Vol.2009-SE-165 No.3
2009/7/2

ベイジアンネットワークを用いた実装プラットフォームの選択支援

風戸広史[†1,†2]　Rafael Weiß[†1]　林晋平[†1]
小林隆志[†3]　佐伯元司[†1]

実装プラットフォームの組み合わせがシステム全体の品質特性にどのような影響を及ぼすかを把握することは重要である．本稿では品質特性と実装プラットフォームの因果関係をベイジアンネットワークでモデル化し，実装プラットフォーム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，業務アプリケーションの例題へ適用することによりその有効性を示す．

A Technique for Choosing Implementation Platforms Using Bayesian Networks

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

It is important to understand how a combination of implementation platforms influences quality attributes on a system. In this paper, we propose a technique to choose implementation platforms by modeling casual dependencies between requirements and platforms 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
Tokyo Institute of Technology.
†2 株式会社 NTT データ 技術開発本部
NTT DATA CORPORATION.
†3 名古屋大学 大学院情報科学研究科
Nagoya University.

1. はじめに

システムの実行環境はハードウェア，OS，プログラミング言語，ライブラリ，フレームワーク，ミドルウェアなどの実装プラットフォームを組み合わせて構成される．実装プラットフォームの背後にはそれぞれ何らかのアーキテクチャ上の仮定があり，そのプラットフォームを採用することでシステムの品質特性に影響を与える．たとえば，ある企業の業務アプリケーションにおいて Struts フレームワークを採用する場合，Web ブラウザをクライアントとする 3 層のクライアント／サーバアーキテクチャという暗黙の仮定が具象化されるため，システムのスケーラビリティ，性能，変更容易性などの品質特性が影響を受ける．

このような品質特性と実装プラットフォームの関係が明確ではないことや，品質特性間にトレードオフの関係があることから，品質特性に基づいて実装プラットフォームの組み合わせを選択することは難しい．我々は実装プラットフォームを選択するために，以下の要件を満たす支援手法が必要であると考えている:

- **トレードオフや不確定性を含んだ状況のモデル化．**品質特性と実装プラットフォームの組み合わせの関係をモデル化し，その組み合わせの範囲で実現可能な品質特性のトレードオフ関係を分析する．本来であれば実装の詳細も品質特性に寄与するが，その部分は不確定要因であることを許容し，実装に先立って品質特性の予測を可能とする．
- **多様な観点・抽象度を持つ要求の捕捉と分析．**抽象的な要求からトップダウンのアプローチだけで要求分析が進むことはほとんどない．ステークホルダは独自の利害関係に基づいた抽象度で要求を表現してくるため，たとえば「Web ブラウザで操作したい」という要求が品質特性にどんな影響を与えるかといった抽象化，要求を実現するにはどんな実装プラットフォームが必要かといった具体化の双方を支援する必要がある．
- **開発者の経験や実績の再利用．**実装プラットフォームの選択基準はシステムに期待される品質特性だけで決まるものではない．実際に運用可能な支援を行うためには，開発者の経験や開発組織における実績などのパラメータを反映させるべきである．

そこで，本稿では品質特性と実装プラットフォーム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 (BN; Bayesian network)[1] でモデル化し，品質特性に基づいた実装プラットフォームの選択を支援する手法を提案する．提案手法では実装プラットフォームの背後にあるアーキテクチャ上の仮定をモデルに明示し，それらを仲介して品質特性の達成レベルと実装プラットフォーム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アーキテクチャ設計という不確定性を含む問題の分析を支援する．

Vol.2009-SE-165 No.3
2009/7/2

本稿の残りの部分は以下のように構成されている．まず，次節で本稿で用いる BN の概念について簡潔に導入する．3 節では BN を用いて品質特性と実装プラットフォームの因果関係をモデル化する手法を提示し，続く 4 節でそれを用いて実装プラットフォームの選択支援を行う事例を示す．5 節は関連研究をいくつか紹介し，提案手法との差異について議論する．最後に，6 節で結論と今後の展望を述べる．

