第 4 章

オブジェクトの進化モデルの構築

4.1 概要

漸進的な開発では，新たに要求される仕様をシステムに取込みながら，クラスやシステムの拡張性を損なわないように注意しなければならない．本研究では，オブジェクトが予測したとおりの進化の過程を経ているか否かを定量的に評価する手段について検討を行った．オブジェクトの進化過程を評価するために，本章ではオブジェクトの種とその進化過程を定量的に違いを示すとともに，オブジェクトの進化過程をソフトウェアアーキテクチャの評価に適用する可能性について議論する．

MVC(Model-View-Controller) アーキテクチャは GUI(Graphical User Interface) を扱うオブジェクト指向システムで一般に導入されているソフトウェアアーキテクチャである．そこで，オブジェクトを種に分類するにあたり，MVC アーキテクチャに基づいたクラスの分類を考えた．本章では，まず第 4.2節でクラスを種に分類し，種と種の環境との関係について議論する．第 4.3節では，アーキテクチャから予想されるオブジェクトの種の進化過程と 3 つのシステム開発事例で実際に観測された進化過程を比較する．続く節ではオブジェクトの種に着目した進化モデルについて議論するとともに，オブジェクトの進化過程をソフトウェアアーキテクチャの評価に適用する可能性についても議論する．

4.2 オブジェクトの種と環境

4.2.1 オブジェクトの種の定義

GUI (Graphical User Interface) を用いるオブジェクト指向システムには，MVC (Model-View-Controller) と呼ばれる階層構造を持ったソフトウェアアーキテクチャを適用することが多い [26]．MVC のうち，View オブジェクトとは，ディスプレイの表示を制御するオブジェクトであり，Controller オブジェクトとは，マウスやキーボード操作を監視するオブジェクトである．また，利用者が Controller を介して操作するオブジェクトは Model オブジェクトと呼ばれる．利用者の操作結果は，View オブジェクトが Model オブジェクトを参照した内容としてディスプレイに表示される．MVC アーキテクチャは，MacAPP，MFC，ET++といった様々なプログラミング言語でも導入され，それぞれの言語で構築されたフレームワークが開発者に提供されている [34] ．このアーキテクチャの目的は，全体の要求変更の 40%に及ぶと言われている利用者インタフェースに関わる変更を [35]，View と Controller といったオブジェクトで吸収し，Model オブジェクトに波及させない構造をシステムに持たせることである [18].

MVC アーキテクチャのような階層構造は，分析工程のモデルにも導入されている．I. Jacobson が提唱する方法論では，オブジェクトを実体オブジェクト，インタフェースオブジェクト，制御オブジェクトの 3 種類に分類する [29]. これらのオブジェクトは，それぞれ Model，View，そして Controller に対応させることができる．

このように，MVC アーキテクチャはオブジェクト指向システム開発において重要なアーキテクチャとなっている．そこで，オブジェクトの種に基づいた進化過程を捉えるにあたり，オブジェクトの分類基準に MVC アーキテクチャを採用することにした．ただし，MVC のうち，View と Controller の関連は強く，いずれも利用者との相互作用が多い点という共通点があるため，これらをまとめてひとつの種と考える．また，汎用クラスの進化についても議論する必要があると考え，本研究では，MVC に基づいた 2 種類のオブジェクトの種にクラスライブラリの種を加えた次の 3 種のオブジェクトについて，その進化過程を議論する．

- 境界領域種：View+Controller クラス群

- 問題領域種：Model クラス群

- 共有領域種：システムで再利用されたクラスライブラリ群

4.2.2 種と環境の関係

3 種類のオブジェクトの種と，オブジェクトの進化に影響を与える 2 つの環境との関係を，進化環境図として図 4.1に示した．図の横軸は利用者環境からの距離を表し，縦軸は技術的環境からの距離を表す．左へ行くほど利用者環境から遠くなり，右へ行くほど利用者環境に近くなる．利用者環境からの距離は，利用者環境変化の影響の受けやすさを概念的に示したもので定量化されてはいない．縦軸は，上へ行くほど技術的環境から遠くなり，下へ行くほど技術的環境に近くなることを表す．技術的環境からの距離は，開発者の設計意図の変化による変更の対象となりやすいことを意味する．したがって，直観的に，境界領域種は利用者環境の変化の影響を受けやすいと考えられるから図中の右下へ，共有領域種は特定の利用者の要求変更の影響を受けることはないので，図中の左上へ位置付けることができる．問題領域種は，MVC アーキテクチャでもアプリケーション依存のオブジェクトとアプリケーション非依存のオブジェクトが混在しているため，個々のオブジェクトの性質によって利用者環境からの距離は異なると予想される．

各環境からの距離は，種に属するオブジェクトの進化の速度にも影響を与える．境界領域種は利用者要求の変更に対応するために早く進化するが，共有領域種は，環境変化の影響を受けないため，ほとんど進化しないか，あるいは緩い進化を起こすだけであろう．問題領域種は，技術的環境の変化，すなわち，開発者が問題領域に対する理解を深めることによって，次第に進化の速度が遅くなると予測される．以上の予測の妥当性を，本論文で取り上げてきた 3 システムのクラスの進化過程によって検証した．

