情報処理学会論文誌 Vol. 51 No. 9 1765–1776 (Sep. 2010)

ベイジアンネットワークを用いたソフトウェア実装技術の選択支援

風戸広史[†1,†2] 林晋平[†1]
小林隆志[†3] 佐伯元司[†1]

ソフトウェアの実行基盤を構成する個々の実装技術が，全体的な品質特性にどのような影響を及ぼすかを予測することは難しい．本稿では品質特性と実装技術の因果関係をベイジアンネットワークでモデル化し，実装技術の選択を支援する手法を提案する．また，ベイジアンネットワークの検証ツール上に提案手法を実装し，例題へ適用することによりその有効性を示す．

Choosing Software Implementation Technologies Using Bayesian Networks

Hiroshi Kazato,[†1,†2] Shinpei Hayashi,[†1]
Takashi Kobayashi[†3] and Motoshi Saeki[†1]

It is difficult to estimate how a combination of implementation technologies influences quality attributes on an entire system. In this paper, we propose a technique to choose implementation technologies by modeling casual dependencies between requirements and technoloies probabilistically using Bayesian networks. We have implemented our technique on a Bayesian network tool and applied it to a case study of a business application to show its effectiveness.

†1 東京工業大学大学院情報理工学研究科計算工学専攻
Department of Computer Science, Tokyo Institute of Technology
†2 株式会社 NTT データ技術開発本部
Research and Development Headquarters, NTT DATA CORPORATION
†3 名古屋大学大学院情報科学研究科情報システム学専攻
Department of Information Science, Nagoya University

1. はじめに

高品質のソフトウェアを開発する際には，利用する実装技術を適切に選択する必要がある．ソフトウェア開発は要求分析に始まり，アーキテクチャ設計を経て，使用するアーキテクチャに従った個々の実装技術の選定に入る．その結果，ソフトウェアの実行環境（プラットフォーム）はハードウェア，ライブラリ，フレームワーク，ミドルウェアなどの様々な実装技術を組み合わせて構成される．しかし，ソフトウェアに求められる品質特性に従って，適切な組合せは異なる．たとえば，企業向けに対話型の住所録アプリケーションを開発することを考える．企業内のある従業員が数十人の同僚の連絡先を管理することを想定する場合は，そのデータを単純なファイルとしてストレージに格納することで十分である．一方，数十万人の顧客を管理し全従業員から利用される場合，バックエンドとして関係データベースを利用することが適切である．このように，アプリケーションが想定するスケーラビリティにより，選択すべき実装技術が異なる．これらの選択は，対象アプリケーションの開発ドメインの経験が乏しい開発者にとっては簡単ではないため，その支援が求められる．

対象ドメインでの開発に明るいアーキテクトの知見や過去の開発資産での選択結果をよく反映した選択支援が有用である．要求に基づき適切な実装技術の選択が可能なアーキテクトは不足している．彼らの経験や，社内にかかえる過去の成功案件が示す知見に基づき，実装技術の組合せを低コストで得られれば都合が良い．その際には，以下の要件を考慮することが好ましい．

品質特性と実装技術の関係の不明確さ アーキテクトが持っている経験に基づく知見は主観的であることが多く，曖昧さを含んでいる．

品質特性間のトレードオフ ソフトウェアが満足すべき品質特性の間にはトレードオフがあり[1)]，容易にすべてを満足しがたい．

品質要求の抽出漏れ 重視すべき品質特性をあらかじめ顧客から抽出しつくすことは難しく，重視すべき特性の一部しか明らかになっていない場合がある．また，顧客は品質特性そのものを暗に期待し，たとえば「Web アプリケーションとして作ってほしい」のように，より低レベルの要求しか陽に言わない場合がある．

そこで，本稿ではソフトウェア品質特性と実装技術の因果関係を確率モデルの一種であるベイジアンネットワーク（Bayesian network; BN）[2),3)] を用いてモデル化し，アーキテクチャの選定後に行う実装技術の選択を品質要求によって支援する手法を提案する．BN は人の持つ確信の度合いをベイズ統計学に基づいて確率的に扱うことが可能なモデルであり，提

案手法では BN の以下の性質を利用して前述の問題を解決する．

予備知識とデータの統合　実装技術を選択する段階ではソフトウェアは未完成であり，品質は不確かさを持っている．開発者の知識や経験に基づく主観的な予測は確率として表すことができるため，開発実績に基づくデータとともに 1 つの確率モデルに統合することができる．

確率推論に基づくアルゴリズム群　既存の事後確率計算アルゴリズムを用いることにより，開発者の選択した複数の品質特性の期待効用を最大化する実装技術の組合せを検出することができる．

欠損データの許容　すべての確率変数の因果関係がモデルに含まれるため，一部の確率変数の値が確定すれば未観測の確率変数の確率分布を予測できる．実装技術を選択するためにすべての品質要求を与える必要がない．また，実績値から学習する場合にも確率変数のデータ欠損に対応できる．

本稿の主要な貢献は，以下の 2 点である．

BN に基づく実装技術選択法　提案手法では，各実装技術の選択根拠となる設計上の決定事項を抽出し，品質特性と実装技術の関係を BN 上に表現する．手法が示す BN の構築手順に基づいて，MVC と Layers の 2 つのアーキテクチャスタイル[4] に基づく開発ドメインそれぞれの BN が実際に構築可能だった．

事例研究による有用性の確認　事例では，Layers アーキテクチャスタイルに基づくアプリケーション開発ドメインの BN が実際に構築可能であり，アーキテクトの判断に合致した選択結果が得られた．

本稿の以降の構成を以下に示す．まず，次章で BN の概念を簡潔に導入する．次に 3 章では BN を用いて品質特性と実装技術の因果関係をモデル化する手順と，そのモデルを用いて実装技術を選択する手法を提示する．続いて 4 章で提案手法を用いて実装技術の選択支援を行う例題を示す．5 章では提案手法に関連する先行研究をいくつか紹介し，提案手法との関連性を述べる．最後に 6 章で本稿の結論と今後の展望を述べる．

2. ベイジアンネットワーク

文献 2)，3) などにも詳しく，よく知られたことであるが，後の説明に必要となるので本章で BN の概念を簡潔に導入しておく．

BN は離散的な確率変数をノード，ノード間の因果関係を有向リンクとする有向非循環グラフ（directed acyclic graph; DAG）によって確率分布を表現するモデルである．確率変数 X，Y の定性的な因果関係を有向リンク $X \to Y$ で表し，X を Y の親ノード，Y を X の子ノードと呼ぶ．ノード X_i の親ノードの集合を $pa(X_i)$ で表し，X_i が親を持つ場合は条件付き確率 $P(X_i \mid pa(X_i))$，親を持たない場合は事前分布 $P(X_i)$ を与える．この $P(X_i \mid pa(X_i))$ は，X_i の親ノードがとる具体的な値の組合せごとに X_i の条件付き確率値を与えた条件付き確率表（conditional probability table; CPT）として表現される．

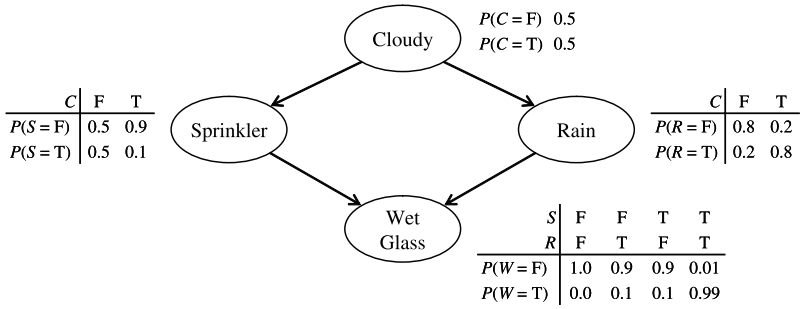

図 1　ベイジアンネットワークの例
Fig. 1　An example of a Bayesian network.

