system as follows (marked as configuration 1):

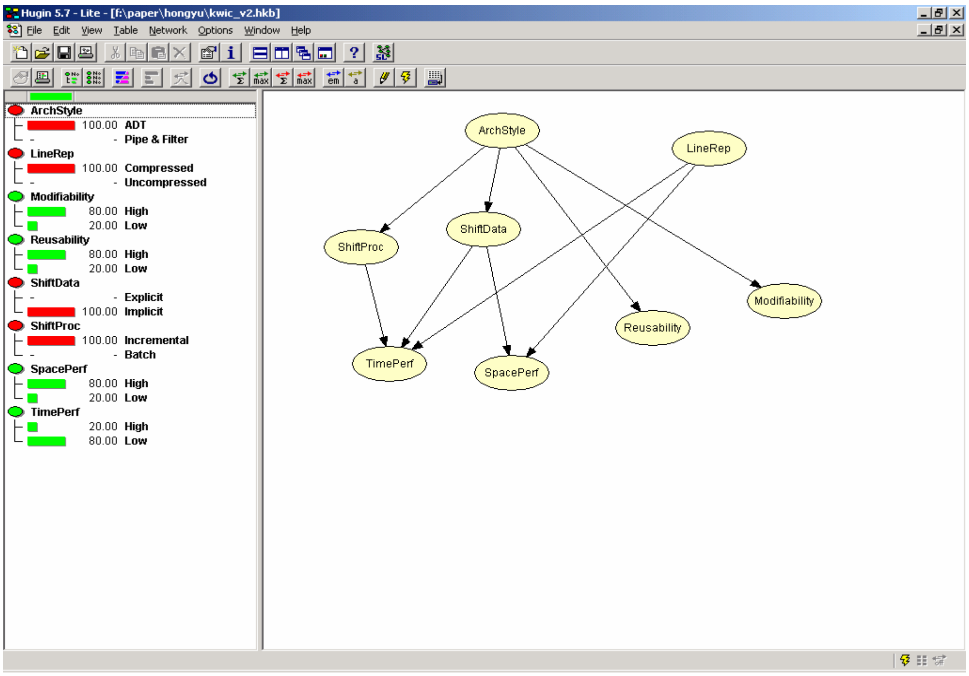
- ARCH_STYLE = ADT
- SHIFT_PROCESSING = Incremental
- SHIFT_DATA = Implicit
- LINE_REPRESENTATION = Compressed

We can predict the quality of the resulting KWIC system through the KWIC Bayesian Network. Assuming the above decisions have been made, we can update the Bayesian Network in Fig. 4 by propagating the decisions over the network and updating the probability of each node. There are many inference algorithms available [27, 20]. In our project, we use the *junction* tree method [20], one of the most efficient and scalable methods for making inferences over a Bayesian Network. We can perform the junction tree inference with the aid of the Hugin$^{\text{TM}}$ tool. Figure 5 shows the updated Bayesian Network.

In the left frame of Fig. 5, nodes that have a value of 100.00 indicate the decisions made. We can predict that the probabilities of system having high space performance, high reusability and high modifiability are all 80%. However, the probability of having a high time performance system is only 20%.

For the same KWIC system, the architect may also have the following design (marked as configuration 2):

- ARCH_STYLE = Pipe & Filter
- SHIFT_PROCESSING = Batch


Hugin 5.7 - Lite - [f:\paper\hongyu\kwic_v2.hkb]
File Edit View Table Network Options Window Help
ArchStyle
LineRep
ShiftProc
ShiftData
TimePerf
SpacePerf
Reusability
Modifiability

ArchStyle
- 100.00 ADT
- - Pipe & Filter

LineRep
- 100.00 Compressed
- - Uncompressed

Modifiability
- 80.00 High
- 20.00 Low

Reusability
- 80.00 High
- 20.00 Low

ShiftData
- - Explicit
- 100.00 Implicit

ShiftProc
- 100.00 Incremental
- - Batch

SpacePerf
- 80.00 High
- 20.00 Low

TimePerf
- 20.00 High
- 80.00 Low

Fig. 5. The updated KWIC Bayesian Network (with configuration 1).

Table 5. The comparison between configuration 1 and configuration 2.