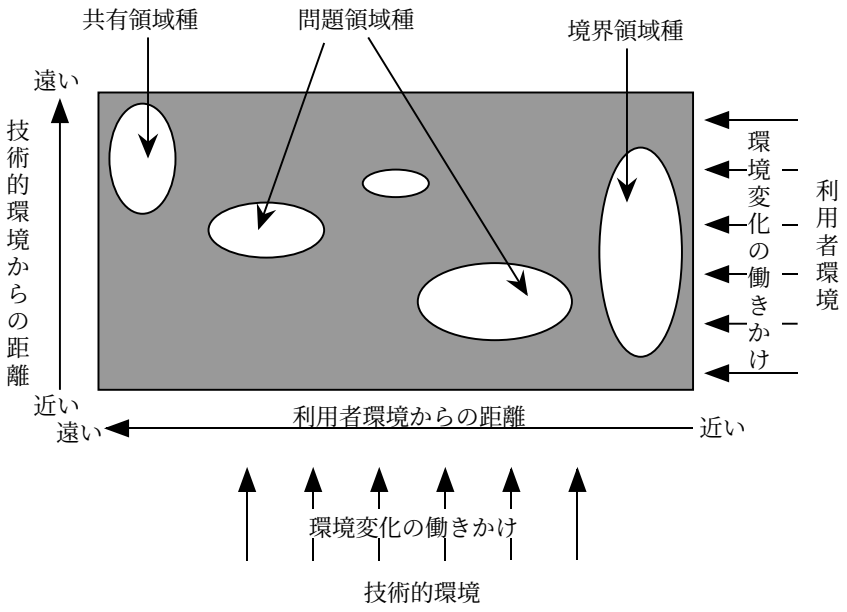

図 4.1: 進化環境図

4.3 種に着目した進化の観測

種別の進化過程を示す前に，3 システムの全クラスの進化過程を図 4.2に示す．縦軸はクラスの進化率を表し，横軸はシステムの開発期間を表す．ここで，進化率とは，第 1章で示した進化率メトリクスを各クラスに適用して求めた値である．図 4.2の左上段は SystemA，右上段は SystemB，そして下段は SystemC に属する全クラスの進化過程を表す．図から，SystemA と SystemB のグラフには類似点が多いが，SystemC は他の 2 システムとは異なる形状をもっていることがわかる．これは，SystemA と SystemB の開発期間中に観測されたクラスの進化率の大きな変動が漸進型開発による利用者環境への適応を表しているのに対して，SystemC のクラスは開発開始から急速に安定状態に到達している．これは，SystemC が落水型開発を採用していたため，開発期間中に利用者環境の変化が発生しなかったためと考えられる．SystemA，SystemB の第 4 版では進化が緩やかになっているが，これは開発の終盤に至って利用者環境の変化が小さくなったためと考えられる．

