ソフトウェア工学 136−15
(2002. 3. 7)

4+1ビューモデルにもとづくソフトウェアアーキテクチャの記述およびその評価法

福澤 公之　　佐伯 元司
東京工業大学 大学院 情報理工学研究科 計算工学専攻

概 要

ソフトウェアアーキテクチャはシステムのソフトウェア品質特性に大きな影響を与えるため非常に重要であり，形式的に記述し解析や評価を行う手法が必要とされている．本稿ではRational Unified Processの4+1ビューモデルにもとづいたアーキテクチャ記述法と，そのアーキテクチャ記述をカラーペトリネットに変換し実行することで品質特性を定量化する手法を提案する．さらに，実際に提案した手法を用いてRMIのアーキテクチャを記述し品質特性を定量化することで本研究の有効性を確認した．

Description and Evaluation of Software Architecture based on 4 + 1 View Model

Kimiyuki Fukuzawa　　*Motoshi Saeki*
Department of Computer Science,
Graduate School of Information Science and Engineering,
Tokyo Institute of Technology

Abstract

Software Architecture has a great influence on achieving software quality characteristics of a system, so analysis and evaluation techniques are necessary in the early phase of development processes. In this paper, we propose a technique for describing architectures based on 4 + 1 View Model in Rational Unified Process, and a technique for quantifying quality characteristics by converting architecture description to Coloured Petri Nets and executing it. To show the effectiveness of our techniques, we describe RMI architecture and quantify its quality characteristics.

1 はじめに

ソフトウェア開発のライフサイクルにおいて, アーキテクチャ設計は要求と実装とのギャップを埋める重要な役割を果たしている. ソフトウェアアーキテクチャ(以下, アーキテクチャ) はシステムの構造や振舞のみならず, システムのセキュリティ, 信頼性, 効率性などのソフトウェア品質特性 (以下, 品質特性) とも密接に関連している. そのため, 設計したアーキテクチャを形式的に記述し解析や評価を行うことで, ソフトウェア開発のより早い段階で目標とする品質特性を達成できるかどうかを調べることができ, 開発の生産性を向上させることができる. この場合, アーキテクチャ記述はシステム構成要素間の関係を明確に定義できる以外に, 構成したシステムが期待した動作をするかどうかをチェックできることが不可欠である. すなわち, チェック可能な形式記述法と解析ツールが必要となる.

現在, 形式的な記述法として, Z(集合論 + 述語論理)[1], CCS, CSP[2][3], π-計算 [4], CHAM[5], 半順序イベント集合 [6] を意味的基礎とするアーキテクチャ記述法がすでに提案されている. しかし, これらの記述法を用いてアーキテクチャを記述するための方法論は確立されておらず, 形式的な記述に慣れていないアーキテクトにとって実際に実用規模のシステムのアーキテクチャを記述するのは難しい.

一方, システムの開発方法論である Rational Unified Process[7](以下,RUP) においてアーキテクチャを記述するための方法論やアーキテクチャとして何を書くべきかを定めた 4+1 ビューモデル [8] が提案されているが, 具体的なアーキテクチャの記述法は定められておらず, アーキテクチャ記述の解析を行う手法やツールも存在しない.

そこで本研究では実用規模のシステムのアーキテクチャを記述するのに適したアーキテクチャ記述法の提案と, そのアーキテクチャ記述をもとに解析を行い, 品質特性を定量化する手法を提案することを目的とする.

実用規模のシステムのアーキテクチャを記述しやすくするために, 本研究では RUP の 4+1 ビューモデルに則した記述法を定義し, その表記としては一般的にオブジェクト指向モデリングに利用されている UML[9] のサブセットおよび表形式記述を採用した. これにより, アーキテクトは RUP の方法論を利用し, UML のモデリングツールを用いてアーキテクチャの記述を行うことができる.

また, アーキテクチャを解析するために, アーキテクチャ記述をカラーペトリネット [10] に変換し, そのカラーペトリネットを実行することで品質特性を定量化できるようにした. アーキテクチャ記述の意味的基礎としてカラーペトリネットを採用した理由は, カラーペトリネットが挙動に関する解析や評価に適しており, 実行にともなってトークンに付加された属性値の計算を行うことが可能であり, シミュレーションや解析の手法と Design/CPN などのツールが存在するからである. 本研究で提案した手法を用いてアーキテクチャの品質特性を定量化することで, アーキテクトはアーキテクチャの評価や選択を行うことができる.

