Vol.2013-SE-182 No.17
2013/10/24

実行シナリオに基づくレイヤアーキテクチャの評価手法

小林　謙太郎[1,a)]　新田 直也[1,b)]

概要：大規模ソフトウェアの開発において, 開発初期におけるアーキテクチャの選択は, その後のプロジェクトの成否を大きく左右する. しかしながら, 実際の開発においてアーキテクチャをどのように選択すればよいかについては, 設計者の経験に依存する部分が大きく, アーキテクチャ選択のための決定的な手法は確立されていないのが現状である. そこで本研究では, レイヤアーキテクチャを対象に, 選択したアーキテクチャが要求仕様から抽出した実行シナリオと適合しているか否かを評価する手法を提案する. 事例研究として, 本研究室で開発した 3D 格闘ゲームの実行シナリオおよび初期アーキテクチャに対して本手法の適用を行った. その結果, 本手法による評価結果が, 当該プロジェクトにおいて初期アーキテクチャをほぼ変更することなく開発完了したという事実とよく一致していることがわかった.

1. はじめに

大規模ソフトウェアの開発において, 開発初期におけるアーキテクチャの選択は, その後のプロジェクトの成否を大きく左右する. アーキテクチャの選択が不適切だった場合, 最終的に十分なパフォーマンスが得られなくなる, 一部の機能が実装できなくなる, 場当たり的な実装を繰り返すことによって保守性が大きく低下するなどといった問題を生じる恐れがある. そこで近年, アーキテクチャを中心的な成果物と捉え, 反復的に洗練させていくアーキテクチャ中心開発手法 [1], [2] が注目されている. しかしながら, 実際の開発においてアーキテクチャをどのように選択していけばよいかについては, 設計者の経験に依存する部分が大きく, アーキテクチャ選択のための決定的な手法は確立されていないのが現状である.

そこで本研究では, 選択したアーキテクチャが, 要求仕様から抽出した実行シナリオと適合しているか否かを評価する手法の確立を目指す. 我々は先行研究 [3], [4] において, 選択したアプリケーションフレームワーク (以下, フレームワークと略) のアーキテクチャが実行シナリオと適合しているか否かを評価する手法の開発を行った. フレームワークを用いて開発されるアプリケーションは, アプリケーション固有の部分とフレームワークとして再利用される部分の 2 層からなるレイヤアーキテクチャを持つと考えることができる. そこで, 本研究では任意数の階層を持つ一般的なレイヤアーキテクチャに対しても適合性の判定ができるように, 先行研究の適応を行う. 先行研究との具体的な差異を以下に列挙する.

- フレームワークは特定のドメイン内で再利用可能な設計およびその実装である. したがって, 先行研究におけるフレームワークの選択は, アーキテクチャのみならず実装の一部までもを固定化することを意味しており, そのことによって各実行シナリオの実装に課される制約は非常に強いものであった. これに対し, 本研究ではアーキテクチャの選択のみを評価の対象としているため, 各実行シナリオの実装に対してはあまり強い制約が課されることはない. 通常は, 実装の指針やガイドラインといったレベルでの制約が課される程度である. ただし, アーキテクチャに基づいた指針やガイドラインから外れる実装の増加は, システム全体の保守性や再利用性を著しく低下させる危険性がある.
- 同様の理由により, 先行研究では選択したフレームワークの適合性を評価するために, フレームワークの実装の詳細にまで立ち入って調査を行う必要があった. これに対し, 本研究では特定の実装を前提とする必要がないため, より高い抽象度で評価を行うことができると考えられる.
- 先行研究では, 不適合性の 1 つの要因としてフレームワークによるリソースの占有に着目した. 本研究では, 一般的なレイヤアーキテクチャを対象とするため, 不適合性を評価する上でどのリソースをどのレイヤに配置するかという点に主に着目する.
- 先行研究では, もう 1 つの不適合性の要因としてフレー