例として，確率変数 C，S，R，W の同時分布を表現する BN を**図 1** に示す．これらの確率変数はそれぞれ真偽値をとり，C：天気が曇る（Cloudy），S：スプリンクラが作動する（Sprinkler），R：雨が降る（Rain），W：芝生が湿る（Wet Glass）という 4 つの事象を表している．図 1 中に示されたグラフの各ノードはそれぞれの確率変数を，付近に付与された表はそれらの CPT を表す．C は親ノードを持たない確率変数で，これには均等な事前分布，すなわち $P(C = \mathrm{F}) = P(C = \mathrm{T}) = 0.5$ が与えられている．また，R は親ノードとして C を持ち，その CPT には C との条件付き確率が表現される．たとえば，表の左上の要素は，$C = \mathrm{F}$ であった場合に $R = \mathrm{F}$ となる確率，$P(R = \mathrm{F} \mid C = \mathrm{F}) = 0.8$ を表している．また，たとえば W などの複数の親を持つ要素には，親ノードの値の組合せに対する確率が記述された CPT を付与する．

BN 上の確率変数の計算を確率推論と呼ぶ．BN のノードのうち，いくつかの値が観測値 e によって固定された場合，未観測のノードに関する事後確率 $P(X \mid e)$ を計算したり，同時分布が最大となる値の組合せとその確率値を見つけたりするための確率推論のアルゴリズムが多数考案されている．本稿では，Junction Tree アルゴリズム[5] と焼きなまし MAP 法[6] を用いる．Junction Tree アルゴリズムでは，観測値が固定されていないすべてのノー

ドに対して，その事後確率を厳密値で計算できる．また，焼きなまし MAP 法では，焼きなまし法を用いて同時分布が局所的最大となる値の組合せとその確率値を発見できる．後者は厳密解を得られないものの，観測したデータに対して妥当な確率変数の値を，組合せとして検出可能な点が優れている．

3. 提案手法

本章では BN を用いて実装技術の選択を支援する手法を提示する．プラットフォームの決定は，大きく分けて，(1) BN による対象ドメインのモデル化，および (2) 構築した BN に基づく実装技術の選択の 2 フェーズからなる．(1) は，開発対象のアーキテクチャに明るいアーキテクトが事例を分析し，開発対象のドメインごとに 1 度だけ行う．これにより，アーキテクトの知見やディスカッション時に検討された過去の開発プロジェクトでの選択結果が BN のリンク構造や定量的な確率にエンコードされる．(2) は，得た BN を同ドメインのアプリケーション開発時に利用する．アプリケーション開発者は，要求分析の結果にともないアーキテクチャを決定した後，品質特性の重視の度合いなどを証拠として対象アーキテクチャをモデル化した BN に入力し，確率推論により採用すべき実装技術の組合せを得る．組合せの特定は BN 上の計算により自動的に行われるため，アプリケーション開発者が対象ドメインに明るくない場合でも，適切な実装技術の組合せを得ることができる．以降では，企業向けの対話型業務アプリケーションの開発を想定し，その実装技術の選択支援を行う事例を用いながら提案手法を説明する．

3.1 BN による対象ドメインのモデル化

3.1.1 確率変数の抽出

提案手法では実装技術の背後にある設計上の決定事項をモデルに明示し，それらを仲介して品質特性の達成レベルと実装技術の選択肢を確率的に関連付けることにより，開発の初期段階における設計という不確定性を含む問題の分析を支援する．提案手法では，BN 上の確率変数を以下の 3 群に分けて抽出する．

(1)　Q 群：対象ドメインで求められる品質特性．
(2)　C 群：実装技術の選択根拠となる設計上の決定事項．
(3)　I 群：対象ドメインで利用予定のある実装技術．

個々の実装技術は具体的すぎ，品質特性との因果関係を明確にしにくい．そこで，それぞれの実装技術を裏付ける設計の根拠を設計上の決定事項として抽出し，C 群を構成する．Q 群，I 群の間に中間層として C 群を設けることにより，実装技術が求める品質を満足する理由を利用者に明示できる，共通する設計に基づく実装技術が存在する場合には後段で付与する CPT を小さくでき BN の保守性に貢献する，などの利点がある．

Q 群の要素は，対象ドメインで求められる品質特性を列挙し，それらの相対的な達成レベルを状態とする離散的な確率変数である．たとえば性能を確率変数とし，{High, Mid, Low} の 3 段階の状態を設ける．要素の候補は既存の非機能要求カタログ[7),8)] から発見することができ，性能以外にも変更容易性，使いやすさ，スケーラビリティ，信頼性などがある．達成度合いにより多くの段階を設けたり，連続的な値を用いたりすることも可能だが，分かりやすさや CPT の保守性を考慮して 2〜5 段階とする．多くの品質特性が，このような数段階の離散値に分類可能なことが分かっている[9),10)]．

I 群では，対象ドメインで利用予定のある実装技術を，競合製品や機能上の類似性を用いてカテゴリへ分類し，そのカテゴリを確率変数，カテゴリに含まれる実装技術を状態とする．たとえば，コンポーネントのライフサイクルを管理するフレームワークのカテゴリを抽出して確率変数とし，{Spring, EJB, Rails, Hand Coding} などを状態とする．そのほか，ユーザインタフェースやデータ永続化のカテゴリも考えられる．

C 群では，I 群に抽出した実装技術を比較して専門家がトレードオフを分析し，選択根拠となる設計上の決定事項を抽出する．ここで抽出した設計上の決定事項も確率変数とし，その選択肢を状態とする．たとえばフレームワークの選択根拠としてプログラミング言語を確率変数とし，{Java, Ruby} を状態とする．トレードオフとして Java は性能面で有利であり，Ruby は変更容易性や開発生産性で有利である．そのほか，クライアント/サーバの形態，データの保存方法などが決定事項の候補となりうる．

図 2 はこの工程により構築された BN の例である（後工程で抽出する確率変数間の関係も含んでいる）．この BN は以降の提案手法の説明で例題として用いるため，著者らが MVC アーキテクチャスタイル[4)] に基づく対話型のアプリケーション開発ドメインにおける実装技術と品質特性の関係をモデル化したものである．図上部から，5 要素の Q 群，3 要素の C 群，3 要素の I 群により構成され，I 群の確率変数はモデル，ビュー，コントローラに用いる実装技術の選択を表す．この例では，実装技術の選定の裏付けとしてプログラミング言語やクライアントの型（Fat，Thin，Rich）が存在するとアーキテクトが分析している．

3.1.2 関係の抽出

次に，前工程で得られた確率変数の間の因果関係を有向リンクで結び，子ノードに CPT を定義する．このとき，群をまたがる有向リンクは C 群のノードを始点とする．これは，C 群に配置された設計上の決定事項は，I 群に配置された実装技術の選定理由になると同時

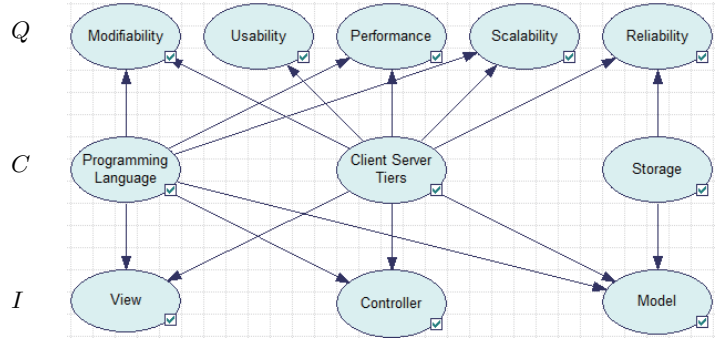

図 2　業務アプリケーションの実装技術選択のための BN
Fig. 2　A BN for choosing implementation technologies of a business application.