2. ベイジアンネットワーク

2.1 グラフィカルモデル

BN は確率変数をノード，確率変数間の確率的関係を有向エッジとする有効非循環グラフ (DAG; directed acyclic graph) によって確率分布を表現するグラフィカルモデルである．BN では確率変数 X, Y の間の条件付き依存性を有向エッジを用いて $X \to Y$ と表し，始点となる X を Y の親ノード，終点となる Y を X の子ノードと呼ぶ．また，ノード X_i の親となるノードの集合を $Pa(X_i)$ で表す．親ノードを持つ確率変数 X_i では親の状態ベクトルに対する事後確率 $P(X_i|Pa(X_i))$ を条件付き確率表 (CPT; conditional probability table) で定義し，親ノードを持たない確率変数については事前分布を与える．

簡単な例として，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．これらはそれぞれ真偽値を状態とる離散的な確率変数であり，C: 天気が曇る，S:スプリンクラーが作動する，R: 雨が降る，W: 芝生が湿るという 4 つの事象を表している．確率変数間の依存関係が有向エッジで表現されており，たとえば S は C に依存し，スプリンクラーが作動する確率は天気が曇っているとき $P(S = T|C = T) = 0.1$，曇っていないとき

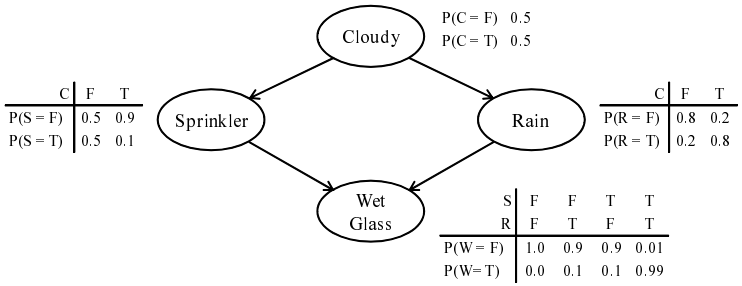
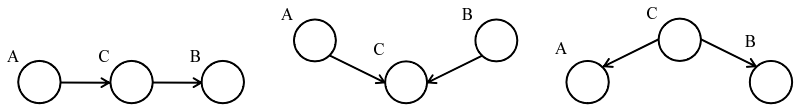


図 1 ベイジアンネットワークの例
Fig. 1 An example of a Bayesian network.



(a) head-to-tail (b) head-to-head (c) tail-to-tail

図 2 有向グラフのパス上のノードの分類
Fig. 2 A classification of nodes on paths in DAGs.

$P(S = T|C = F) = 0.5$ である．直感的には，この BN は天気が曇りかどうかによってスプリンクラーが作動したり，雨が降ったりする事象が発生し，それらに依存して芝生が湿るという事象が発生することを表現しているモデルである．

2.2 条件付き独立性

BN の構造において条件付き独立性という重要な概念がある．確率変数 A の状態が与えられたとき，確率変数 B, C が条件付き独立であるとは，A で条件づけられた B, C の同時分布がそれぞれの事後周辺分布の積として

$$P(B, C|A) = P(B|A)P(C|A) \quad (1)$$

の形に因数分解できることであり，これを $A \perp B \mid C$ と表す．BN ではこの確率変数間の条件付き独立性を，DAG の構造を用いた有向分離 (d-separation) と呼ばれる概念を用いて判別することができる．DAG のノードの部分集合 A, B, C を互いに素な集合とするとき，有向分離は以下のように定義される．

(1) DAG のパス上に出現するノードは，接続された有向エッジの方向の組み合わせに応じて**図 2** に示す head-to-tail, head-to-head, tail-to-tail のいずれかに分類される．

(2) A の任意のノードから B の任意のノードへ至るパスが以下のいずれかを満たすとき，そのパスは遮断されているという．
  (a) 集合 C に含まれるあるノードで，パスに含まれる有向エッジが head-to-tail または tail-to-tail となる．
  (b) パスに含まれる有向エッジがあるノードで head-to-head であり，そのノード自身を含めたすべての子孫が集合 C に含まれない．

(3) すべてのパスが (2) によって遮断されているとき A, B は C により有向分離されるといい，このとき条件付き独立性 $A \perp B \mid C$ が成立する．