[1] 甲南大学大学院 自然科学研究科
Graduate School of Natural Science, Konan University
[a)] m1324005@center.konan-u.ac.jp
[b)] n-nitta@konan-u.ac.jp

ムワークによる制御機構の占有にも着目した．これは，フレームワーク側のコードが必要に応じてアプリケーション側のコードを呼び出す制御の反転と呼ばれる仕組みを考慮したものである．しかしながら，制御の反転はフレームワーク特有の仕組みであるため，本研究では対象としない．その代わり，本研究では下位レイヤ内で占有されているリソースの更新を上位レイヤから指示する際に，他の複数のリソースも連鎖的に更新されてしまうような下位レイヤ内の制御構造に着目する．

事例研究として，本研究室で開発した 3D 格闘ゲームの実行シナリオおよび初期アーキテクチャに対して本手法の適用を行った．その結果，初期アーキテクチャが実行シナリオと適合することがわかった．これは，当該プロジェクトにおいて初期アーキテクチャをほぼ変更することなく，実行シナリオをすべて実装する形で開発完了したという事実とよく一致しており，本手法の有効性を示唆しているといえる．

2. レイヤアーキテクチャ

ソフトウェアアーキテクチャの厳密な定義は確立されていないが，一般的にはソフトウェア全体を複数のサブシステムに分割し，それらの間の関係を定義したものとされ，静的側面と動的側面の両面によって記述される場合が多い．**レイヤアーキテクチャ** [5] は複数のレイヤによって構成され，それらの間に利用-被利用の関係が定義できるアーキテクチャをいう．ここで，利用する側のレイヤを**上位レイヤ**とよび，利用される側のレイヤを**下位レイヤ**とよぶ．レイヤアーキテクチャにおいては，レイヤ間で利用-被利用の関係が循環してはならない．フレームワークを用いて開発されるアプリケーションは，アプリケーション固有の部分を上位レイヤとし，フレームワークとして再利用される部分を下位レイヤとするレイヤアーキテクチャを持つと考えることができる．

3. レイヤアーキテクチャの記述法

1 節で述べたように，本研究では，一般的なレイヤアーキテクチャの実行シナリオに対する適合性を評価できるように，フレームワークのアーキテクチャのみを評価の対象としていた先行研究の拡張を行う．先行研究では，実行シナリオを実装する上で，以下の 2 種類の制約が生じる可能性を考慮して評価を行った．

(1) フレームワークがリソースを占有し，アプリケーション側からのリソースへのアクセスが制限されることによって生じる制約
(2) フレームワークがアプリケーション全体の制御機構を占有し，アプリケーション側のコードが実行されるタイミングが限定されることによって生じる制約

本研究ではこれら 2 種類の制約と関連して，以下の 2 種類のアーキテクチャ上の指針をレイヤアーキテクチャの主要な構成要素であると考える．

(1') 各リソースをどのレイヤに配置し，上位レイヤ側から下位レイヤ内のどのリソースを制御するようにするのかという指針 (**リソースの配置と制御**)
(2') 上位レイヤ側から下位レイヤ内のあるリソースを更新する際に，同時に別のリソースも更新するようにするのか否かという下位レイヤ内の制御構造に関する指針 (**リソース更新の連鎖**)

