ソフトウェアを正しく作る方法と正しさを保証する方法

(研究課題名：オブジェクト指向分析モデルの形式的構築法と検証法)

「機能と構成」領域　　青木　利晃

要　　旨

現在、ITの進歩により、ソフトウェアの規模、及び、新規ソフトウェアの需要が共に急激に増加し続けています。それらのうち、誤りを含むものは少なくありません。また、ソフトウェアの規模が大きくなると、ソフトウェアを実現しているプログラムではなく、その設計書の役割が重要になってきます。そこで、この研究では、ソフトウェアの設計書の正しさを保証する手法を提案しました。

1．研究のねらい

現在、ITの進歩により、ソフトウェアの規模、及び、新規ソフトウェアの需要が共に急激に増加し続けています。それらのうち、誤りを含むものは少なくありません。最近でも、航空管制システム、銀行システム、携帯電話などの誤りにより、混乱が引き起こされたことがあります。今後、ますますソフトウェアの規模が大きくなり、社会システムの様々な部分が電子化されていきます。これらのことを考えると、ソフトウェアの正しさを保証する手法の確立が急務です。

ソフトウェアの規模が大きくなると、プログラムの直接的な扱いではなく、それを抽象的に記述した分析モデルや設計モデルの重要性が大きくなります。これらのモデルはプログラムを作成する前に作成し、ソフトウェアのアーキテクチャや粒度の大きい単位を記述します。そのため、実装後に誤りが発見されると、コストの高いバックトラックを生じることになってしまいます。幸い、最近では、UML（Unified Modeling Language）など、これらのモデルを詳細に記述するためのダイアグラムが提案されており、それによって記述されたモデルを検証することにより、分析・設計段階で誤りを取り除くことが可能になります。

ソフトウェアの検証のためのアプローチとしては、定理証明手法とモデル検査手法があります。定理証明手法では、ソフトウェアの性質を推論規則などを用いて対話的に証明することにより、正しさを保証します。モデル検査手法では、ソフトウェアを有限状態で表現し、

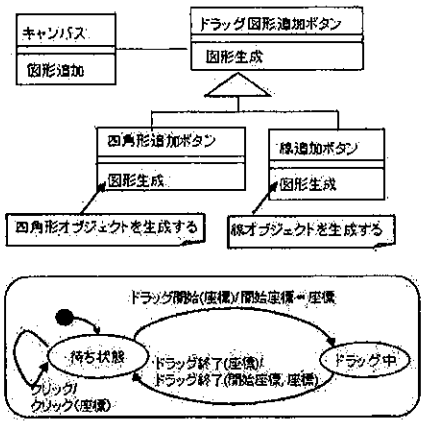
網羅的に探索して自動的に正しさを保証します。これらの手法には長所と短所があります。前者は、高い記述能力と検証能力を持っていますが、対話的に証明を行うためコストが高くなります。一方、後者は、記述能力が低くソフトウェアを有限状態に抽象化する必要がありますが、自動的に検証を行うことができます。

そこで、UMLで記述された分析・設計モデルを、定理証明手法やモデル検査手法を用いて正しさを保証する方法について研究を行っています。さきがけ研究では、定理証明手法の適用に主眼を置いて研究を行いましたが、もう一方のモデル検査手法の適用についても並行して研究を行いました。以下では、定理証明手法による分析・設計モデルの検証法に焦点をあてて説明します。

2．研究方法と成果

2．1　研究課題

現在、ソフトウェアの設計書はUMLと呼ばれるグラフィカルな記法を用いて作成するのが標準になりつつあります。図1の左にUMLによる記述の例を示します。これは、図形エディタの設計書の一部です。このように、四角形や線などの図形、および、自然言語で説明を付加することにより設計書を記述します。一方、定理証明技術では、証明などを支援する定理証明システムというソフトウェアを持ちます。定理証明システムでは、紙などを用いて証明をするかわりに、計算機上で証明を行います。これにより、厳密な証明が可能であり、証明の一部を自動的に行うことができます。図1の右に定理証明システムの例を示します。これは、銀行の入金、出金処理の一部を書いたものです。∀や⇒を用いた論理式で記述し、それらの処理の正しさを推論規則を用いて証明します。このように、UMLによる設計書と



wd(total, book, x) =
  if x ≤ total then (total – x, (– x)::book) else (total, book)
dp(total, book, x) =
  if x ≤ total then (total + x, x::book) else (total, book)
命題:
(0≤0 ∧ sum([])=0)∧
(∀x total book. 0≤ total ∧sum(book) = total ⇒ 0 ≤ total' ∧ sum(book') = total') ∧
(∀x total book. 0≤ total ∧sum(book) = total ⇒ 0 ≤ total'' ∧ sum(book'') = total'')
 where sum ([]) = 0 | sum(x::xs) = x + sum(xs),
       (total',book') = wd(total, book, x) ,
       (total'', book'') = dp(total, book, x)