Vol.2009-SE-165 No.3
2009/7/2

たとえば，図 1 では S から R に至る 2 つのパスがあり，(S, C, R) は C において tail-to-tail，(S, W, R) は W において head-to-head である．そこで，C を条件付けに用いると W はこの集合に含まれず子孫も持たないため，いずれのパスもこの集合で遮断され，$R \perp S \mid C$ となる．また，C から W に至る 2 つの パス $(C, S, W), (C, R, W)$ はそれぞれ S, R において head-to-tail であるため，S, R を条件付けに用いると $W \perp C \mid S, R$ が成立する．これらの条件付き独立性と確率の乗法定理を用いると，C, S, R, W の同時分布は以下のような周辺分布の積に分解できる．

$$P(C,S,R,W) = \prod_i P(X_i|Pa(X_i)) \tag{2}$$
$$= P(C) \cdot P(S|C) \cdot P(R|C,S) \cdot P(W|C,S,R) \tag{3}$$
$$= P(C) \cdot P(S|C) \cdot P(R|C) \cdot P(W|S,R) \tag{4}$$

2.3 確率推論アルゴリズム

BN のノードのうち，いくつかの状態が観測値 e によって固定された場合，残ったノードに関する事後確率 $P(X|e)$ を計算したり，同時分布が最大となる状態の組を見つけたりするための推論アルゴリズムが多数考案されている．

リンクの方向性を無視したとき，グラフ構造にループが存在しない BN では確率伝播法 (BP; belief propagation) と呼ばれるアルゴリズムで事後確率を厳密に計算できる．ここでは**図 3** に示すような $X_1 \rightarrow X_2$,$X_2 \rightarrow X_3$ の依存性を持つノード X_1, X_2, X_3 からなる単純な BN を用いて BP を説明する．事後確率を計算したいノード X_2 とし，それより上流のノードに関する証拠を e^+，下流のノードに関する証拠を e^- に分ける．$e^- \perp e^+ \mid X_2$ とベイズの定理を用いれば，事後確率 $P(X_2|e)$ は以下のように因数分解できる．

$$P(X_2|e) = P(X_2|e^+, e^-) \tag{5}$$
$$= \frac{P(e^-|X_2, e^+)P(X_2|e^+)}{P(e^-|e^+)} \tag{6}$$
$$= \alpha P(e^-|X_2)P(X_2|e^+) \tag{7}$$

ただし，$\alpha = \frac{1}{P(e^-|e^+)}$ は X_2 の値によらない定数値である．

ここで，式 (7) の第 2 項は親ノードから $P(X_2|e)$ へ伝播する確率であり，これを $P(X_2|e^+) = \pi(X_2)$ と表す．この $\pi(X_2)$ は X_2 の CPT と $P(X_1|e^+)$ を用いて X_1 について周辺化することにより，次式で求められる．

$$\pi(X_2) = \sum_{X_1} P(X_2|X_1)P(X_1|e^+) \tag{8}$$

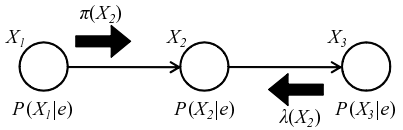


図 3 BP の簡単な例
Fig. 3 A simple example of BP.

式 (8) は X_1 が観測されているか，X_1 に親ノードが存在せず事前分布が与えられている場合には直ちに値が定まる．それ以外の場合には $\pi(X_1)$ を親ノードについて再帰的に計算する．

また，式 (7) の第 1 項は子ノードから $P(X_2|e)$ へ伝播する確率であり，これを $P(e^-|X_2) = \lambda(X_2)$ と表す．この $\lambda(X_2)$ は X_3 の CPT と $\lambda(X_3)$ を用いて次式のように表せる．

$$\lambda(X_2) = \sum_{X_3} P(e^-|X_2, X_3)P(X_3|X_2) \tag{9}$$
$$= \sum_{X_3} P(e^-|X_3)P(X_3|X_2) \tag{10}$$

ただし，(10) 式への展開では $e^- \perp X_2 \mid X_3$ であること用いた．式 (10) は X_3 が観測されている場合には直ちに値が定まる．それ以外の場合 X_3 に子ノードが存在しなければ X_3 のすべての状態について等しい値を持つ一様確率分布として扱い，子ノードがある場合には $\lambda(X_3)$ を再帰的に計算する．

