

LA-004

コンポーネントベース開発のためのアプリケーションフレームワーク Application Framework for Component based Development

井上 晶広[†]蔵川 圭[†]中西 正樹[†]渡邊 勝正[†]

Akihiro INOUE

Kei KURAKAWA

Masaki NAKANISHI

Katsumasa WATANABE

1. はじめに

ソフトウェアはその機能を日々増加させ、それに伴ってその複雑性を増大している。その複雑性に対応するために新しいソフトウェアパラダイムが開発されてきた。そして近年、コンポーネントベース開発がその中心的役割を果たしてきている。

また、新しいソフトウェアの形態として個人に特化したパーソナルソフトウェアを実現することが有効であると考えられる。

ソフトウェアの更なる複雑化に対応し、パーソナルソフトウェアを実現するためにコンポーネントベース開発を基盤とした新しいソフトウェアパラダイムのフレームワークを提唱する。

2. コンポーネントベース開発

ソフトウェアの複雑性に対応するために、オブジェクト指向を基にして考えられたのがコンポーネントベース開発である。

コンポーネントベース開発は、再利用可能なソフトウェア部品の組み合わせによってソフトウェアを製作する開発手法であり、これまでの開発で行われてきたモジュールやオブジェクトの再利用の考え方をさらに発展させたものである。そのために、ソフトウェア部品であるコンポーネントを開発する者と、コンポーネントを組み合わせることによってアプリケーションソフトウェアを開発する者を分離することを可能にした。

しかしながら、この開発手法にも限界が見えつつある。それは、コンポーネントの数が増大するにつれ、その組み合わせは複雑になり多くのコンポーネントから必要なコンポーネントを見つけ出すことが困難になってきていることである。さらに、コンポーネント間の接続にも問題がある。これは、コンポーネントは固定されたインターフェイスを介して接続されるため、インターフェイスが適合しないと接続することができないことによる。

3. アクティブソフトウェア

本稿で提案しているフレームワークは、新しいソフトウェア概念であるアクティブソフトウェアを実現するためのものである。

アクティブソフトウェアはその特徴として

- 自己を認識する
- 自己を調整する
- 自己を訂正する
- 自由に移動する

を有していると定義されている[1]。これによって、ソフトウェア自身がその複雑性を解決し、強健性を保つことができるようになると考えられる。

このアクティブソフトウェアを実現されるための技術的要素として

1. self-adaptive software
2. negotiation
3. tolerant software and interface
4. physically grounded software

が必要である。これらの技術を達成するためのフレームワークを提案する。

4. 関連研究

これまで行われてきたいろいろなアーキテクチャの提案のなかで、本研究と関連があるものを挙げる。

- Feature Driven Development (FDD)[2]
FDD はソフトウェアをユーザの要求に分解し、その要求ごとに設計、構築を繰り返してソフトウェア全体を製作するプロセスである。
- Model Driven Architecture (MDA)
MDA は、モデルを中心として進めていく開発手法であり、その中心となるモデルは2階層からなり、PIM (Platform Independent Model) とPSM (Platform Specific Model) である。PIM はプラットフォームに依存しないモデル、PSM はプラットフォームに特化したモデルである。ここで、プラットフォームとは、CPU や OS、ミドルウェア、言語などを指す。
- Service Oriented Architecture (SOA)
エンタープライズシステム開発において有望視されている開発手法である。サービスの組み合わせによってシステムを構築するという手法であり、サービス間のメッセージングによってシステムを構築する。

5. 提案するフレームワーク

本研究において提案するフレームワークは基本的な考え方として、アプリケーションの概念を捉えなおすことによってソフトウェアの開発手法を見直そうというものである。

その具体的な方針として、現在の主流開発手法であるコンポーネントベース開発において行われている、コンポーネントの発見・接続の自動化を図る。

5.1 コンポーネント

コンポーネントの発見・接続の自動化を考えた場合、システムのトップレベルにおいてソフトウェアが必要とするコンポーネントを発見し、そのコンポーネントを接

[†]奈良先端科学技術大学院大学, Nara Institute of Science and Technology

続させていくという手法が考えられるが、これは、開発プロセスにおけるウォーターフォールモデルに対応するもので、トップレベルにおける設計が非常に困難となってしまう。

このために、コンポーネント自身が接続するコンポーネントを発見・接続する手法が有用であると考えている。これにはソフトウェアの目的を理解しコンポーネントがその目的を達成するために接続する相手を発見するという能力が必要となってくる。

