

An Illustrated Example using Enterprise Architect

*Embedded Systems Development using **SysML***

*By Doug Rosenberg
with Sam Mancarella*



目次

はじめに	– Back to the Future	3
第 1 章	– SysML の概要と Enterprise Architect での SysML について	4
第 2 章	– 音楽プレイヤーの要求	14
第 3 章	– 音楽プレイヤーの振る舞い	18
第 4 章	– 音楽プレイヤーの構造	26
第 5 章	– オーディオプレイヤーの制約とパラメトリック	32
第 6 章	– 音楽プレイヤーのハードウェアの実装	39
第 7 章	– 音楽プレイヤーのソフトウェアの実装	49

はじめに - Back to the Future

このドキュメントには、私がこの 25 年の間に ICONIX 社で行ってきたこととは異なることが書いてあります。ICONIX 社での活動は、ほとんどソフトウェアエンジニアリングを対象にしていたのですが、今回は組み込みシステムを対象としています。これは、かつて学んだ電子工学にある、私の「基礎」に戻ったと言えます。

私の学位は電子工学であり、コンピュータサイエンスではありません。私がプログラマーとして、南カリフォルニアにある航空宇宙の研究組織で VLSI の CAD ツールの開発にあたり、その後シリコンバレーでも働いていく中で、ソフトウェアエンジニアリングに没頭していきました。SysML は私にとっては新しい領域ですが、SysML を利用することで役に立つ問題については、本質的によく理解しています。

大学卒業後（1980 年代前半）の最初の 4 つの仕事は、VLSI の設計に関するものでした。そのうちの 3 つ（2 つは TRW で 1 つは Hughes 研究所）は、VHSIC (Very High Speed Integrated Circuits) と呼ばれるプロジェクトでした。ちなみに、VHDL の V は、VHSIC を示しています。TRW での私の仕事はギガヘルツレベルの速度を可能にするための複雑な製造プロセスを対象にした、初期の「設計ルールチェック」ソフトウェアを拡張するものでした。私は、もっとも初期の「回路シミュレータ」の一つである SPICE に関する仕事も行いました。

その後、短い間ですがシリコンバレーにある Calma という会社で「シンボリックレイアウトおよび圧縮」と呼ばれる仕事を行いました。その後 TRW に戻り、「階層レイアウトの検証」と呼ばれるアプリケーションの設計とプログラミングを行いました。このアプリケーションは、完全な IC のレイアウトをサブセル (SysML のブロック) に分解する作業を再帰的に行うものです。そして、入出力のポート (これは SysML のポートの概念に似ています) を決定し、物理的な設計ルールと電気的な接続を確認します。

この間、TRW での私の上司であった Jim Peterson は初期のハードウェア設計言語 (HDL) の一つを開発していました。これは、THDL (TRW Hardware Description Language) と呼ばれていました。THDL 自体は CIF (Caltech Intermediate Format¹) の拡張でした。CIF は Carver Mead の研究グループ内で Jim が Caltech の学生であったときに開発されました。Jim の THDL に関する研究は VHSIC との契約で資金が提供されていました。VHDL のコンセプトのうちのいくつかは THDL から生まれています。

TRW での2度目の仕事の後には、私はカリフォルニアの Malibu にある Hughes 研究所で働きました。ここでは、CAD システムのインターフェースを開発しました。これは、VHSIC Electron Beam Lithography System と呼ばれていました。この開発は大がかりなプロジェクトであり、以前の製造技術よりもはるかに優れたものでした。電子ビームで、シリコンウエハーの上に 0.1 ミクロンの線を描いていました。

このドキュメントについて Sparx Systems から依頼があったときに、Enterprise Architect の SysML の実装についてほぼ全ての責任を持つ Sam Mancarella が、私と同じような考え方・背景を持っていることに気づきました。Sam は、このドキュメントで利用している音楽プレーヤーのサンプルを作成しました。このサンプルは完全であり、また包括的なものになっていましたので、このドキュメントの作成はとても容易でした。このサンプルについての著作権のすべては Sam が所有していることを、ここで改めてはっきりとさせておきたいと思います。そして、私の仕事は、単にこのサンプルについての文章を書くことだけでした。私には過去の電子工学の知識があったので、Sam のサンプルがどれだけ優れたものかをすぐに理解することができました。そして、その知識を活用することで、このサンプルの部品をどうやってまとめあげると良いかということがわかりました。

¹ Introduction to VHDL By R. D. M. Hunter, T. T. Johnson, p.17-18

第 1 章 SysML の概要と Enterprise Architect での SysML について

組込み機器の設計開発のロードマップ

まず、SysML の概要について、私たちが今まで別の場所で紹介している内容とは少し異なる方法で伝えたいと思います。既存の SysML に関する書籍やチュートリアルの中は SysML のダイアグラムの種類や意味の説明であり、SysML の複数のダイアグラムを組み合わせて意味のあるモデルにするための方法は説明されていませんでした。

私たちは、この数年の間に ICONIX プロセスを利用して大きな成功を収めることができます。この ICONIX プロセスは、曖昧ではない設計開発プロセスを定義し、設計開発のプロセスを「ロードマップ」という形で示しています。私たちは、この「プロセスロードマップ」を、ユースケース駆動のソフトウェアの開発・ビジネスモデル・デザイン駆動テスト・アルゴリズム強調のソフトウェア設計の 4 つに対して具体的に定義しました。このドキュメントでは、このような定義を、組込み機器の設計開発、つまりハードウェアとソフトウェアの両方が連携するシステムの設計開発に対する「プロセスロードマップ」を定義していきます。このドキュメントの前半の章で、このロードマップについて説明します。そして、それ以降のそれぞれの章では、ロードマップの内容のそれぞれのアクティビティ（活動）に対して、詳細に説明します。さらに、このロードマップに沿って、Enterprise Architect をどのように利用するかを説明します。この説明では、具体的な例を利用して、図も含めて説明します。

このドキュメントでは SysML 1.1 の全ての図についての利用方法を説明すると同時に、Enterprise Architect が持つ UML からのソースコード生成機能（Verilog や VHDL 等のハードウェア記述言語からのソースコード生成も含まれます）についても説明します。さらに、Enterprise Architect で利用できる SysML のパラメトリック図のシミュレーション機能についても説明します。これらの機能を組み合わせた設計開発については、このドキュメントの後半で説明します。

特に、

- 5 章では、Enterprise Architect のパラメトリック図のシミュレーション機能について説明します。制約（Constraint）モデルからグラフとして動作結果が見えるようになります。
- 6 章では、ハードウェア記述言語のサポートについて説明します。具体的には、Verilog、VHDL および SystemC について、状態マシン図からのソースコード生成について説明します。
- 7 章では、状態マシン図・シーケンス図およびアクティビティ図からの C、C++、C#、Java および VBNet のソースコードを生成する機能について説明します。
-

これらの機能は、それぞれ単独でも利用することは可能です。しかし、システム設計にうまく活用することで、全体の「馬力」を向上させることができます。これらの機能を、「プロセスロードマップ」に組み込んで活用することで、さらに「馬力」を増幅させることができます。

図 1 は、「組込み機器向け ICONIX プロセス」の最上位のロードマップです。

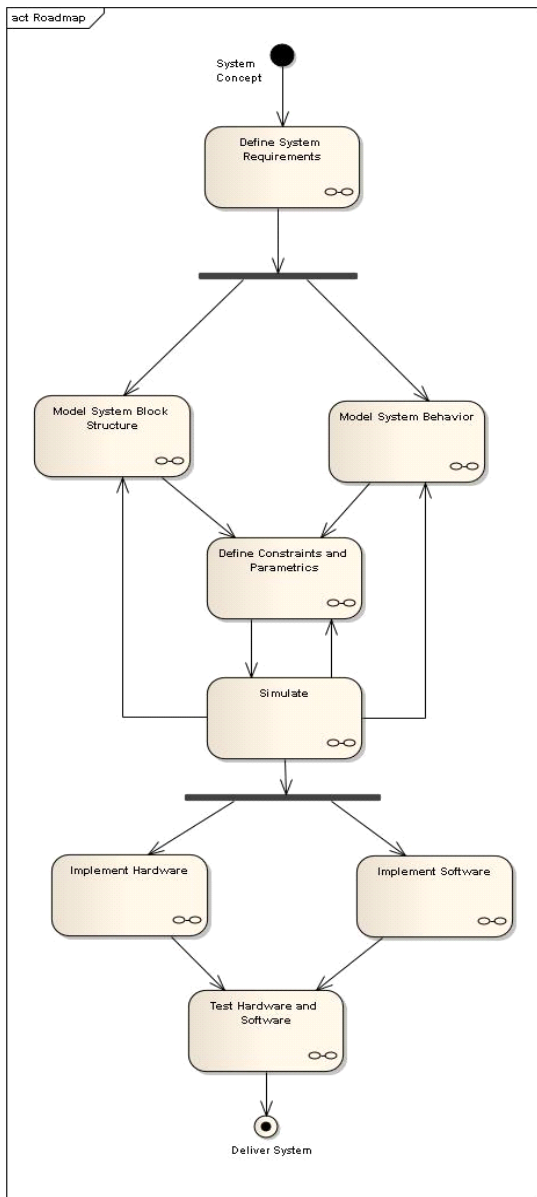


図 1 - 組み込み機器開発向け ICONIX プロセスロードマップ

この図でわかるように、私たちのロードマップは、まずシステム要求を定義するところから始まります。次に、システムの振る舞いやブロック構造のモデリングを行い、それから制約やパラメトリック図を定義します。シミュレーション機能はここで活用します。その後、ハードウェアとソフトウェアの両方を実装します。この章では、この図のアクティビティ（活動）の概要を紹介します。2章から7章までで、それぞれのアクティビティの詳細を、実際の例（音楽プレーヤー）を元に説明します。

要求・構造・振る舞い・パラメトリック - SysML の 4 つの柱

私たちの組み込み機器向け設計開発プロセスロードマップは、一般的に4つのセクションで構成される SysML モデルを構築することで進んでいきます。システムモデル全体を構成する4つのモデル（要求・構造・振る舞い・パラメトリック）は、「SysML の4つの柱（“The Four Pillars of SysML”）」とも呼ばれます。²

² OMG Systems Modeling Language Tutorial, INCOSE 2008

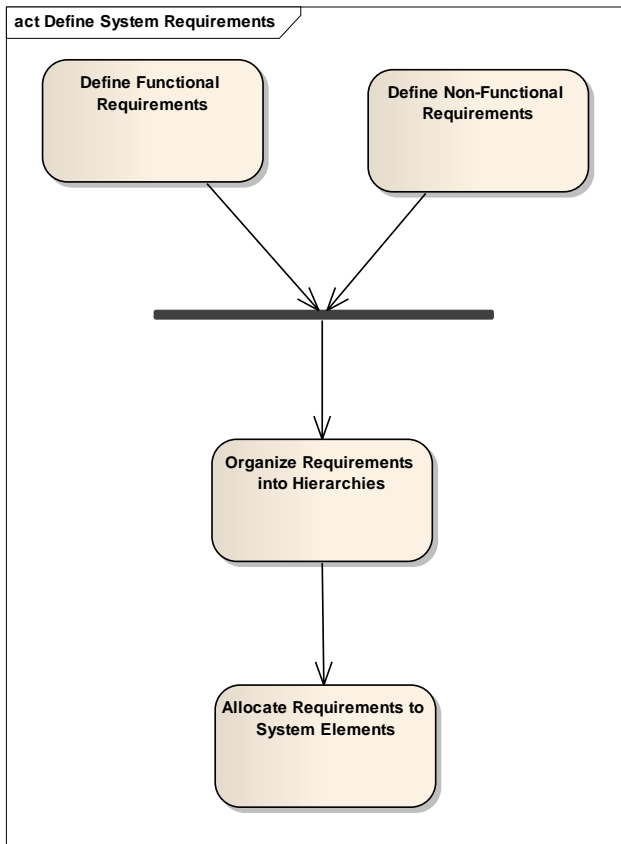


図 2 - ロードマップ:システム要求の定義

要求

ここでの要求は、一般的には機能要求として分類されます。機能要求はシステムで利用できる機能を表現します。非機能要求は、性能・信頼性などの要求です。要求は、要求ダイアグラム上に階層状に分けることが可能です。Enterprise Architect では、定義済みの要求を他の要素に簡単に割り当てることが出来ます。単に要求を、対象の要素に対してドラッグ&ドロップするだけです。要求と要素間の関係をマトリックスで確認・追跡することもできます。

ロードマップにおける要求定義の手順は左の図の通りです。なお、要求をシステム要素に割り当てる作業はモデルの開発を通して行われる作業であり、他の詳細ロードマップのアクティビティの実行中にも頻繁に発生することに注意して下さい。要求定義に関しては、第 2 章でより詳しく説明します。