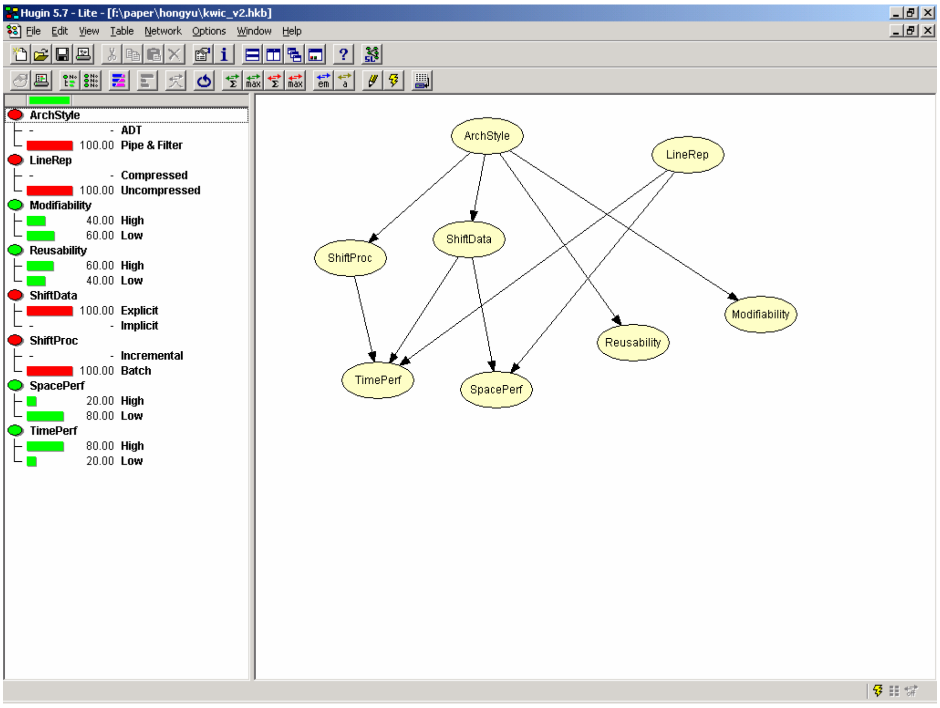
	TIME_PERFORMANCE	SPACE_PERFORMANCE	REUSABILITY	MODIFIABILITY
Configuration 1	20%	80%(+)	80%(+)	80%(+)
Configuration 2	80%(+)	20%	60%	40%

- SHIFT_DATA = Explicit
- LINE_REPRESENTATION = Uncompressed

We can predict from Fig. 6 that this time, the probability of having a high time performance system is 80%, the probability of having a high space performance system is 20%, the probability of having a highly reusable system is 60%, and the probability of having a highly modifiable system is 40%.

Table 5 shows the comparison between configuration 1 and configuration 2. The numbers shown are the probabilities of having "High" quality. The "+" sign indicates the superior configuration with respect to the quality attribute.

Configuration 1 is better than configuration 2 with respect to space performance, reusability, and modifiability. However, configuration 1 may lead to a system that is more likely to have low time performance. Configuration 2 is superior with respect to time performance. However, it may result in a system that is more likely


Hugin 5.7 - Lite - [f:\paper\hongyu\kwic_v2.hkb]
File Edit View Table Network Options Window Help
ArchStyle
LineRep
ShiftProc
ShiftData
TimePerf
SpacePerf
Reusability
Modifiability