エージェントの分野で利用されている BDI アーキテクチャや、その発展系である INTERRAP アーキテクチャの技術が応用できる可能性があるのではないかと考えられる。これは動的な環境の下で目的を達成するための対応型アーキテクチャであり、自律エージェントが自らが置かれた状況を把握し、目的達成のために行動を起こす必要があれば必ずすべき行為を決定して、これを実行するというものである。

5.2 インターフェイス

図1に従来の手法との比較を示す。従来手法におけるコンポーネントはパズルのようなもので、インターフェイスが合致しなければ組み合わせることができない。これを解決するために、システムにおいてインターフェイスの不一致を吸収し情報のやり取りを円滑に行えるようにすることを提案している。

基盤として、SOAにおけるメッセージング・インフラストラクチャの技術を応用することを検討している。コンポーネントの入出力情報をXMLなどの形式を用いて交換することにより、過大な情報は無視し、不足情報は他のメッセージなどから収集することが可能となるので、柔軟な対応をすることができるようになるという考えである。

5.3 要求仕様

5.1で説明したとおり、コンポーネントにソフトウェアの目的を伝えることで、コンポーネント間の接続を行う。コンポーネント群に対して要求仕様を与え、その要求仕様に従ってソフトウェアの構築を進める。

そのためには、要求仕様とコンポーネントの機能仕様を解釈して、その間の関連を分析しなければコンポーネントの発見・接続を行うことはできない。さらに、対象のドメインによって要求仕様の解釈が異なってしまうという問題もあるので、多くの解決すべき問題が残っている。

5.4 環境や仕様の変更

現在のソフトウェアでは、ソフトウェアの置かれている状況が変化したり、仕様に変更されたりした場合、迅速な対応を取ることができない。

これに対応するために、コンポーネント間の接続を動的に変更できる必要がある。それにはコンポーネントに自らが置かれている状況を把握させる必要がある。

5.5 モデル

コンポーネントが状況を把握するためには、ソフトウェア自身とその周りの環境のモデル化が必要である。このモデルによってソフトウェアが大きく異なってしまうため、モデルの構築が重要なポイントとなる。モデル

対象物	従来の手法		提案手法	
	Component	Software	Component	Software
製作者	Person	Person	Person	Computer
イメージ				

図1: 従来手法との比較

を動的に改良していくことが必要になり、この仕組みを明らかにしていく計画である。

また、5.3で述べたように、対象のドメインによって要求仕様の解釈が異なってしまう。よって、ドメインごとに異なったモデルを用意する必要性も出てくる。

6. 基盤システム

5.で述べたフレームワークを実現するために基盤となるシステムが必要である。このシステムの動作概念は図2であり、その動作を以下に示す。

1. システムは、ユーザからの要求 (Requirement) を解釈し適切なモデルを用いてコンポーネント (Active-Component) に要求を伝える
2. コンポーネントは要求と周りの状況を判断して接続すべきコンポーネントを発見、接続を行う
3. コンポーネントの接続の結果としてアプリケーション (Application) が形成される
4. 要求や仕様の変化を監視し、変更が必要となった際にはコンポーネントの接続関係を変更する

このような方式に基づいてアプリケーションはシステム内部でのコンポーネント間の接続によって形成され、その状況によって絶えず変化するものとして定義される。

このように、アプリケーションの概念を新しく捉えなおすことによってアプリケーションは柔軟で、状況や仕様の変化に対して迅速に対応できるものとなる。

6.1 アプリケーションの製作過程

上述したシステムの動作を開発者の視点に立つと、アプリケーションの製作過程は以下になる。

1. ユーザがシステムに対して要求を出す
2. システムが要求を解釈し、その要求を満たせるようなモデルを構築する
3. 構築したモデルに従って各コンポーネントが接続する
4. コンポーネントが接続しあうことによってアプリケーションが構築される
5. 要求や状況の変更に対応してコンポーネント間の接続が動的に変更される