証明:

$$\frac{\{x\le total\} \vdash x \le total \qquad \dfrac{\{sum(book)=total\} \vdash sum(book) = total}{\{sum(book)=total\} \vdash sum(book) - x = total - x}\ \text{推論規則}}{\dfrac{\{x\le total\} \vdash 0 \le total - x \qquad \{sum(book)=total\} \vdash x \le total \Rightarrow 0 \le sum(-x :: book) = total - x}{\{0\le total\ ,\ sum(book) = total\} \vdash 0 \le total' \wedge sum(book') = total'}}$$

$$\vdash \forall x\ total\ book.\ 0\le total \wedge sum(book) = total \Rightarrow 0 \le total' \wedge sum(book') = total'$$

図1　UMLと定理証明システムの記述

定理証明システムの記述では異なります。そこで、まず、UMLを定理証明システムに取り扱い可能な形式に変換する方法が必要になります。これが、１つ目の研究課題、ソフトウェアの正しさを保証する方法、です。

設計書が定理証明システムの記述に変換できると、望ましい性質が成立するかどうかを定理証明システムを用いて証明することによりソフトウェアの正しさを保証します。図１の右の証明は、証明過程をいくらか省略してあり、厳密に証明を行うと、約10ステップかかります。定理証明システムの自動推論機能を用いればかなりの部分の自動化が行えますが、それでも、適切な定理や推論規則を探すために２、３回の人の介入が必要です。このように、入金、出金処理の一部でもこれだけの証明手続きが必要なため、大規模なソフトウェア開発では膨大な手続きが必要となってしまいます。本来、ソフトウェアの正しさの保証は、ソフトウェアを作成するより本質的に難しい問題であるため、これは当然のことです。そのため、定理証明システムは、軍事、航空、宇宙などの、多くの人手や時間をかけても良い分野で用いられており、家電製品用ソフトウェアなどの一般のソフトウェア開発には使えませんでした。そこで、このようなコストを下げるソフトウェアの作り方を提案することが２つ目の研究課題、ソフトウェアを正しく作る方法、です。

２．２　ソフトウェアの正しさを保証する方法

１つ目の研究課題は、UMLで定義されている概念を定理証明システムで取り扱い可能な形式に変換する方法を提案することにより解決できます。UMLは図形や自然言語を使って記述できるため色々なことを表現できます。しかしながら厳密性に欠けます。定理証明システムでは厳密に定義された述語論理に基づいて記述されたものしか取り扱うことができません。そのため、UMLで書かれた設計書から定理証明システムで取り扱い可能な述語論理に変換する時に、UMLで取り扱える概念を厳密に定義しなおす必要があります。UMLでは、オブジェクト、オブジェクトへの参照、オブジェクトの実体化、継承、状態遷移などの概念があります。本研究では、それらを厳密に定義し、述語論理の定義や式に変換する手法を提案しました。

このようにUMLの概念を厳密に定義すると、検証可能になる一方で、表現能力が下がりすぎてしまう可能性があります。また、述語論理でUMLの概念を実現するために様々な仕組みを作る必要があり、余分なオーバヘッドがかかる可能性もあります。そこで、提案した変換手法で、ソフトウェアの設計で典型的に用いられるデザインパターンのいくらかを記述してみました。図２にその中の１つであるObserverPatternを示します。これは、片方