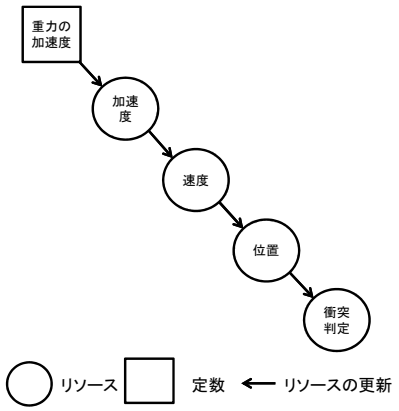


図 1 物理演算ドメインのリソース依存グラフ

3.1 リソース依存グラフ

対象領域の中で持続的に存在し，その変化を外部から観測できる属性もしくは実体を**リソース**とよぶ．**リソース依存グラフ**とは，問題ドメインが要求するリソース間の依存関係を記述したグラフである．リソース依存グラフは問題ドメイン固有の情報を記述したものであるため，アーキテクチャの選択には依存しない．**図 1** に物理演算のドメインにおけるリソース依存グラフを示す．物理演算ドメインにおいては物体の運動をシミュレートするために，例えば物体の加速度，速度，位置，他の物体との衝突判定といった情報を扱う必要がある．これらの情報をリソースとみなしたとき，それらの間には図に示したような依存関係が存在している．リソース依存グラフにおいて，円はリソース，矩形は定数，矢印はリソースまたは定数からその状態が影響を与えているリソースへのリソース間の関係をそれぞれ示す．例えば，図 1 において，速度から位置への矢印は，速度の値が位置の変化に影響を与えるという物理演算ドメインにおける依存関係を示している．

3.2 リソース更新グラフ

リソース更新グラフは，選択したレイヤアーキテクチャによってリソースがどのようにして更新されるかという観点から上記 (1'), (2') で示したアーキテクチャ上の指針を表現したグラフである．リソース更新グラフは，当該ド

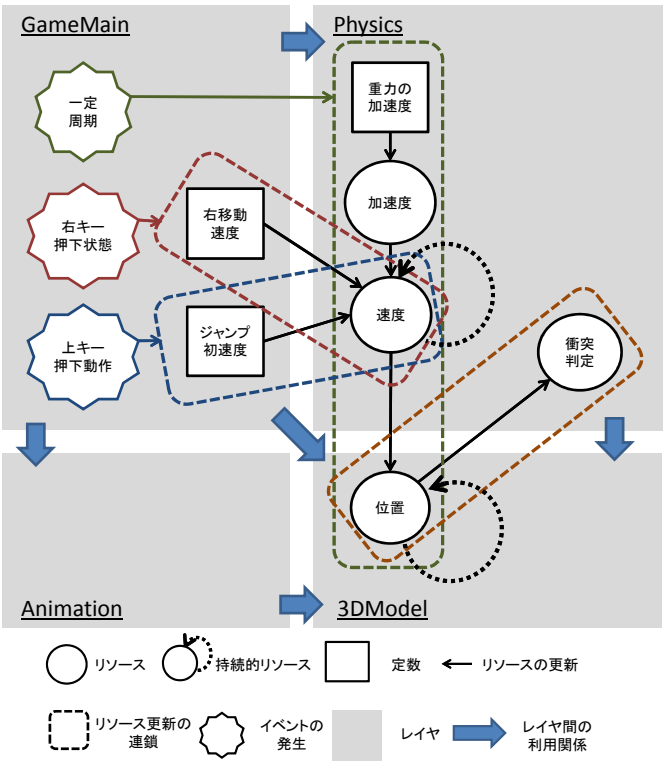


図 2　RadishFight のリソース更新グラフ

メインのリソース依存グラフを拡張して作成される．アーキテクチャの選択肢が複数存在する場合は，アーキテクチャの候補の数だけリソース更新グラフが作成されるが，元となるリソース依存グラフは共通である．図 2 に，本研究室で開発した 3D 格闘ゲーム RadishFight[6] の開発当初のアーキテクチャを元に作成したリソース更新グラフを示す．RadishFight は，GameMain, Physics, Animation, 3DModel の 4 つのレイヤからなるレイヤアーキテクチャを持つ．GameMain はゲーム全体の制御，Physics は物理演算，Animation は 3D 物体の定型的な動作，3DModel は構造化された 3D 物体の形状および幾何演算をそれぞれ扱う．3DModel が最下位に位置し，Physics と Animation はその上位，GameMain は他のすべてのレイヤの上位に位置する．図 2 では，大きな矩形で各レイヤを，大きな矢印でレイヤ間の利用関係を表している．リソース依存グラフ同様，円はリソース，小さな矩形は定数を表す．なお，小さな矢印は当該アーキテクチャにおいて，どのリソースの情報を元にどのリソースの情報を更新するかを示している．破線領域はリソース更新が連鎖する範囲を示し，星はイベントの発生，星からの矢印はイベントの発生によって更新されるリソースを表している．計算過程の中で一時的に存在するリソースを**一時的リソース**とよび，システムの実行の中で持続的に存在するリソースを**持続的リソース**とよぶ．持続的リソースには更新前の状態から更新後の状態への影響を表す点線の矢印によるループを記す．図 2 のリソース間の更新の関係は，図 1 のリソース間の依存関係と一致していることがわかる．