構造

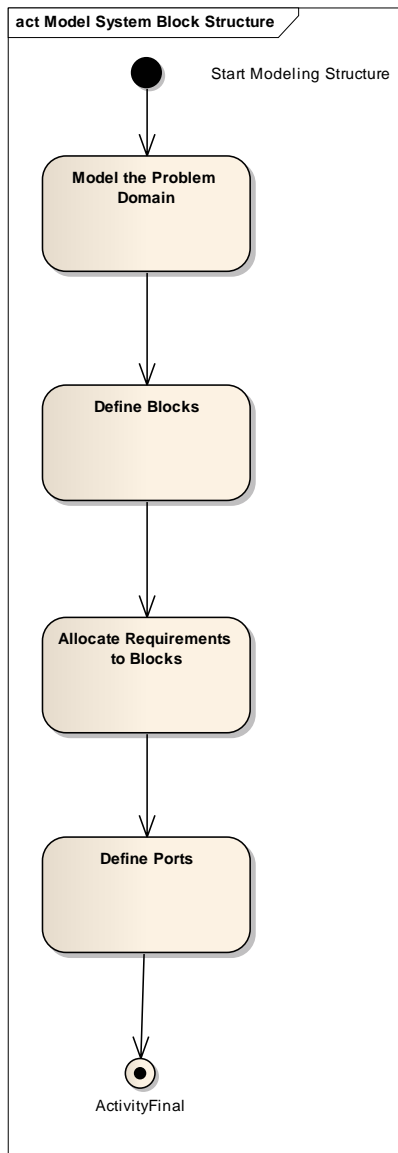


図 3 - ロードマップ:モデルシステムのブロック構造

ブロック (Block) 要素はハードウェア・ソフトウェアを表現するために利用します。また、場合によってはこれら以外の「何か」を表現するために利用することもあります。ブロック定義 (Block definition) 図は、システムの構成を表現するために利用します。内部ブロック (Internal block) 図は、ブロックの内部、つまりパート・ポート・コネクタなどを表現するために利用します。UML と同様に、パッケージはモデルに構造を定義し、階層的に管理するために利用します。

もし、ブロックを電子回路 (ブロックが電子回路であるというのはよくある一つの例です) であると考えた場合には、ポートは回路への入力信号や、回路からの出力信号を定義します。SysML では、入力信号や変換について詳細に定義することができます。Enterprise Architect では、定義した内容を元にシミュレーションを行うことができます。シミュレーションの結果はグラフで表現したり、CSV 形式で出力したりすることができます。この機能の詳細については、第 5 章で説明します。ブロック構造を定義することはパラメトリックを定義したりシミュレーションを実行したりする前に行う必要があります。

左の図は、システムの構造を定義する範囲のプロセスロードマップです。第 3 章では構造をモデリングすることについて、さらに深く説明します。

振る舞い

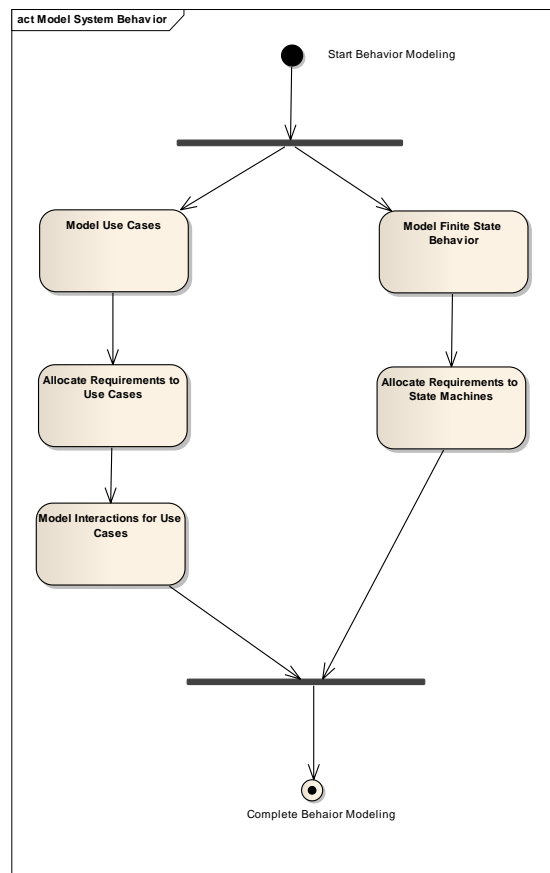


図 4 - ロードマップ:モデルシステムの振る舞い

SysML では、システムの振る舞いを異なる 4 つの視点で分析し、表現します。ユースケース・アクティビティ図・シーケンス図・ステートマシン図です。

私たちのロードマップでは、振る舞いのモデリングを行う場合には、2 つの作業を変更して行います。片方の作業はユースケース³から始まります。ユースケースでは利用者とシステムとの相互作用がどのようになるかを、シナリオという形で表現します。ユースケースは一般的に、対象のシナリオが成功するような「晴れの日」のシナリオと、通常ではない条件・例外・失敗などの場合を示す複数の「雨の日」のシナリオで構成されます。ユースケースは、シーケンス図で詳細を明確にします。

ロードマップのもう片方の作業では、ステートマシン図を利用して、システムの一部について、イベント駆動の状態が定義される振る舞いを定義します。簡単な例として、音楽プレイヤーにおける電源制御回路の状態の振る舞いを考えます。Enterprise Architect の機能を利用すると、ステートマシン図からソースコードを生成することができます。後で紹介するように、このようなステートマシンはソフトウェアか、あるいはハードウェア記述言語 (HDL) で実現します。

要求は、ユースケースと状態の両方に割り当てられます。第 4 章で、この振る舞いのモデリングの詳細について説明します。

³ ユースケースについての詳しい情報は、“Use Case Driven Object Modeling with UML: Theory and Practice” by Doug Rosenberg and Matt Stephens を参照のこと

Enterprise Architect Suite システムエンジニアリング版の追加機能

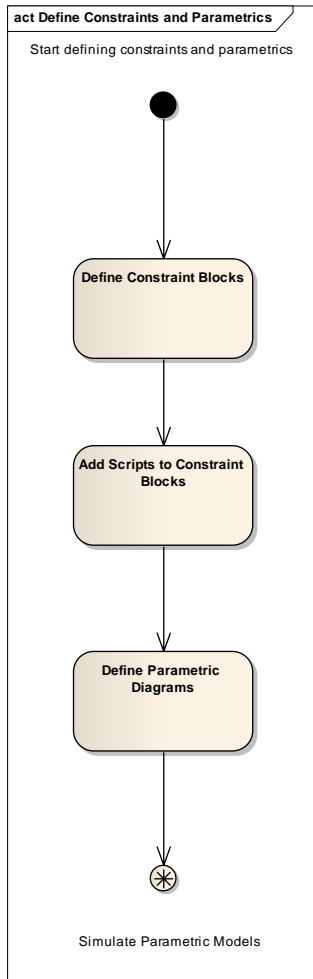


図 5 - ロードマップ:制約とパラメトリックの定義

Enterprise Architect Suite システムエンジニアリング版では、他のエディションでは利用できない独自の機能をいくつか搭載しています。これらの機能は、組み込み機器の設計開発に役立ちます。システムエンジニアリング版では、シミュレーション可能な SysML のパラメトリック図や UML モデルからのソースコードの生成機能（Verilog や VHDL 等のハードウェア記述言語も含む）を搭載しています。そのほか、VisualStudio や Eclipse との連携機能も利用できます。これらの機能を利用することで、UML や SysML と、Visual Studio や Eclipse の開発環境とを密接に連携させることができます。私たちのプロセスロードマップでは、特徴的なこれらの機能についても、開発プロセスに取り入れています。

パラメトリック

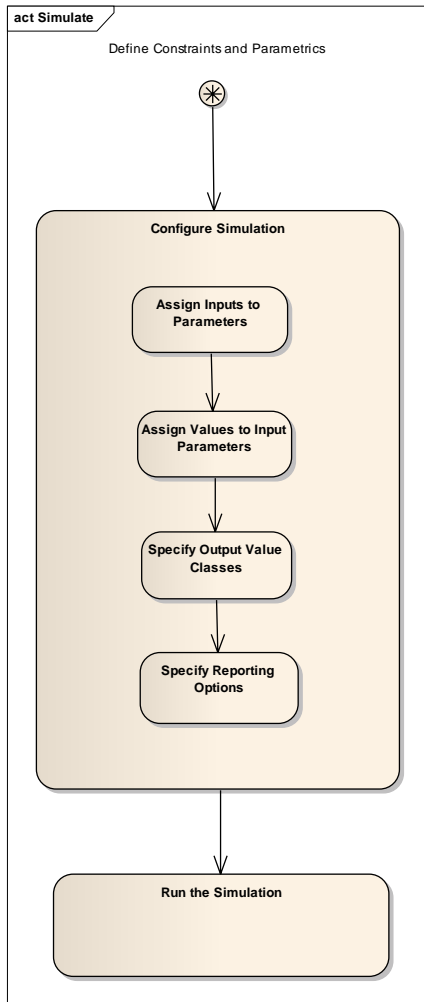


図 6 - ロードマップ:シミュレーション

パラメトリックは、詳細な特性・物理的な法則・制約などをシステムのブロックに対して定義することができます。これにより、システムがどのように振る舞うかをシミュレーションすることが可能になり、技術的なトレードオフを考慮することが可能になります。そして、指定された要求を満たすようにするまでシミュレーションを繰り返すことができます。

私たちのロードマップでは、このパラメトリックの領域で 2 つのアクティビティを定義しています。最初のアクティビティでは制約ブロックとパラメトリック図を定義します。2 番目のアクティビティでは、パラメータに渡す値を指定し、シミュレーションを実行します。

Enterprise Architect のシミュレーション機能を利用することで、今までは外部のシミュレーションツールにモデルの情報を出力する必要がなくなります。この機能は、Sparx Systems の SysML 機能独自の大きな特徴です。

Enterprise Architect のスクリプト機能のサポートやシミュレーション結果のグラフ表示機能を利用することで、モデルのトレードオフを調査・検討し、システム全体の要求に適合するかどうかを効率よく確認することができます。この機能の詳細については、第 5 章で説明します。

ハードウェアの実装

ハードウェア記述言語を利用することにより、ソフトウェアのような表現で電子回路の仕様を定義することができます。

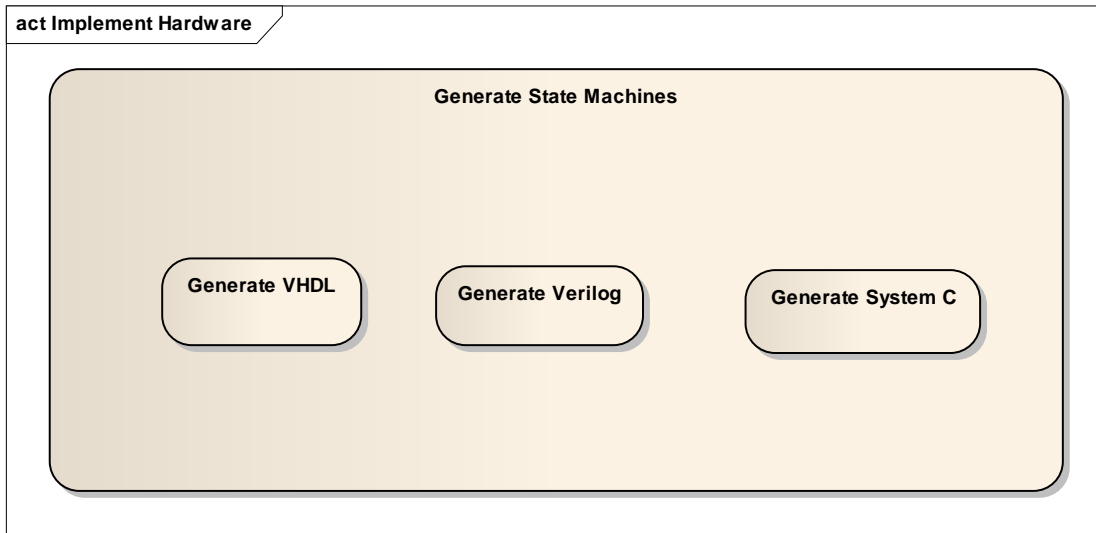


図 7 - ロードマップ:ハードウェアの実装

Enterprise Architect の主要な機能であるソースコード生成の機能が、Verilog、VHDL および System C の 3 つのハードウェア記述言語 (HDL) をサポートするように拡張されました。プロセスロードマップの観点で見れば、ソースコードの生成と SysML の間には直接の依存関係はありませんので、SysML のモデルをハードウェアとソフトウェアの両方実装に利用することができます。HDL のソースコードを生成すれば、チップ上のハードウェアとして実現することができます。

この詳細については、第 6 章で説明します。

ソフトウェアの実装

Enterprise Architect Suite システムエンジニアリング版に含まれる強力な機能を利用することで、ソフトウェアの実装を効率的に行うことができます。2 つの主要でありかつ特徴的な機能は次の通りです。

- 振る舞いモデル (ステートマシン図・アクティビティ図・シーケンス図) から処理を含むソースコードを生成する機能
- Eclipse や VisualStudio などの開発環境に Enterprise Architect のモデルを連携する機能