2 品質特性とその定量化手法

2.1 ソフトウェア品質特性

本研究ではソフトウェア品質のモデルとして ISO/IEC9126 を採用した. ISO/IEC9126 では, ソフトウェアの品質を 6 つの特性と 21 の副特性に分類して定義している.

以下に, 品質特性とその概要を挙げる.

品質特性	概要
機能性	ユーザ要求を満足する一連の機能
信頼性	一定の期間, 条件下での所定性能維持能力
使用性	使い勝手の良さ
効率性	一定条件下での性能, 資源効率
保守性	変更のしやすさ
移植性	別環境への移行しやすさ

本研究ではこれらの品質特性のうち, アーキテクチャの振舞に関係があり, カラーペトリネットの実行で定量化できるものとして, 以下の 4 つの品質副特性の定量化を行う.

機能性	
セキュリティ	不正アクセスの排除能力
信頼性	
成熟性	潜在障害の低率さ
効率性	
時間効率性	応答時間, 処理時間, 処理能力
資源効率性	資源の量, 使用時間

2.2 セキュリティ・成熟性・時間効率性

カラーペトリネットで表現したアーキテクチャの実行は, 図 1 のように処理の流れとともにトークン

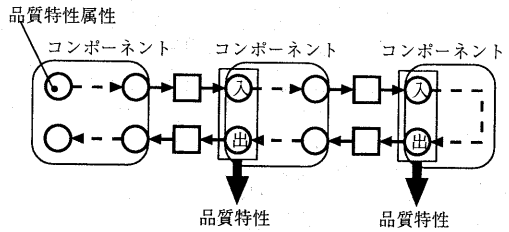

図 1: セキュリティ 成熟性 時間効率性

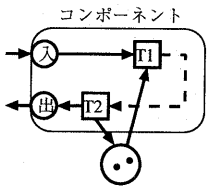

図 2: 資源効率性

が移動していくことで表現される．セキュリティ，成熟性，時間効率性の定量化は，トークンにこれらの品質特性に関する品質特性属性を持たせ，処理の流れに伴って属性の値の計算を行い，全ての処理の終了後にアーキテクチャコンポーネントのインターフェイスごとに入力時と出力時の属性値の変化を平均することで行う．

以下，それぞれの品質特性に関して品質特性属性の計算方法を説明する．

セキュリティに関する品質特性属性は「通信により情報が漏洩する危険性」を表し，通信を表すトランジションにおいて以下の式のように危険性が加算される．

$$\text{属性値} \mathrel{+}= \text{盗聴危険性} \times \sum_{\text{転送データ}} \left(\text{情報重要度} \times \prod_{\text{暗号}} (1 - \text{暗号強度}) \right)$$

セキュリティはインターフェイスの入力時と出力時の属性値の差として定量化する．

成熟性に関する品質特性属性は，「処理が正常に達成される確率」を表し，トランジションにおいて以下の式のように成功率が乗算される．

$$\text{属性値} \times= \text{成功率}$$

成熟性はインターフェイスの出力時の属性値を入力時の属性値で割った結果として定量化する．

時間効率性に関する品質特性属性は「処理や通信にかかる時間」を表す．処理を表すトランジションにおいては以下の式のように処理時間が加算され，

$$\text{属性値} \mathrel{+}= \text{処理時間}$$

通信を表すトランジションにおいては以下の式のように通信時間が加算される．

$$\text{属性値} \mathrel{+}= \text{呼出時間} + \left(\sum_{\text{転送データ}} \text{サイズ} \right) \div \text{スループット} \times \text{オーバーヘッド}$$

時間効率性はインターフェイスの入力時と出力時の属性値の差として定量化する．

これらの計算で用いるパラメタはアーキテクトが経験や測定にもとづいて与える．

2.3 資源効率性

資源効率性は CPU の使用率として定量化する．

カラーペトリネットで表現したアーキテクチャにおいて，計算資源である CPU は図 2 の下部にあるプレース中のトークンのように表現される．この図では，トランジション T1 で CPU の割り当てを行い，トランジション T2 で CPU の解放を行っている．CPU の割当時間は T1 と T2 の発火時刻の差で表される．