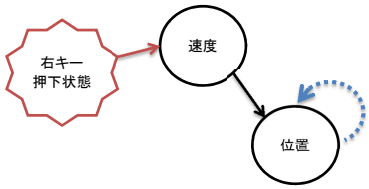


図 3　S1 のシナリオグラフ

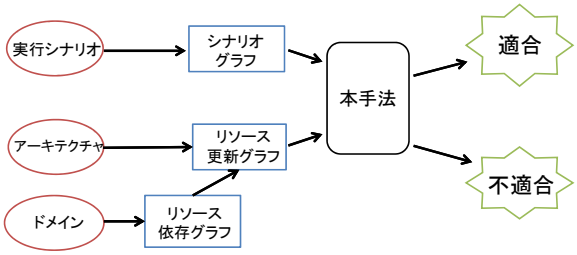


図 4　手法の概要

4. 実行シナリオに基づくレイヤアーキテクチャの評価法

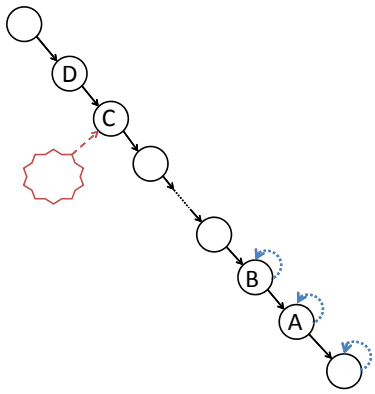
本節ではレイヤアーキテクチャを実行シナリオとリソースの更新という観点から評価する手法について説明する．本手法はレイヤアーキテクチャから導かれるリソースの更新に関する指針と，実行シナリオが要求するリソースの振る舞いを比較し，適合性の評価をおこなう．そのため，レイヤアーキテクチャから導かれる指針は前節で述べたリソース更新グラフで，実行シナリオが要求するリソースの振る舞いは以下で述べるシナリオグラフで表し，それらを比較して評価を行う．

4.1 シナリオグラフ

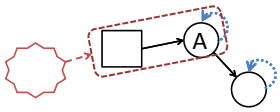
シナリオグラフはある実行シナリオが要求するリソースの振る舞いを表したグラフである．シナリオグラフは必要なリソースとイベント発生による更新を含み，必要に応じて複数の実行シナリオを一つのグラフで表すことがある．またシナリオグラフはアーキテクチャからは独立している．図 3 シナリオグラフの例を示す．シナリオグラフの記法は基本的にリソース更新グラフと共通であるが，レイヤに関する記述やリソース更新の連鎖に関する記述は含まない．

4.2 評価手法

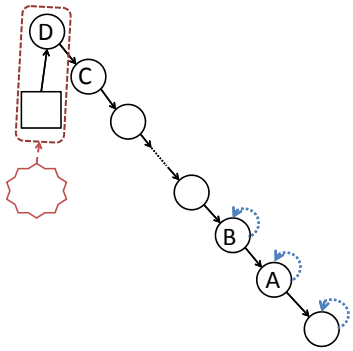
評価手法の概要を図 4 に示す．本手法では**リソースの再現性**に着目してアーキテクチャの評価を行う．リソースの再現性とはシナリオグラフによって要求されるリソースの更新を，アーキテクチャ上で定義されているリソースの制御のみで再現できるか否かを示す．再現できる場合を再現可能といい，そうでない場合を再現不能という．