に，Q 群に配置された品質特性を満足する理由にもなるためである．また，仮に Q に属するノードと I に属するノードを直接リンクした場合，品質特性の達成度と実装技術の因果関係を直接 CPT に表す必要が生じてしまう．前述した，共通した設計に基づくグルーピングの利点を担保するため，そのようなリンクを避けられるようグラフを設計する．なお，品質特性間の依存関係，設計上の決定事項間の依存の連鎖，実装技術間の組合せの制約を表現するため，各群の内側の要素間の依存関係は存在してよい．また，親ノードを持たない C 群上の要素には，均等な事前分布を与えておく．

図 2 はこの工程により抽出された確率変数間の関係も含んでいる．これらの関係は，アーキテクトが持つ様々な知見から特定された．品質特性の予測については，たとえば，性能の達成度合いはクライアント/サーバアーキテクチャの形態とプログラミング言語に依存していることをアーキテクトは経験的に把握していた．そこで，それらの組合せごとに性能の達成レベルを考え，**表 1** のように CPT を設定した．たとえば，*Performance* = High となる確率に注目する．この CPT では，*Programming Language* = Java を観測した際は *Client Server Tiers* ごとに 0.90，0.60，0.80 と高い一方，*Programming Language* = Ruby の場合は 0.30，0.10，0.10 と低く設定されている．また，いずれのプログラミング言語の場合も，Fat クライアントが選択された場合の値は 0.90，0.30 と，Thin クライアントが選択された場合の値 0.60，0.10 に比べて高い．すなわち，Ruby よりも Java，Thin クライアントよりも Fat クライアントの性能が高くなるという知見を反映させている．同様に，実装技術のカテゴリの 1 つであるコントローラの選択基準がクライアント/サーバアーキテクチャ

表 1　性能に関する CPT
Table 1　CPT for *Performance*.

Programming Language		Java			Ruby		
Client Server Tiers		Fat	Thin	Rich	Fat	Thin	Rich
Performance	High	0.90	0.60	0.80	0.30	0.10	0.10
	Mid	0.08	0.35	0.15	0.65	0.80	0.70
	Low	0.02	0.05	0.05	0.05	0.10	0.20

表 2　コントローラに関する CPT
Table 2　CPT for *Controller*.

Programming Language		Java			Ruby		
Client Server Tiers		Fat	Thin	Rich	Fat	Thin	Rich
Controller	Spring	0.50	0.60	0.40	0.00	0.00	0.00
	EJB	0.00	0.30	0.50	0.00	0.00	0.00
	Rails	0.00	0.00	0.00	0.00	1.00	1.00
	Hand Coding	0.50	0.10	0.10	1.00	0.00	0.00

の形態とプログラミング言語に依存しているという見解から，Java の場合には Spring フレームワークや EJB コンテナを選択できるが，Ruby の場合は Ruby on Rails 以外の利用実績がないことを考慮し，CPT を**表 2** のように調整した．この表の定義から，プログラミング言語が Ruby の場合にはクライアント/サーバアーキテクチャの形態に応じて，コントローラの実装方法が以下のように一意に定まる．

$$P(\text{Rails} \mid \textit{Programming Language} = \text{Ruby}, \textit{Client Server Tiers} \neq \text{Fat}) = 1$$
$$P(\text{Rails} \mid \textit{Programming Language} = \text{Ruby}, \textit{Client Server Tiers} \neq \text{Fat}) = 0$$
$$P(\text{Hand Coding} \mid \textit{Programming Language} = \text{Ruby}, \textit{Client Server Tiers} = \text{Fat}) = 1$$
$$P(\text{Hand Coding} \mid \textit{Programming Language} = \text{Ruby}, \textit{Client Server Tiers} = \text{Fat}) = 0$$

このように，親となるノード（I 群または C 群）の組合せ条件によって選択できない実装技術がある場合には，その実装技術の条件付き確率を 0 とし，他の代替案の条件付き確率の合計が 1 になるように調整する．ここでは，Fat クライアントの場合はいずれの言語もフレームワークを適用できず独自実装が必要になる可能性があるという知見も CPT に反映されている．

3.2　実装技術の選択支援

要求分析中に品質特性の達成レベル，設計上の決定事項，実装技術の選択肢などの要求を抽出した場合，確率変数の値を観測したと考える．どの抽象度の要求を観測した場合でも，

それらを証拠（evidence）として BN に与えて推論アルゴリズムを適用することにより，他の確率変数の事後確率を計算する．アプリケーション開発者は要求分析中に証拠を得たとき，それを対象開発ドメインの BN に与えることにより，以下のように実装技術の選択を行う．

(1)　MAP 推定を用い，同時分布が最大となる実装技術の値の組合せを得る．

(2)　Junction Tree アルゴリズムなどを用い，全確率変数の周辺分布を得ることにより，(1) で得た組合せの正当性を分析する．また，その一部を新たな証拠として BN に与え，提案手法を繰り返し適用することによりより精度の高い組合せの選択を行う．

3.1 節で構築した BN に対し，「性能を高くしたい」「変更容易性は中程度でよい」という 2 要件を獲得した場合を考える．これは，証拠として *Performance* = High，*Modifiability* = Mid を観測したと見なすことができる．これを入力とし，確率変数 *View*，*Controller*，*Model* の値の適切な組合せを MAP 推定する．BN 処理ツール GeNIe[11] に実装された焼きなまし MAP 法を用いてこれを求めたところ，0.129 の確率で (*View* = Swing, *Controller* = Spring, *Model* = Hand Coding) の組合せが検出された．これは，*Q* 群に変更容易性よりも性能を優先させる品質要求を与えたために，*C* 群では Ruby よりも性能が高い Java と，アプリケーションロジックまで含めてクライアント側で動作する Fat クライアントの形態が採用された結果である．これを受けて，*I* 群ではビューに Java の GUI ツールキットである Swing を用い，モデルでは性能を稼ぐためにハンドコーディングを行うという実現方式が選択された．また，Java の Fat クライアントにおけるコントローラの実装技術は表 2 から Spring フレームワークとハンドコーディングを選択可能だが，一定の変更容易性を確保するために，Spring フレームワークが選択された．確率 0.129 は一見高くないが，この値はモデル，ビュー，コントローラの実装技術の選択肢の直積集合における確率であることに注意されたい．また，同様の証拠をもとに厳密推論により全ノードの周辺分布を GeNIe を用いて計算した．その結果を**図 3** に示す．図からは，各実装技術に関する事後確率では，$P(\mathit{View} = \text{Swing}) = 0.410$，$P(\mathit{Controller} = \text{Spring}) = 0.439$，$P(\mathit{Model} = \text{Hand Coding}) = 0.601$ がそれぞれ最尤値となっていることが分かる．この結果は焼きなまし MAP 法で得た結果と一致しており，この結果は前結果が誤った局所解に陥っていないことを示唆している．また，図 3 は第 2 位以降の確率も示している．たとえば，*Model* の選択では他の実装技術の選択に比べ，Hand Coding が他の実装と比べて強く優位であることがグラフから見てとれる．

