解決困難な課題に対するUML によるアプローチ

STRJ経済性検討委員会 フォーラム

林晋　京都大学大学院文学研究科
現代文化学系、情報・史料学専修
平成17年11月10日(木)15：00－17：00
総評会館 4階 401会議室

2005/11/9　1

要旨

- ソフトウェア工学の基本技術となりつつある、モデリング言語、UML について、その基本思想、歴史、長所・短所等について述べる。
- それをもとに、LSI設計・製造への適合具合を考えてみる。

2005/11/9　2

お勧めの教科書

- 何もしらないときに読むと良い本
  - 簡単UML, オージス総研著, SE Shoeisha
- 少しわかってきたら読むと良い本
  - UML Distilled (Third Edition), Martin Fowler, Addison-Wesley.
- UML2.0の解説ページ　by 佐藤(林研究室卒業生)。UML2.0 のポイントがうまくまとまっている。特に古いUMLを知っている人には最適。
  - http://www.smartmodeling.jp/uml/

2005/11/9　3

まず、ソフトウェア工学とは何か？

- ソフトウェアの作成方法の改善を目指す工学
- ソフトウェア科学という言葉との境界は曖昧。結局はスタンスの違いだけ。

2005/11/9　4

何を改善するか？

- 生産効率：　時間、コスト, etc
- 製品の品質: 安全性、パーフォマンス、使いよさ
- 製品の保守・管理: 修正、改変、統合
- 製品の規模・複雑度: 巨大化、チーム
- etc.etc…

2005/11/9　5

ソフトウェア工学の研究

- 言語のパラダイム、言語そのもの
  - 構造化プログラミング、関数型、論理型、オブジェクト指向; Fortran, C, Java,….
- 開発法
  - ソフトウェアシステムの作り方を研究する: waterfall model, spiral model, agile methods
  - 非ソフト系工学でいえば、Taylorism, Fordism, TPS, などのIEにあたる。また、経営学にも似ている。
- 仕様記述、モデリング
- 検証・テスト技法
- ソフトウェア・メトリック
- Etc. etc….

2005/11/9　6

UMLの位置づけ (1)

- UML は「モデリング言語」。モデルを記述する言語。
- 現在のソフトウェア工学では、モデリング・モデルという言葉が氾濫しているが、この概念がわかりにくい。
- いくつかの異なった意味を包含しており、また、明瞭に境界が決まっていない用語。しかし、実際のシステム開発には便利な用語。
- 詳細は、たとえば、
  - I. Sommerville: Software Enginnering, 7th Edition, Addison-Wesley, 2004,
  - M. A. Jackson: Problem frames, Addison-Wesley, 2001.

2005/11/9 7

UMLの位置づけ (2)

- おそらく、プログラミングと仕様記述が、どんな行為かは、すでに十分知られているだろう。UML は良くプログラミング言語か仕様記述言語の一種だと誤解される。しかし、
- UMLはプログラミング言語ではない
  - プログラムはシステムの動作の**HOW**を書いたもの。つまり、アルゴリズムとデータ構造の記述。
  - しかし、UML モデルは全く実行できない場合さえある。
- UMLは仕様記述言語でもない
  - 仕様記述とはシステムの**(FOR) WHAT**を詳細に書いたもの(プログラムのWHATと見ても良い)。簡単に言うと出力と入力の関係を記述したもの(本当は、こんなに単純ではない)。
  - StateMachine や Action Semantics は実行できてしまうので、仕様記述というには、いささか HOW 過ぎる。

2005/11/9 8

UMLの位置づけ (3)

- 80年代ころまでは、この HOW と WHAT の強烈な2分法があった:Hoare 論理, Z notations (林:「プログラム検証論」、共立出版参照)
- しかし、現実の世界では、この2分法は成り立たない。そのため80年代の終わりころから、この2分法の境界が曖昧になってきた: B methods, Refinements, Executable spec., ….
- UML の model とは、この WHAT と HOW の混在したもので、WHATが中心であるもの。あるいは、HOW の要素を取り入れた WHAT (仕様記述)と考えるとよい。