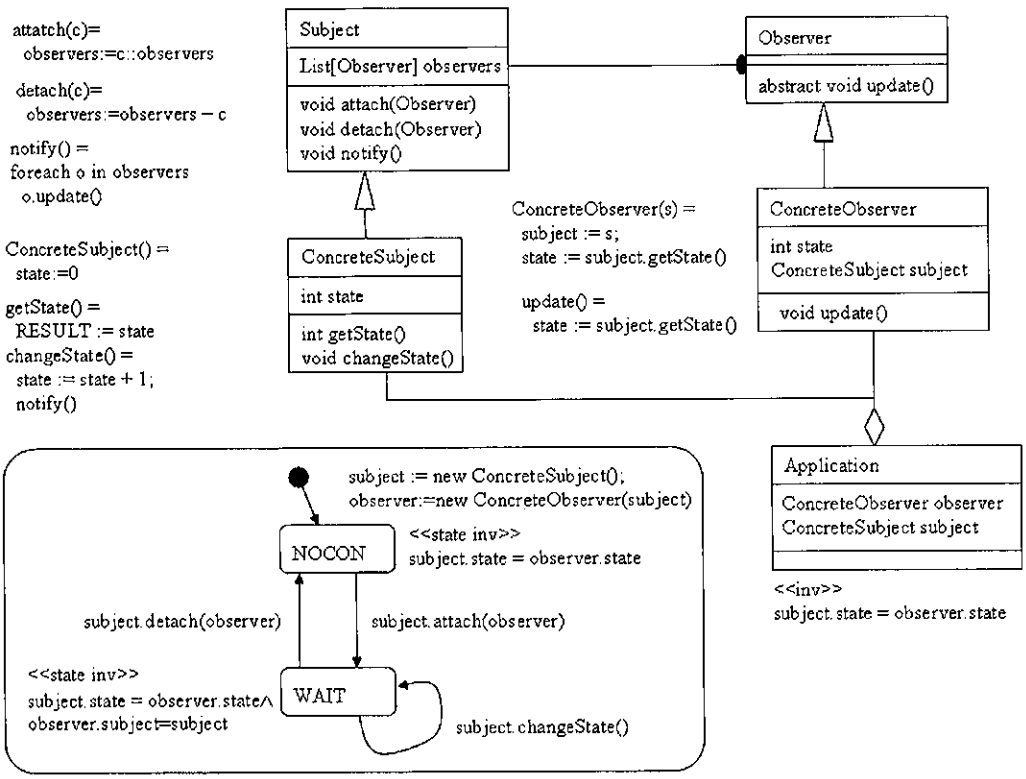


図２　典型的な設計の記述例

（ConcreteSubject）の状態の変化を、他方（ConcreteObserver）に通知することで、両方の状態を一致させるための典型的な設計です。この記述は、定理証明システムHOLで約200行で記述することができます。また、状態が常に反映されるという性質の証明では、４つの補題を証明することにより保証できます。それぞれの補題はすべて１ステップで証明できました。これらのことは、提案した手法が、十分な表現能力を持っていること、および、変換された記述で証明を行う場合のオーバヘッドが小さいことを意味しています。

２．３　ソフトウェアを正しく作る方法

実際のソフトウェアの設計書の検証では、複雑なデータ型や動作を含んでいる場合、正しさ証明が難しくなります。特に、証明過程で帰納法や場合分けが必要な場合、証明ステップや判断が難しくなります。本の貸し出し、返却、登録などを管理するシステムの単純な設計モデルを作成して本が無くならないことを検証しましたが、証明では十数ステップかかりました。ステップ数は多くはないですが、いくらかの場合分けや帰納法の適用が必要であり、適切なそれらを発見するのに時間が多くかかりました。そこで、一般のソフトウェア開発でも適用できるくらいコストを下げるために、証明の再利用と正しい設計書の段階的構築に基づいた手法を提案しました。この手法を、表明主導手法（Assertion Driven Approach）と呼

ぶことにします。証明の再利用では、重複した証明を再度行う必要が無いのでコストを下げることができます。また、段階的構築では、早期に誤りが発見できるため、手戻りコストを下げることができます。

図３に表明主導手法の概要を示します。この手法は２つのフェーズに分かれています。１つ目は、個々のソフトウェア開発ではなく、ソフトウェアの集合を考えるフェーズ、すなわち、領域分析です。一般に、個々のソフトウェアはそれぞれ全く異なるものではなく、共通の性質を持つソフトウェア集合に分けることができます。その集合を領域と呼びます。領域分析では、そのようなソフトウェアに共通に出現する概念や、それらの間で異なる概念を抽出して分析します。表明主導手法では、領域分析のフェーズで、領域に現れる共通な動作を抽出し、それにまつわる性質を証明して蓄積し、領域ライブラリを作成します。この動作と性質は、すでに検証済みの設計書からも抽出することができます。２つ目のフェーズは、個々のソフトウェアを作成するフェーズ、すなわち、通常のソフトウェア開発です。このソフトウェア開発では、領域ライブラリから適切な動作を探し、アレンジします。そして、設計書に段階的に追加していきます。このアレンジと、動作の追加では、動作や設計書の正しさが崩れないようにします。本研究では、このような領域ライブラリと、それを用いた正しい設計書の段階的構築のための基礎理論を提案しました。