資源効率性 (CPU 使用率) はハードウェアごとに CPU の総割当時間をシステムの総実行時間とそのハードウェアの CPU 数で割った結果として定量化する．

3 アーキテクチャ記述法

本研究ではアーキテクチャ記述に RUP の 4+1 ビューモデルを詳細化した図を用いる．本節では 4+1 ビューモデルの説明を行った後，本研究で提案するアーキテクチャ記述法において各ビューをどのように記述するかを説明する．

3.1 4+1 ビューモデル

4+1 ビューモデルは論理ビュー，プロセスビュー，実装ビュー，配置ビュー，ユースケースビューの 5 つのビューからなり，アーキテクチャに関わるそれぞれの役割に応じてビューを使い分けることができるという特徴がある．

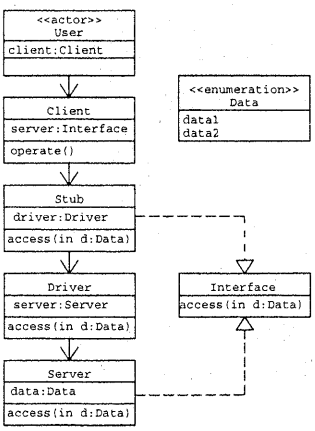

図 3: 論理ビュー (クラス図)

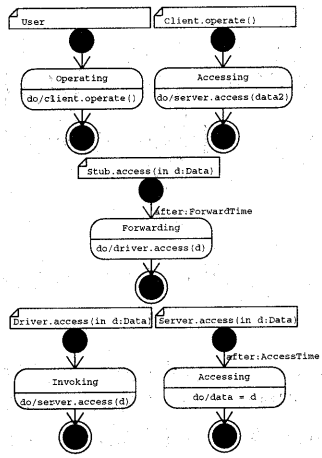

図 4: 論理ビュー (状態図)

RUP では論理ビューにはアーキテクチャの構造や振舞を記述し, プロセスビューには動作の並行性や並列性を記述し, 実装ビューにはシステムで使用しているモジュールやプロトコルを記述し, 配置ビューにはハードウェアや構成要素の配置を記述し, ユースケースビューには他のビューの理解を促進させるためにシステムの動作の典型例を示すシナリオを記述すると定めている.

3.2 論理ビュー

論理ビューにはアーキテクチャの構造と振舞を記述する. 構造は図 3 のようにクラス図として記述し, 振舞は図 4 のように状態図として記述する. ここでは例として RMI(Remote Method Invocation) のアーキテクチャを記述した.

クラス図ではアーキテクチャの構成要素をクラスとして記述し, ユーザなどのアクターを actor ステレオタイプを付けたクラスとして記述する. 構成要素は属性と操作を持つことができ, アクターは属性を持つことができる. 構成要素の振舞はその操作ごとに状態図として記述し, アクターの振舞は 1 つの状態図として記述する.

振舞を表す状態図において, 状態のアクションには属性値の変更, 他のオブジェクトの操作呼出, オブジェクトの生成を記述することができ, 遷移には遷移が起こる条件であるガードと, 遷移にかかる時間を記述することができる.

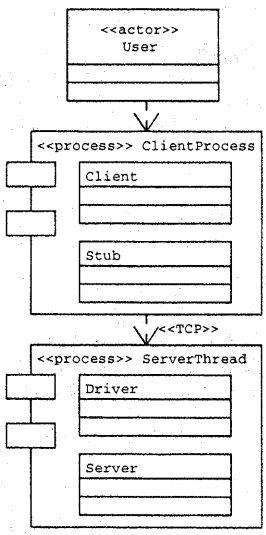

図 5: プロセスビュー

3.3 プロセスビュー

プロセスビューにはプロセスとそれらの間の通信を図 5 のようにコンポーネント図で記述する.

プロセスビューでは, プロセスを process ステレ

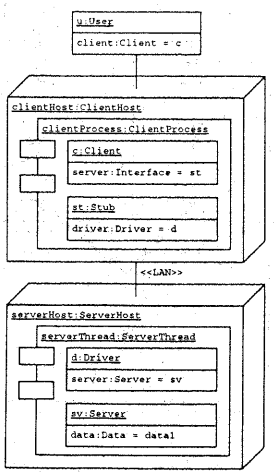

図 6: 配置ビュー

オタイプを付けたコンポーネントとして記述し, アクターを actor ステレオタイプを付けたクラスとして記述し, それらの間で起きる通信をその通信の種類やプロトコルを表すステレオタイプを付けた依存性[1] として記述する.