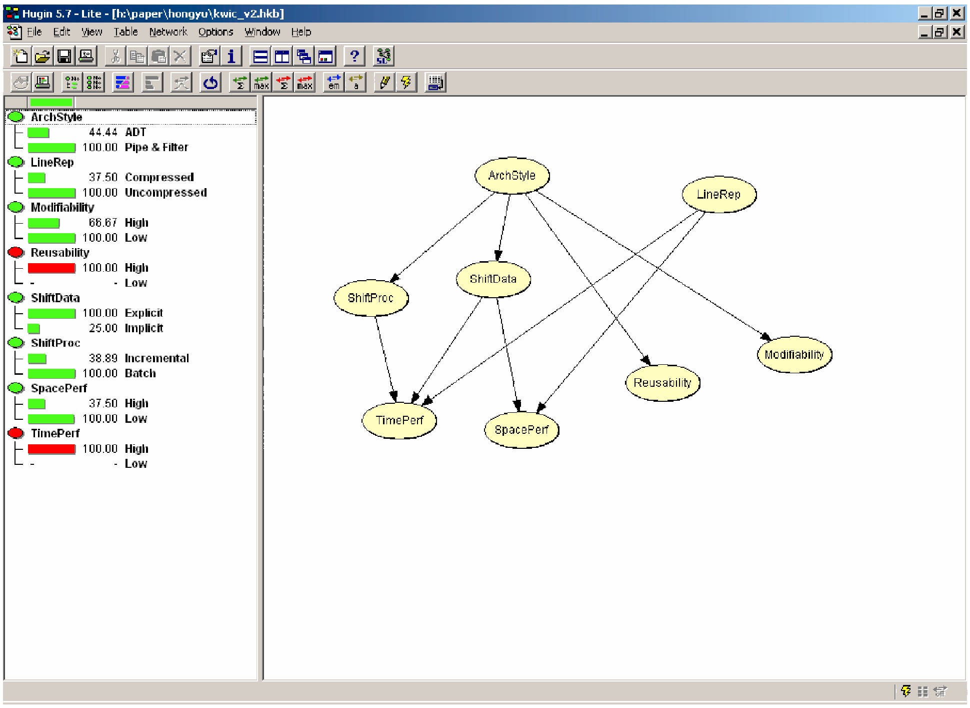
```
ArchStyle
  -          ADT
  100.00     Pipe & Filter
LineRep
  -          Compressed
  100.00     Uncompressed
Modifiability
  40.00      High
  60.00      Low
Reusability
  60.00      High
  40.00      Low
ShiftData
  100.00     Explicit
  -          Implicit
ShiftProc
  -          Incremental
  100.00     Batch
SpacePerf
  20.00      High
  80.00      Low
TimePerf
  80.00      High
  20.00      Low
```

Fig. 6. The updated KWIC Bayesian Network (with configuration 2).

to have low space performance and low modifiability. If the customer's preference of quality attributes is reusability $\gg$ modifiability $\gg$ space performance $\gg$ time performance (where "$\gg$" means "much more important than"), then the configuration 1 is preferable. If the customer's preference is time performance $\gg$ space performance $\gg$ reusability $\gg$ modifiability, then configuration 2 is preferable.

To evaluate more precisely a design with respect to customer's preference on multiple quality attributes, we can also use more rigorous techniques such as the Analytical Hierarchy Process (AHP [31]), which assigns weights to each quality attribute and calculates an overall score for each configuration of design decisions. The better design is the one that has a higher overall score.

Not all combinations of design decisions are valid. Design decisions could be mutually dependent or exclusive. By performing inference over the Bayesian Network, we can also check the consistency of our decisions. For example, if we select "ARCH_STYLE = Pipe & Filter" and "SHIFT_DATA = Implicit" together, an inconsistency error will be reported during inference (due to the inter-dependency between SHIFT_DATA and ARCH_STYLE). Consistency checking helps us verify if a given set of design decisions is indeed valid. It is especially helpful when we deal with large and complex design spaces.


Hugin 5.7 - Lite - [h:\paper\hongyu\kwic_v2.hkb]
File Edit View Table Network Options Window Help
ArchStyle
LineRep
ShiftData
ShiftProc
TimePerf
SpacePerf
Reusability
Modifiability

```
ArchStyle
   44.44 ADT
  100.00 Pipe & Filter
LineRep
   37.50 Compressed
  100.00 Uncompressed
Modifiability
   66.67 High
  100.00 Low
Reusability
  100.00 High
       - Low
ShiftData
  100.00 Explicit
   25.00 Implicit
ShiftProc
   38.89 Incremental
  100.00 Batch
SpacePerf
   37.50 High
  100.00 Low
TimePerf
  100.00 High
       - Low
```

Fig. 7. Automating the design decision making.

4.2. *Automating the design decision making*

In the examples given in the last section, a software architect inputs different design decisions. Our Bayesian Network then estimates the resulting system's quality and helps the architect select among design alternatives. It is a "top-down" approach and requires human intervention. In this section, we introduce a "bottom-up" approach, which automates the design decision-making process.