2005/11/9 9

多面的なUML

- UML は多くのシステム記述法を統合したもので、その記述法には、
  - WHAT の記述用のもの(Usecase 図, Sequence 図)
  - HOW よりのもの(動作:Statemachie, Seqeunce図, 構造: Class図,)かなり具体的に動かせるもので、しかし、詳細は抽象化されている
  - 構造を記述するもの(WHAT)の一部。
  - など、色々な要素が入っている。

2005/11/9　10

UML の生い立ち (1)

- Unified Modeling Language
- 80-90年代の OMT法、Booch法などのオブジェクト指向システム開発に使用されていた記法を統合し、明確な文法と意味論を与えたもの。
- Three Amigo と呼ばれる、3人の有名なソフトウェア工学者(G. Booch, I. Jacobson, J. Rambaugh)を中心に、彼らの技術を核にして開発が始まり、現在、この3人の手をほぼ離れ、OMG という非営利団体が開発の責任をもっている。

2005/11/9　11

UML の生い立ち (2)

- 1995: UML 0.8
- 1997: UML 1.0, UML 1.1
- 1998-2002: UML 1.2-1.5
- このころまでは、「寄せ集め」という悪口があったが、それは妥当だった。
- 2003: 大きな変更があり、UML 2.0 が登場。これにより、真に統合された言語になった。

2005/11/9　12

注意

- 初期のUMLは「寄せ集め的」性格と、数学的（理論的）な不正確さが批判された。
- 統合が、ほぼ完成した 2.0 以後では、formal methods の研究者を中心にして、数学的・理論的な欠点・不完全性への批判が多い。
- StateMachine の意味論などでは、確かに、まだ、問題があり、ツール開発の際など、不完全さがプラクティスの障害になることがある。しかし、アカデミズムに多いformal methods 研究者からの批判は、意図的に完全を排することにより、現実性を保つ、という UML の思想を理解できてないためであることも多い。

2005/11/9 13

現在の応用領域

- 主にエンタープライズ系(e-commerce, 銀行のシステム, DB等）のソフトの開発に適しているとされる。
- UMLによるエンタープライズ系のシステム記述のやり方は、わかったので、次は埋め込みソフトという認識が広がっている。
- しかし、現在のUML2.0 は時間概念を持つが、実質は非実時間システム用だと思った方がよい。すくなくとも、ハード・リアルタイム・システムと、UMLの抽象的アプローチは、本質的に相性が悪いところがある（後述）。
- しかし、これは埋め込みソフトとの相性の話であり、半導体設計・生産との相性が悪いことを意味しないはずだ（これも理由は後述）。

2005/11/9 14

階層とUML(1)

- MOF (Meta Object Facility)という規格記述の規格のようなものがあり、UML の標準は、それで記述されている。
- つまり、標準記述の標準、メタ基準言語！
- MOF に代表される、Metamodel, metametamodel の考え方により、UML は拡張を最初から考慮した作りとなっており、改変や拡張に強い。
- UML であることを保ちながら、ユーザーが改造することも可能。半導体への応用を考える場合、この機能の活用が必要となるだろ

2005/11/9 15

階層とUML(2)
Four-layer Metamodel Architecture

- UML は次の4階層を使って定義される
  - meta-metamodel
  - metamodel
  - model
  - user objects

2005/11/9 16

階層とUML(3)
Meta model , Meta-meta model

1. User object がプログラム実行の際のインスタンスに対応
2. Model がプログラマが書く、それぞれのクラスに対応
3. 例えて言うと、Metamodel は Jude (Java で作成されている)でクラスを作る機構を実現するためのJava のクラスに対応。そのクラスは、UML の the classes を表すための Java の a class であることに注意！
4. Meta-metamodel は、3のアナロジーを使おうと Java の文法と意味論に対応

2005/11/9 17

UMLの図　(1)