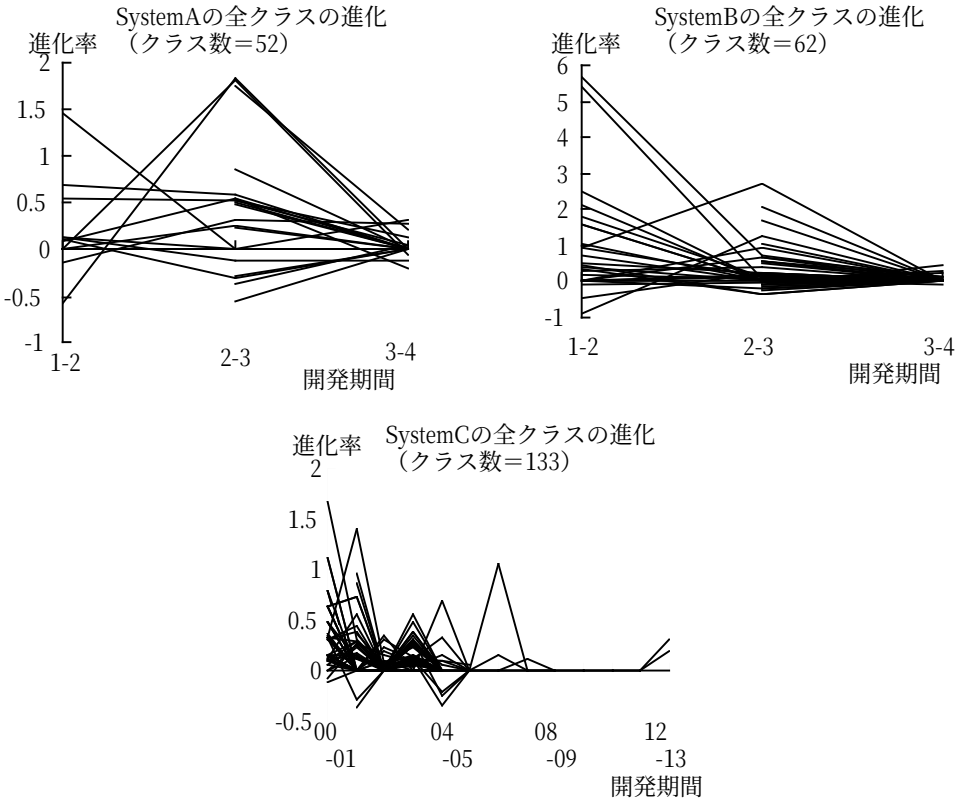

図 4.2: システム全体で観測したオブジェクトの進化過程

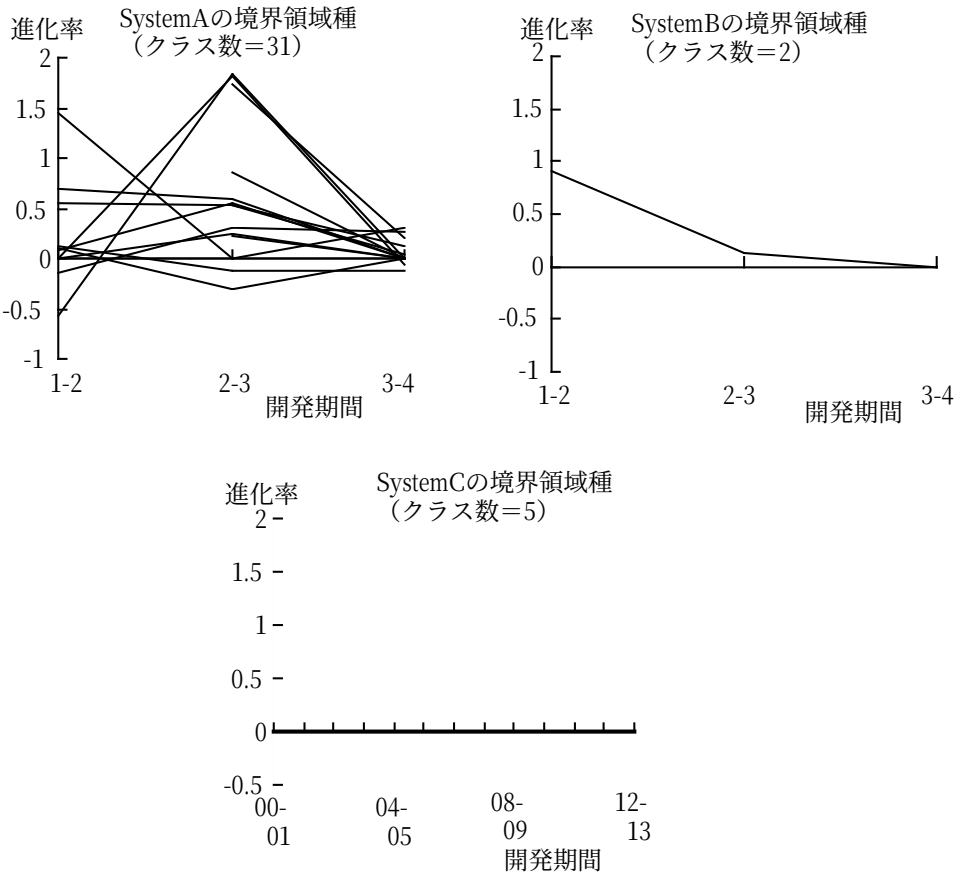

図 4.3: 境界領域種の進化過程

4.3.1 境界領域種の進化過程

3 システムのうち，SystemB と SystemC の境界領域種はコンポーネントを再利用して開発されていた．そのため，SystemB では 2 つのクラスを，また，SystemC では，5 つのクラスを境界領域種に分類できただけである．SystemA では，表示クラス木と編集クラス木に属する最大 31 のクラスを境界領域種に分類できた．

図 4.3の左上段は SystemA，右上段は SystemB，そして下段は SystemC の境界領域種の進化過程を示す．

SystemA の境界領域種には，クラスの進化率に大きな変動を観測できる．また，SystemB では，境界領域種に分類できた 2 クラスのうち 1 クラスに進化

率の変動を観測した．SystemC では，境界領域種の全クラスで進化は観測されなかった．SystemC で観測した現象は，開発期間中に利用者環境の変化が起きていなかったことに起因すると考えられる．

SystemB の第 2 版から第 3 版にかけた開発期間で，全要求変更の 70%が利用者インタフェースの変更要求であったにもかかわらず，SystemA で観測された進化率の変動が観測されていない．境界領域種に変化がないという点では，SystemC と共通している．すなわち，SystemB の境界領域種は，利用者環境の変化の影響を受けていなかったと結論づけることができる．SystemB の利用者インタフェースの開発には，コンポーネントウェアが適用されていたことから，70%の利用者インタフェースの要求変更は，コンポーネントウェアで構築された部分で吸収されたと解釈できる．

コンポーネントウェアをシステム開発に導入することによって，変化が大きい利用者インタフェースは，単純なボタンやスクロールバー，ウィンドウなどのコンポーネントを組み合わせて短期間で構築できるようになった．実際にコンポーネントを組み合わせて構築されたソフトウェアの保守性は高くないが，開発期間を大幅に短縮できるという点で，作り捨て開発に向いている．これは，従来のシステム開発ではシステムの寿命を長くさせることが重要と考えられた時代から，早期開発から作り捨てという利用者環境の変化に合わせた新たなライフサイクルが生まれていることを意味していると考えられる．したがって，SystemB で観測された境界領域種は，コンポーネントウェアを導入することによって，利用者環境の変化をシステム内部に及ぼさないことに成功した例と見ることもできる．

開発者へのインタビューによって，図 4.3に示された SystemB のクラスはコンポーネントを生成するクラスの成長過程を表していることがわかった．しかし，このクラスがコンポーネント用の再利用可能な部品となるまでの進化過程を知るには，さらに長期に渡る観測と他のシステムでの利用状況の観測が必要である．