Assuming a customer wants the system to have high time performance and high reusability and the customer "doesn't care" about the other two quality attributes (space performance and modifiability). Using the constructed Bayesian Network, we can infer a set of "good" design choices that is most likely to satisfy the customer's requirements. Figure 7 shows the updated Bayesian Network. We first assume that the resulting system has the quality of high time performance and high reusability, we then perform a *max-propagation* [20] and infer the most likely combination of design decisions that cause the system to have such quality. In Fig. 7, a state with the value 100.00 belongs to a most probable configuration of variables. We can see that in order for time performance and reusability to be High, we could choose the following design configuration:

- ARCH_STYLE = Pipe & Filter

- SHIFT_PROCESSING = Batch
- SHIFT_DATA = Explicit
- LINE_REPRESENTATION = Uncompressed

5. Discussions and Comparisons

Our Bayesian Network captures design knowledge and experiences of domain experts. The graphical model represents the design space intuitively. The nodes in a graphical model represent design decisions and quality attributes; the edges represent the influential relationship among design decisions and quality attributes. After constructing a Bayesian Network, we could store it into an organization's knowledge base, together with other software assets such as UML models, software architecture, components, etc. During software maintenance and reuse, we can retrieve the Bayesian Network from knowledge base, and examine the design alternatives and quality attributes that were considered in previous projects. We can also perform qualitative analysis over the Bayesian Network to understand the relationships among design decisions and qualities. The following design errors usually caused by inexperience could be reduced:

- Desired quality attributes are not addressed in the design;
- Important design decisions are not considered;
- Possible design alternatives are not considered;
- The impact of design decisions on software quality is not evaluated.

We use subjective probabilities to quantify the graphical model. The use of subjective probability helps us make reasonable estimations at early design stage before the actual system is built. Subjective estimations are necessary in software engineering as "an important aspect of software engineering is the ability to do estimations of process and product attributes" [16]. Many subjective estimations based methods have been proposed, such as Putnam's method for estimation of program length [30] and Höst and Wohlin's method for effort estimation [16]. The subjective probabilities used in a Bayesian Network reflect domain experts' beliefs, which can be elicited from domain experts through a process such as knowledge engineering. The probability assessments can also be refined by the group-consensus technique [14, 7], which assesses the probability from a group of experts. For some design decisions and quality attributes, the assessments could be done through the collection of objective data from similar projects.

Like all formalisms, understanding the Bayesian Network formalism (including the graphical model, the probability theory and the probabilistic reasoning) requires some training, but it rewards us with the ability to better understand and analyze complex problems.

In the situation that many design decisions influence one quality attribute, it may be difficult to assess all the conditional probabilities. For example, if there are

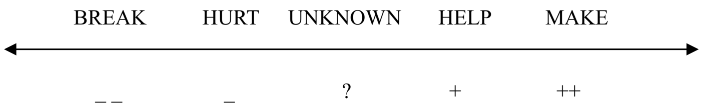


Fig. 8. The contribution catalogue in NFR framework.

n design decisions that influence one quality attribute and each of them has two possible values, we need to assess 2^n probabilities. We can use the Noisy-OR method [20], which only requires assessing n probabilities, thus significantly reducing the size of the CPT.

In a Bayesian Network, after knowing the probability of B given A, we can infer the probability of A given B using the Bayes rules. Although there is only one direct arrow between two nodes in a Bayesian Network, the inference could be bi-direction. Thus, there is no cycle between two nodes in a Bayesian Network (therefore a DAG is sufficient). In our experimentation, we did not encounter the situation in which we need to model a cyclic effect between design decisions.

The construction of a Bayesian Network is a knowledge management effort, thus it is not a one-time activity. Instead, we should continuously maintain and update the Bayesian Network to reflect the new knowledge and experiences accumulated from practices.

Chung *et al.* proposed a non-functional requirement (NFR) framework [5, 24], which aims to improve software quality and has been studied for a variety of information systems. In NFR framework, non-functional requirements (quality attributes) are treated as soft-goals (goals that do not have a clear-cut criteria for their satisfaction) and represented diagrammatically as *softgoal interdependency graphs* (*SIGs*). A SIG explicitly models NFRs, design alternatives, and the reasoning process of satisfying non-functional requirements. The contributions of design alternatives to quality attributes are classified into five types: MAKE (sufficient positive support), BREAK (sufficient negative support), HELP (partial positive support), HURT (partial negative support), and UNKNOWN (contribution unclear). Figure 8 illustrates these contribution types in a spectrum, ranging from sufficient negative support ("--") on the left to sufficient positive support ("++") on the right.