図 8 は、ソフトウェアの実装における最上位のロードマップです。

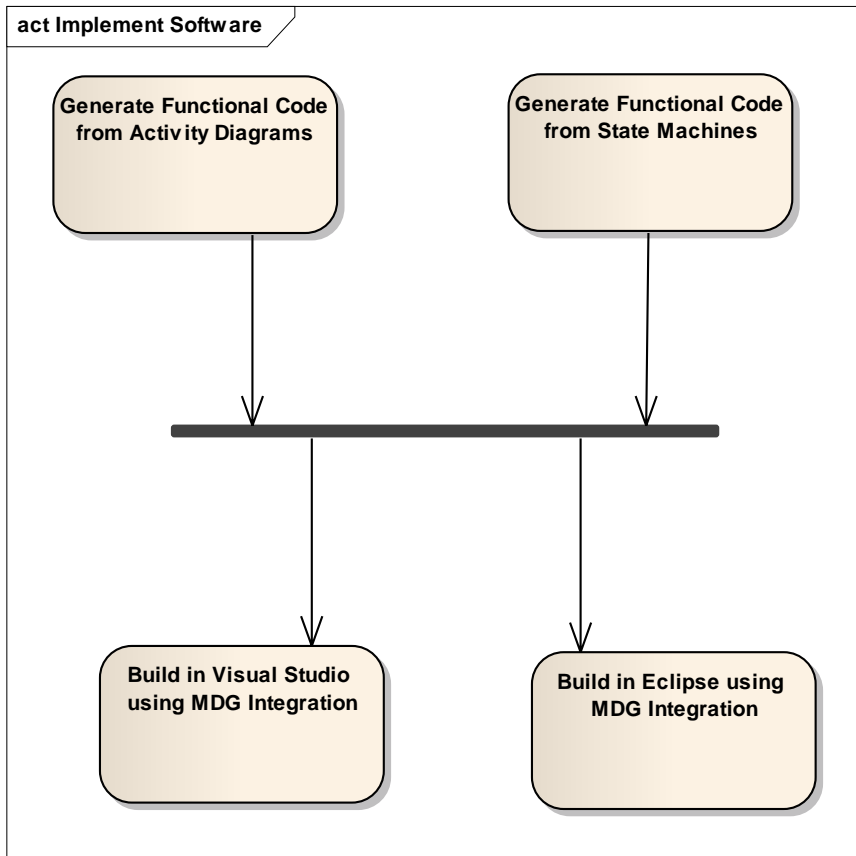


図 8 - ソフトウェアの実装

これらの機能の詳細や利用方法については、第 7 章で説明します。

音楽プレーヤーのサンプルについて

このドキュメント全体を通して、今回のプロセスの手順を説明する際にはサンプルプロジェクトに含まれるダイアグラムを利用します。私たちのサンプル（作成者は Sam Mancarella です）は、多くの人に馴染みのある「音楽プレーヤー」のハードウェアおよびソフトウェアを題材にしています。

次の図は最上位のパッケージ図です。このサンプルモデルがどのように構成されているかがわかります。

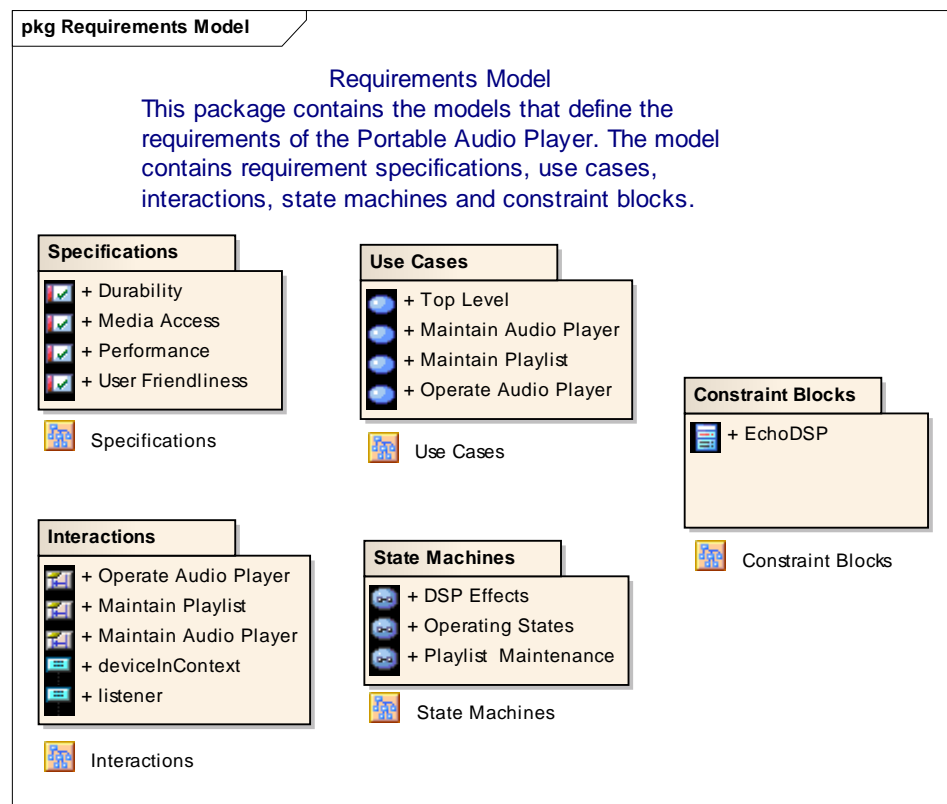


図 9 - ハードウェアとソフトウェアの実装モデルを含む、要求、振る舞い、構造、制約、パラメトリックに分類された SysML モデル

これから先の章では私たちのロードマップに従って、さまざまなアクティビティを説明するために、このサンプルモデルを利用します。よって、この Sam の音楽プレーヤーのサンプルモデルの内容はすぐに理解できるようになるでしょう。

第 2 章 - 音楽プレイヤーの要求

要求のロードマップ

要求は SysML の基礎になります。モデリングする対象のシステムの目的は、この要求を満たすことです。よって、想像されているかと思いますが、私たちのロードマップは要求を定義することから始まります。

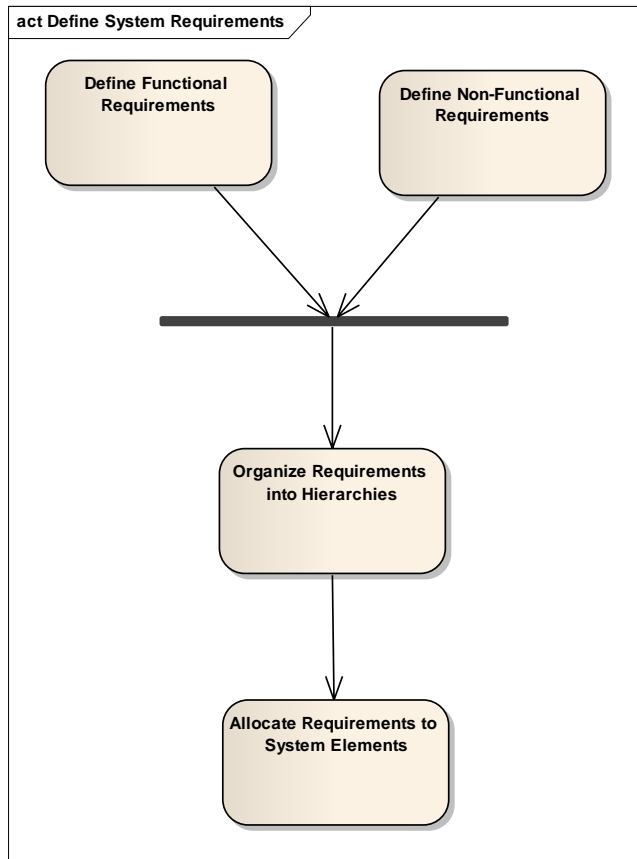


図 1 - 要求定義のロードマップ:

大まかに言いますと、通常、要求は**機能要求**（システム固有の特性や能力を表現した要求）と**非機能要求**（容易に利用できる、などの固有の特性に適用できない要求）に分類されます。これらの要求を効果的に分類することは重要です。そうしなければ、要求モデルは「**機能不全** (Dysfunctional)⁴」に陥ってしまいます。

SysML モデルにおける要求を考える際には、ハードウェア要求・ソフトウェア要求・システムが関係する環境に關係する要求を考えます。例えば、音楽プレイヤーの例では、騒音・天気などの「音楽を聴くときの環境」や、利用者の服装などを考慮する必要があります。

⁴ 機能不全要求を防ぐ方法の詳細は、Use Case Driven Object Modeling with UML - Theory and Practice, by Doug Rosenberg and Matt Stephens を参照

これらのドメインの観点は、より下流の要求を導きます。具体的には、振動への耐性・防水などの要求です。音楽プレーヤーがさらされる環境 (listeningConditions) を説明しているのは、このブロック図 (ibd) です。下の図 2 で定義される listeningDomain で、音楽プレーヤーが動作することが期待されています。

ドメイン「システム」の各部分を表現するために、下の図 2 の SysML の「ブロック」を分解していきます。

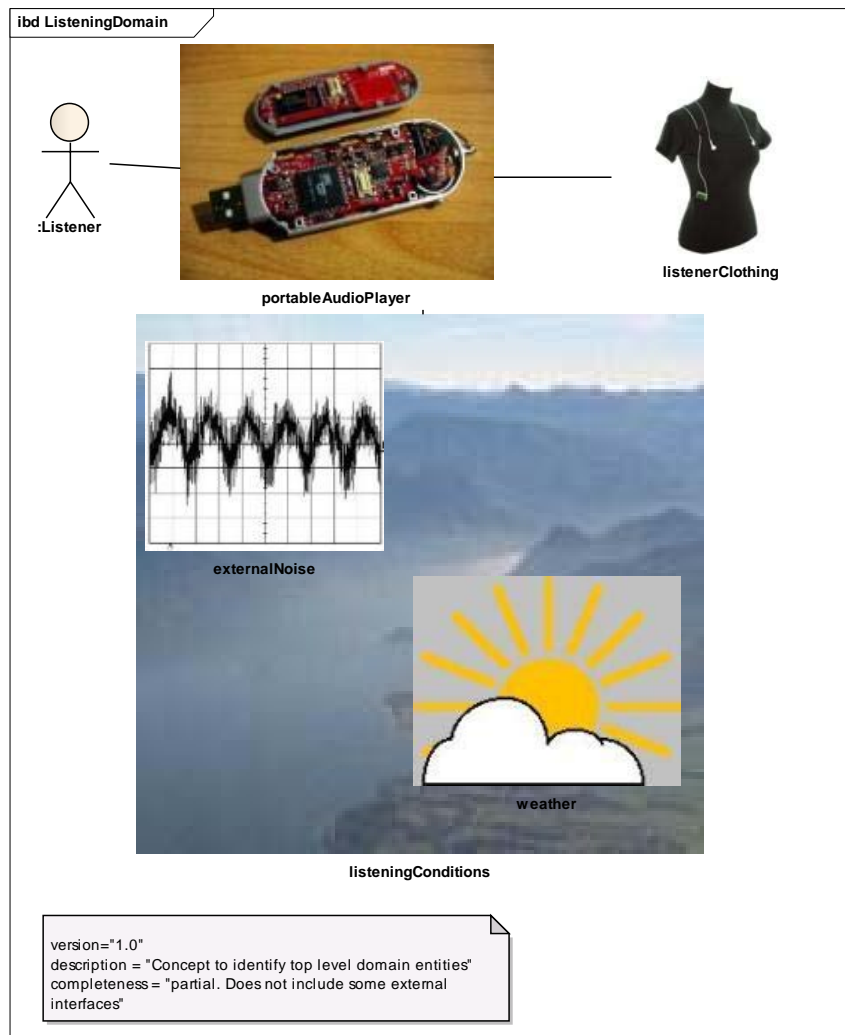


図 2 - ハードウェア、ソフトウェア、システム操作において必要な環境を含む SysML モデル

モデリングテクニック - Enterprise Architect では、画像を簡単に使えます

上の例を見ればわかるように、写真やイラストなどの画像をモデルに追加することで、より理解しやすいモデルを作成することができます。Enterprise Architect では、画像を追加することは簡単です。具体的な操作方法にはいくつかありますが、クリップボードに画像をコピーして貼り付ける場合には、ダイアグラムの背景で右クリックして「書式設定」→「クリップボードの画像を貼り付け」を選択して下さい。数秒後には、その画像がモデルに表示され、読みやすさを向上させることができます。

音楽プレーヤーの要求

SysML では、要求間の関係として、7 種類の関係を定義しています。これらの関係は 2 つに分類することができます。「containment」、「derive」、「copy」の 3 つは要求間の関係です。要求と他のモデル要素との関係には、「satisfy」、「verify」、「refine」、「trace」を利用します。

下の図は音楽プレーヤーの要求図です。○ の中に + が書かれている接続は「containment」であり、要求間の階層関係を構成するために利用します。例えば、Specifications Package は要求を「所有」していることを示しています。それぞれの要求は、さらにその子要求を「所有」していることとなります。

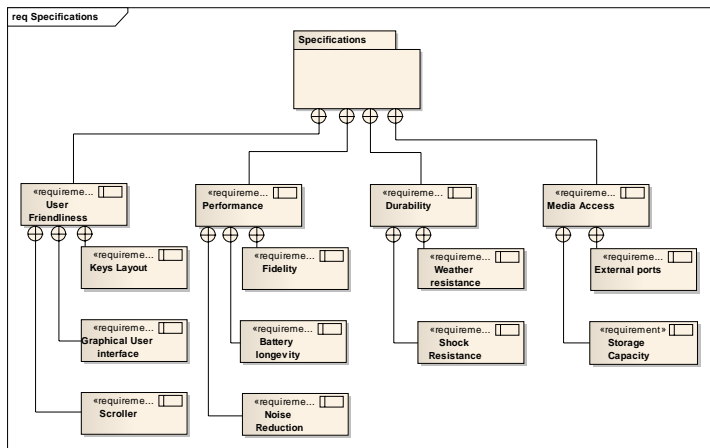


図 3 - 音楽プレーヤーに必要な要求は、利便性、性能、持続性などのカテゴリーに分類される

Enterprise Architect には、要求とモデル要素間の関係 (satisfy) を簡単に定義することのできる機能がいくつか搭載されています。「関係マトリックス」を利用すると、このような関係を確認することができます。