同様の確率推論が，品質特性ではなく実装技術や設計根拠の一部を決定した場合にも行えるため，アプリケーション開発者は要求分析により得た既知の情報から柔軟に未決定の項目の分析を行うことができる．このようにして，品質特性に関する要求から実装技術を選択するトップダウンの分析や，実装技術の選択が影響する品質特性を確認するといったボトムアップの分析の両方を支援できる．実際の要求分析では双方のアプローチを混在させ，提示された事後確率から判明した新たな要求を証拠として入力したり，入力済みの証拠を取り消したりすることを繰り返してよい．

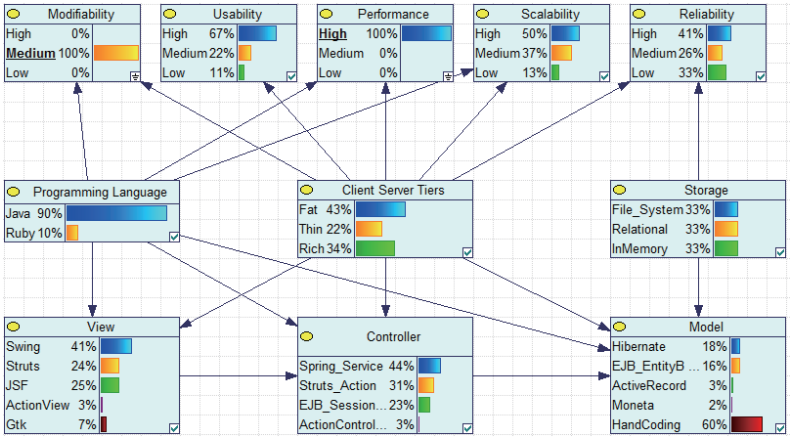

図 3　例題における周辺事後確率
Fig. 3　After observation of two NFRs.

4. 事例による評価

Layers アーキテクチャスタイル[4] に従ったアプリケーションの開発ドメインを対象とし，実際に実装技術の選択を行う事例により，提案手法の評価を行った．評価のポイントは，(1) MVC だけでなく，他の種類の Layers アーキテクチャに基づいて BN の構築が可能なこと，(2) 構築された BN により得た実装技術の選択が，アーキテクトの判断と合致すること，(3) BN の局所的な変更によって新たなノードがグラフに追加できること，の確認である．

4.1 Java EE 5 層レイヤのモデル化

まず，アーキテクトが Java EE プラットフォームを採用した 20 件の開発プロジェクトを分析して BN を構築した．対象としたプロジェクトはすべて，企業などの組織における事務処理を行う対話型の情報システムの開発を行っており，Java EE のレイヤアーキテクチャを用いて 1 年から 3 年程度の期間で実施したものである．

文献 12) では，Java EE における Layers スタイルの具体例として Presentation，Appli-

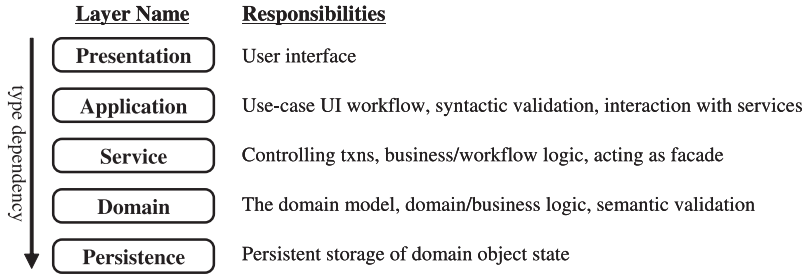

図 4　Java EE の 5 層レイヤアーキテクチャ[12)]
Fig. 4　Five-layer architecture for Java EE [12)].

cation，Services，Domain，Persistence の 5 層からなるレイヤ構造を定義している．**図 4** はそれぞれのレイヤの名称と責務を示しており，アーキテクトはこのレイヤ構成をもとに，以下のように**図 5** に示す BN をモデル化した．

- I 群では 5 つのレイヤの各層を確率変数とし，各層の実装技術の選択肢を確率変数の代替案として抽出した．なお，Layers アーキテクチャでは隣接するレイヤ間でインタフェースを整合させる必要があるため，隣接するレイヤの実装技術の選択結果が自レイヤの実装技術の選択に影響する．本章の例題では，「サービス層に Spring を選択した場合はドメイン層には軽量オブジェクトである POJO を採用することが多い」というように，アーキテクトがトップダウンのアプローチで判断を行っており，I 群の中で上位のレイヤから下位のレイヤへ確率変数間のリンクを設定している．これとは反対に，ボトムアップに判断する，すなわち，下位のレイヤから上位のレイヤにリンクを設定することも可能である．
- Q 群では Java EE システムに求められる品質特性として，クライアントの運用性，クライアントの応答時間，他のシステムとの相互運用性，トランザクションのスループット，ドメインの変更容易性の 5 つをあげ，それぞれ {High, Mid, Low} の 3 段階の値をとる確率変数として抽出した．
- C 群では設計上の決定事項を確率変数として抽出し，それらの品質特性への影響を以下のように分析した．

  クライアントの実行環境　{Web ブラウザのみ，Web ブラウザのプラグイン，仮想マシン} のいずれかの値をとる．クライアントの運用性と応答性に影響する．

  クライアントとサーバ間の通信メッセージ　{WWW フォーム，XML，バイナリ} のいずれかの値をとる．クライアントの応答性と相互運用性に影響する．

  分散コンポーネントの必要性　{サービスの公開，リソースの公開，分散しない} のいずれかの値をとる．相互運用性，トランザクションのスループット，ドメインの変更容易性に影響する．

  SQL ステートメントの記述　{自動生成，手動で記述} のいずれかの値をとる．トランザクションのスループットとドメインの変更容易性に影響する．
- C 群の確率変数から I 群，Q 群のそれぞれの確率変数へのリンクを定義し，アーキテクトの経験に基づいて条件付き確率表を定義した．付録 A.1 に I 群，Q 群のすべてのノードの条件付き確率表を添付する．

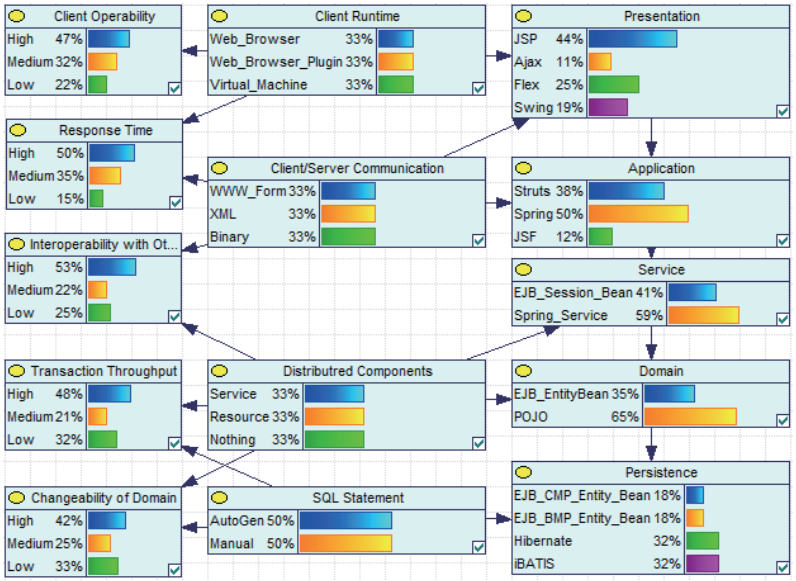

図 5　Java EE のレイヤ実装技術を選択するための BN
Fig. 5　A BN for choosing implementation technology for Java EE layers.