Figure 9 shows an example of SIG for the KWIC domain. We could see that similar to our Bayesian Network, a SIG also codifies the knowledge of making architectural design decisions and helps to systematically guide selection among design alternatives. It is also an improvement over *ad hoc* approach. However, we point out the following limitations of the NFR framework:

- SIGs are difficult to scale up. A SIG explicitly shows the impact of each design alternative on quality attributes. The diagram could grow exponentially. This could make a SIG difficult to read and maintain.
- SIGs do not show the influential relationships and conditional dependencies/

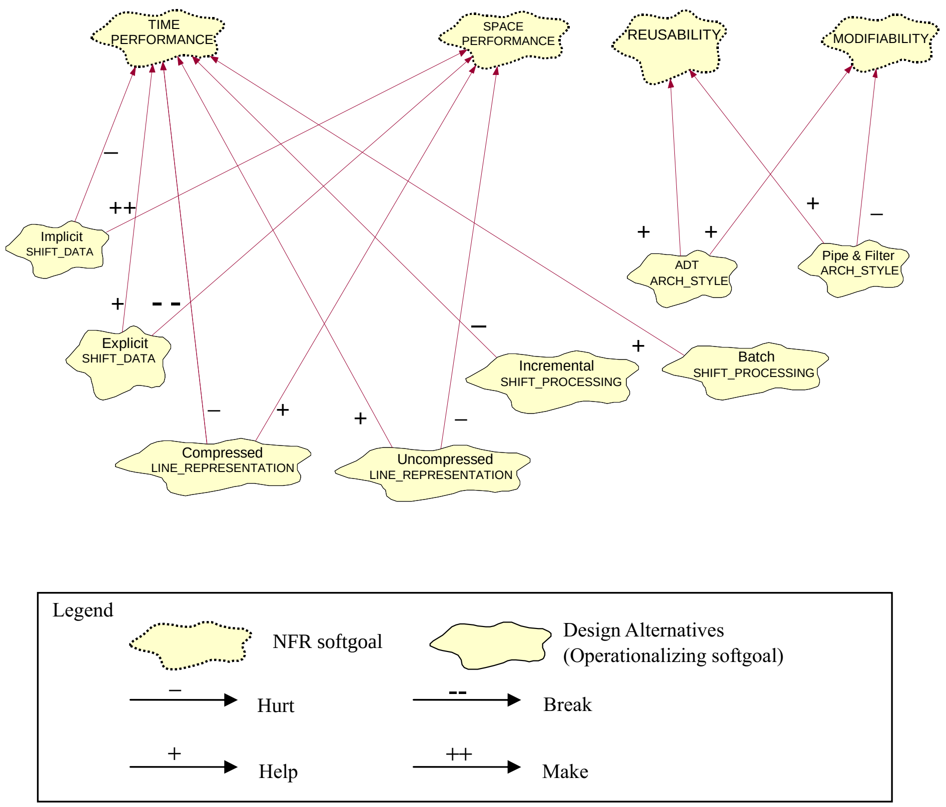


Fig. 9. A SIG for the KWIC domain.

independencies among the design decisions. In qualitative reasoning, the influential relationships and conditional dependencies/independencies reveals additional relationships among design decisions, which could otherwise be considered completely independent.

- SIGs do not have an underlying quantitative model, which hampers automated reasoning. Whereas our Bayesian Network contains both graph-based model (for representing uncertain knowledge in a domain) and underlying probabilistic model (for quantitative reasoning).

6. Related Work

In their influential paper "A Rational Design Process: How and Why to Fake It" [26], Parnas and Clements pointed out that a rational design is important even though we cannot develop software in a completely rational fashion, because a rational design could guide architects in the design process rather than let them proceed on an *ad hoc* basis.

Parnas and Clements addressed the problem of software documentation and recommended post-mortem documentation of a software product. The rationalized software product documentation communicates important "lessons learned", but it is not the only source of past experiences that can be reused for future projects. At the end of the paper, Parnas and Clements suggested that we should also record all of the design alternatives that we considered and rejected.

Much has happened since Parnas and Clements wrote their paper. SEI formulated CMM [32] that classifies development processes according to their maturity levels. "Experience factory" approaches [1] were researched to facilitate accumulation of project experiences in order to continuously improve software development practices. These projects provide many useful guidelines that help companies progress from *ad hoc* development towards the "rational design process". The experienced factory approach collects and manages a variety of experiences including project data, technology reports and lessons learned, and provides repository services for this experience. As a representation of design knowledge, our Bayesian Network could also be stored and managed in an experience factory.