	Battery longevity	Durability	External ports	Fidelity	Graphical User interface	Media Access	Noise Reduction	Performance	Processor	Shock Resistance	Storage Capacity	User Friendliness	Weather resistance
Buttons		↑			↑							↑	↑
Charging Unit - ADP2291	↑	↑											
Clothing										↑			
Codec with Amplifier - TLV32...				↑				↑	↑				
Environment										↑			↑
Li-Ion Battery Monitoring Sys...	↑												
ListeningDomain								↑					
Memory - MT42L32M64DKH...											↑		
Noise								↑					
Panasonic Li-Ion CGR18650AF	↑												
Portable Audio Player				↑				↑		↑			↑
PowerSubsystem	↑												
Processing Subsystem								↑					
Processor - TMS320VC5507								↑					
R5232			↑										
Touch-screen - Toppoly TD0...					↑								
Transport Subsystem	↑		↑										
USB - PL-2528			↑										
User Interface					↑								
Weather													↑

図 4 - Enterprise Architect の要求マトリックスは、要求のブロック割り当てを簡単に表すことができる

モデリングテクニック - ドラッグ & ドロップで要素に要求を割り当てる

モデル要素（ブロックやユースケース）に、関係する要求を割り当てる方法は、Enterprise Architect ではとても簡単です。プロジェクトブラウザにある要求要素を、ダイアグラム内の対象の要素までドラッグ & ドロップするだけです。Enterprise Architect は対象の要素間の関係を自動的に追加します。

また、関係マトリックスの機能を利用することで、モデル要素と要求間の関係をマトリックス形式で参照・編集することができます。

	Battery longevity	Durability	External ports	Fidelity	Graphical User Interface	Keys Layout	Media Access	Noise Reduction	Performance	Scroller	Shock Resistance	Storage Capacity	User Friendliness	Weather resistance
Adjust Volume								↑						↑
Charge Battery	↑													
Connect To Computer			↑				↑		↑					↑
Copy track from external media								↑					↑	
Create Playlist	↑				↑	↑		↑				↑	↑	
Download track	↑	↑	↑					↑					↑	
Listen Audio				↑		↑		↑						↑
Maintain Audio Player							↑							
Maintain Playlist							↑							
Operate Audio Player		↑			↑	↑			↑	↑	↑			↑
Pause						↑			↑					↑
Play				↑		↑		↑	↑					↑
Power On	↑													
Record Audio				↑				↑	↑					↑
Replace Battery	↑	↑									↑			↑
Replace Headphones			↑											↑
Replace Skin		↑			↑	↑					↑		↑	↑
Stop						↑			↑					↑
Top Level														
View Existing Playlist							↑						↑	
View New Tracks							↑						↑	

図 5 - Enterprise Architect の要求マトリックスは、要求のユースケース割り当てを簡単に表すことができる

モデリングテクニック - 関係マトリックスを利用して要求と要素間の関係 (satisfy) を確認する

ドラッグ & ドロップで割り当て関係を定義したり、要素のプロパティダイアログの「要求」タブを利用したりして定義した後は、要求と、任意の種類要素との相互関係をマトリックス形式で確認できます。「表示」 → 「関係マトリックス」を実行します。その後、ソースとターゲットのそれぞれの範囲（パッケージ）と種類を指定します。すると、関係がマトリックスに表示されます。表示される内容はオプションボタンから呼び出せる機能で、CSV ファイルに出力することができます。

この先の章では、音楽プレーヤーの SysML モデルの中のさまざまな局面で、ここで定義した要求がどのように実現されていくのかを見ていきます。

第 3 章 - 音楽プレイヤーの振るまい

振る舞いのモデリングのロードマップ

振る舞いのモデリングでは、利用者や環境に対してシステムがどのようなやりとりをするのか、システムの動的な振る舞いを記述します。利用者とシステムとのやりとりのモデリングには、ユースケースとシーケンス図を利用します。利用者が直接関係しないようなイベント駆動の振る舞いを記述するためには、ステートマシン図を利用します。以下の図が、私たちのロードマップのアクティビティです。

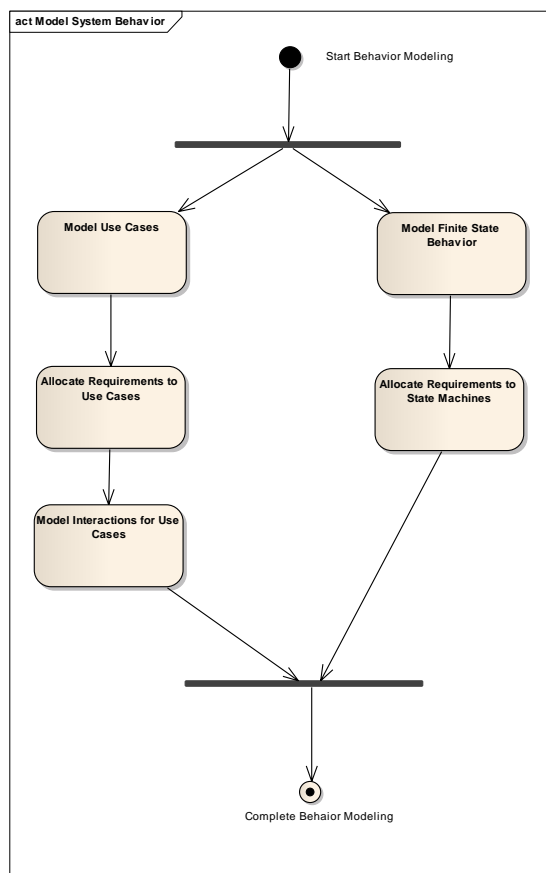


図 1 - 振る舞いのモデリングのロードマップ

このロードマップを見るとわかるように、振る舞いのモデリングは 2 つの平行な枝に分かれます。一方はユースケースであり、他方はステートマシン図です。それぞれの枝では、要求をモデル要素（ユースケースや状態）に割り当てる作業も含まれます。ユースケースについては、概要は自然言語で内容を記述します。詳細はシーケンス図で記述します。

この章では、音楽プレイヤーのサンプルの動的な振る舞いを対象に、ロードマップの内容を確認していきます。その後、第 4 章では期待する振る舞いに対応するようなシステムの構造について見ていきます。私たちは、モデルの構造部および振る舞い部について、「静的」と「動的」という言葉を使います。構造は「静的」です。一度定義されると変更されません。振る舞いは「動的」です。利用者の操作や外部のイベントに応じて変わります。

音楽プレーヤーの振る舞いモデル

ここでは、音楽プレーヤーについて、動的なモデルの 2 つの枝について見ていきます。利用者中心のシナリオ、例えば音楽プレーヤーの操作はユースケースとしてモデリングされます。一方、デバイスの動作の状態などはステートマシン図でモデリングします。なお、プレイリストの編集はユースケースで記述され、同時にステートマシン図でも記述されていることに気をつけて下さい。ストーリー（振る舞い）を説明するために役立つのであれば、両方のダイアグラムを利用することもできます。

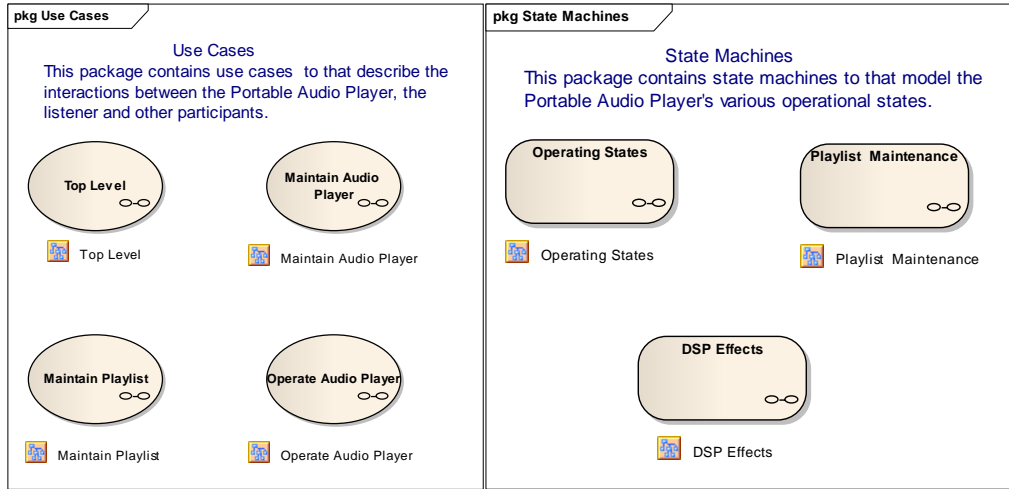


図 2 - ユースケース、やりとり、ステートマシン図を含む振る舞いのモデル

モデリングテクニック- モデルにはストーリーが必要

モデルの第一の目的は、利害関係者・利用者・分析者・設計者・ソフトウェアおよびハードウェアの開発者・品質保証担当者の間において、対象のシステムについて共通の理解を促進する、コミュニケーションの道具となることです。モデルの読みやすさは常に最適に保ち、明確に、曖昧ではない形で「ストーリーを語る」モデルであることを常に確認する必要があります。

以下の図は、音楽プレーヤーの最上位のユースケースです。ユースケース要素に表示されている眼鏡（あるいは無限大）マークのアイコンは子ダイアグラムが存在することを示しています。この子ダイアグラムでは、ユースケースの詳細について記述されています。

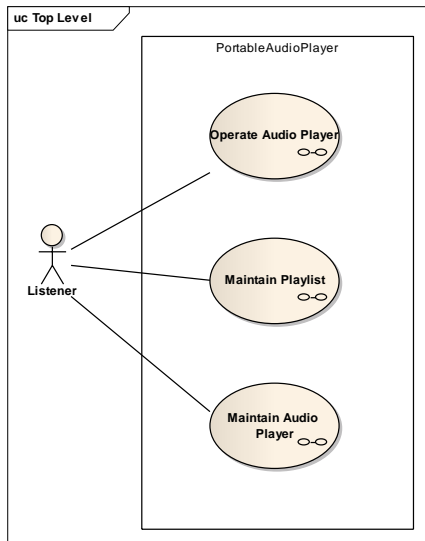


図 3 - 音楽プレーヤーの最上位ユースケース

Enterprise Architect では、このように子ダイアグラムを作成できることで、モデルを深耕していくことが容易です。以下の図は、音楽プレーヤーの管理についての子ダイアグラムです。

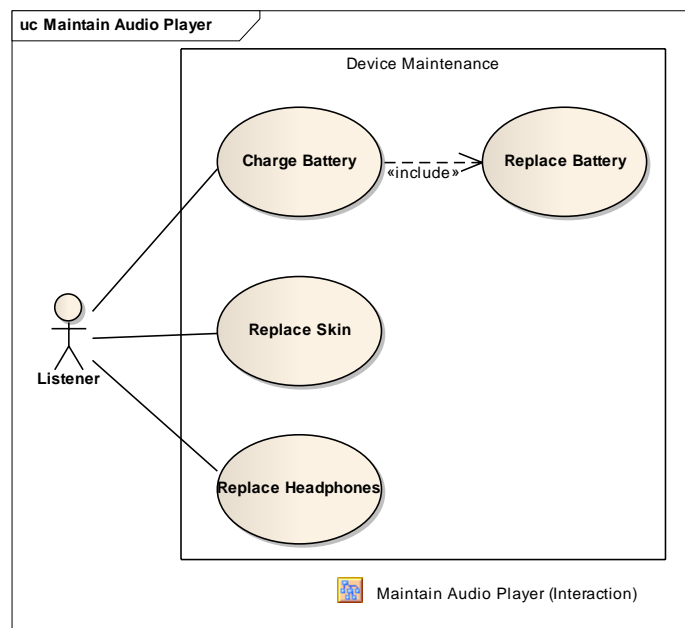


図 4 - 充電、バッテリー交換、外装（スキン）やヘッドホン交換を含む音楽プレーヤーのハードウェアメンテナンスのためのユースケース

Enterprise Architect では、ハイパーリンクを作成することであるダイアグラムから別のダイアグラムに移動するためのリンクを作成・配置できます。上の図にあるリンクは図 5 のシーケンス図にリンクし、図 5 のシーケンス図には上の図に戻るリンクが配置されています。