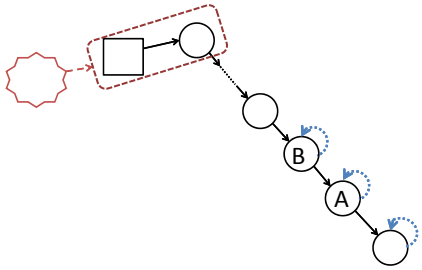
(a) シナリオグラフ例



(b) リソース更新グラフ例 1



(c) リソース更新グラフ例 2



(d) リソース更新グラフ例 3

図 5 リソースの再現性の評価

リソースが再現不能と判断された場合, そのアーキテクチャは実行シナリオと適合しないと評価する. 以下のいずれかの場合, リソースが再現不能になる.

(1) シナリオグラフ上で定義されたリソースの観測可能な振る舞いを再現するために必要な情報が, リソース更新グラフで定義されたリソースに含まれていない. 例えば, **図 5**(a) に示したシナリオグラフが要求するリソースの振る舞いを図 5(b) のようなリソース更新グラフによって再現できるか否かについて考える. シナリオグラフではイベントの発生によってリソース C が更新されることが要求されているが, C の影響を受けて間接的に更新される B の値をイベント発生時に直接計算することによって, B の状態のうちイベント発生からの影響部分は再現することができる. しかしながらこのリソース更新グラフでは更新前の B の状態の情報が欠如しているため, その影響を受けるリソース B および A の状態を完全には再現することができない.

(2) リソース更新グラフ上でのリソースの更新が間接的で, シナリオグラフによって要求されているリソースの振る舞いを実現できない. 例えば図 5(a) に示したシナリオグラフが要求するリソースの振る舞いを図 5(c) のようなリソース更新グラフによって再現できるか否かについて考える. シナリオグラフではイベントの発生によってリソース C が更新されることが要求されているが, リソース更新グラフではリソース D を更新することによって間接的に C を更新するように定義されている. しかしながら, D を通じた間接的な制御によって, C を直接制御するのと同等の振る舞いを再現できる保証はない. 例えば, 3D ゲームにおいて右キーを押したときにプレイヤーが瞬時に右移動を始める動作は, プレイヤーの速度を直接制御することによって実現することが可能であるが, プレイヤーの加速度を制御する方式では一時的に加速度として無限の値を与えなければならなくなり, 実現することができない.

上記の 2 つの場合について考えたとき, 図 5(a) のシナリオグラフの要求を満たすのは, 図 5(d) に示したように, イベントの発生によって C から B までの間のいずれかのリソースが更新される場合のみである. このようにシナリオグラフとリソース更新グラフの形状のみからリソース再現性を形式的に判定することができる.

5. レイヤアーキテクチャの評価事例

5.1 評価事例

提案手法を, 我々の研究室で開発した RadishFight という 3D 格闘ゲームのアーキテクチャに適用して本手法の有効性の評価を行う. 3 節で説明したように, この RadishFight は 4 つのレイヤ GameMain, Physics, Animation, 3DModel によって構成されている. それぞれの関係は図 1 に示した

通りである.

また本事例で用いた実行シナリオは以下の通りである. (RadishFight の全実行シナリオの一部である.)

S1 プレイヤーが水平な地面の上にいるとき, 右移動 (左移動) ボタンを押すと, プレイヤーは右 (左) を向いて地面の上を移動する.

S2 プレイヤーが左または右に移動した際に, 足元に地面がなくなると, プレイヤーは自由落下を開始する.

S3 プレイヤーが水平な地面の上または空中にいるとき, 上ボタンを押すと, プレイヤーはジャンプし始める.

S4 プレイヤーが物体にめりこんだとき, めりこんだ物体の法線方向にプレイヤーを押し戻す.

これらのシナリオからシナリオグラフを作成したものを図 3 と図 **6**, 図 **7** に示す.