式 (8),(10) を式 (7) に代入すると事後確率 $P(X_2|e)$ は以下のように表現できる．

$$P(X_2|e) = \alpha\pi(X_2)\lambda(X_2) \tag{11}$$

同様にして，任意のノード X_i の事後確率は次式で求められる．このアルゴリズムは BN の大きさに対して線形オーダ $O(N)$ となる．

$$P(X_i|e) = \alpha\pi(X_i)\lambda(X_i) \tag{12}$$

3. 提 案 手 法

本節では BN を用いて実装プラットフォームの選択を支援する手法を提示する．この手法は我々が開発しているモデル駆動型開発アプローチ ACCURATE (A Configurable Code generator Unifying Requirements Analysis TEchnique) の一部であり，単に実装プラットフォームの組み合わせを選択するだけでなく，選択結果に基づいたソースコードの自動生成までを行うことを目的としている．本稿では ACCURATE の詳細は割愛するが，まず 3.1

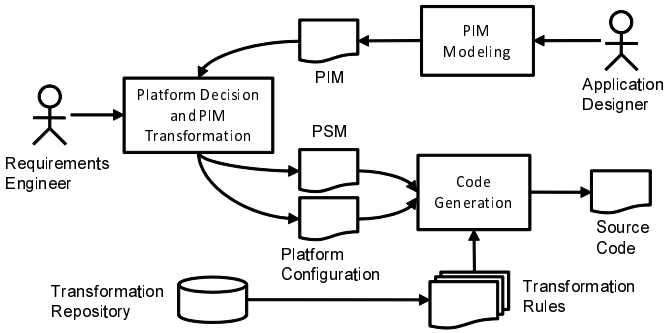


図 4 ACCURATE アプローチの概要
Fig. 4 An overview of the ACCURATE approach.

節でその全体像を説明し，3.2 節以降で実装プラットフォームの選択支援に関する手法について詳述する．

3.1 ACCURATE の概要

ACCURATE における開発プロセスは**図 4** に示すように，1) PIM モデリング，2) プラットフォーム決定と PIM 変換，3) コード生成の 3 つのアクティビティから構成される．本稿の提案手法は 2) のプラットフォーム決定を BN を用いて支援するものである．

この開発プロセスではまず，アプリケーション設計者が開発対象となるアプリケーションの構造や振る舞いを実装プラットフォームに非依存なモデル (PIM; platform-independent model) として記述する．次に，要求分析者やアーキテクトが品質特性に関する要求 (非機能要求) を元に適切な実装プラットフォームの組み合わせを選択する．この選択に従って PIM をプラットフォームに特化したモデル (PSM; platform-specific model) へ自動的に変換するとともに，コード生成ツールのためのプラットフォーム設定を出力する．これらの成果物は最後のアクティビティでコードを生成するために利用される．このように，アプリケーションの機能性と品質特性を個別に開発し，それらを統合することに特徴がある．

ACCURATE では既存のコード生成ツールを流用して PSM からソースコードへの変換を行うことを前提としている．特に，openArchitectureWare[2] や AndroMDA[3] のように変換規則と変換の実行エンジンを分離し，複数のプラットフォームに対応したコード生成ツールが必要である．これらのツールは様々な実装プラットフォーム向けの変換規則を変換リポジトリに蓄積しており，本稿の提案手法と連携することで品質特性に基づいたソースコードの生成支援を可能にする．

3.2 BN による対象ドメインのモデル化

ここでは，特定の開発対象ドメインにおける品質特性と実装プラットフォームの因果関係をベイジアンネットワークでモデル化する手法を提示する．BN を構築する技術者は対象ドメインに精通していることを想定しており，この作業を通じてその技術者の経験や知識が BN のリンク構造や定量的な確率にエンコードされる．

まず，対象ドメインで考慮する品質特性，実装プラットフォームの選択肢，アーキテクチャ上の決定事項をそれぞれ離散的な確率変数として抽出する．

(1) 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に応じた離散的な確率変数とする．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．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 3〜5 段階とする．

(2) 対象ドメインで利用予定のある実装プラットフォームを，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プラットフォームを状態とする．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, Ruby on Rails, Hand Coding} などを状態とする．

(3) (2) で抽出した実装プラットフォームを比較して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．ここで抽出したアーキテクチャ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．