4.3.2 問題領域種の進化過程

それぞれのシステムで問題領域種に分類できるクラスは以下のクラス木に属するオブジェクトである．

- SystemA
  - 計算クラス木：最大 16 クラス
  - 物理量クラス木：5 クラス
- SystemB
  - 仲介クラス木：最大 7 クラス
  - 一覧表操作クラス木：最大 7 クラス
  - 永続化クラス木：最大 11 クラス
  - 表抽出クラス木：最大 7 クラス
  - 上記クラス木から独立したクラス：最大 13 クラス
- SystemC
  - 証券領域クラス木：最大 68 クラス
  - 仲介クラス木：最大 25 クラス
  - 定数表クラス木：最大 8 クラス

図 4.4に SystemA の問題領域種の進化過程を示し，図 4.5に SystemB の問題領域種の進化過程を示す．それぞれ縦軸はクラスの進化率を表し，横軸はシステムの開発期間を表す．

SystemA の計算クラス木と，SystemB のクラス木から独立したクラスを除くクラスは，いずれも開発が進むにしたがって徐々に進化の速度が緩やかになっている．SystemB の開発期間第 2 版から第 3 版にかけて，一覧表操作クラス木で，大きな進化率の変化を示しているクラスは，このクラス木に属するすべてのクラスのスーパークラスである．図 4.5では，すべてのサブクラスに共通な進化をスーパークラスが集約することで，サブクラスの進化を緩やかにするという継承の効果を観測できる．

SystemB のクラス木から独立したクラスの進化過程は，SystemA の境界領域種に似た進化過程を示しているが，開発者へのインタビューから，これらのクラスの進化が，偶然的な利用者要求の変更によるものではなく，開発者が，

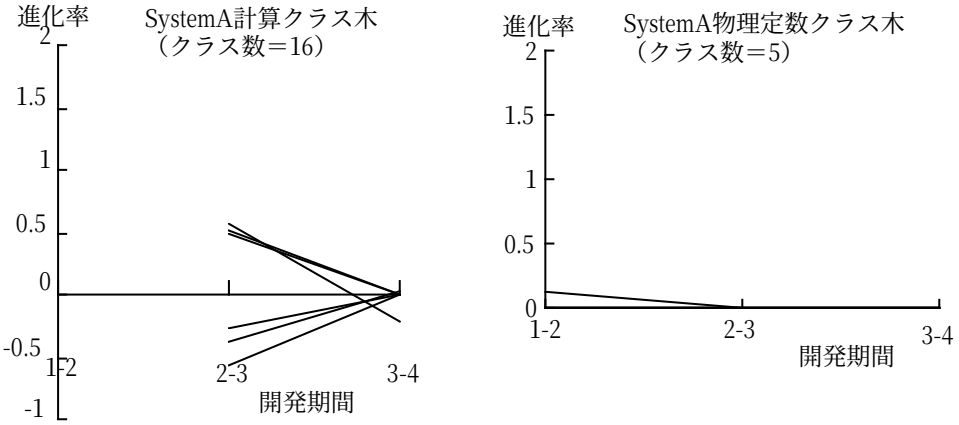

図 4.4: SystemA の問題領域種の進化過程

将来のシステム開発に向けて再利用可能な共有領域種として扱えるようにクラスの分割や継承構造の変更を行った技術的要因によるものであることが明らかになった．このグループに属しているクラスの進化過程の詳細は，開発者の設計意図が進化環境図上でどのように働くかという観点から，後で再び触れることにする．

4.3.3 共有領域種の進化過程

SystemB では，日付け管理を行う 2 クラスと検索アルゴリズムを保持する 11 クラスを共有領域種に分類できた．SystemC の共有領域種は，2 クラスで，これらのクラスは SystemB で再利用された日付け管理を行うクラスと同じクラスである．SystemA には，共有領域種に分類できるクラスはなかった．

図 4.6に共有領域種の進化過程を示す．縦軸は進化率を表し，横軸はシステムの開発期間を表す．図から，いずれのクラスにも開発初期に小さな進化が起き，その後は安定していることがわかる．

この進化のパターンは，SystemC の境界領域種で観測した利用者環境の変化が発生していない場合のオブジェクトの進化パターンと類似している．すなわち，これらのクラスは利用者環境や技術的環境の変化の影響をほとんど受けていないと考えることができる．