- 一見、図の集まりと思える UML だが、その中心部は意味論と構文論とそれらを管理するアーキテクチャ。これこそ UMLの真髄ともいえる。
- とはいうものの UML は図中心で使用され、それが実用性の大きな保証に寄与している。文字列の並びより、図の方がわかりやすい！
- 構造をあらわす Structure 図と、動きを表す Behavior 図に大別される。Behavior 図は、さらに Interaction をあらわすものと、そうでないもの(単体の動きを表すもの)に分類される。

2005/11/9 18

UMLの図と、その分類

http://www.smartmodeling.jp/uml/notation.html

- **Structure図**
  - Class図
  - Object図
  - Package図 。
  - Composite Structure図
  - Component図
  - Deployment図
- **Behavior図**
  - Interaction図
    - Sequence図
    - Communication図
    - Interaction Overview図
    - Timing図
  - State Machine図 。
  - Activity図
  - Use Case図

2005/11/9 19

UMLの長所と短所

1. UML は抽象的
2. UML は多面的
3. UML はグラフィカル
4. UML は巨大な階層的標準で記述されている

2005/11/9 20

UMLは抽象的 (1)

- 重要なポイント: UMLはプログラミング言語ではない。モデリング言語である。
- モデリングはプログラミングと仕様記述の中間。デザインに近いのだった。
- そのため、データ構造、アルゴリズムのプラットフォーム(ハード、OS, ライブラリ、言語 etc.)に依存する部分を書くべきではないし書けない。
- これを書かない、つまり、抽象的であることが、現在までの応用では力を発揮している。
- 逆に言うと、そういう抽象をしても構わないシステム、抽象するとわかりやすくなるシステムの記述に使われている。

2005/11/9 21

UMLは抽象的 (2)

- UML の成功は、そういう抽象を行っても大丈夫な応用領域を発見したことと、その領域が十分大きかったことが貢献している。
- しかし、いずれにせよ、抽象的ということは、何を捨象している。
- たとえば、「時間」！
- UML2.0では、Timing diagram が導入されているが、たとえば、これと StateMachine を合わせて使うのは困難。StateMachine には、時間概念が、実質的には無い！
- このため、hard real time system の開発に際しては、timing を UML の言語外のこととして別途扱うしかない。

2005/11/9　22

Timing の問題

- Real time 概念の欠落は、現在のコンピュータシステム、特に、ソフトウェア体系に共通する「欠陥」。
- Real time性は platform に強く依存する。
  - 例：プロセスAが、中間ファイルを使って、プロセスBにデータを渡すとし、それを A が出力を終えた後の一定時間内で完了したいとする。しかし、普通のOS では、file system のキャッシュ、IO のバッファリングの問題があるので、いつ B がデータをファイルから読めるかは判らない。
- 通常のシステムでは、この問題は「捨象」できる。つまり、「現場」の努力にまかせることにしても、何とか「現場」が乗り切れる。
- 組み込みソフトでは、これは無理。Real time OS など、新しいソフトウェア体系の必要性が認識されている。
- UML が、こういう問題を乗り切れるかどうかは、まだ、わからない。

2005/11/9　23

LSI設計の場合は？

- おそらく、Real time 性、timing の問題は、組み込みソフトの場合より少ない。
- 理由：LSIには、OS のようなプラットフォームがない！
- OS などのプラットフォームは、ブラックボックス化されたサービスを提供するためにある（system call）。これが抽象ということで、real time 性を気にしなくて良いならば、これは長所。
- しかし、real time 性が問題ならば、これは制御不能ということに等しいから致命的短所。これが timing の問題を引き起こしていた。

2005/11/9　24

UML と LSI 設計 (1)

- しかし、LSI設計の場合には、もともと、そういうものがないのではないか？あっても、少ないのではないか？
- これが正しければ、SystemC などによる設計の場合と同様に、untimed 記述を UML でやり、timed 記述を後付で行うだけでなく、UML 自体で timed 記述までやることも不可能ではないだろう。

2005/11/9 25

UML と LSI 設計 (2)

- SystemC などの Cベース設計の利点として、HDL などの特殊な言語を知らなくても、C や C++ を知っていれば書けるということがある。UML が普及すれば、C よりさらに良い？
- また、SystemC などは、C++ を拡張しているわけだが、UML には、もともと、そういう拡張のメカニズムがある。つまり、ソフトウェア技術者の常識として UML を知っていれば、LSI 設計用 Profile だけ読めば、LSI 設計用言語を学習できることになる。
- 実際、すでに、そういう取り組み、あるいは、UML を使って SoC 設計をするという取り組みは存在する。
  - ミラノ大などの、SystemC 用 UML2.0 Profile, SoC 用
  - シンガポール大学、富士通研究所, etc. etc.