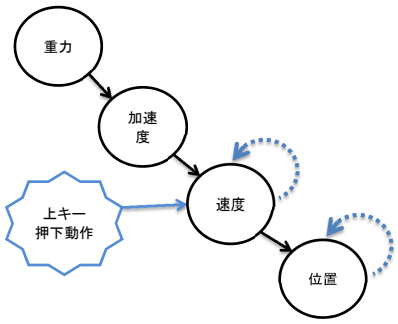


図 **6** S2 及び S3 のシナリオグラフ

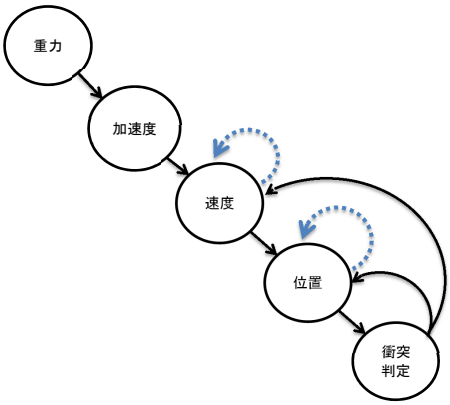


図 **7** S4 のシナリオグラフ

S2 と S3 は 1 つのシナリオグラフで表現している. その理由はジャンプキーを押下直後から自由落下をし始めるためである. RadishFight の開発当初のアーキテクチャを元に作成したリソース更新グラフは図 2 に示した通りである.

ところで最初にアーキテクチャを選択するに当たって, 以下の 2 つのアーキテクチャ案も同時に検討された.

A 案 移動時, またはジャンプ時に位置を直接更新する.

B 案 移動時, またはジャンプ時に外力が加わると考え, 重力と合成する形で加速度を更新する.

これらをリソース更新グラフとして表したものを, それぞれ図 **8**, 図 **9** に示す.

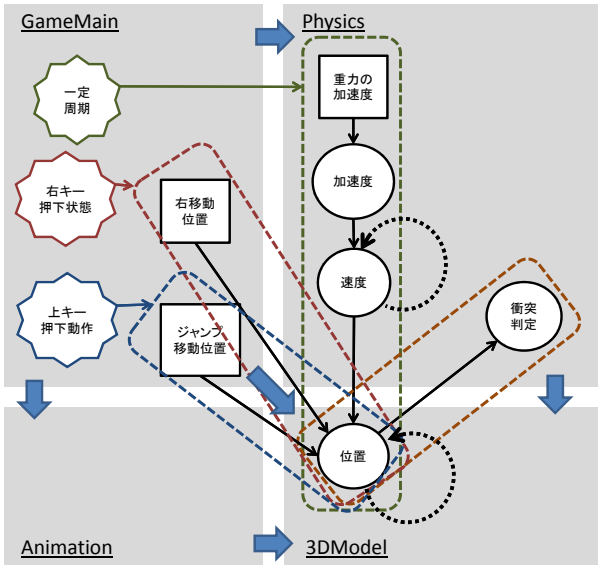


図 **8** A 案のリソース更新グラフ

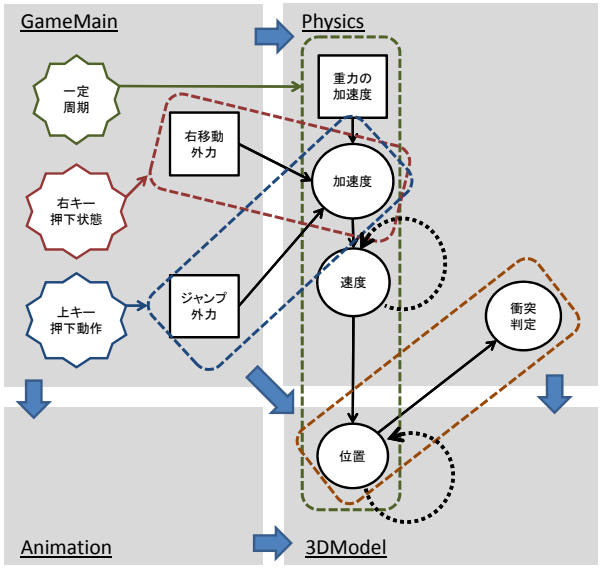


図 **9** B 案のリソース更新グラフ

最終的に選択されなかったこれらの案も本手法を用いて評価することによって, 本手法が開発当初のアーキテクチャの選択理由を説明できるか否かの検証を行う.