次に，これらの確率変数の間の因果関係を有向エッジで結び，子ノードに CPT を定義する．このとき，有向エッジはアーキテクチャ上の決定事項を表すノードを始点とし，品質特性または実装プラットフォームを表すノードを終点とする．このように有向エッジを定義することで，品質特性のノード集合 Q と実装プラットフォームのノード集合 I の要素間を結ぶ任意のパスはアーキテクチャ上の決定事項を表すノード集合 A で tail-to-tail となり，$Q \perp I \mid A$ が成立する．この条件付き独立性を利用することで，Q に属するノードでは親となるアーキテクチャ上の決定事項によるその品質特性への寄与度，I に属するノードでは親となるアーキテクチャ上の決定事項に対してきその実装プラットフォームを選択する確率が CPT で表現される．

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プラットフォームの因果関係を直接 CPT に表す必要が生じる．我々は 1 節でそのような因果関係は明確でないと主張をしており，そのようなリンクを避けられるように提案手法を設計した．

3.3 実装プラットフォームの選択支援

要求分析中に品質特性の達成レベル，アーキテクチャ上の仮定，実装プラットフォーム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 に与えて厳密推論の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．

このようにして，品質特性に関する要求から実装プラットフォームを選択するトップダウンの分析や，実装プラットフォーム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し，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す必要がある．

4. 適 用 事 例

本節では，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，提案手法を適用して実装プラットフォームの選択支援を行う事例を示す．開発対象のアプリケーションの機能は単純に住所録を管理するものである．住所録には複数のエントリを登録することが可能であり，個々のエントリには氏名，生年月日，住所，電話番号，メールアドレスを記録する．

さて，読者はこのアプリケーションにどんな実行プラットフォームを想像しただろうか．ある従業員が数十人の同僚の連絡先を管理するために用いるファットクライアント型アプリケーションかもしれないし，数十万人の顧客を管理し全従業員から利用される Web アプリケーションかもしれない．これらにはそれぞれ異なる品質特性が期待されるため，その結果として実行プラットフォームの選択も異なってくる．

4.1 BN の構築

要求分析に先立ち，業務アプリケーションの構築について経験を持つアーキテクトと提案手法の専門家がディスカッションを行い，品質要求と実装プラットフォームの関係を BN でモデル化した．作成した BN のグラフ構造を**図 5** に示す．

要求分析で考慮する品質特性として変更容易性，使いやすさ，性能，スケーラビリティ，信頼性が挙げられた．実装プラットフォームとして Java や Ruby のライブラリやフレームワークを分類し，カテゴリとしてユーザインタフェース，コンポーネント管理，データ永続化の 3 つを定義した．これらの品質特性やカテゴリを表すノードが BN に追加されている．また，実装プラットフォームの選択基準としてプログラミング言語，クライアント／サーバの形態，データの保存方法を手掛かりとしていることが判明したため，それらをアーキテクチャ上の決定事項としてノードを追加した．

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，それらの組み合わ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．ここでは，Ruby よりも Java，Thin クライアントよりも Fat クライアントやリッチ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，実装プラットフォームのカテゴリの 1 つであるコンポーネント管理の選択基準がクライアント／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も我々は知っていた．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．ファット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．

4.2 支援ツールの試作と適用

我々は Bayesian Network Tools in Java(BNJ)[4] を用いて前節の BN を実装した．BNJ は Java で開発されたオープンソースソフトウェアであり，BN を構築するためのグラフィカルエディタと任意の構造の持つ BN について厳密推論を実行可能な Junction Tree アルゴリズムを備えている．アーキテクチャ上の決定事項を表すノードに均等な事前分布を与え，ツールの厳密推論により全ノードの周辺分布を計算した結果を**図 6** に示す．

この状態から「変更容易性を高くしたい」「変更容易性は中程度でよい」という 2 つの要件を獲得したため，観測された証拠として $Modifiability = Mid$，$Performance = High$ を与えて再計算した結果を**図 7** に示す．アーキテクチャ上の決定事項に関する事後確率は

$$P(Programming\ Language = Java) = 0.986$$
$$P(Client/Server\ Architecture = ThinClient) = 0.613$$
$$P(DataStorage = RelationalDB) = 0.793$$