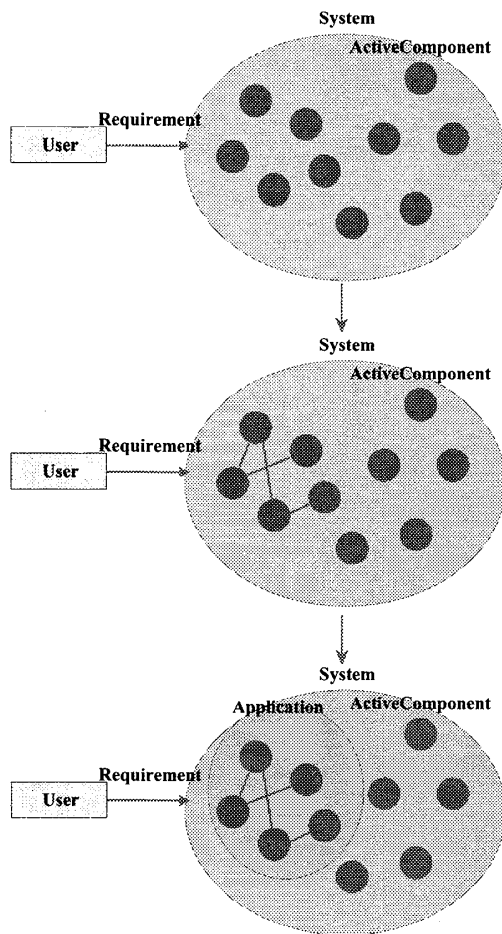


図 2: 動作概念図

このように、ユーザがシステムに対して要求を出すだけでソフトウェアがシステムによって構築されるので、これまでのように人の手によってコンポーネントの接続を行ってアプリケーションを構築する必要性が減少すると考えている。

6.2 パーソナルソフトウェア

上述したように、アプリケーションの製作・変更が容易に行えるようになるため、特定の個人に最適化したアプリケーションをユーザ自身が作り出すことが可能になる。

これによって、ユーザが求める動作を的確に表現したアプリケーションを提供することができるので、生産効率をあげることができると考えられる。

これは、従来の手法によるソフトウェアのパーソナライゼーションはあるアプリケーションへの機能の追加や、ユーザインターフェイスの変更などによって行っていたが、本論文で提示した新しい“アプリケーション”の概念によって当該のアプリケーション自体が変化することによって個人の要求に対応することになる。そのためにより柔軟に要求に対応することができる。

6.3 スケーラビリティ

大規模なアプリケーションを考えた場合、スケーラビリティが大きな問題となってくる。提案するフレームワー

クが完全なものである場合は、どのような大規模なアプリケーションであってもコンポーネントの接続は問題なく行えるが、安定性、効率性を考慮した場合にはある程度の粒度を持ったコンポーネント群をひとつのコンポーネントとして扱える仕組みが必要となる。

しかし、ある程度の粒度を持ったコンポーネントもこれまでのように固定されたものであっては、状況の変化への対応が難しくなる。そこで、要求仕様によってコンポーネントを定義できる仕組みを用意することによってこの問題は解決される。

7. まとめ

ソフトウェアは大規模化・高機能化に伴って、複雑さを増してその強健さを失ってきた。それに対応するための新しいソフトウェアパラダイムとしてコンポーネントベース開発を基盤とした新しいフレームワークを提案した。

本論文ではフレームワークとして設計プロセスの新しいコンセプトの提案を行い、これを実現させるための技術的要素を列挙することによってフレームワークの構築のための礎とした。

今後、今回挙げた技術的要素をひとつずつ解決していく必要がある。特にモデルの作成とその動的な変更が重要なポイントであると考えられる。この部分を中心にして実現方法の研究を進める必要があると考える。

また、このフレームワークに対応するコンポーネントの作成支援の方法、要求仕様の記述方法、モデルの記述方法など確定するための支援環境の構築の研究も必要であると考えられる。

参考文献

- [1] Robert Laddaga, Active Software, In *Self-Adaptive Software, First International Workshop, IWSAS 2000*, LNCS 1936, Springer, 2000, pp. 11-19.
- [2] スティーブン・R・パルマ, ジョン・M・フェルシング, アジャイル開発手法 FDD, ピアソンエデュケーション, 2003.
- [3] Herbert A. Simon, The Science of Artifact Third Edition, MIT Press, 1996.
- [4] 渡邊勝正, 小林郁典, 木村晋二, 堀山貴史, 中西正樹. アクティブソフトウェアの構成と表現について. In 日本ソフトウェア科学会第 18 回大会論文集, 2001, 4D-2.(CD-ROM)
- [5] 井上品広, 中西正樹, 渡邊勝正. アクティブソフトウェア構築のための Java API. In 情報科学技術フォーラム 2003 (FIT2003), 2003, B-016, (CD-ROM)
- [6] W・ブリー, デザインパターンプログラミング, トッパン, 1996.

謝辞

本研究は一部、日本学術振興会科学研究補助金基盤研究(C)(2)15500023、および大川情報通信研究助成の支援による。アクティブソフトウェアについて、日頃議論している研究室の皆さんに感謝します。