また, プロセスを表すコンポーネントの中にはそのプロセスで実行される構成要素をクラスとして記述する.

図 5 では Client と Stub が ClientProcess で実行され, Driver と Server が ServerProcess で実行され, ClientProcess と ServerProcess の間で TCP の通信が起こるということが記述されている.

3.4 配置ビュー

配置ビューにはハードウェアとそれらの間の通信, ハードウェア中のプロセス, システム実行開始時における構成要素の配置を図 6 のように配置図で記述する.

配置ビューではハードウェアをノードインスタンスとして記述し, アクターをアクタークラスのインスタンスとして記述し, それらの間で起きる通信をその通信の種類を表すステレオタイプを付けたリンクとして記述する.

[1] コンポーネント図ではコンポーネント間の関係として通常依存性を用いる

また, ハードウェアを表すノードインスタンスの中にプロセスを表すコンポーネントのインスタンスを記述し, その中にシステム実行開始時における構成要素の配置をオブジェクトして記述する.

図 6 ではクライアントのハードウェア (clientHost) とサーバのハードウェア (serverHost) が LAN で接続され, Client, Stub, Driver, Server がそれぞれ 1 つずつ配置されている構成が記述されている.

3.5 実装ビュー

実装ビューは UML ではなく表を用いて品質特性属性の計算に用いるパラメタを記述する.

記述するパラメタは他のビューで記述した要素 (キー) ごとに以下の表で定義される.

キー	パラメタ		
型	サイズ	情報重要度	暗号強度
操作	暗号強度	成功率	呼出時間
プロセス間通信	オーバーヘッド		
ハードウェア間通信	スループット	盗聴危険性	成功率
ハードウェア	CPU 数		
時間変数	時間		

操作呼出のパラメタの型となるクラスやインターフェイス, 列挙型にはその型のデータのサイズと情報重要性, かけられている暗号の強度をパラメタとして与え, 構成要素の振舞を表す操作にはその呼び出し時に転送するデータ全体にかける暗号の強度と呼び出しの成功率, 呼び出しにかかる時間をパラメタとして与え, プロセス間の通信の種類やプロトコルを表すステレオタイプにはその通信のオーバーヘッドをパラメタとして与え, ハードウェア間の通信の種類を表すステレオタイプにはその通信路のスループットと盗聴危険性, 通信の成功率をパラメタとして与え, ハードウェアには CPU の数をパラメタとして与え, 状態図中で遷移時間を表すのに用いる時間変数には時間をパラメタとして与える.

RMI のアーキテクチャにおける記述例を表 1 に示す.

3.6 ユースケースビュー

ユースケースビューは品質特性の定量化には関与しないが, 他のビューの理解を促進するために図 7 のようにコラボレーション図として記述する.

表 1: 実装ビュー

型	サイズ	情報重要度	暗号強度
Data	100	1	0

操作	暗号強度	成功率	呼出時間
Client.operate	0	1	0
Stub.access	0	1	0
Driver.access	0	1	2
Server.access	0	1	0

プロセス間通信	オーバーヘッド
TCP	1.1

ノード間通信	スループット	盗聴危険性	成功率
LAN	1000	0.9	0.99999

ノードインスタンス	CPU 数
clientHost	1
serverHost	1

時間変数	時間
ForwardTime	0.2
AccessTime	0.1

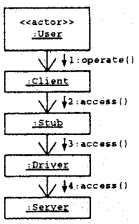

図 7: ユースケースビュー

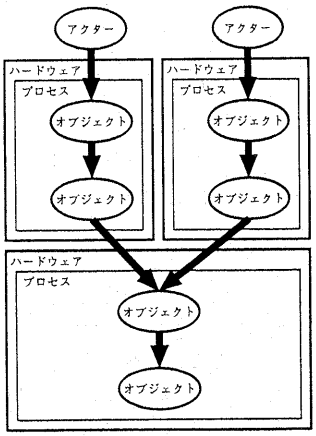

図 8: 変換イメージ (全体)

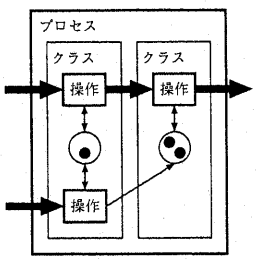

図 9: 変換イメージ (プロセス)

4 変換手法

本節ではアーキテクチャ記述をカラーペトリネットに変換する手法を説明する.