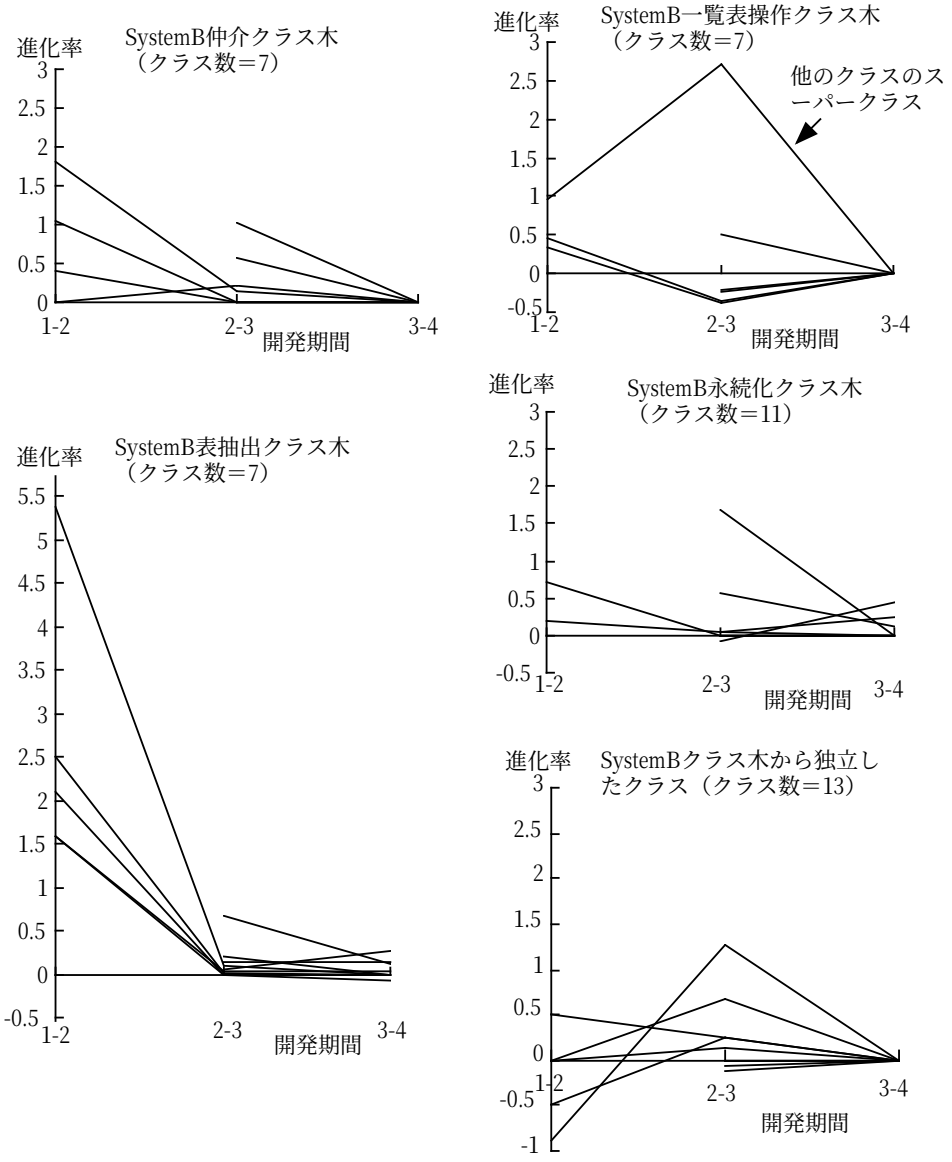

図 4.5: SystemB の問題領域種の進化過程

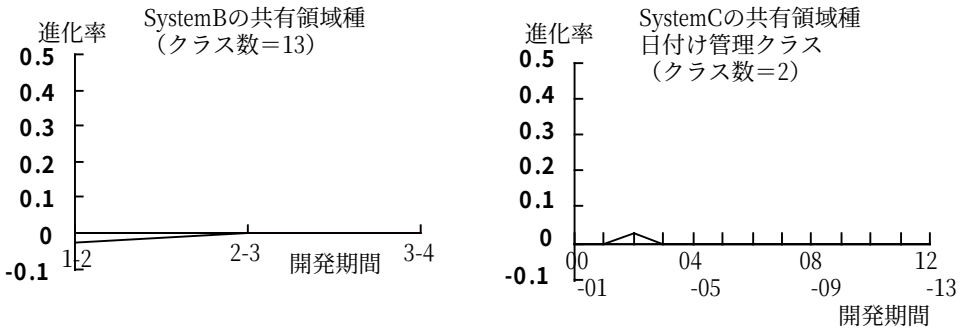

図 4.6: 共有領域種の進化過程

4.3.4 種による進化の違い

以上の観測結果から，多くのオブジェクトは，アーキテクチャから予測された進化パターンを経ることが確認できた．しかし，予測されたとおり，境界領域種には開発期間中に大きな進化率を示すクラスがある一方で，利用者環境の変化が及ばず，その結果，進化しないクラスがあることも確認した．

また，問題領域種は，開発が進むにつれて進化の速度は緩やかとなる傾向があり，この進化の過程で，開発者は問題領域種を安定させるために継承構造の変更や分割を行っていることが明らかとなった．このような進化のパターンは，技術的環境へ適応した結果，オブジェクトが利用者環境から徐々に離れた位置へ移動する過程を表していると理解することができる．そのような進化の過程で，開発者は共有領域種として取り扱いが可能なオブジェクトを発生しようとしていることも確認した．

共有領域種は，新しいシステムに再利用された当初，個々の環境へ適応するために微少の進化が起こるが，その後は安定し，利用者環境や技術的環境の変化の影響を受けることがないことを確認した．

個々の種に属するオブジェクトについて，その進化過程と設計意図とを関連づけ，進化の特徴を詳細にみていくことにする．

SystemA における種の進化

SystemA の計算クラス木に属するオブジェクトは，第 1 版では定義されていなかった．計算クラス木は，第 2 版で「熱交換制御のためにいくつかの熱交換アルゴリズムが提供され，シミュレーション設備の種類を増やすように要求された」ため，シミュレーションの複雑化にともなって表示クラスからシミュレーションの計算部分だけを分割して形成されたクラス群である．このような新たな種の出現は，アーキテクチャからは予想できない進化の形態を表す．進化環境図では，右端の境界領域種からより左方に位置する問題領域種が第 2 版で発生したことになる．