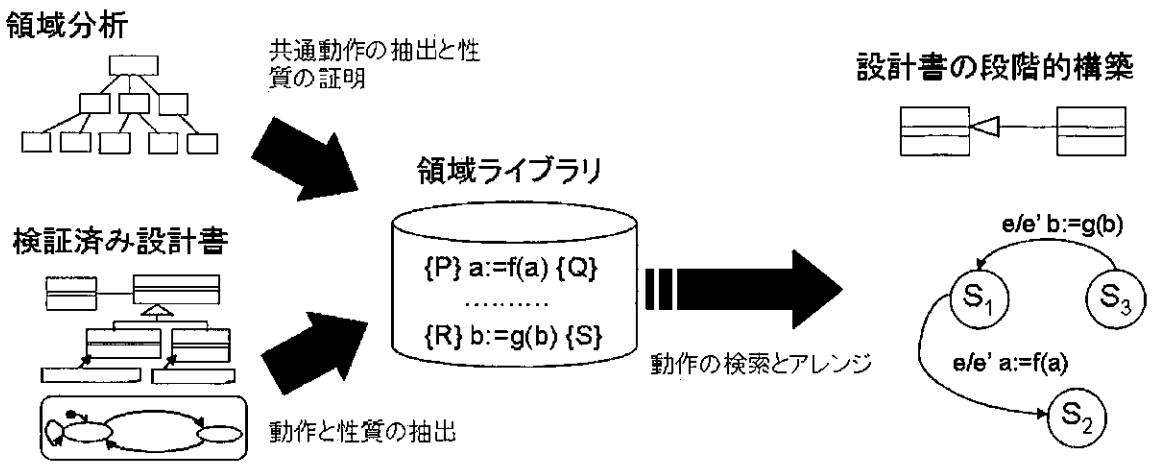


図３　表明主導手法の概要

２．４　検証支援ツール

本研究では、以上で提案した手法を支援するツールF-Developerを開発しています。このツールでは、設計書をプロトタイプ実行するF-Prototyperと、設計書を自動的に定理証明システムHOLで取り扱い可能な形式に変換して証明を支援するF-Verifierから構成されています。現在は、F-Developer ver.0.2 (2003 Dec.版) for Windows, F-Developer ver.0.12 (2001 Jun.版)