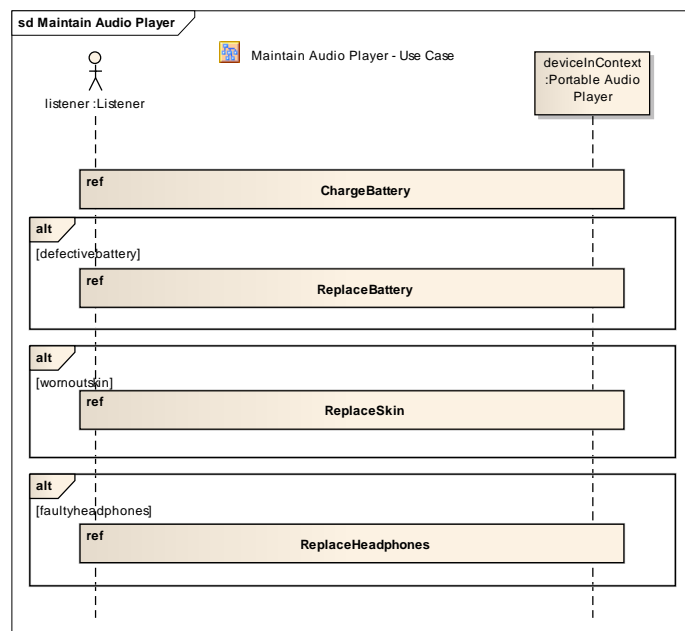


図 5 - 3 つの代替手段（バッテリー不良、外装摩耗、ヘッドホン不良）と通常行為（バッテリー充電）を表した音楽プレーヤー維持のためのやり取り図

下の図は、音楽プレーヤーの基本的な操作についてのユースケース図です。

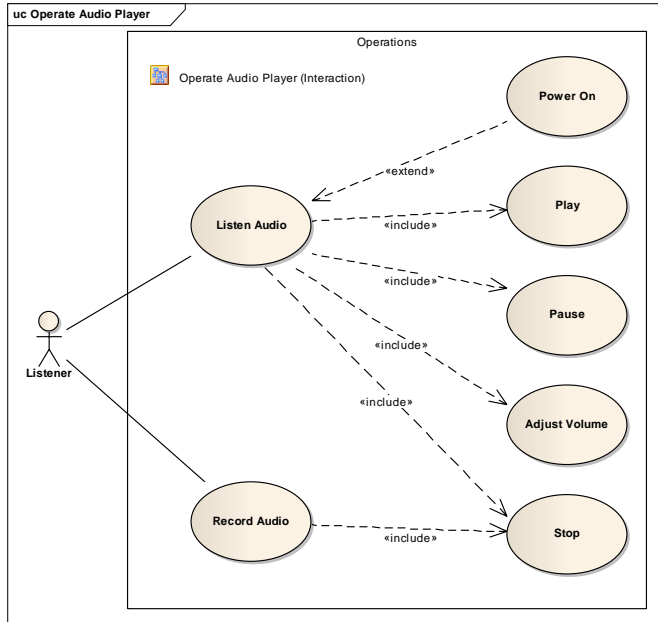


図 6 - リスニングとレコーディングのための音楽プレーヤーユースケース

管理のユースケースについて、ユースケース図とシーケンス図（図 7）は Enterprise Architect のダイアグラム参照機能を使用して、互いに参照できるようになっています。このダイアグラムでは、動作なし・再生・一時停止とボリュームの調整についての流れが並行に行われる可能性があることを示しています。

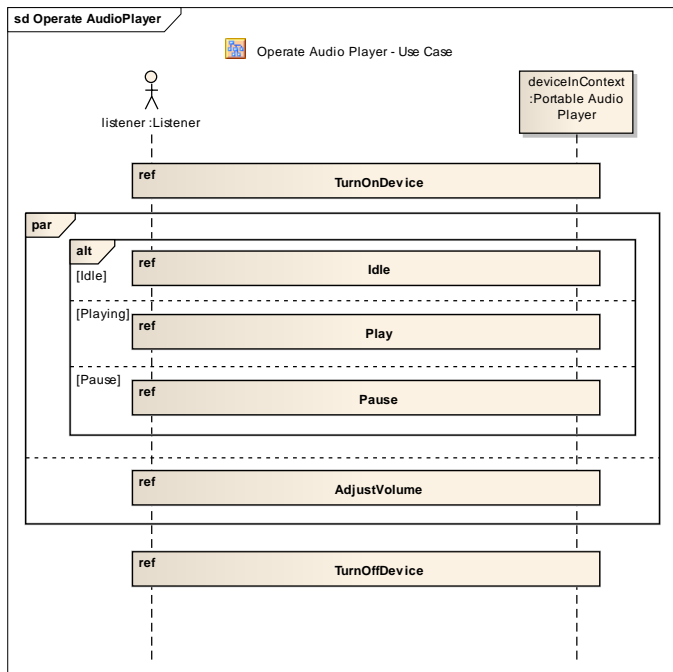


図 7 - リスニングとレコーディングのための音楽プレーヤーのやり取り図

ユースケースは、利用者がどのようにシステムと相互作用を行うのかを記述します。この後、ステートマシン図を利用してどのようにイベント駆動の振る舞いを記述するかを見ていきます。

音楽プレーヤーの状態モデル

組み込みシステムでは、状態の操作・イベントの発生やシステムの処理の観点で記述することには多様な利点があります。SysML（ここでは UML も同じ）では、システムに関するいくつかの状態の観点で記述するために、ステートマシン図を利用します。

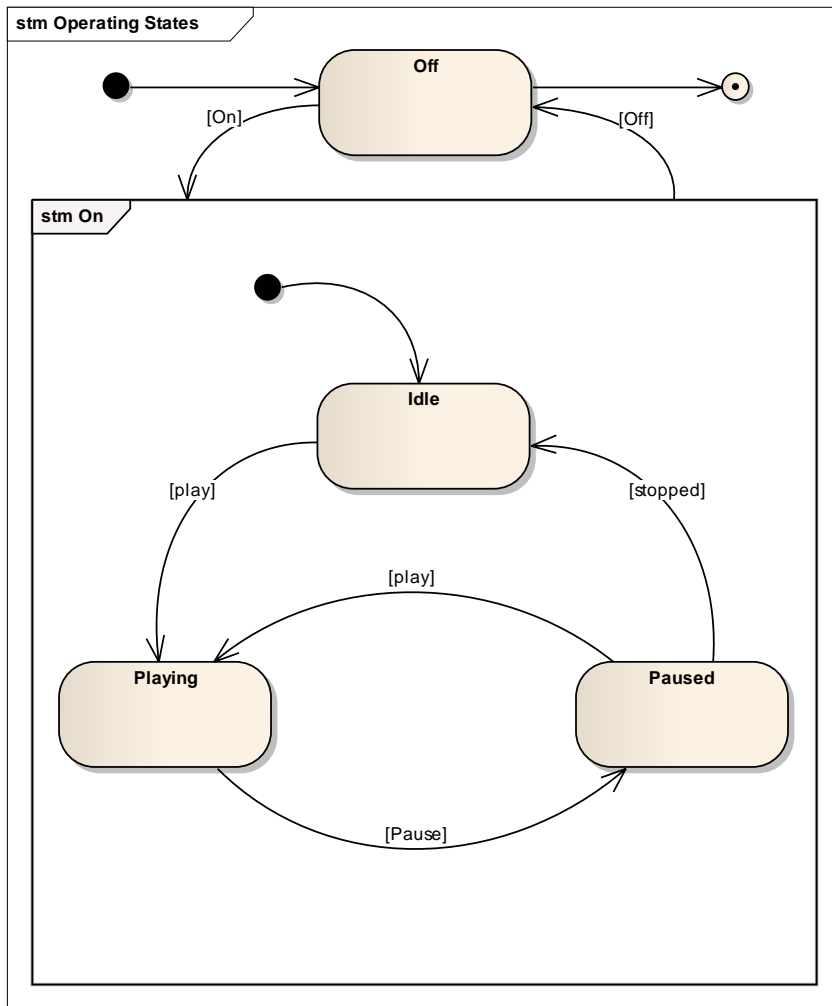


図 8 - 音楽プレーヤーの操作状態

ステートマシン図では、1つのダイアグラムでサブ状態を「入れ子」にすることができます。図 8 では音楽プレーヤーの On の状態の詳細についての詳細な振る舞いと、On/Off の振る舞いを 1つのダイアグラム内で表現しています。なお、要求を状態に関連づけるには、単にプロジェクトブラウザにある要求をダイアグラム内の状態要素の上にドラッグ & ドロップするだけです。

ステートマシン図はシステム（あるいはブロック）の状態と、「ボタンが押される」とようなイベントの発生とを関連づけます。図 9 では「オーディオ EQ」ボタンによって、さまざまな音楽効果が循環するというシステムの振る舞いを表現しています。

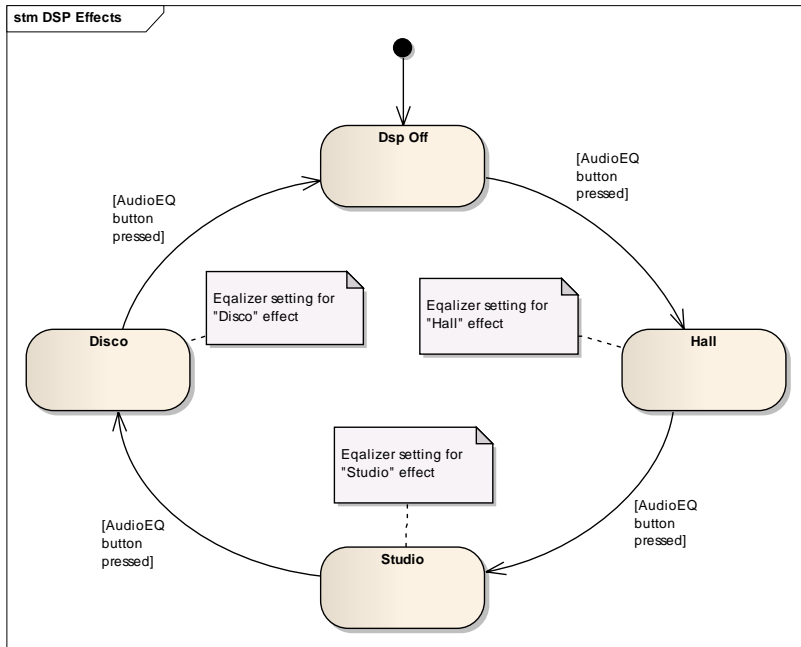


図 9 - デジタル信号プロセスによるオーディオ効果のためのステートマシン図

ユースケース図やステートマシン図を利用して「ストーリーを語る」場合に、適用する必要がある絶対的なルールは存在しません。最適なガイドラインとしては、「ストーリーを語る」場合に、単に最適なダイアグラムを利用する、ということだけです。場合によっては、両方を利用することが最適です。

ユースケース図とステートマシン図を結合する

次の例は、音楽プレーヤーのシステムについて、ユースケース・シーケンス図・ステートマシン図を利用して、どのように動作するかについて異なる観点から記述する方法です。

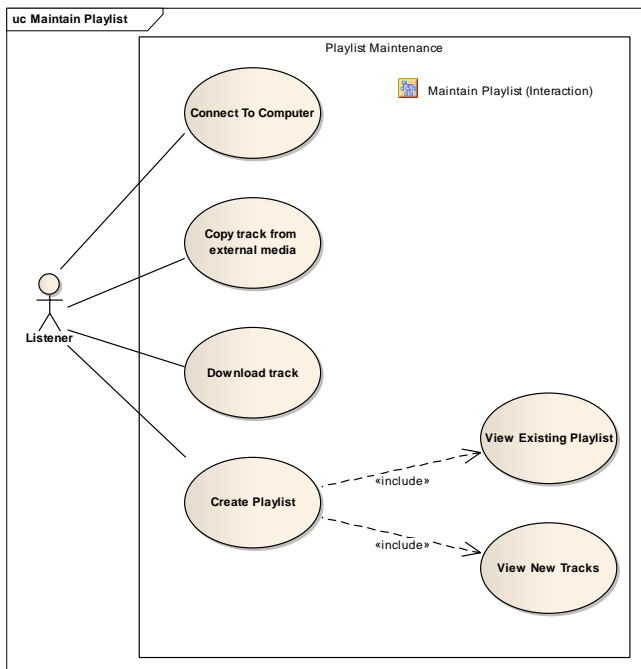


図 10 - プレイリストのメンテナンスのためのユースケース図

それぞれのダイアグラムでは対象のシステムに対する異なる観点（ビュー）を提供しています。要求の定義からハードウェア・ソフトウェアの実装までの間を通して誤解のないようにするために、「ストーリーを語る」ために必要なだけのビューを利用することができます。図 11 はプレイリストの管理のためのさまざまなシナリオを示しています。図 12 はイベント駆動の観点です。