4.2　Java EE システムの実装技術の選択支援

次に，この BN を用いて 2 つのシステムの実装技術を選択した．それぞれのシステムにおける品質要求を以下に示す．

システム 1　インターネット上に公開される商用サイトのアプリケーションである．利用者

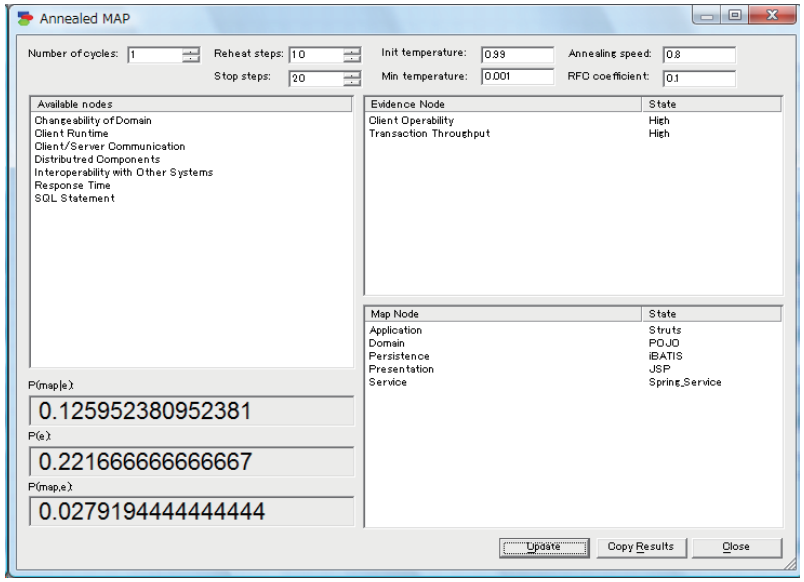

図 6　焼きなまし MAP 法を実行するためのツール画面
Fig. 6　A tool dialog for calculating annealed MAP.

が不特性多数であるため，利用者にとって運用性が高いシステムにしたい．また，トランザクションの処理性能の要求は毎分 10,000 件であり，BN を構築した際の分析対象のプロジェクトと比較して高水準であると判断した．

システム 2　企業内で利用されるデータ入力アプリケーションである．よく訓練された利用者が効率良くデータの入力作業を行うことが目的であり，応答速度が重要となる．このシステムはデータベースに入力データを蓄積することが目的であり，他のシステムとは連携する必要がない．

これらの品質要求をもとに，まずシステム 1 では *Client Operability* = High，*Transaction Throughput* = High を証拠として BN に与え，MAP 推定を行った．焼きなまし MAP 法を用いて MAP 値を求めるための GeNIe の画面を**図 6** に示す．ユーザは，画面左部の確率変数から証拠として観測値を与えるものを選択し，具体的な値を振る．値は画面右上部のリストに表示されている．その後，Update ボタンをクリックすると，画面右下のリストに同時分布が局所的最大となる値の組合せが表示される．この例では，(*Presentation* = JSP, *Application* = Struts, *Service* = Spring Service, *Domain* = POJO, *Persistence* = iBATIS) の組合せが選択され，同時確率値は 0.126 であった（**表 3**）．

表 3　2 つの Java EE システムに対する選択結果
Table 3　Evaluated results for two Java EE systems.

システム	システム 1	システム 2
BN に与えた証拠	*Client Operability* = High *Transaction Throughput* = High	*Response Time* = High *Interoperability* = Low
Presentation	JSP	Flex
Application	Struts	Spring
Service	Spring Service	Spring Service
Domain	POJO	POJO
Persistence	iBATIS	Hibernate
最大事後確率（MAP）	0.126	0.100

実際には，アーキテクトは Web ブラウザをクライアントとして利用とすることで，ソフトウェアの配布や更新におけるユーザの運用負荷を低減するという判断を行った．Web ブラウザを用いる場合，プレゼンテーション層の実装技術には JSP，Ajax のいずれかを選択可能であるが，この選択に影響する品質要求は与えられていないため，経験が豊富な JSP と Struts をプレゼンテーション層，アプリケーション層の実装技術としてそれぞれ提示した．また，トランザクション処理性能を重視するためにオブジェクトやトランザクションを分散しないことを決定し，サービス層とドメイン層にそれぞれ Spring と POJO を採用するとともに，永続化層では SQL の性能チューニングを行いやすい iBATIS を実装技術として提示した．

一方，システム 2 では *Response Time* = High，*Interoperability* = Low を証拠として与えた結果，(*Presentation* = Flex, *Application* = Spring, *Service* = Spring Service, *Domain* = POJO, *Persistence* = Hibernate) の組合せが選択され，同時確率値は 0.100 であった（表 3）．アーキテクトは Web ブラウザ単体より多少運用性が劣るものの，Web ブラウザに Flash プラグインをインストールして高度な応答性を実現する必要があると判断し，プレゼンテーション層の実装技術として Flex を提示した．また，外部システムとの接続を考えないため分散オブジェクト技術である EJB は利用せず，アプリケーション層とサービス層の実装技術には Spring フレームワーク，ドメイン層の実装技術に POJO をそれぞれ提示した．永続化層の実装技術には特に対応する要件が提示されていないが，上位層であるドメイン層で POJO を選択したため，POJO と組み合わせた利用経験が最も豊富な Hibernate を提示した．以上のように，MAP として推定された実装技術の組合せは経験者の判断とよ

く一致していた．

なお，図 6 の左下に表示された 3 つの確率値のうち，中段の値 $P(e) = 0.22$ は証拠として与えたノード自体の確率であり，BN のモデルにおいて，この品質要求の発生確率は 22%であることを表している．たとえば，すべての品質要求を High に設定した場合，$P(e) = 0.017$ という非常に低い値が得られるため，この品質要求が妥当でないという判断に用いることができる．その際には，いくつかの品質特性に対する観測値を破棄し，より制約の弱い条件での実装技術選択を試みるなどの判断が行える．

4.3　新たな品質特性の導入

ここでは，品質特性を表す Q 群のノードに，これまで考慮していなかった品質特性として，使いやすさを考慮する必要が発生した状況を考えよう．I 群には変更を行わず，これまで用いてきた Java EE の 5 層レイヤアーキテクチャでこの品質要求を扱いたい．

そこで再び，同ドメインに明るいアーキテクトが BN を変更する．変更後の BN の構造を**図 7** に示す．まず Q 群に使いやすさ（*Usability*）のノードを導入した．この品質特性を達成する手段は多様に考えられるが，アーキテクトは C 群にユーザ入力の補助（*User Assistance*）を導入し，{ダイアログによる入力補助，ユーザ入力のサジェスト，補助しない} をその選択肢とした．また，アーキテクトは経験上，ユーザ入力の補助は使いやすさ以外にも Q 群の応答性（*Response Time*）や I 群のプレゼンテーション層（*Presentation*）に影響することを知っていたため，ユーザ入力の補助がそれらの親となるようにリンクを接続し，条件付き確率表を再定義した．更新された箇所の CPT を付録 A.2 に添付する．

このように変更した BN に対し，変更前と同様に焼きなまし MAP 法を適用し，以下の 2 点を確認した結果を**表 4** に示す．

(1)　変更後の BN に対して，変更前の事例におけるシステム 1 と同じ品質要求を与えたとき，提示される実装技術の選択結果が変わらないこと．

(2)　上記に加えて使いやすさが高い（*Usability* = High）という品質要求を与えたとき，アーキテクトの判断に照らして妥当な実装技術の組合せが選択できること．