5.2 評価結果

本評価事例の結果を表 1 に示す. A 案がシナリオ S2 と S3 に適合しないのは, S2 と S3 の自由落下において要求される更新前の速度の情報がこのアーキテクチャの中に存在しないためである. 具体的にはジャンプする際に位置を上方向に更新したとしても, 初速度が設定されていないので, 即座に落下し始めてしまう. B 案が S1 に適合しないのは, 4.2 節で説明したように, 加速度を制御することで, S1 が要

表 1　評価結果

アーキテクチャ＼シナリオ	S1	S2	S3	S4
開発当初	○	○	○	○
A 案	○	×	×	○
B 案	×	×	×	○

求している動作を再現しようとすると, 一時的に無限の加速度を指定しなければならないためである.

6. 考察

前節で述べた評価結果から, 本手法によって RadishFight の開発当初になぜ A 案でも B 案でもなく, 現行のアーキテクチャを選択したのかの一定の説明ができることがわかった. 今後, 他のソフトウェアのアーキテクチャの選択事例にも適用していくことで, 本手法の有効性を確認していきたい. 先行研究ではフレームワークの実装を再利用することによる制約を考慮して低い抽象度で評価を行っていたが, 本研究ではそのような制約を考慮する必要がないため, より上流工程においてアーキテクチャ評価が可能であると考えられる. 今後の課題としては本手法の一部を自動化したツールの実装が挙げられる. ツールの支援によりリソースが膨大なプロジェクトでも, 比較的短時間でレイヤアーキテクチャを評価することができると期待される. またアーキテクチャ抽出技術として, ソースコードからリソース更新グラフを自動抽出するツールを実装することも今後の課題として挙げられる. そのためには, ソースコードとシナリオ更新グラフの関係性をさらに詳細化し, モデル化することが必要であると考えられる.

7. おわりに

選択したアプリケーションフレームワークのアーキテクチャが実行シナリオと適合しているか否かを評価する先行研究の手法を拡張して, 選択したレイヤアーキテクチャと実行シナリオの間の適合性を評価する手法の構築を行った. また, 本研究室で開発した 3D 格闘ゲームの実行シナリオおよび初期アーキテクチャに対して本手法の適用を行ったところ, 本手法による評価結果が, 当該プロジェクトにおける初期アーキテクチャの選択理由をうまく説明できることがわかった.

今後, 本手法の中で提案したアーキテクチャ記述法に基づいて, アーキテクチャの自動抽出技術を構築することを試みたい. また, 本手法の一部を自動化することを検討していきたい.

参考文献

[1] Ivar Booch, Grady Rumbaugh, James Jacobson, The Unified Software Development Process, Addison Wesley (1999).
[2] Anthony J. Lattanze, 橘高 陸夫 訳, アーキテクチャ中心設計手法, 翔泳社 (2011).
[3] 手塚 裕輔, 新田 直也：リアルタイムアプリケーションの開発におけるフレームワークの選択支援手法, 情報処理学会研究報告, 2010-SE-170(3) (2010).
[4] Naoya Nitta, Kume Izuru, Takemura Yasuhiro : A Method for Early Detection of Mismatches between Framework Architecture and Execution Scenarios, In *Proc. of the 20th Asia-Pacific Software Engineering Conference (APSEC)*, (2013), to appear.
[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal, Pattern-oriented software architecture, a system of patterns, John Wiley & Sons (1996).
[6] 新田 直也, 久野 剛司, 久米 出, 武村 泰宏 : 3D ゲームエンジン Radish の開発とそのアーキテクチャ比較への応用, 日本デジタルゲーム学会, デジタルゲーム学研究, Vol. 4, No. 1, pp.1-12 (2010).