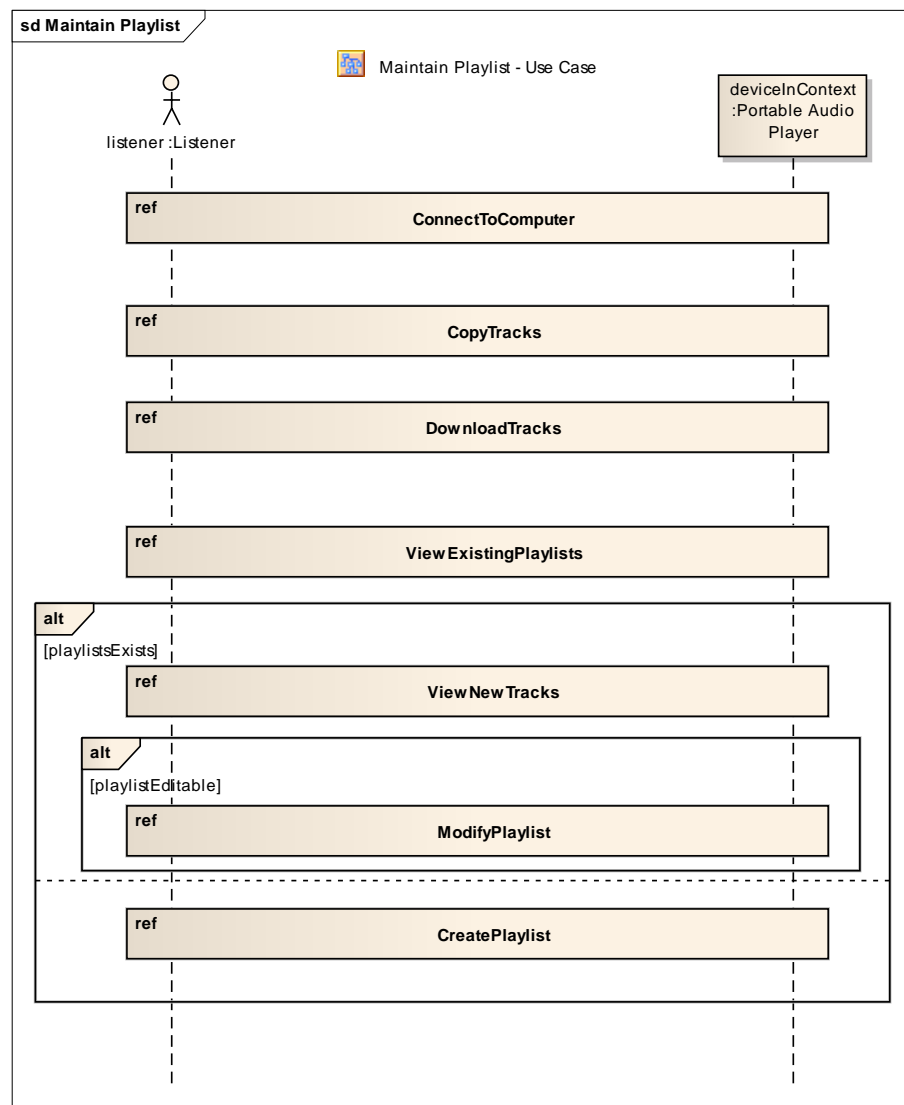


図 11 - プレイリスト保持のためのシナリオ

プレイリストの編集をするためには、プレイリストが既に存在し、編集可能でなければならないことに注意して下さい。しかしながら、この条件とは関係なく、曲はダウンロードしたりコピーしたりすることができます。

図 12 のステートマシン図では異なる観点を提供しています。最上位のステートマシン図では音楽プレーヤーと音楽サーバーとの接続・切断に依存する振る舞いを示しています。この図を見るとわかるように、プレイリストの管理に関する振る舞いは、機器が接続されている場合に発生します。

アクティビティ図は音楽プレーヤーが接続された場合の振る舞いを記述するために利用されています。アクティビティ図内のフォークとジョイン（黒い太線）は平行な処理を記述するために利用します。

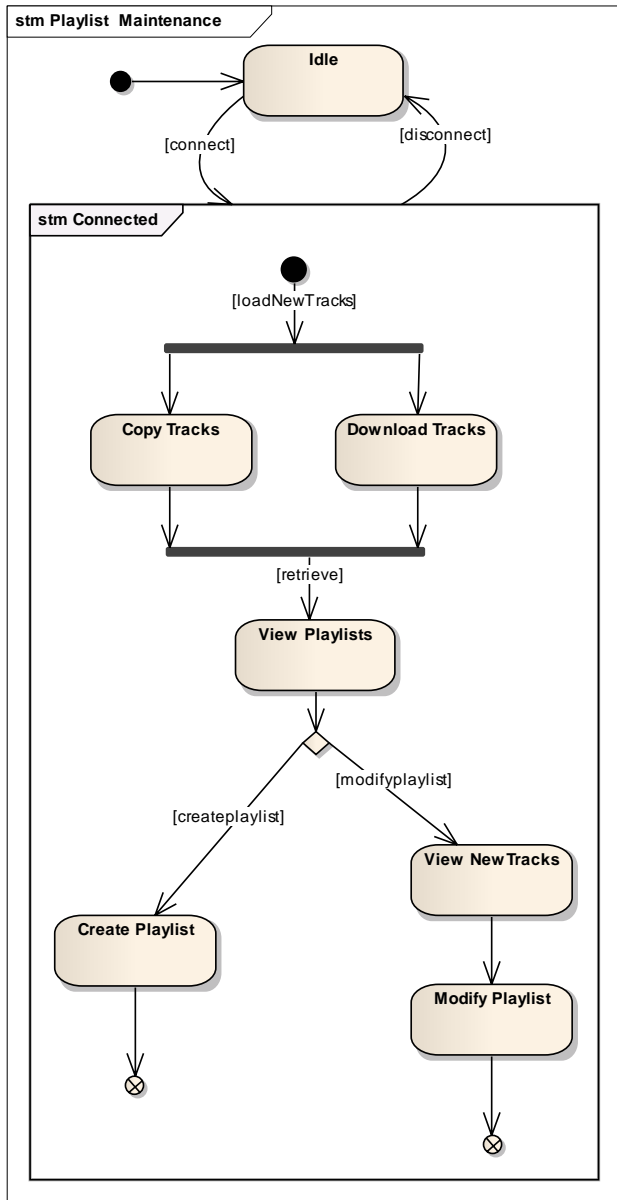


図 12 - プレイリストのメンテナンスのための状態/イベントの振る舞い

ユースケース・シーケンス図・ステートマシン図の組み合わせやアクティビティ図はシステムの詳細な動的振る舞いを特定するために利用できます。さらに、ユースケース・状態・アクティビティやその他のモデル要素に対して要求を割り当てることができます。

以上で、ロードマップの作業内容を網羅しました。次の章ではブロックを利用して、システムの構造を定義するためのロードマップを見ていきます。

第 4 章 音楽プレイヤーの構造

これまで、要求と振る舞いのモデリングについてみてきました。次に、SysML と Enterprise Architect を利用してどのようにシステムの構造を記述するか、についてみていきます。今までと同様に、音楽プレイヤーの例を利用して説明していきます。

ロードマップ:構造の定義

図 1 は、構造のモデリングのロードマップです。

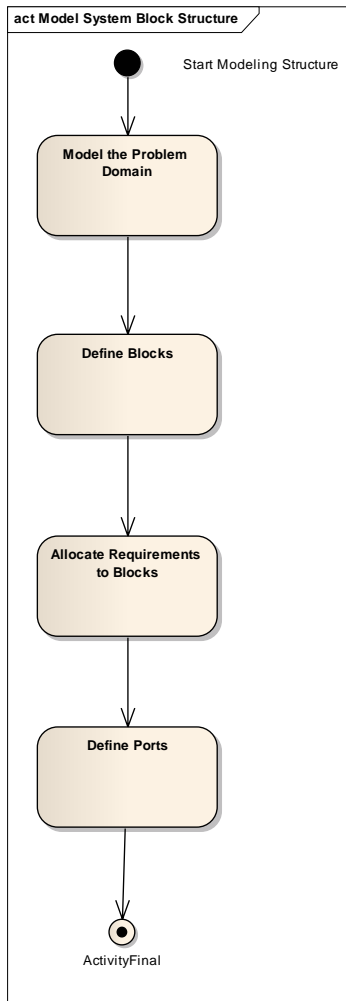


図 1 - ロードマップ - モデル構造

SysML では、ブロックは構造を記述するための主要な単位になります。ブロックはハードウェアやソフトウェアの要素を示します。ブロックはポートを持つことができます。ポートは、ブロックからの入出力を示します。

問題ドメインのモデリング

構造モデリングのロードマップは ICONIX プロセスを知っている人であれば、誰もが知っているステップ、つまりドメインモデリングから始まります。ドメインモデルを作成する際には、「実際の世界のもの」を基準にして、抽象化されたものを定義します。図 2 は音楽プレーヤーのドメインモデリングの内容です。この図を見るとわかるように、ドメインモデルには、洋服や外部環境などのシステムの外部にある実際世界の要素も含まれます。

問題ドメインのモデルの目的は、音楽プレーヤーが動作する「システム」を記述することです。要求から実装までを通して、装置の設計の「コンテキスト」を記述するために「システム」のモデルを利用します。図 2 では、「超システム工学 (System of Systems)」として知られる考え方を元に、音楽プレーヤーがどのシステムと相互作用を行うかを示しています。

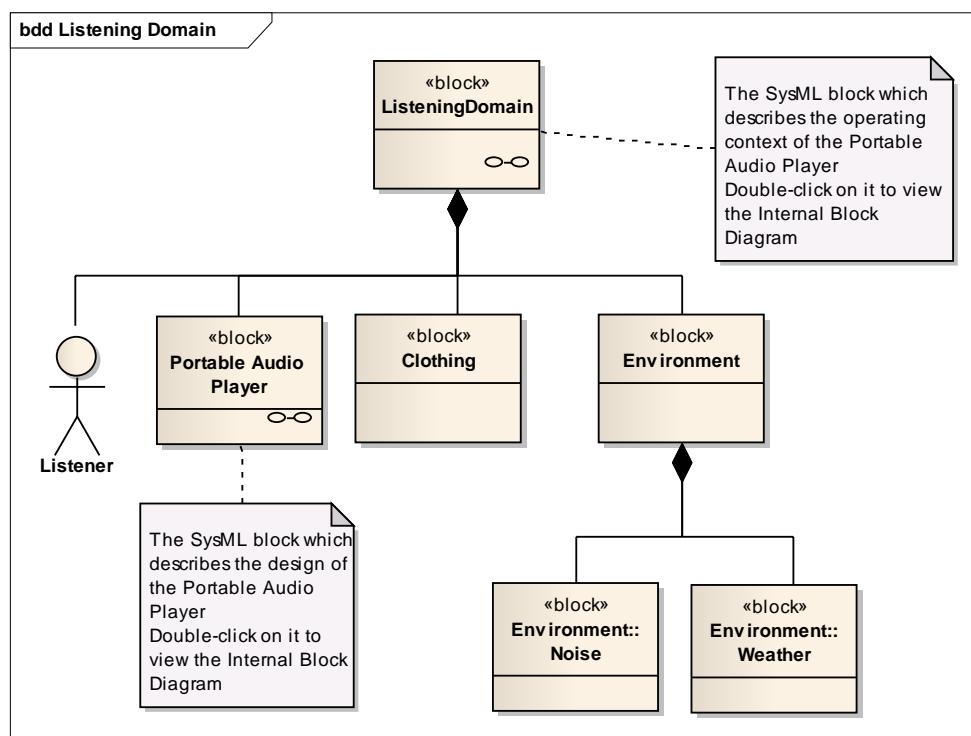


図 2 - 音楽プレーヤーのドメインモデル

図 2 において黒い菱形のマークを持つ関係は、コンポジションの関係です。この例では、「Noise」と「Weather」は「Environment」を構成するもの、ということを示しています。

SysML ではブロック図には 2 つのレベルがあります。ブロック定義図 (BDD) と内部ブロック図 (IBD) です。この章では、これらの 2 つの図について説明します。

ブロック構造のモデリング (ブロック定義図)

図 3 は音楽プレーヤーの上位の構造の詳細を説明する「子ブロック定義図」です。ブロック定義図の目的はコンポジションの関係を利用して、関係するブロックを相互に結びつけることで、ブロックの構成を示すことです。

この図を見るとわかるように、音楽プレーヤーや 4 つの主要なサブシステムで構成されています。Power, Processing, User Interface, Transport です。この 4 つのサブシステムのそれぞれはブロックとしてモデリングされ、それぞれのブロックはサブブロックで構成されています。

Embedded Systems Development using SysML

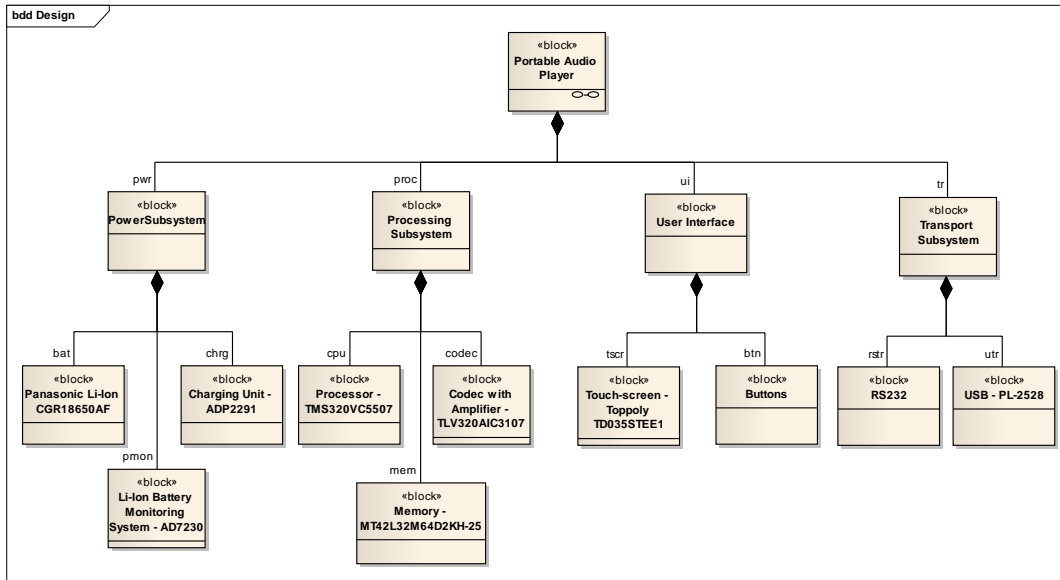


図 3 - 音楽プレーヤーのブロック構造

図 4 は Power Subsystem の詳細です。このブロックは、Lithium-Ion Battery, Charging Unit, Monitoring System で構成されています。それぞれのブロックにはポートがあり、音楽プレーヤーを操作するための電流を示しています。