アーキテクチャ全体は配置ビューをもとに図 8 のように変換する. この図はクライアントを 2 つに増やした RMI のアーキテクチャを表している.

アーキテクチャの実行はアクターの振舞を表す状態図の実行によって開始される. 状態図の実行中に他のオブジェクトの操作呼出が行われると, 対象オブジェクトの操作の振舞を表す状態図が実行される.

また, プロセスで状態図を実行している場合, プロセス内のオブジェクトへの外部からの操作呼出はロックされ, ハードウェア中で同時に起きる状態遷移の数はそのハードウェアの CPU 数以下となる.

提案したアーキテクチャ記述法ではオブジェクトの動的生成を記述することができるが, カラーペトリネットではネットの構造を動的に変更することができないため, プロセスはプロセスビューをもとに図 9 のように変換する.

プロセスはそのプロセスで実行する構成要素を表すクラスを変換したものからなり, クラスは振舞を表す操作を変換したものと, そのクラスのインスタンスを表すトークンを保持するプレースからなる. あるオブジェクトの操作の実行中にはそのオブジェクトが必要に応じて参照され, オブジェクトの生成時には対象となるプレースに新しいトークンが出力される.

状態図は図 10 のように状態や遷移ごとにネットに変換し接続される. 状態図の実行はネット上を対象となるオブジェクトの識別子や操作のパラメタ, 品質特性属性を保持したトークンが遷移していくことで表現される. この例において中央の状態では操作呼出を行っているので, 対応するネットの斜線のプレースが呼出対象の操作を変換したネットに通信を表すトランジションを介して接続される.

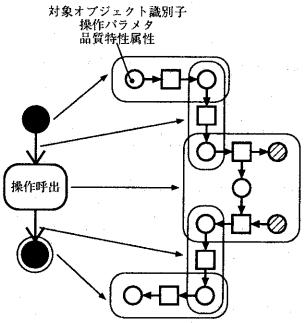

図 10: 変換イメージ (状態図)

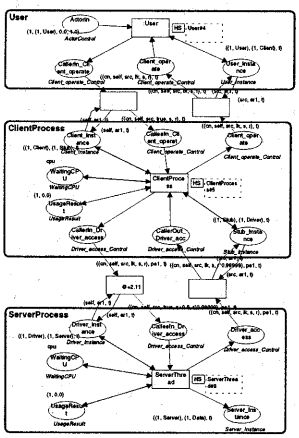

図 11: カラーペトリネット (全体)

5 適用実験

本研究の有効性を確かめるために, 適用実験として例で示した RMI のアーキテクチャの品質特性の定量化を行った.

RMI のアーキテクチャ記述を変換したカラーペトリネットのうち, アーキテクチャ全体を表すページ, プロセスを表すページ, 状態図を表すページをそれぞれ図 11, 図 12, 図 13 に示す. アーキテクチャ全体を表すページは配置ビュー, プロセスビュー, 実装ビューをもとに変換され, プロセスを表すページはプロセスビュー, 論理ビュー, 実装ビューをもとに変換され, 状態図を表すページは論理ビュー, 実装ビューをもとに変換される.

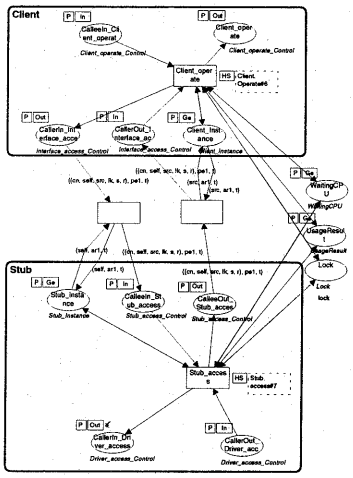

図 12: カラーペトリネット (ClientProcess)