(1) については，BN の変更前と同じ (*Presentation* = JSP, *Application* = Struts, *Service* = Spring Service, *Domain* = POJO, *Persistence* = iBATIS) の組合せが選択され，その同時確率値は 0.078 であった．また，(2) ではプレゼンテーション層とアプリケーション層の実装技術がそれぞれ Ajax と Spring に変更されて (*Presentation* = Ajax, *Application* = Spring, *Service* = Spring Service, *Domain* = POJO, *Persistence* = iBATIS) の組合せが選択され，その同時確率値は 0.128 であった．アーキテクトの判断では，ユーザ入力を予測

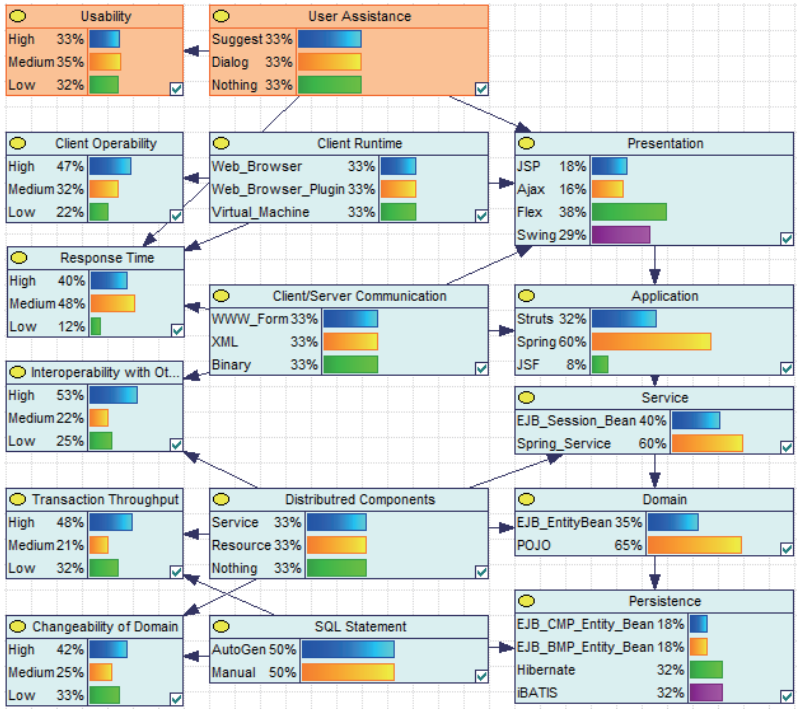

図 7　新たなノードを 2 つ追加した後の BN
Fig. 7　A modified BN with two new nodes.

表 4　変更後の BN を用いた実装技術の選択結果
Table 4　Evaluated results after modification.

品質要求	要求 1（システム 1 と同等）	要求 2
BN に与えた証拠	*Client Operability* = High *Transaction Throughput* = High	*Client Operability* = High *Transaction Throughput* = High *Usability* = High
Presentation	JSP	Ajax
Application	Struts	Spring
Service	Spring Service	Spring Service
Domain	POJO	POJO
Persistence	iBATIS	iBATIS
最大事後確率（MAP）	0.078	0.128

して候補を提示するサジェスト機能が使いやすさに最も貢献し，またクライアントの運用性を高めるためには Web ブラウザを実行環境とすることが有効であると考えているため，この 2 つを両立できる Ajax をプレゼンテーション層の実装技術に採用することは妥当である．また，Ajax と組み合わせやすいアプリケーション層とサービス層の実装技術として Spring フレームワークを使用し，さらに Spring と組み合わせやすい POJO をドメイン層に用いるというアーキテクトのトップダウンの思考とも，この結果はよく一致している．永続化層の実装技術についてはトランザクションのスループットを重視する品質要求があるため，トップダウンの選択ではなく，SQL をハンドコーディングでチューニング可能な iBATIS を使うという判断をしている．

このように，グラフの一部を変更した場合においても提案手法は有効に機能する．BN の条件付き確率表はノードの直接の親となるノード集合を条件付けに用いるため，グラフの変更の影響が局所化され，新たに導入したノードとリンクに直接関係する箇所の条件付き確率表のみを更新するだけでよい．

4.4 議　　論

本章の適用事例では，BN に含まれる 14 個のノードのうち品質特性に関する 2 個の値を観測することにより，他のノードの事後確率の計算結果が実装技術の選定に役立つことを確認できた．提案手法が必ずしも正しい実装技術の組合せを与えるとは限らないものの，BN の構造やパラメータが適切に構成された場合，過去の経験や実績に基づき妥当性が最も高いと想定される候補を参考情報として与える点が有用である．

しかしながら，提案手法のフェーズ 1 によって，最初から妥当なモデルが構築できるとは限らない．完成したソフトウェアの品質が確率推論の結果と著しく異なったり，提示された候補が熟練者の判断と一致しなかったりする状況が生じる．文献 13) などの機械学習のアルゴリズムを用いて，前者は失敗事例として同様の選択を避け，後者は正解事例として同様の選択を促進するように CPT のパラメータを更新し，BN を繰り返し利用する過程でモデルの妥当性を向上させることができると考えている．提案手法では品質特性，設計上の決定事項，使用した実装技術の値がアンケート票などで蓄積しやすい形になっており，パラメータの学習による効果を実際に確認することが今後の課題である．また，ノードやリンクの過不足といった BN の定性的なグラフ構造の誤りを機械学習で修正するには多量のデータが必要であり，提案手法への適用は現実的ではない．一般的なソフトウェアの開発と同様に，複数の開発者が作成やレビューに参加することでグラフ構造の妥当性を高める必要がある．

厳密推論に必要な計算時間は汎用的な PC（Pentium M 1.60 GHz，メモリ 1.5 GB）でも 0.1 秒以内であり，実装技術の組合せを得る操作も数回のクリックだけで済むため，要求分析の過程でストレスのない効果的な支援が可能であった．

5. 関 連 研 究

アーキテクチャスタイルを用いて品質特性とアーキテクチャの関係を定量化する試みには以下のような先行研究がある．Klein らはアーキテクチャスタイルにおける問題，解決策の枠組みに品質特性の見積り方法を組み合わせた Attibute-Based Architecture Styles（ABAS）を考案した[14]．ABAS ではアーキテクチャスタイルを品質特性の種類ごとに分類しており，アーキテクチャスタイル上の要素に与えた属性値と計算式から品質特性を見積もる．Petriu らは UML のコラボレーション図，シーケンス図とアーキテクチャスタイルを用いてアーキテクチャの構造や相互作用を表現し，それを Layered Queueing Network（LQN）に変換することで性能を見積もる手法を提案した[15),16)]．これらは重要度の高い品質特性について精密なモデルを構築して見積りを行う手法であり，決定したアーキテクチャに対して妥当性を検証する用途に適している．我々の提案手法は要求分析の過程で複数の品質特性間のトレードオフを分析し，意思決定の局面へ素早くフィードバックするための軽量的な支援を目的としており，これらの手法とは相補的に活用できる．

また，ソフトウェア開発における不確定性をモデル化するために BN を適用する手法もすでにいくつか提案されている．Tang らはアーキテクチャの設計における設計要素と設計根拠の連鎖構造を BN の状態変数で表現し，親ノードの変更から受ける影響の度合いを確率で表すことによって，設計変更の影響範囲を予測する手法を提案した[17]．Pendharker らはソフトウェア開発の工数見積りにおいて，工数，開発方法論，CASE ツールおよびその利用経験などのコストドライバと開発工数の関係を BN でモデル化し，ニューラルネットワークや CART アルゴリズムより高精度の見積りが可能なことを示した[18]．これらの研究ではプロジェクト管理における管理項目の因果関係や，開発プロセスにおける成果物の追跡性などのソフトウェア開発の定性的な側面を BN のリンク構造で表現し，人的要因や外部環境などによりリンクが持つ不確定性を確率でモデル化する．我々の提案手法は設計，実装における再利用資産と品質特性の定性的な関係に対して，品質特性の達成度合いを不確定性としてとらえたものであり，ソフトウェア工学分野における BN の新たな適用例である．