また，第 3 版から第 4 版に向かう期間で進化が起きなかったクラスは，境界領域種の場合は 46%にすぎなかったのに対し，問題領域種では 77%を占めていた．これは，境界領域種が問題領域種よりも利用者環境の変化に敏感に適応していた証拠であろう．以上の結果から，SystemA の計算クラス木に属するオブジェクトを進化環境図に位置付けるとすると，境界領域種よりも利用者環境から遠い左側に位置すると考えられる．これはアーキテクチャから予測できた種の位置づけと一致する．

計算クラス木が開発初期に大きな進化率を示していたのに対して，物理量クラス木に属するオブジェクトは，開発の初期に 5 クラスのうちの 1 クラスがメソッド数を 1 だけ増加させただけで，他の進化を観測できなかった．これは，これらのクラスが物理量という不変の値を保持しているため，利用者環境の変化や技術的環境の変化の影響を受けることがなかったためと考えられる．このようなオブジェクトは，アプリケーション依存のオブジェクトというよりは，物理量計算の問題領域に特化したオブジェクトとみなせる．動的メトリクスを適用した計測結果から，物理量クラス木に属するクラスのオブジェクトは，アプリケーションのクラスからメッセージを受けることはあっても，メッセージをアプリケーションクラスに向けて発信することがないことを確認した．この結果は，これらのクラスが単体で再利用可能なことを意味している．したがって，物理量クラス木に属するオブジェクトは，その進化の性質から進化環境図の左上に位置付けることができ，その仕様から，共有領域種としての取り扱いが可能であると解釈できる．

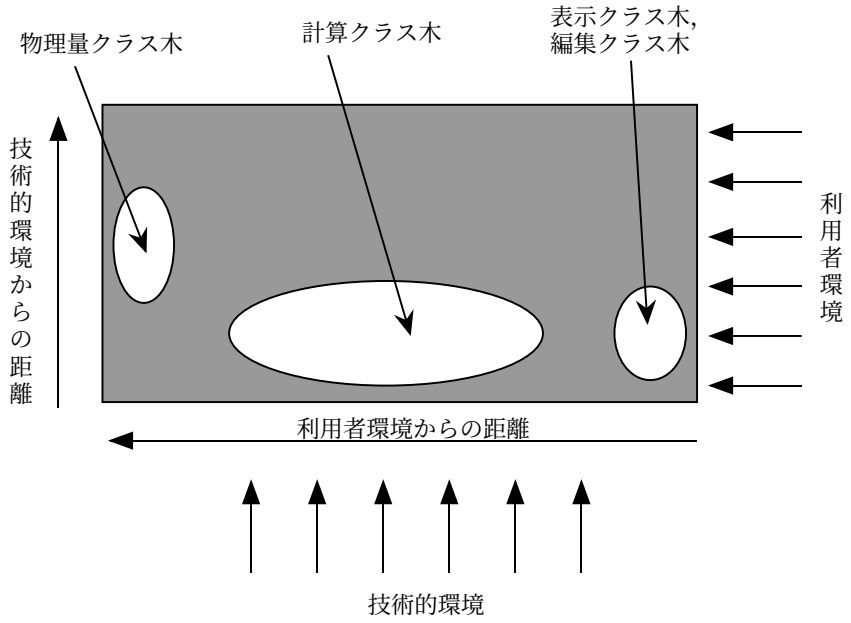

図 4.7: SystemA の種の進化環境図

以上の観測結果から，図 4.7に SystemA のオブジェクトの進化環境図を表す.

SystemB における種の進化

SystemB も，MVC アーキテクチャに基づいて開発されたが，SystemA とは異なり，ほとんどの境界領域種のオブジェクトがコンポーネントウェアで開発され，観測対象となった境界領域種はコンポーネントを生成する 2 クラスであった．2 クラスのうち，進化を示したクラスは開発当初に早い進化を示し，継続的に進化しながら徐々に安定していったが，コンポーネントを生成するクラスは，将来，共有領域種に分類できるクラスである．今後，これらのクラスが他のアプリケーション開発でどのように再利用されるかを観察すれば，共有領域種の形成過程を明らかにすることができるだろう.

SystemB の境界領域種はコンポーネントウェアが利用者環境の変化に適応することで，境界領域種の進化は緩やかであったが，次に注目する問題領域種の一覧表操作クラス木では，第 2 版から第 3 版にかけて要求された一覧表の操

作に関する要求変更にスーパークラスが適応し，サブクラスの進化を緩やかなままであった．この一覧表操作クラス木に属するオブジェクトは，直接 GUI コンポーネントとメッセージの送受信を行うオブジェクトであるため，利用者環境の変化を受けやすい性質を持つ．これらのオブジェクトを進化環境図へ位置付けるならば，右下方の利用者環境に近い位置にスーパークラスを，より左上方にそのサブクラスを置くことができる．

その他の仲介クラス木，永続化クラス木，表抽出クラス木に属するオブジェクトは，発生した当初に早い進化が現われ，急速にその進化の速度を緩やかにしてゆき，やがて進化が停止する．これらの進化のパターンは，開発者が問題領域の理解を深めることで，MVC アーキテクチャが目指す安定した Model オブジェクトの構築が徐々に完成されていったことを表していると思われる．

次に，以上のクラス木から独立したクラスに注目してその進化過程を見ていこう．これらのオブジェクトの進化過程は，SystemA の境界領域種の進化のパターンと似ている．しかし，これらのオブジェクトの進化は，クラスの分割，統合，継承木からの独立といった，アプリケーション非依存の再利用可能なオブジェクトとなるための進化でり，明らかな技術的環境への適応によるものである．開発者へのインタビューでも，これらのクラスが共有領域種として再設計されたクラスであることを確認することができた．この観測結果から，図 4.5のクラス木から独立したクラスで観測される進化過程は，進化環境図の右下方から左上方へ移動する際に起きた進化の過程と解釈できる．