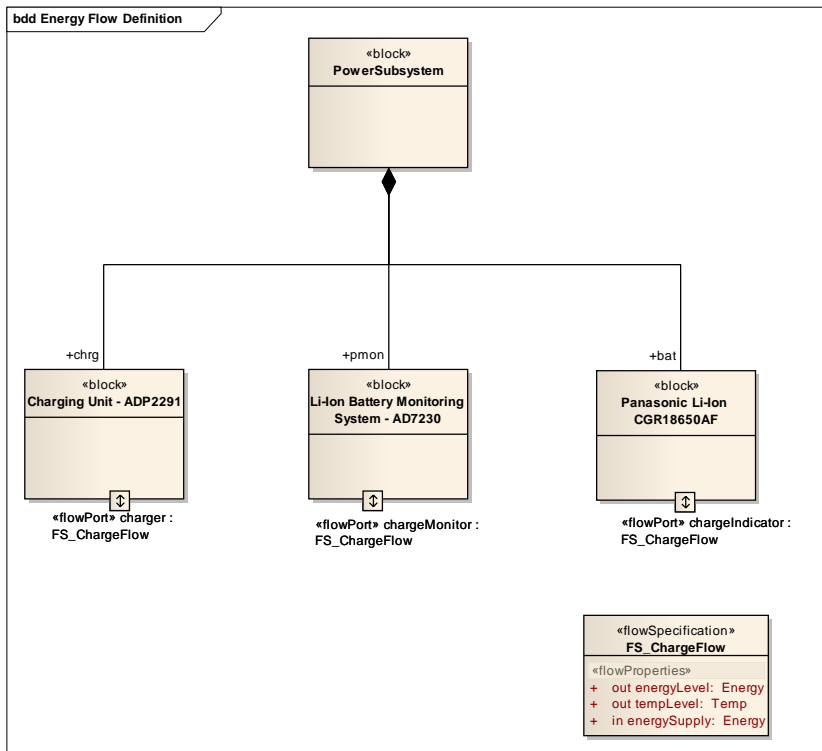


図 4 - 音楽プレーヤーのPower Subsystem

ブロック内部のモデリング（内部ブロック図）

図 5 は Power Subsystem の内部ブロック図（IBD）です。内部ブロック図の目的は、対象のサブシステムを構成するためにブロックの部分をどのようにつなげるかを詳細に記述することです。別の言葉で言えば、内部ブロック図はブロックの構成の「何を」と「どのように」について記述します。それぞれのパート要素は先ほど説明したブロック定義図でのコンポジションの関係を表現しています。

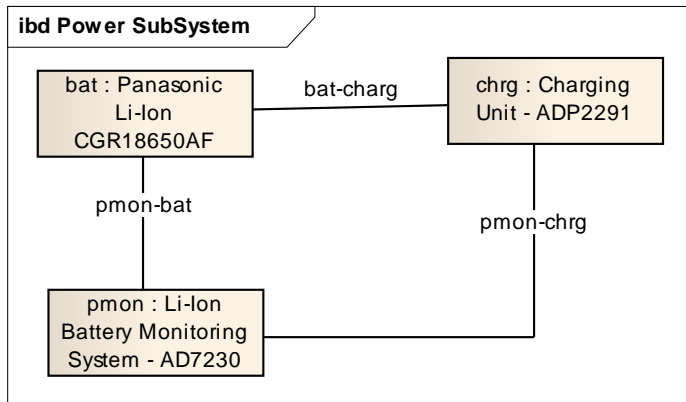


図 5 - Power Subsystem の IBD

内部ブロック図ではブロック内のパートの接続を記述します。図 6 を見るとわかるように、一つの内部ブロック図で複数のレベルをネストして表現することができます。

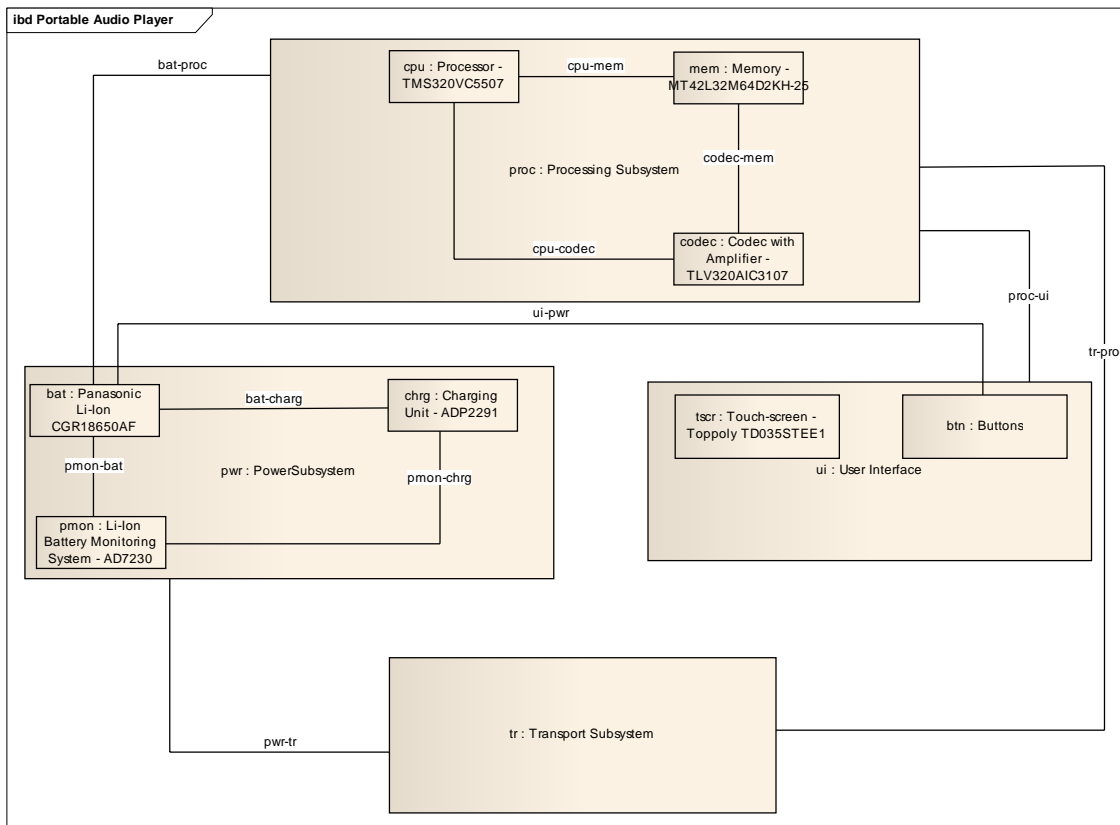


図 6 - 音楽プレーヤーのパーツ配線を表す様々なレベルの IBD

図 7 は Processing Subsystem の内部について記述しています。この図を見るとわかるように、CPU は Memory Unit および Codec/Amplifier と接続しています。

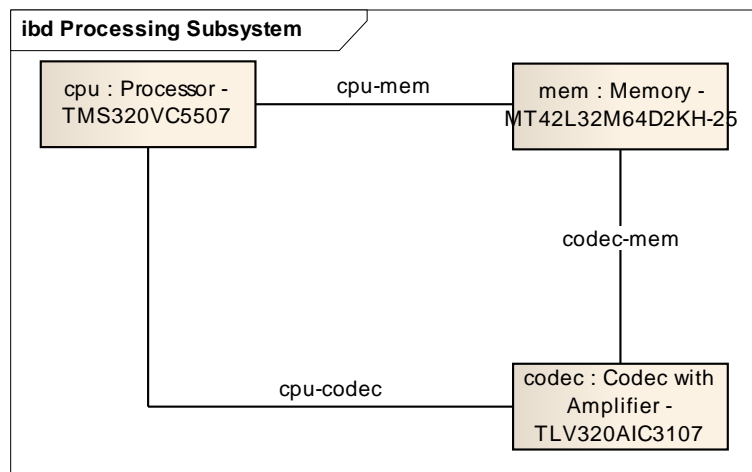


図 7 - 音楽プレーヤーの Processing Subsystem のブロック内部

ポートの定義

構造モデリングのロードマップの最後は、ポートを定義することです。図 8 では User Interface, Processing Subsystem, Transport Subsystem, USB, RS-232 間のデータの流について記述しています。

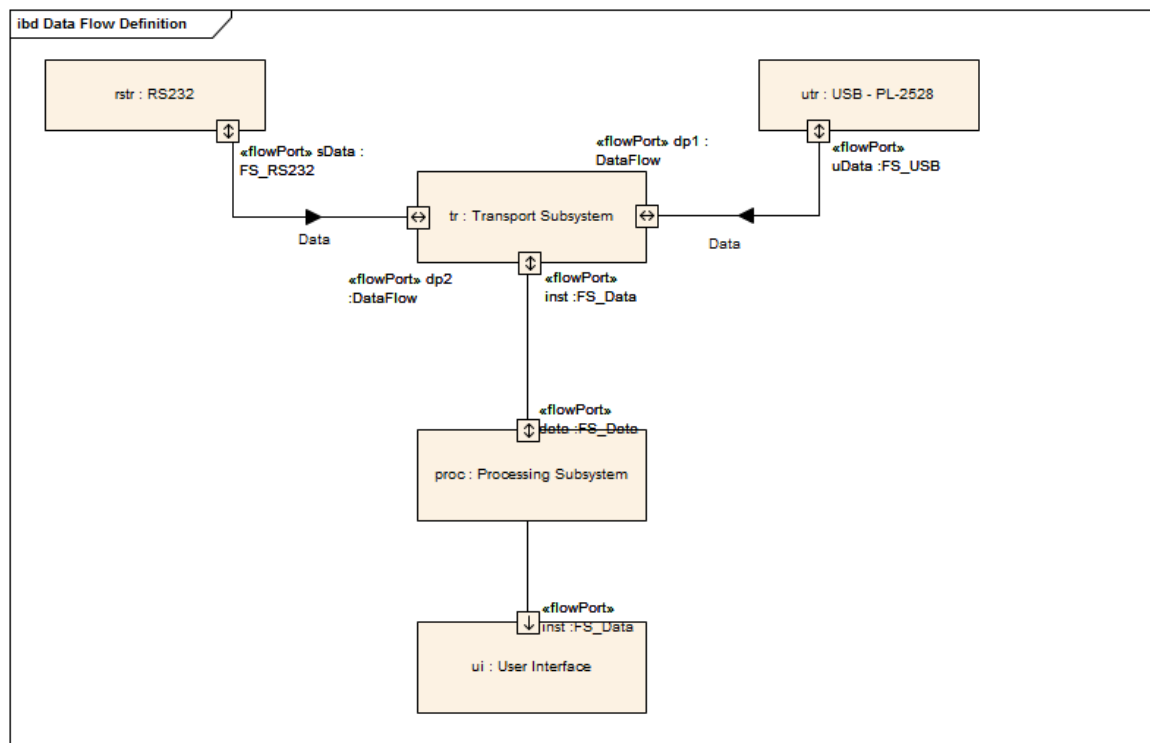


図 8 - Subsystem 間の音楽プレーヤーのデータフロー

図 8 には flowPort という種類のポートがあります。SysML の flowPort は、ポートのプロパティやポートに関する振る舞いを定義する FlowSpecification によって定義されています。

flowPort は、ポートを流れる情報の向き (in/out/conjugate) についても記述します。SysML には standardPorts というポートも定義されています。このポートはインターフェースを提供したり要求したりすることができます。ItemFlow (接続の上にある矢印) は接続やポートを通して流れる内容について記述しています。上の例では、Data が接続を通過して流れていることを示しています。

音楽プレーヤーのハードウェア部品

最後に、図 9 では音楽プレーヤーのハードウェア部品を示しています。

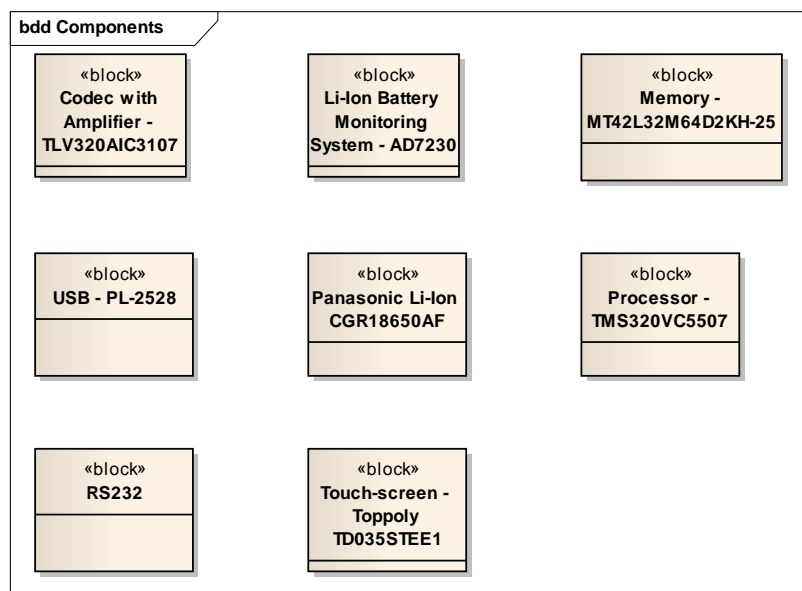


図 9 - ブロックとしてモデル化されたハードウェアコンポーネント

ブロックとしてハードウェア部品をモデリングすることにより、要求をハードウェアに割り当てることができるようになります。繰り返しになりますが、Enterprise Architect では要求をモデル要素に簡単に割り当てることができることを忘れないで下さい。