6. お わ り に

本稿では品質特性と実装技術の因果関係を BN を用いてモデル化し，実装技術の組合せの

選択を支援する手法を提案した．提案手法では要求分析で部分的に獲得した品質特性の目標レベル，設計上の決定事項，実装技術の要求を確率変数の観測値として BN の推論アルゴリズムに与えることで，未観測の確率変数に関する事後確率などを計算し，品質特性の予測や実装技術の選択を支援する．提案する BN の構築手順により，実際に実装技術の選択に有用な BN を 2 アーキテクチャに対して記述可能なことを確かめた．さらに，提案手法を BN の検証ツール上に実装し，品質特性に基づいて業務アプリケーションの実装技術を Layers アーキテクチャ上で分析する事例を用い，正しい選択結果が得られることを確認した．

提案手法は情報システムの構築における開発の複雑さを低減させ，品質やコストを改善することを目指した取り組みの最初の一歩である．今後も継続的に提案手法の改善を行う予定であり，以下のような課題に取り組みたい．

さらなる事例の構築と評価　本稿では Layers アーキテクチャに基づく対話型の企業アプリケーションの実装技術の選択事例を示した．今後はこの事例の BN により多くのノードを導入し，実用的な支援が行えるかどうか再評価したい．追加するノードの種類として品質特性に対する副特性や，アーキテクチャスタイルと関連のあるデザインパターンなど，より細粒度の因果関係を BN に含めることも検討する．

パラメータの学習　本稿の適用事例では品質特性や対象領域のアーキテクチャ，実装技術などに詳しい経験者が BN の構造と CPT を定義したが，多数の親を持つノードでは CPT の定義に多くの労力が必要であった．BN は十分な学習データを与えることにより，その CPT を学習できることが知られている[13)]．実開発のデータを訓練集合として CPT のパラメータを機械学習させることでこの負担を軽減できるとともに，より精度の高い分析が可能になると考えている．さらに，アーキテクトが定めた BN の構造の修正にもデータに基づくアプローチが適用可能であると考える．

他のアーキテクチャスタイルへの拡張　本稿では MVC と Layers の 2 つのアーキテクチャスタイルに対する BN を構築したが，他のアーキテクチャスタイルにも提案手法を拡張し，実装技術の選択を BN で支援することが可能か検討したい．たとえば Broker アーキテクチャではブローカやクライアント/サーバそれぞれの実装技術と性能，信頼性，セキュリティなどの品質特性の関係を BN で表現できると考えている．一方，Pipes and Filters アーキテクチャではパイプやフィルタの組合せによりアーキテクチャの構成要素数が変化するため，実装技術と品質特性の関係を静的な BN で表現することが難しいと予想する．

参 考 文 献

1) Kazman, R., Klein, M. and Clements, P.: ATAM: Method for Architecture Evaluation, Technical Report CMU/SEI-2000-TR-004, CMU/SEI (2000).
2) 繁桝算男，本村陽一，植野真臣：ベイジアンネットワーク概説，培風館 (2006).
3) Bishop, C.M.：パターン認識と機械学習（下）：ベイズ理論による統計的予測，シュプリンガー・ジャパン (2008).
4) Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M.: *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, Inc. (1996).
5) Lauritzen, S.L. and Spiegelhalter, D.J.: *Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems*, pp.415–448, Morgan Kaufmann Publishers Inc. (1990).
6) Yuan, C., Lu, T.C. and Druzdzel, M.J.: Annealed MAP, *Proc. 20th Conference on Uncertainty in Artificial Intelligence* (*UAI '04*), pp.628–635, AUAI Press (2004).
7) ISO/IEC 9126: Information Technology – Software product evaluation – Quality characteristics and guidelines for their use (1991).
8) Chung, L., Nixon, B.A., Yu, E. and Mylopoulos, J.: *Non-Functional Requirements in Software Engineering*, Springer (1999).
9) NTT データ：非機能要求グレード検討会．http://www.nttdata.co.jp/nfr-grade/
10) JIS X. 0133-1 (1999).
11) GeNIe & SMILE. http://genie.sis.pitt.edu/
12) Marinescu, F.: *Ejb Design Patterns: Advanced Patterns, Processes, and Idioms with Poster*, John Wiley & Sons, Inc. (2002).
13) Heckerman, D., Geiger, D. and Chickering, D.M.: Learning Bayesian networks: The combination of knowledge and statistical data, *Machine Learning*, Vol.20, No.3, pp.197–243 (1995).
14) Klein, M. and Kazman, R.: Attribute-Based Architectural Styles, Technical Report CMU/SEI-99-TR-022, CMU/SEI (1999).
15) Petriu, D.C. and Wang, X.: From UML Descriptions of High-Level Software Architectures to LQN Performance Models, *Proc. International Workshop on Applications of Graph Transformations with Industrial Relevance* (*AGTIVE'99*), pp.47–62, Springer-Verlag (2000).
16) Petriu, D.C. and Shen, H.: Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications, *Proc. 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools* (*TOOLS '02*), pp.159–177, Springer-Verlag (2002).
17) Tang, A., Nicholson, A., Jin, Y. and Han, J.: Using Bayesian belief networks for

change impact analysis in architecture design, *Journal of Systems and Software*, Vol.80, No.1, pp.127–148 (2007).
18) Pendharkar, P.C., Subramanian, G.H. and Rodger, J.A.: A Probabilistic Model for Predicting Software Development Effort, *IEEE Trans. Softw. Eng.*, Vol.31, No.7, pp.615–624 (2005).

付　　録

以下の表では適宜，WWW Form を W，XML を X，Binary を B，Service を Sv，Resource を R，Nothing を N，Suggest を Sg，Dialog を D と省略している．

A.1　4 章における BN の CPT

P(*Client Operability* | *Client Runtime*)

Client Runtime		Web Browser	Web Browser Plugin	Virtual Machine
Client Operability	High	0.9	0.3	0.2
	Mid	0.05	0.6	0.3
	Low	0.05	0.1	0.5

P(*Response Time* | *Client Runtime*, *Client Server Communication*)

Client Runtime		Web Browser			Web Browser Plugin			Virtual Machine		
Client Server Communication		W	X	B	W	X	B	W	X	B
Response Time	High	0.15	0.3	0.9	1/3	0.35	0.9	1/3	0.35	0.9
	Mid	0.7	0.4	0.05	1/3	0.6	0.05	1/3	0.6	0.05
	Low	0.15	0.3	0.05	1/3	0.05	0.05	1/3	0.05	0.05

P(*Interoperability with Other Systems* | *Distributed Components*, *Client Server Communication*)

Distributed Components		Service			Resource			Nothing		
Client Server Communication		W	X	B	W	X	B	W	X	B
Interoperability with Other Systems	High	0.6	0.6	0.6	0.9	0.9	0.9	0.05	0.2	0.05
	Mid	0.3	0.3	0.3	0.05	0.05	0.05	0.25	0.6	0.05
	Low	0.1	0.1	0.1	0.05	0.05	0.05	0.7	0.2	0.9

P(*Transaction Throughput* | *Distributed Components*, *SQL Statement*)