In software requirements engineering context, quality attributes are also called non-functional requirements (NFRs) and goal-oriented approaches have been proposed to address NFRs [29, 5, 23]. In the last section, we briefly compared our approach with the NFR framework approach proposed by Chung *et al.*

ATAM [6] and its predecessor SAAM [2] evaluate an architecture mainly after the architecture is designed. These methods require a brainstorming section to generate a list of possible scenarios to test the architectures for single systems. Our approach can be used at an early stage of architectural design. We explore design alternatives through careful domain analysis, and produce Bayesian Network that is reusable in a domain. It would be interesting to explore how ATAM and our approach can complement each other.

Architectural styles [33] and design patterns [10] are related to our work in that they both capture reusable design knowledge accumulated in practices. The difference is that the architectural styles capture structural pattern of software architectures, the design patterns capture a set of collaborating objects that represent a common design solution, while our Bayesian Network captures the knowledge of making architectural design decisions.

There are some performance models proposed [28] to establish a link between software performance and software architecture. However, these performance models do not address other quality attributes such as maintainability or reusability. There are also many models for making tradeoffs between reliability and cost [22, 35]. These models facilitate cost-efficient progress toward a quantified system reliability goal. Our proposed approach considers many quality attributes and enables subjective estimations based on design decisions.

7. Conclusion

Although getting an architecture right is very important, achieving a rational architectural design is difficult. Different architectural design decisions may lead to systems that satisfy the same set of functional requirements but differ in certain quality attributes. Software architects have to make decisions on adopting or rejecting a design from a web of complex, and often uncertain information. In this paper, we proposed a Bayesian Network based approach to help software architects make better decisions during architectural design. Our Bayesian Network captures the knowledge about making architectural design decisions. We construct a graphical model to intuitively record the relationships among design decisions and quality attributes, and use subjective probabilities to quantify the graphical model. Various analyses, such as prediction of software quality given design decisions and automated design decision making, can be performed over a constructed Bayesian Network. We illustrated our approach using the KWIC example.

We believe our approach helps software architects analyze design alternatives and make more informed decisions at early stage of software development. Our approach could also automate the design decision making process, making it possible for a software agent to design a software architecture. The space of design decisions for KWIC domain described in this paper is small. In the future, we should evaluate the effectiveness and scalability of our approach in large-scale industrial projects. Better methods and tools shall be developed to facilitate the acquisition of probabilities from experts. For some design decisions and quality attributes, the assessment of probabilities could be improved through the collection of objective data. We will also integrate the proposed approach with our frame-based product line development methods [37, 17, 18]. We hope that our approach could be an integral part of knowledge management practice in a software organization.

Acknowledgments

The authors have applied the approach described in this paper to software product line engineering [37]. The authors wish to thank Dr. Yang Bo and Keith Frampton for their valuable suggestions for this project, and thank all anonymous reviewers for their insightful comments.

References

1. V. Basili and G. Caldiera, Improve software quality by reusing knowledge and experience, *Sloan Management* **37**(1) (1995).
2. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.
3. B. Boehm, Software risk management: Principles and practices, *IEEE Software* **8**(1) (1991) 32–41.
4. E. Charniak, Bayesian networks without tears, *AI Magazine*, AAAI, 1991.
5. L. Chung, K. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, 2000.