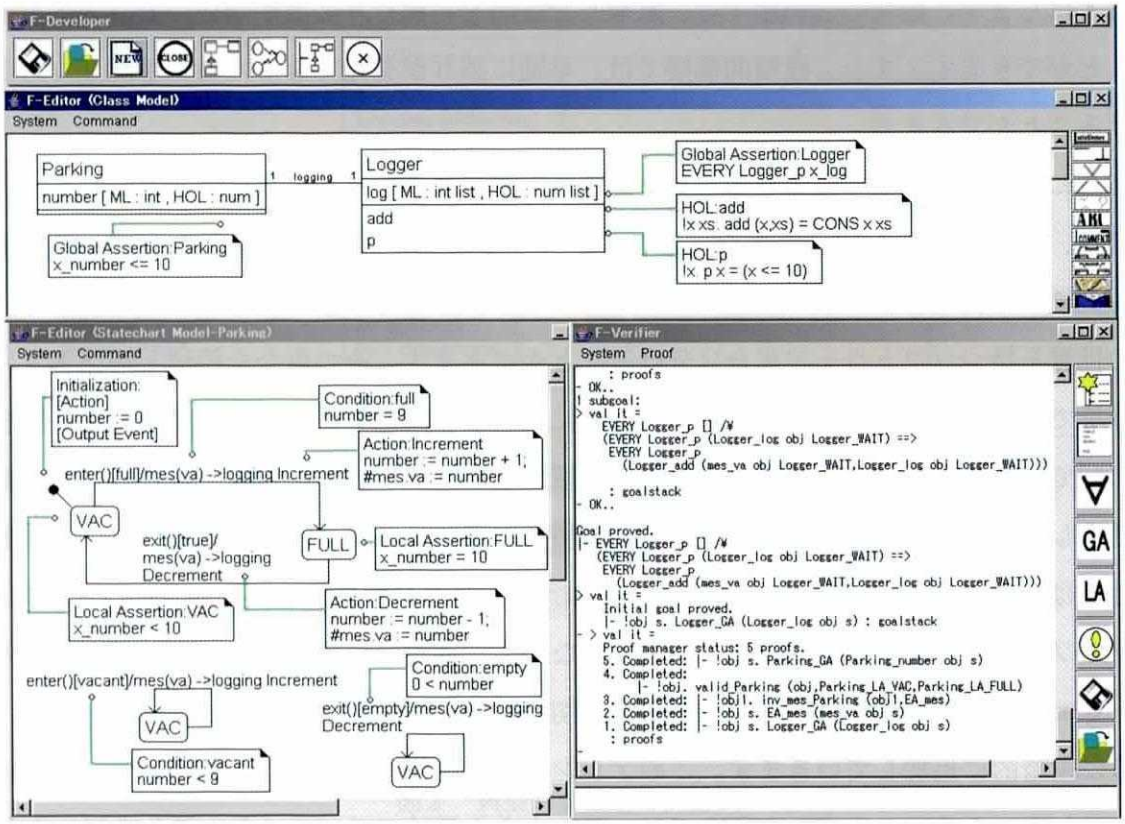


図４　計算機支援環境の画面スナップショット

を公開しています。

３．今後の展望

UMLから述語論理の変換法については、クラス図とステートチャート図、アクション言語、制約言語に関しては、ほぼ提案は完了しました。UMLには、その他にも、アクティビティ図やシーケンス図といった記法もあるので、それらの取り扱いを考えることが今後の課題です。表明主導手法では、領域ライブラリのための枠組は提案しましたが、まだ、中身がありません。今後、動作と性質の蓄積を行う必要があります。これらの蓄積は、検証済みの設計書から抽出できるので、日頃のソフトウェア開発活動で本手法を用いることにより、徐々に大きくなっていくでしょう。そしていずれ、設計したいことすべてをカバーできるようになると、効率的で高信頼性を持つソフトウェア開発が可能になることが期待できます。そのためには、提案手法をすべて扱うことができるF-Developerを完成させることが必要です。一方で、それらが蓄積されていくと、動作と性質の検索やアレンジが困難になります。そこで、それらの自動化手法が必要となるでしょう。これは今後の課題の課題です。

4．成果リスト

論　文

1．青木利晃，片山卓也：オブジェクト指向分析モデルの検証支援環境，第19回　日本ソフトウェア科学会 全国大会論文集（CD-ROM），2002.

2．岡崎光隆，青木利晃，片山卓也：並行オブジェクトモデルから並行スレッドモデルへの変換法，情報処理学会 ソフトウェア工学研究会 研究報告 2003-SE-140, pp.39-46, 2003.

3．立石孝彰，青木利晃，片山卓也：振舞い近似手法を用いたステートチャートに対する不変性の検証，情報処理学会論文誌 Vol.44 No.6, pp.1448-1460, 2003.

4．青木利晃，片山卓也：状態遷移図の段階的構築のための論理的基盤，情報処理学会 ソフトウェア工学研究会 研究報告 2003-SE-143, pp.21-28, 2003.

5．青木利晃，岸　知二，片山卓也：定理証明システムとモデル検査ツールを併用した設計モデルの検証実験，日本ソフトウェア科学会　第１回ディペンダブルソフトウェア研究会，pp.49-58, 2004.

6．青木利晃，片山卓也：オブジェクト指向分析モデルにおけるデータフローの形式化と解析手法，日本ソフトウェア科学会 学会誌 コンピュータソフトウェア，Vol.21, No.4,pp.1-26, July, 2004.

7．青木利晃，岸　知二，片山卓也：センサーのモデル化とモデル検査技術の適用について，組込みソフトウェアシンポジウム2004, pp.118-125, 2004.

8．Takaaki Tateishi, Toshiaki Aoki and Takuya Katayama: Successive Behavior Approximation Method for Verifying Distributed Objects, Third International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'02), pp.439-446, 2002.

9．Mitsutaka Okazaki, Toshiaki Aoki and Takuya Katayama: Extracting threads from concurrent objects for the design of embedded systems, Ninth Asia-Pacific Software Engineering Conference 2002, pp.107-116, 2002.

10. Mitsutaka Okazaki, Toshiaki Aoki and Takuya Katayama: Formalizing sequence diagrams and state machines using Concurrent Regular Expression: Proceedings of 2nd International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools SCESM'03, pp.74-79, 2003.

11. Kenro Yatake, Toshiaki Aoki and Takuya Katayama: Collaboration-based Verification of Object-Oriented models in HOL, Proceedings of the 2nd International Workshop on Verification and Validation of Enterprise Information Systems, VVEIS 2004, pp.78-80, 2004.

12. Toshiaki Aoki and Takuya Katayama: Foundations for Evolutionary Construction of State Transition Models, the Seventh International Workshop on Principles of Software Evolution 2004, pp.143-146, 2004.