これらの問題領域種に属するクラスの進化過程と，SystemA で観測された境界領域種から分離して定義された問題領域種の発生状況との共通点から，開発者は，利用者環境の変化を受けて，利用者環境の変化の影響を受けて進化するクラスと進化する必要のないクラスとを切り分け，分割や継承構造の変更などの設計変更を進めながら，より安定したクラスを作り出す設計意図を持っていることがわかる．

では，SystemB の共有領域種についてもその進化過程の特徴をみていこう．SystemB の共有領域種には 2 種類のオブジェクトが含まれている．まず第一の共有領域種は日付け管理に関するクラスで，SystemC でも再利用されたクラスである．第二の共有領域種は検索アルゴリズムを保持するクラスで，システムの開発中第 3 版までは再利用されていたが，第 4 版では削除された．ただ

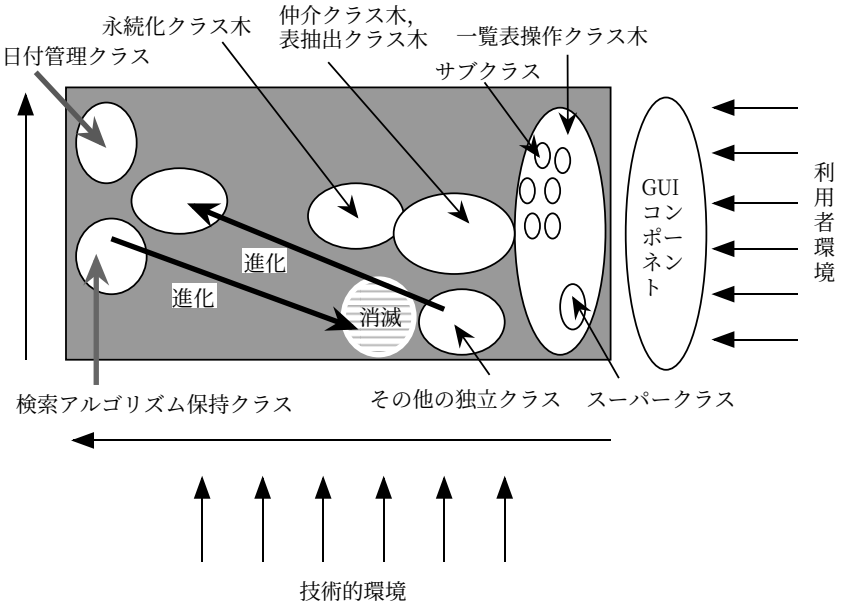

図 4.8: SystemB の種の進化環境図

し，クラスが提供していた一部の役割は他の問題領域種のクラスのメソッドに取り込まれていた．どちらの共有領域種にも再利用を行った初期に，1 メソッド程度の小さな進化が観測されたが，その後の進化は起きていなかった．

第一の共有領域種の進化過程は，共有領域種として予測された進化のパターンを示している．これらのオブジェクトは利用者環境が変化していた期間でも進化していないから，進化環境図の左上に位置付けることができる．しかし，第二の共有領域種の進化過程は，再利用部品を再利用しながらシステムを構築していくボトムアップ型の開発による種の進化過程として解釈できるかもしれない．ここで観測した例は，進化環境図の左方から右方へ移動する際にオブジェクト自身が消滅した例と考えられる．

図 4.8に以上の SystemB の種を進化環境図へ位置付けた図を示す．

SystemC における種の進化

SystemC では，図 4.2に示すように，開発開始後 1 か月でほとんどのオブジェクトの進化が止まっている．

ここで，共有領域種の 2 クラスについて，SystemB で観測された同じクラ

表 4.1: 共有領域種の進化率の分布 (%)

System B: 共有領域種			
	第 1 版-第 2 版	第 2 版-第 3 版	第 3 版-第 4 版
中央値	-0.7	0.0	0.0
最小値	-1.4	0.0	0.0
最大値	0.0	0.0	0.0
クラス数	2	2	3
System C: 共有領域種			
	第 00 版-第 05 版	第 05 版-第 09 版	第 09 版-第 13 版
中央値	0.7	0.0	0.0
最小値	0.0	0.0	0.0
最大値	1.4	0.0	0.0
クラス数	2	2	2

スの進化過程とを比較してみよう．表 4.1に，2 つのシステムで観測された 2 クラスの進化過程を示す．これらのクラスは，同じクラスであるにもかかわらず，一方の SystemB では負の進化率を示し，SysmteC では正の進化率を示している．いずれの変化も微小ではあるが，再利用時に異なった進化が起きたことを表している．クラスを再利用する際，再利用クラスを取り込む側のアーキテクチャと再利用クラスが開発されたアーキテクチャとではアーキテクチャミスマッチと呼ばれる不具合が生じると言われている [17]．これらの微小な進化はアーキテクチャミスマッチを解消するためのものであり，技術的環境への適応と考えられる．