となり，Java 言語で実装された Thin クライアント型の 3 層クライアント／サーバアーキテクチャの組み合わせが推奨されることがわかる．また，実装プラットフォームに関する事後確率では以下の選択肢がそれぞれ最尤値となる．

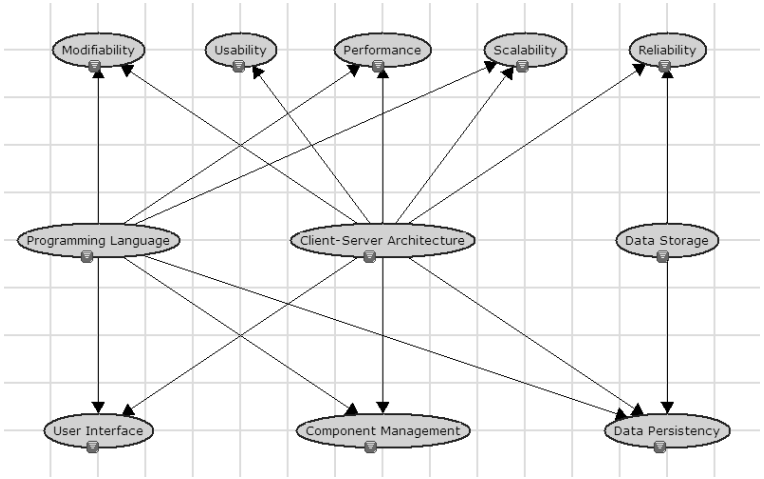


図 5 業務アプリケーションの実装プラットフォーム選択のための BN
Fig. 5 A BN for choosing implementation platform of a business application.

表 1 性能に関する CPT
Table 1 CPT for performance.

Programming Language		Java			Ruby		
Client-Server Architecture		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 コンポーネント管理に関する CPT
Table 2 CPT for component management.

Programming Language		Java			Ruby		
Client-Server Architecture		Fat	Thin	Rich	Fat	Thin	Rich
Component Management	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

$$P(UserInterface = Struts) = 0.506$$
$$P(ComponentManagement = SpringFramework) = 0.535$$
$$P(DataPersistency = Hibernate) = 0.455$$

また，要求として明示的に指定されていない信頼性，スケーラビリティなどの品質特性も厳密推論により以下のように事後確率が計算されている．

$$P(Reliability = High) = 0.634$$
$$P(Scalability = High) = 0.702$$

4.3 議　　論

本節の適用事例では，BN に含まれる 11 個のノードのうち，品質特性に関する 2 個を要求として固定することにより，他のノードの事後確率の計算結果が実装プラットフォームの選定に役立つことを確認できた．厳密推論に必要な計算時間は汎用的な PC (Pentium M 1.60GHz, メモリ 1.5GB) でも 0.1 秒以内であり，観測値を与える操作もノードを 1 回クリックするだけであるため，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．

推論の正しさについては BN の構築中に訓練データを用意し，手動で CPT の補正を 3 回実施した結果，アーキテクトの経験と近い事後確率が得られている．事後確率が最大となる状態はその開発組織で最も実績がある選択肢であるため，経験の浅い開発者は BN の推論結果に従うことで誤った選択による失敗を避けられる．

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)[5] を考案した．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る手法を提案した[6][7]．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．

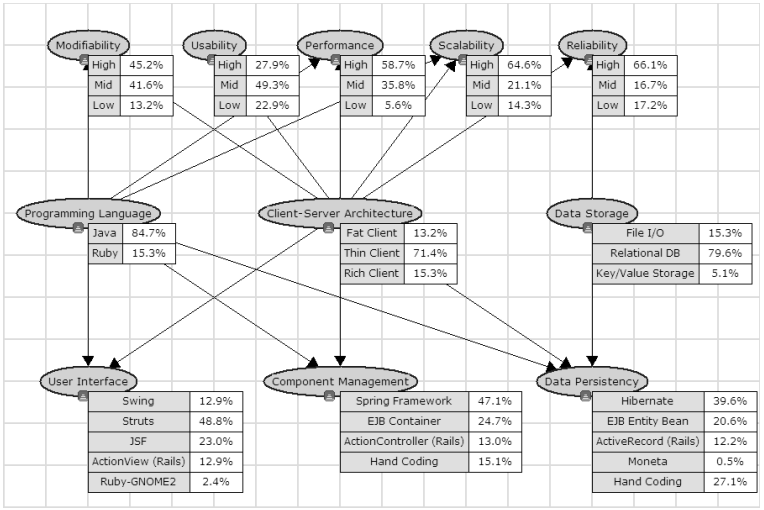


図 6 適用事例における周辺事後確率 — 初期状態
Fig. 6 Case study — initial state.