2005/11/9 26

UML は SystemC より優れているか？

- この問いかけは実は正しくない！ SystemC は、C++ の拡張であり、UMLと比較すべきは C++ ということになる。
- つまり、たとえば「UML は、C++ より LSI 設計、特に SoC 設計用言語のベースとして優れているか？」と聞くべきだ。
- この答えに対する、林の答えは、

■ YES

2005/11/9 27

いくつかの理由

- 未来の UML base SoC 設計用言語が SystemC より優れていると考える根拠は、数多くある。
- たとえば、graphical な記述も、その一つ。
- しかし、最大の利点は、豊富な構造記法と、UML が「ITの標準言語」になりつつあることだろう。
- これらをリストしつつ、未来(近未来か？)の UML ベースの SoC 設計を夢想してみると…

2005/11/9　28

UML の利点：構造化記述メカニズム

- UML2.0 の構造化記述メカニズム
  - UML2.0は、「階層構造の業界標準」ともいえる D'Souza の Catalysis に基づく、構造化システム記述法 composite　structure 図をもっている。
  - Component, StateMachine, Class, Deployment, Package などの各種の図も構造化設計を支援する機能を持つ。
  - これらを使うとカスタムメードの構造化記法体系を定義することさえできる。しかも、それを標準の中で行うための Profile というメカニズムが(一応)ある。

2005/11/9　29

UML の利点: 未来の IT 標準言語

- ITの教科書を見ると、Pascal もどきの言語で、アルゴリズムやプログラムが記述してあることが多い。
- UML 図が、段々と、この役割を果たすようになってきている。典型例は、デザイン・パターンの本。DB, 埋め込みソフト等の教科書でも、段々と、この傾向が強まるだろう。
- つまり、UMLはIT技術者の「共通言語」になる可能性が高い。
  - 新人の学習が容易
  - 他の origanization との情報交換が容易。

2005/11/9　30

UML の利点:経営戦略との連動

- ビジネスモデルを UML のようなモデリング言語で書く「ビジネスモデリング」の試みも始まっている（林は幾分懐疑的）。
- また、要求管理システムとUMLツールの連動も構想されている。（林は、こちらに賛成！）
- これらを使って、たとえば、プロダクトライン開発の経営戦略情報から、ソフト、さらにはハードの設計までを、一体化して、標準的媒体（言語）で記述するということが可能になる可能性がある。（UML は、自然言語の文書なども取り込めるようになっているので、UML の記法で書けないものもOK!）

2005/11/9 31

UML の利点: その他

- UML は graphical な設計ができる。
- Codesign に適する。ソフトウェアがUMLで記述されている機会が増加するはず。
- Profile メカニズムなどにより、標準との整合性を極力保ちつつ、カスタマイズできる。
- さらに、Composite Structure を合わせて使えば、他の organization との、部分的に手の内を隠しながらの共同作業も管理し易い筈。
- 豊富な開発ツール
  - 多種、多様なツールがあり、コードを生成するものも多い。たとえば、BridgePoint を使えば、SystemC のコードを吐く model compiler を比較的簡単に作れるだろう。

2005/11/9 32

結論

- UML は、SystemC などに比べて、特にアルゴリズムやデータ構造の記述のような「局所的問題」が書きやすくなるわけではない。すくなくとも、今の標準のままでは、かえって書きにくいはず。
- しかし、それは解消可能である。
- そして何より重要なのは、これから、ますます重要となると予想される「大域的問題」こそ、UML の最も得意とする分野。
- そして、今のところ、そういう問題に対処する記述法で、広く使われつつあるものは、UML 以外には見当たらない。

2005/11/9 33