Distributed Components		Service		Resource		Nothing	
SQL Statement		AutoGen	Manual	AutoGen	Manual	AutoGen	Manual
Transaction Throughput	High	0.5	0.75	0.05	0.05	0.6	0.9
	Mid	0.4	0.2	0.2	0.05	0.35	0.05
	Low	0.1	0.05	0.75	0.9	0.05	0.05

P(*Changeability of Domain* | *Distributed Components*, *SQL Statement*)

Distributed Components		Service		Resource		Nothing	
SQL Statement		AutoGen	Manual	AutoGen	Manual	AutoGen	Manual
Changeability of Domain	High	0.8	0.3	0.05	0.05	1	0.3
	Mid	0.15	0.45	0.2	0.05	0	0.65
	Low	0.05	0.25	0.75	0.9	0	0.05

P(*Presentation* | *Client Runtime*, *Client Server Communication*)

Client Runtime		Web Browser			Web Browser Plugin			Virtual Machine		
Client Server Communication		W	X	B	W	X	B	W	X	B
Presentation	JSP	1	0	1	1	0	0	1	0	0
	Ajax	0	1	0	0	0	0	0	0	0
	Flex	0	0	0	0	0.7	0.95	0	0.3	0.3
	Swing	0	0	0	0	0.3	0.95	0	0.7	0.7

P(*Application* | *Presentation*, *Client Server Communication*)

Presentation		JSP			Ajax			Flex			Swing		
Client Server Communication		W	X	B	W	X	B	W	X	B	W	X	B
Application	Struts	0.5	0.5	0.5	0.2	0.2	0.2	0.3	0.3	0.3	0.3	0.3	0.3
	Spring	0.3	0.3	0.3	0.5	0.5	0.5	0.7	0.7	0.7	0.7	0.7	0.7
	JSF	0.2	0.2	0.2	0.3	0.3	0.3	0	0	0	0	0	0

P(*Service* | *Application*, *Distributed Components*)

Application		Struts			Spring			JSF		
Distributed Components		Sv	R	N	Sv	R	N	Sv	R	N
Service	EJB Session Bean	0.3	1	0.05	0.05	1	0.05	0.3	1	0.05
	Spring Service	0.7	0	0.95	0.95	0	0.95	0.7	0	0.95

P(*Domain* | *Service*, *Distributed Components*)

Service		EJB Session Bean			Spring Service		
Distributed Components		Service	Resource	Nothing	Service	Resource	Nothing
Domain	EJB EntityBean	0.3	0.9	0.3	0.05	0.9	0.05
	POJO	0.7	0.1	0.7	0.95	0.1	0.95

P(*Persistence* | *Domain*, *SQL Statement*)

Domain		EJB EntityBean		POJO	
SQL Statement		AutoGen	Manual	AutoGen	Manual
Presistence	EJB CMP Entity Bean	1	0	0	0
	EJB BMP Entity Bean	0	1	0	0
	Hibernate	0	0	1	0
	iBATIS	0	0	0	1

A.2　4 章の BN の変更により更新された CPT

P(*Usability* | *Input Assistance*)

User Assistance		Suggest	Dialog	Nothing
Usability	High	0.7	0.3	0.0
	Mid	0.25	0.6	0.2
	Low	0.05	0.1	0.8

P(*Response Time* | *Client Runtime*, *Client Server Communication*, *User Assistance*)

Client Runtime				Web Browser			Web Browser Plugin			Virtual Machine		
Client Server Communication				W	X	B	W	X	B	W	X	B
User Assistance	Sg	*Response Time*	High	0.297	0.263	0.365	0.324	0.291	0.451	0.368	0.364	0.552
			Mid	0.328	0.474	0.485	0.398	0.596	0.517	0.403	0.571	0.435
			Low	0.375	0.263	0.15	0.278	0.113	0.032	0.229	0.065	0.013
	D	*Response Time*	High	0.283	0.231	0.362	0.311	0.252	0.5	0.366	0.358	0.672
			Mid	0.389	0.597	0.575	0.483	0.725	0.499	0.481	0.637	0.328
			Low	0.328	0.172	0.063	0.206	0.023	0.001	0.153	0.005	0
	N	*Response Time*	High	0.323	0.289	0.45	0.366	0.358	0.672	0.429	0.5	0.841
			Mid	0.4	0.6	0.52	0.481	0.637	0.328	0.465	0.499	0.159
			Low	0.277	0.111	0.03	0.153	0.005	0	0.106	0.001	0

P(*Presentation* | *Client Runtime*, *Client Server Communication*, *User Assistance*)

Client Runtime				Web Browser			Web Browser Plugin			Virtual Machine		
Client Server Communication				W	X	B	W	X	B	W	X	B
User Assistance	Sg	*Presentation*	JSP	0	0	0	0	0	0	0	0	0
			Ajax	1	1	1	0	0	0	0	0	0
			Flex	0	0	0	0.85	0.75	0.9	0.5	0.333	0.6
			Swing	0	0	0	0.15	0.25	0.1	0.5	0.667	0.4
	D	*Presentation*	JSP	0.9	0.7	0.8	0	0	0	0	0	0
			Ajax	0.1	0.3	0.2	0	0	0	0	0	0
			Flex	0	0	0	0.75	0.6	0.8	0.333	0.2	0.425
			Swing	0	0	0	0.25	0.4	0.2	0.667	0.8	0.575
	N	*Presentation*	JSP	0.9	0.7	0.8	0	0	0	0	0	0
			Ajax	0.1	0.3	0.2	0	0	0	0	0	0
			Flex	0	0	0	0.75	0.6	0.8	0.333	0.2	0.425
			Swing	0	0	0	0.25	0.4	0.2	0.667	0.8	0.575

(平成 22 年 1 月 6 日受付)
(平成 22 年 6 月 3 日採録)

風戸　広史（正会員）

2001 年東京工業大学工学部情報工学科卒業．2003 年同大学大学院情報理工学研究科計算工学専攻修士課程修了．2010 年同専攻博士後期課程修了．2003 年より株式会社 NTT データ技術開発本部勤務．博士（工学）．ソフトウェア設計方法論やソフトウェア開発環境等の研究に従事．

林　　晋平（正会員）

2004 年北海道大学工学部情報工学科卒業．2006 年東京工業大学大学院情報理工学研究科計算工学専攻修士課程修了．2008 年同専攻博士後期課程修了．2009 年より同専攻助教．博士（工学）．ソフトウェア変更やソフトウェア開発環境等の研究に従事．IEEE，ACM 各会員．

小林　隆志（正会員）

1997 年東京工業大学工学部情報工学科卒業．1999 年同大学大学院情報理工学研究科計算工学専攻修士課程修了．2004 年同専攻博士後期課程修了．2002 年同大学学術国際情報センター助手．2007 年名古屋大学大学院情報科学研究科特任助教授．2009 年より同研究科情報システム学専攻准教授．現在に至る．博士（工学）．ソフトウェア開発技法，ソフトウェア理解，複合メディアコンテンツ検索等の研究に従事．IEEE-CS，ACM，ソフトウェア科学会，電子情報通信学会，日本データベース学会各会員．

佐伯　元司（正会員）

1978 年東京工業大学工学部電気電子工学科卒業．1980 年同大学大学院工学研究科情報工学専攻修士課程修了．1983 年同専攻博士後期課程修了．同大学助手，助教授を経て，2000 年より同大学大学院情報理工学研究科計算工学専攻教授．工学博士．要求工学やソフトウェア開発技法等の研究に従事．IEEE-CS，ACM，ソフトウェア科学会，人工知能学会，電子情報通信学会各会員．