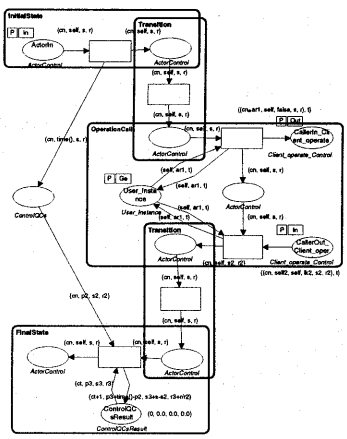

図 13: カラーペトリネット (User)

表 2: 品質特性 (クライアント ×1)

	セキュリティ	成熟性	時間効率性
User	0.9	0.99998	2.41
Client.operate	0.9	0.99998	2.41
Stub.access	0.9	0.99998	2.41
Driver.access	0.0	1.0	0.1
Server.access	0.0	1.0	0.1

	資源効率性
ClientHost	0.083
ServerHost	0.041

表 3: 品質特性 (クライアント ×3)

	セキュリティ	成熟性	時間効率性
User	0.9	0.99998	2.51
Client.operate	0.9	0.99998	2.51
Stub.access	0.9	0.99998	2.51
Driver.access	0.0	1.0	0.1
Server.access	0.0	1.0	0.1

	資源効率性
ClientHost	0.077
ServerHost	0.115

このカラーペトリネットを実行し定量化した品質特性を表 2 に示す. また, 配置ビューを変更してクライアントの数を 3 つにした構成で定量化した品質特性を表 3 に示す.

これらの結果を比べると RMI のアーキテクチャでは, クライアントの数を増やすことがセキュリティや成熟性には影響を与えないが, 時間効率性や資源効率性に影響を与えることがわかる.

本節では同じアーキテクチャを異なる構成で記述し品質特性を定量化することで, そのアーキテクチャ単体の評価を行ったが, 同じ機能を実現する異なるアーキテクチャを記述して比較することで, アーキテクチャ間の品質特性のトレードオフを評価し, アーキテクチャ選択の指針とすることもできる.

6 おわりに

本研究では実用規模のアーキテクチャを記述するのに適したアーキテクチャ記述法と, アーキテクチャ記述をカラーペトリネットに変換し実行することで品質特性を定量化する手法を提案した. また, 実際に RMI のアーキテクチャを記述し, その品質特性を定量化することで本研究の有効性を確認した.

今後の課題としては,

- 本研究で定量化した品質特性以外の品質特性を定量化できるようにすること
- アーキテクチャスタイルやパターンに関して, 適用することができるビューや適用することによる品質特性の変化を調べること
- 本研究ではオブジェクトの動的生成は記述できるがプロセスの動的生成は記述できないので, それを可能とすること

などが考えられる.

参考文献

[1] G. Abowd, R. Allen, and D. Garlan. Using Style to Understand Descriptions of Software Architecture. In *Proceedings of the ACM SIGSOFT '93 Symposium on the Foundations of Software Engineering*, pp. 9–20, December 1993.

[2] R. Allen and D. Garlan. A formal basis for architectural connection. In *ACM Transactions on Software Engineering and Methodology*, July 1997.

[3] R. Allen and D. Garlan. Formalizing architectural connection. In *Proceedings of the Sixteenth International Software Engineering*, Italy, May 1994.

[4] J. Magee, J. Kramer, and D. Giannakopoulou. Software architecture directed behaviour analysis. In *Proceedings of Ninth International Workshop on Software Specification and Design*, April 1998.

[5] P. Inverardi and A. L. Wolf. Formal specification and analysis of software architectures using the chemical abstract machine model. *IEEE Trans. Software Eng.*, Vol. 21, No. 4, April 1997.

[6] D. C. Luckham and J. Vera. An Event-Based Architecture Definition Language. In *IEEE Transactions on Software Engineering*, pp. 717–734, September 1995.

[7] P. Kruchten. *The Rational Unified Process, An Introduction.* Addison Wesley, 1998.

[8] P. B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, Vol. 12, No. 6, pp. 42–50, November 1995.

[9] OMG. Omg unified modeling language specification, September 2001.

[10] K. Jensen. *Coloured Petri Nets*, Vol. 1-2. Springer-Verlag, 1992.