Executable UML(実行可能なUML)とは，仕様をグラフィカルに表現する言語である．それは，実行可能なセマンティクスとタイミング規則をUML(統一モデリング言語)表記法のサブセットと組み合わせたものである．つまり，この言語を用いれば，プログラムのように実行できるクラス・モデル，状態モデル，プロシージャ・モデルからなる完全に実行可能なシステム仕様を構築できる．従来の仕様とは異なり，実行可能な仕様で動作させることができ，したがってテスト，デバッグ，パフォーマンスの測定ができる．テストされた仕様(モデル)は，それから目的のソース・コード(ターゲット・コード)に変換することができる．

● リアルタイム・システムのためのコード

ターゲット・コードは，単一プロセッサ，もしくは，分散プロセッサ上で動作する．従来のあいまいな仕様とは異なり，実行可能なモデルでは，詳細なアプリケーションのタイミングや同期的なふるまい，相互に排他的なモード，リソースの競合などの問題を解決することができる．ターゲットとなるマルチタスクとマルチ・プロセッサに対して，最適な速度で効率的なコードを生成するようにパフォーマンスと配置の判断から，変換元であるモデルをカラーリング[注1]することができる．多くのモデル・コンパイラでは，経験豊かなプログラマが変換プロセスを調整し，かつ有意に調節できるように変換のしくみを公開している．これは，ベンダ所有のコンパイル・プロセスに開発チームは左右されないことを意味している．変換のしくみの公開は，生成されたコードがカスタマイズされたハードウェアやソフトウェア環境において効果的に動作することを保証するために必要である．したがって，Executable UMLは，複雑なリアルタイム分散システムや組み込みシステムの仕様を記述する上で適切である(かつ，広範囲に使用できる)．

● 仕様と実装は常に分離している

変換のプロセスでは，ターゲット・コードが生成されたあとでも，元の仕様が完全に変更がないかたちで残る．これは，元の仕様が「満たされている」にもかかわらず，実装を行うためにそれらを歪めてしまう作り込みのプロセスとは明らかに対照的である．変換を行うことにより，元の仕様は常に実装から分離され，実装による影響を受けなくなる．したがって仕様モデルの開発やテストは，実装技術(新しい言語，新しいプロトコル，開発中のハードウェア)が進化している間も，そのまま変わらずに進めることができる．

Executable UMLとは何か

なぜExecutable UMLを使うのか

ソフトウェアを設計する上でもっとも困難なことがコードを書くことではないということは，経験豊かな開発者であればだれでも知っている．コードを書くことは簡単な部分である．

困難な部分，つまり，プロジェクトが中止になったり，スケジュールが6ヶ月から6年に延びたり，あるいはプロジェクトをキャンセルすることになり，それによってマネージャが時期尚早に解雇される原因となる部分とは，

問題を明らかにすること

である．

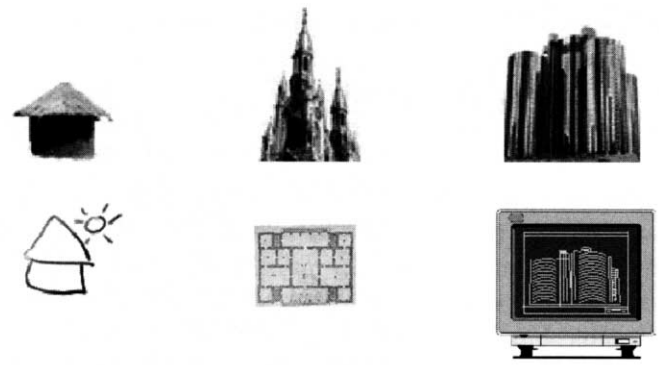
注1：カラーリングについてはp.30に詳細を記す．

● 仕様が重要

これは，とくに大規模で複雑なシステムにおいて真実である．道具小屋の仕様が誤解された程度ならば，その不完全な道具小屋を修正したり，置き換えたりすることは比較的簡単である．しかし，もし夢のマイホームや摩天楼，または極超音速航空機の仕様を誤って規定した場合，その誤解に対するコストは天文学的となる可能性がある．

物理的な建設の世界では，仕様のためのツールは，何百あるいは何千年もの歳月をかけて発展してきた．今日では，青写真やCADシステム，そして数学モデルを使用せずに物理的な資源を費やすことなどありえないだろう．現在の工学分野における多くの驚嘆すべきことは，仕様化技術なしでは不可能である．