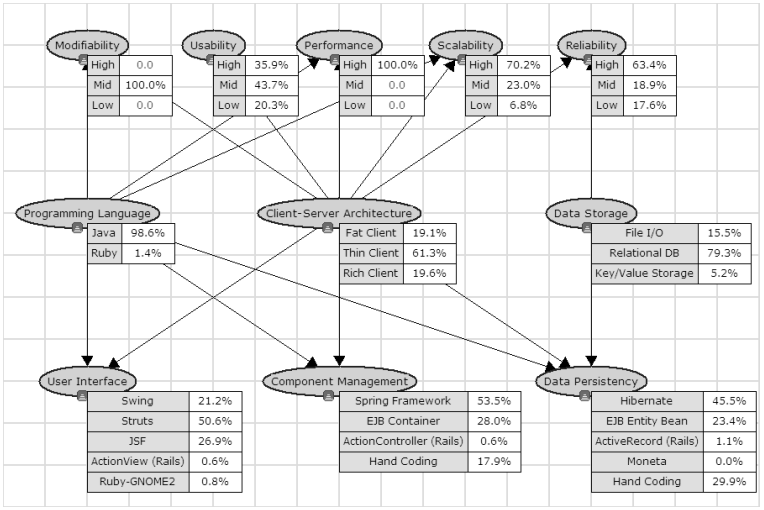


図 7 適用事例における周辺事後確率 — 2 つの非機能要求の観測後
Fig. 7 Case study — after observation of two NFRs

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[8] を提案した．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[9]．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捉えたものであり，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プラットフォームの因果関係を BN を用いてモデル化し，実装プラットフォームの組み合わ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，アーキテクチャ上の決定事項，実装プラットフォームの要求を確率変数の観測値として BN の厳密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を計算し，品質特性の予測や実装プラットフォームの選択を支援する．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プラットフォームを分析する事例を用いてその有用性を確認した．

提案手法は COTS ベースのシステム構築における開発の複雑さを低減し，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**．本稿では例題としてクライアント／サーバ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プラットフォーム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，実装プ

ラットフォーム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．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．

- **モデル駆動開発ツールとの連携**．3.1 節で述べたように，本稿で提案した手法は我々が開発しているモデル駆動型開発アプローチ ACCURATE の一部である．今後は ACCURATE に提案手法を統合し，BN を用いた実装プラットフォームの選択結果に基づいてソースコード生成まで一貫して支援する事例を示したい．

参 考 文 献

1) 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).
2) openArchitectureWare.org: Official openArchitectureWare Homepage, `http://www.openarchitectureware.org/`.
3) AndroMDA.org: AndroMDA.org - Home, `http://www.andromda.org/`.
4) Kansas State University Lab for Knowledge Discovery in Databases: Bayesian Network tools in Java (BNJ), `http://bnj.sourceforge.net`.
5) Klein, M. and Kazman, R.: Attribute-Based Architectural Styles, Technical Report CMU/SEI-99-TR-022, CMU/SEI (1999).
6) Petriu, D.C. and Wang, X.: From UML Descriptions of High-Level Software Architectures to LQN Performance Models, *AGTIVE '99: Proceedings of the International Workshop on Applications of Graph Transformations with Industrial Relevance*, Springer-Verlag, pp.47–62 (2000).
7) Petriu, D. C. and Shen, H.: Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications, *TOOLS '02: Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, Springer-Verlag, pp.159–177 (2002).
8) 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).
9) Pendharkar, P.C., Subramanian, G.H. and Rodger, J.A.: A Probabilistic Model for Predicting Software Development Effort, *IEEE Transactions on Software Engineering*, Vol.31, No.7, pp.615–624 (2005).