4.4 オブジェクトの進化モデル

ここでは，以上で調査したオブジェクトの種別の進化パターンを整理し，進化モデルを提示することを試みる．以下に観測された各種の進化パターンの特徴をまとめた．

1. 境界領域種の進化パターン

境界領域種に属するオブジェクトの進化は，変化の大きい利用者環境に適応し続けるため，進化率が大きく，継続する．

2. 問題領域種の進化パターン

問題領域種に属するオブジェクトは，利用者環境と技術的環境に適応しながら進化する．また，そのオブジェクトの性質によって，次のような進化パターンに分類することができる．

- 利用者環境の影響を大きく受けるオブジェクト
  境界領域種に似た進化率が大きく継続する進化パターンを持つ．
- アプリケーション固有のオブジェクト
  初期の大きい進化から安定期へ向かう進化パターンを持つ．
- アプリケーション固有のオブジェクトから利用者環境の影響を受けないオブジェクトへ変化するオブジェクト
  進化率の大きい進化後，進化が止まる進化パターンを持つ．進化率の大きい進化は利用者環境の影響を受けないオブジェクトへ変化するための技術的環境への適応による進化である．
- 利用者環境の影響を受けないオブジェクト
  初期の微少な進化があるが，その後に安定する進化パターンを持つ．

3. 共有領域種の進化パターン

初期の微少な進化後，安定期へ至り，進化が止まる進化パターンを持つ．

以上の進化パターンをもとにして，進化環境図を用いたオブジェクトの進化モデルを図 4.9にまとめた．

利用者環境の変化にシステムが適応するためには，進化率が大きくなることを予め予想して開発された部分と進化率が小さくなると予想された部分とを切り分けた構造をソフトウェアのアーキテクチャとして定義する必要がある [4, 53]．MVC アーキテクチャは，階層構造を用いることによって利用者環境の変化を VC のオブジェクトに集中させ，これらのオブジェクトが利用者環境に適応することによって，システムが利用者要求を満足させる構造である．ここ

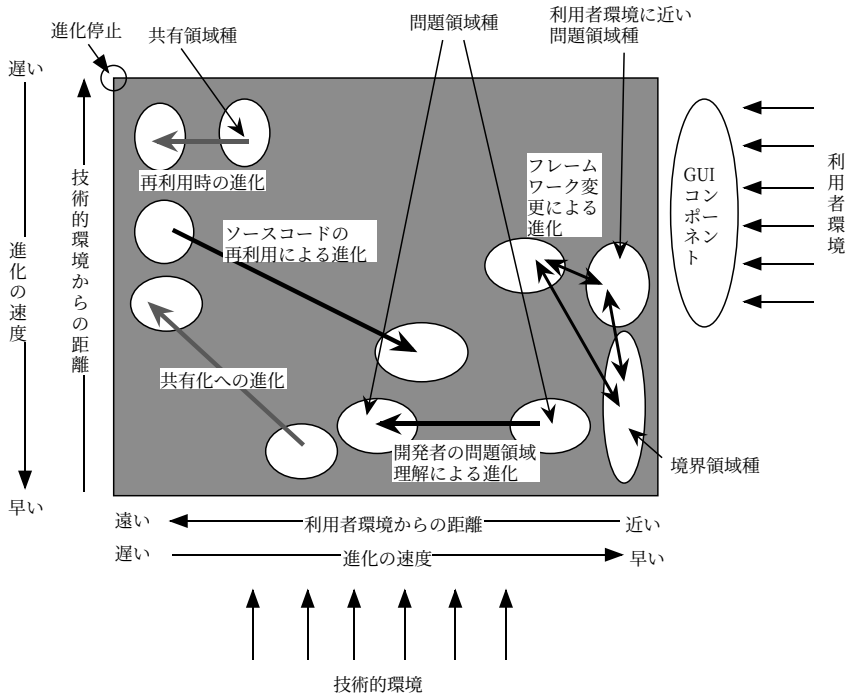

図 4.9: オブジェクトの進化モデル

では境界領域種が進化率の大きい部分となり，問題領域種は進化率の小さい部分となる．もし，MVC アーキテクチャを用いて開発したシステムのオブジェクトが，予想とは異なる進化パターンを持たざるをえないとしたら，それは，開発者の意図的な設計変更の結果であるか，あるいは，そのソフトウェアアーキテクチャがシステムの要求変更の傾向に合致しないことを表していると考えられる．

4.5 考察

本章では，クラスの種による進化過程を調査した結果について議論し，その成果を進化モデルという形でまとめた．さらに多くの事例について調査し，進化パターンと進化環境図中のオブジェクトの位置との関係を検証する必要がある．また，ここで示したオブジェクトの進化モデルを一般化するには，より多くの事例調査が必要であり，ソフトウェアアーキテクチャの妥当性を検証す

るためにも，他のアーキテクチャを導入して開発された事例を調査する必要がある．

本研究で，共有領域種の，他の種とは異なる進化パターンを定量的に明らかにできたことは，新しい再利用クラスの評価基準を提示できたことになる．これまで，クラスの再利用性に関する議論は，その仕様や再利用に対する開発者の積極性といった人間的側面から行われることが多かったが，共有領域種の進化パターンという側面から，客観的にクラスの再利用性を評価する基準を示すことができた．すなわち，あるクラスが再利用に適する度合いは，そのクラスの進化過程が利用者環境と技術的環境の変化の影響を受けず，進化しないパターンであるかを評価することによって定量的に示すことができる．以上の研究成果から，再利用可能クラスとして登録されているクラスには，その仕様や再利用方法だけでなく，進化の実績データも必要であることが明らかとなった．