6. P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures — Methods and Case Studies*, Addison-Wesley, 2002.
7. R. Cooke, *Experts in Uncertainty — Opinion and Subjective Probability in Science*, Oxford University Press, 1991.
8. N. Fenton and M. Neil, A critique of software defect prediction models, *IEEE Trans. on Software Engineering* **25**(3) (1999).
9. N. Fenton, B. Littlewood, M. Neil, L. Strigini, A. Sutcliffe, and D. Wright, Assessing dependability of safety critical systems using diverse evidence, *IEE Proc. Software Engineering* **145**(1) (1998) 35–39.
10. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns — Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
11. D. Garlan, G. E. Kaiser, and D. Notkin, Using tool abstraction to compose systems, *IEEE Computer* **25** (1992) 30–38.
12. D. Heckerman, E. Horvitz, and B. Nathwani, Towards normative experts systems: Part I. The Pathfinder project, *Methods of Information in Medicine* **31** (1992) 90–105.
13. D. Heckerman, J. Breese, and K. Rommelse, Decision-theoretic troubleshooting, *Communications of the ACM* **38** (1995) 49–57.
14. O. Helmer, *Social Technology*, Basic Books, New York, 1966.
15. E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse, The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users, in *Proc. Fourteenth Conf. on Uncertainty in Artificial Intelligence*, Madison, WI, 1998.
16. M. Höst and C. Wohlin, Experimental evaluation of subjective effort estimation techniques, in *Proc. 20th Int. Conf. on Software Engineering*, Kyoto, Japan, April 1998, pp. 332–339.
17. S. Jarzabek and H. Zhang, XML-based method and tool for handling variant requirements in domain models, in *Proc. Fifth IEEE Int. Symp. on Requirements Engineering* (*RE'01*), Toronto, IEEE Press, August 2001, pp. 166–173.
18. S. Jarzabek, P. Bassett, H. Zhang, and W. Zhang, XVCL: XML-based Variant Configuration Language, in *Proc. 25th Int. Conf. on Software Engineering* (*ICSE 2003*), 2003, pp. 810–811.
19. F. V. Jensen, H. I. Christensen, and J. Nielsen, Bayesian methods for interpretation and control in multi-agent vision systems, in *Proc. Applications of Artificial Intelligence*, Orlando, Florida, 1992.
20. F. V. Jensen, *An Introduction to Bayesian Networks*, UCL Press, London, 1996.
21. B. Kitchenham, Software Metrics, *Software Reliability Handbook*, ed. P. Rook, Elsevier Applied Science, London, New York, 1990.
22. W. Kuo and V. Prasad, An annotated overview of system-reliability optimization, *IEEE Trans. on Reliability* **49**(2) (2000) 176–187.
23. B. A. Nixon, Management of performance requirements for information systems, *IEEE Trans. on Software Engineering* **26** (2000) 1122–1146.
24. J. Mylopoulos, L. Chung, S. Liao, H. Wang, and E. Yu, Exploring alternatives during requirements analysis, *IEEE Software*, January/February, 2001.
25. D. L. Parnas, On the criteria to be used in decomposing software into modules, *Communications of the ACM* **15**(12) (1972) 1053–1058.
26. D. L. Parnas and P. Clements, A rational design process: How and why to fake it, *IEEE Trans. on Software Engineering* **12**(2) (1986).
27. J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988.

28. D. Petriu, C. Shousha, and A. Jalnapurkar, Architecture-based performance analysis applied to a telecommunication system, *IEEE Trans. on Software Engineering* **26** (2000) 1049–1065.
29. G. C. Roman, A taxonomy of current issues in requirements engineering, *IEEE Computer* **18** (1985) 14–23.
30. L. H. Putnam and A. Fitzsimmos, Estimating software costs, *Datamation*, September 1979.
31. T. Saaty, *The Analytical Hierarchy Process*, McGraw-Hill, New York, 1980.
32. SEI, The Capability Maturity Model: Guidelines for Improving the Software Process, Addison-Wesley, Reading, Massachusetts, 1995.
33. M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.
34. H. Singh, V. Cortellessa, B. Cukic, E. Gunel, and V. Bharadwaj, A Bayesian approach to reliability prediction and assessment of component based systems, in *Proc. Int. Symposium on Software Reliability Engineering*, Hong Kong, November 2001, pp. 12–21.
35. S. Wadekar and S. Gokhale, Exploring cost and reliability tradeoffs in architectural alternatives using a genetic algorithm, in *Proc. Int. Symposium on Software Reliability Engineering*, Boca Raton, FL, November 1999, pp. 104–113.
36. D. Wooff, M. Goldstein, and F. Coolen, Bayesian graphical models for software testing, *IEEE Trans. on Software Engineering* **28**(5) (2002) 510–525.
37. H. Zhang, S. Jarzabek, and B. Yang, Quality prediction and assessment for product lines, in *Proc. 15th Int. Conf. on Advanced Information Systems Engineering* (*CAiSE'03*), Klagenfurt/Velden, Austria, June 2003, Springer-Verlag LNCS 2681, pp. 681–695.
38. H. Zhang and S. Jarzabek, An XVCL approach to handling variants: A KWIC product line example, in *Proc. 10th Asia-Pacific Software Engineering Conference* (*APSEC 2003*), Chiangmai, Thailand, IEEE Press, December 2003.