青写真と数学モデルが奪われてしまったら，たちまち泥小屋での生活や，つり橋を渡るためにラバを追い立てる生活に戻ってしまうだろう．



進んだ仕様化技術がなければ，現代の物理的建造物を造ることは不可能である．

泥小屋とつり橋の世界は，現在のソフトウェアの現実とさほど違わない(前回デスクトップ・コンピュータ・システム[注2]が壊れたのはいつだろう)．残念なことにソフトウェアの仕様化技術は物理的な世界ほど発展しておらず，その結果，人々は毎日苦労している．未来的なソフトウェアの摩天楼は，先進の仕様化技術なしでは存在不可能だろう．

しかし，すべてがむだなわけではない．ソフトウェアの仕様化技術は，ここ数年で劇的な飛躍をとげている．実行可能で，かつ変換可能な仕様は最新の成果である．どのようにして仕様がこれらの技術的成果を取り入れ発展したのか，その概要をみてみよう．

仕様の発展

● 文字による仕様

当初，仕様はすべて文字によるものだった．見事に装丁された300ページにもおよぶ機能仕様を受けとった(または届けられた)ときのことを思い出していただきたい．そこには巧妙にインデントされた黒丸の付けられた項目がある．またIPアドレスよりも多くの数字とピリオドからなる多段にインデントされた段落の表題がある．

注2：この文は，Linux/UNIXユーザを対象にしていない．好みではない他のデスクトップ・ソフトウェアをその代わりとして考えていただきたい．

この重苦しい仕様を読むよりもさらにまずいことに，それをレビューするために受付係[注3]を除く全社員が参加して10時間(18時間のように思える)の会議を3日間も行っていた．

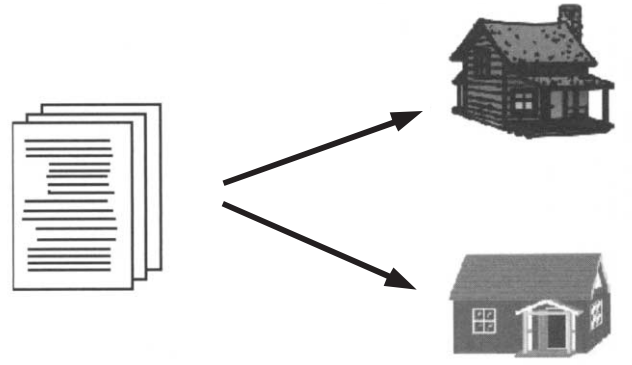
文字の限界は，これらの恐ろしい会議ですぐに露呈する．複雑，もしくは議論があるような問題は，「それは設計フェーズで解決しよう」というマントラ[注4]のもとに，しばしば後回しにされてしまう．もちろんこれらのいくつかは本当に設計の問題だが，考えることが必要となるがために，その多くは現在解決することができない．

逆にコードを書く場合には，書いているコードがうまく動くかを考えているはずである．詳細は整理されていなければならない．そうでなければ，解決しなければならない複雑な問題の解法について議論することができない．他方，文字による仕様を書く場合には，自分に都合のよいように詳細レベルを選択することができる．それは，重要ではあるが細かいことをつくろうには都合がよい．実際，注意深さのレベルは，常に現在のページ数に反比例している．

すべてのレビューが完了し，実際に何かを造ろうとしたときに文字による仕様の本当の問題が具現化する．本当の問題を解決し始めようとすると，仕様において使われている多くの用語がはっきりと定義されていないことに気付くだろう．同じ用語が異なる意味で使用され，異なる用語が同じものを意味する．その問題の解決により深く取り組めば，より多くの矛盾を発見することになるだろう．例えば，40ページ目のきわめて重要な機能が，128ページのきわめて重要な機能を否定していることがある．

このすべての不明瞭さは，開発者に軽蔑され，大部分が無視されるため，オリジナルのドキュメントの完全性は損なわれてしまう．ユーザとアプリケーションの専門家は意見を求められるが，彼らは最終的に造られるモノの「仕様を定義した」と自分自身をごまかしてしまう．実際は，「提案した」，「導いた」という用語のほうが現実に則している．

実際に物語的な文章のように，本質的に1次元の媒体を使用して無数のレイヤやプロトコル，そしてプロセッサの相互作用をともなう複雑なソフトウェア・システムを定義しようとすること自体がおかしな話である．夢のマイホームをテキスト文章に定義して，それを建築業者に手渡しただけで望むものが得られるだろうか．まずそのようなことはありえない．建築業者に渡す前に，それらの青写真をよく見る必要があるだろう．



システムがマイホームと同じように単純だとしても，文字による仕様を信頼する人はいない．

注3：全員をプロセスの一部にしたいので，受付係も招待することにしよう．

注4：*The American Heritage Dictionary*，第3版：マントラ　名詞．祈りの中で繰り返される神聖な言葉の決まり文句，神への祈願としての瞑想や呪文，手品のまじない，または神秘的な潜在能力を含む聖書の一部あるいは音節．