以上で、ブロック・パート・ポートについての説明を終わります。これまでの 3 章で、要求・振る舞いモデリング・構造モデリングについて SysML でモデリングすることを説明し、SysML の基礎部分を作成しました。この次の 3 つの章では、SysML と Enterprise Architect Suite システムエンジニアリング版のより詳細な内容について説明します。次の章では、制約とパラメトリックについて説明します。そして、ハードウェアとソフトウェアの実装に進みます。

第 5 章 オーディオプレーヤーの制約とパラメトリック

SysML と UML の最大の違いの一つは、SysML のモデルの一部をシミュレーションできることです。シミュレーションは、システムの主要な側面を説明するために数学的・物理的法則に基づいて行われます。Enterprise Architect と他の SysML モデリングツールとの最大の違いの一つはモデリングツール内でシミュレーションが実行できることです。外部のシミュレータと連携するものではありません。この章では、今までと同様にプロセスロードマップに沿って、これらの機能について説明していきます。

制約とパラメトリックのロードマップ

制約とパラメトリックのロードマップは、大きく 2 つのステップに分けられます。最初のステップでは、図 1 のように制約とパラメトリックを定義します。次のステップでは、図 2 のようにシミュレーションの設定を行い、実際にシミュレーションを行います。この 2 つのステップは Enterprise Architect Suite システムエンジニアリング版で行うことができます。これにより、要求を満たすための解決策に向かって加速集中することができます。この章では、音楽プレーヤーの例を元に、これらのステップを確認していきます。

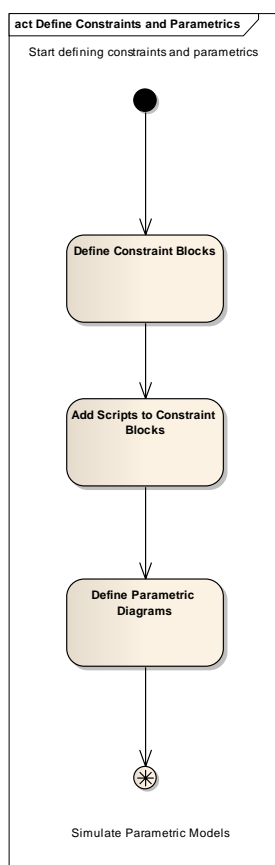


図 1 - ロードマップ:制約とパラメトリックの定義

SysML のパラメトリックモデルではシステムを決定づけるパラメータの分析を支援します。この内容には、パフォーマンス・信頼性やその他の物理的特性などの主要な特性の評価が含まれます。複雑な数学的関係に基づく実行時の制約を把握することで、パラメトリックモデルは要求モデルと設計モデルとを結びつけます。SysML では、パラメトリックモデルは実行時の要求を記述するために利用することもできます。(つまり、内部の燃焼エンジンは、付属されたパラメトリックの特性に合わせて、そのトルクを伝えます。パラメトリックはエンジンに対するトルク曲線を記述するために、グラフで記述することができます。)

図 1 のように、パラメトリックモデルを定義することの前には、制約ブロックの定義や制約ブロックへのスクリプトの追加・パラメトリックスの定義の作業があります。パラメトリックモデルが定義できた後は、図 2 にあるようにシミュレーションが実行可能になります。

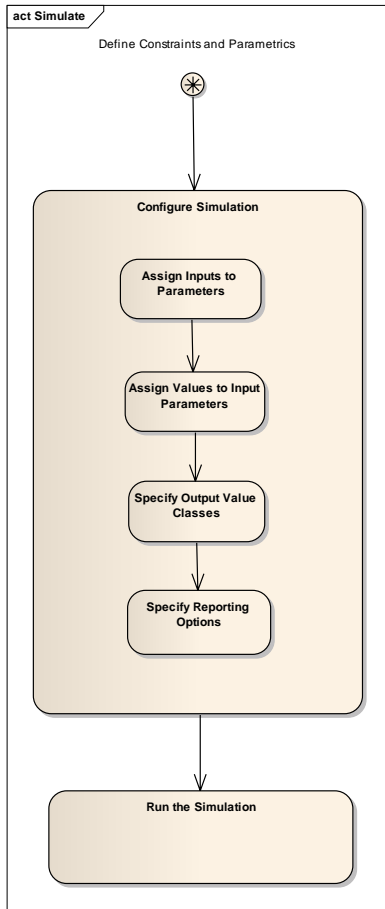


図 2 - ロードマップ: 設定とシミュレーション実行

SysML のパラメトリックモデルをシミュレーションする手順は、単にシミュレーションの設定を行い、そして実行するだけです。このシミュレーション機能は、Enterprise Architect 内部で完結していますので、高速に、そして簡単に工学的なトレードオフを検証することができます。また、結果のフィードバックと結びつけることで、要求に合致するかどうかの解決のために集中することができます。

制約ブロックの定義

パラメトリックモデルを構築するためには、シミュレーション対象のモデル内に既に作成済みの制約の機能を示す SysML の制約ブロックを作成します。それぞれの制約ブロックには入出力のパラメータを指定します。さらに、制約を実行可能とするためのスクリプトを定義します。図 3 では、音楽プレーヤーが動作するための基礎をなす数学的な関数の制約ブロックを定義しています。

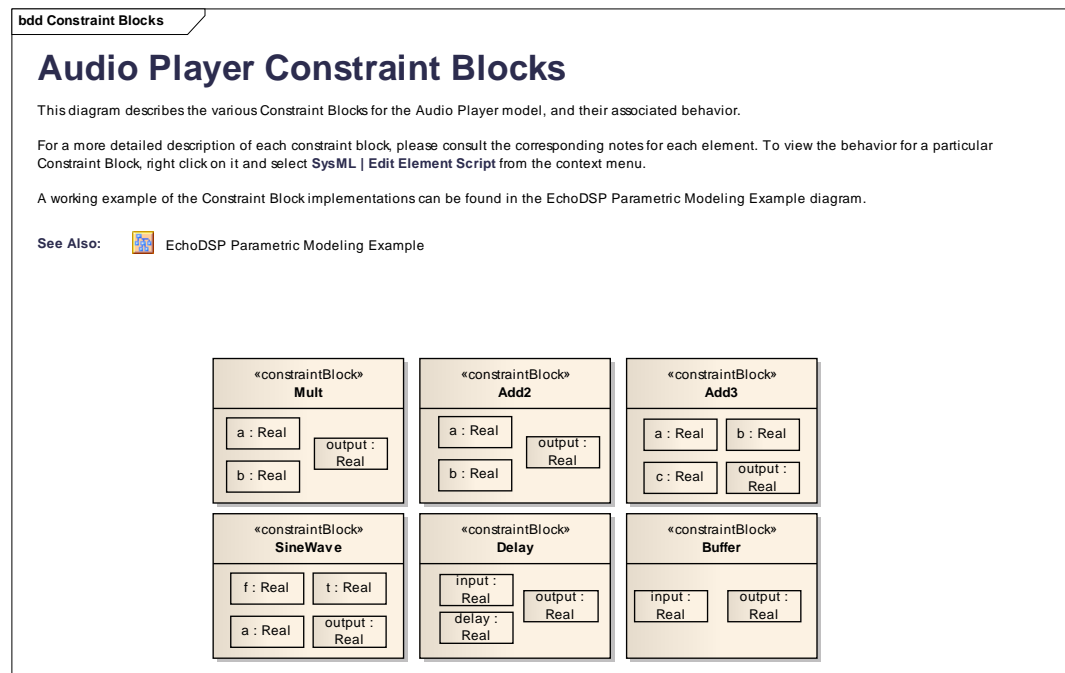


図 3 - 音楽プレーヤー関数のための制約ブロック

次に、シミュレーション対象のパラメトリックモデルを含む制約ブロック要素を作成します。今回の例では、Echo Digital Signal Processing (DSP) 関数をシミュレーションします。

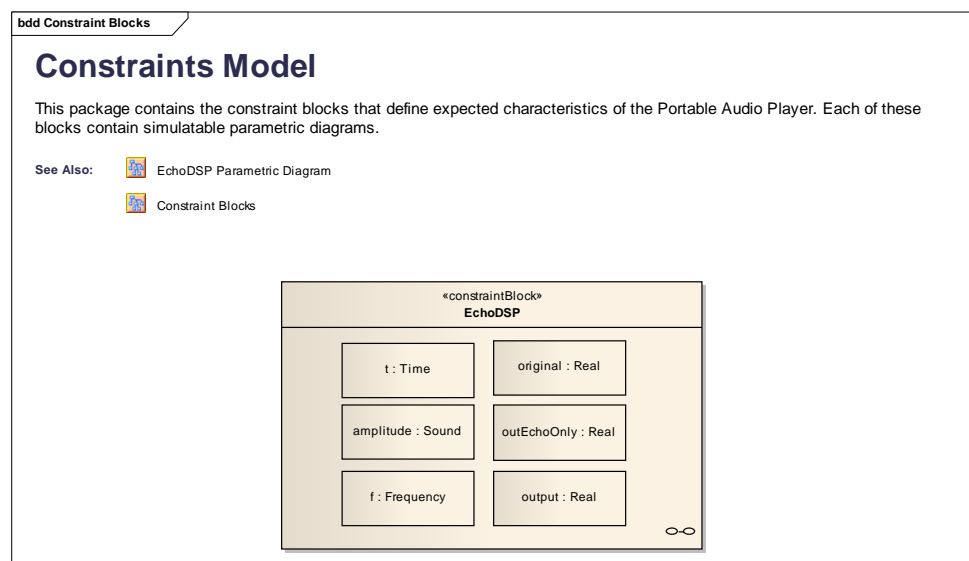


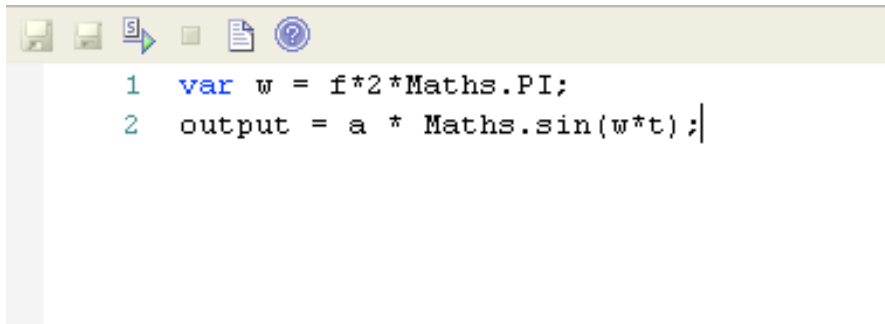
図 4 - Echo DSP 関数のための制約ブロック

図 4 でわかるように、Echo 関数は Sine 波の振幅と周波数で構成される元の信号・エコーを生成するための時刻による信号を入力値とします。また、出力としては、エコーされた信号のみ・元の信号とエコーされた信号を合成した信号の両方となります。この Echo 関数を構成するには、図 3 にある基本的な制約ブロック (Sinewave, Delay, Add) などを利用します。

制約ブロックにスクリプトを追加する

制約ブロックの作成が完了したら、次にスクリプトを追加します。制約ブロックの関係や振る舞いを、実行可能なスクリプトとして表現します。Enterprise Architect で制約ブロック要素にスクリプトを追加するには、制約ブロック要素を右クリックして「SysML」→「スクリプトの追加」を実行します。

図 5 は SineWave 制約ブロックのスクリプトを示しています。その他の制約ブロックにも同様のスクリプトが定義されています。



```

1  var w = f*2*Maths.PI;
2  output = a * Maths.sin(w*t);
    
```

図 5 - SineWave の制約ブロックのためのスクリプト

制約ブロックにスクリプトを関連づけることで、シミュレーションの実行に必要な数学的関数を提供できます。それぞれのブロックの正確な振る舞いは等式として表されます。スクリプトでは入力値と出力値を名前指定して適切に利用することで、シミュレーションを実行可能にします。

モデリングのヒント: Enterprise Architect のスクリプトで利用する言語

スクリプトを作成する際には、JavaScript, Jscript, VBScript のいずれかを利用することができます。また、これらの言語を通して既存のアセンブリ・コンポーネント・API などを利用することもできます。

なお、シミュレーションを実行する場合、対象の制約ブロックの全てが同じ言語で作成されていないことに注意して下さい。

パラメトリック図の定義

パラメトリックモデルは制約ブロックの発生（インスタンス）が 制約プロパティ要素という形で配置され、パラメトリック図でそれぞれが接続されます。

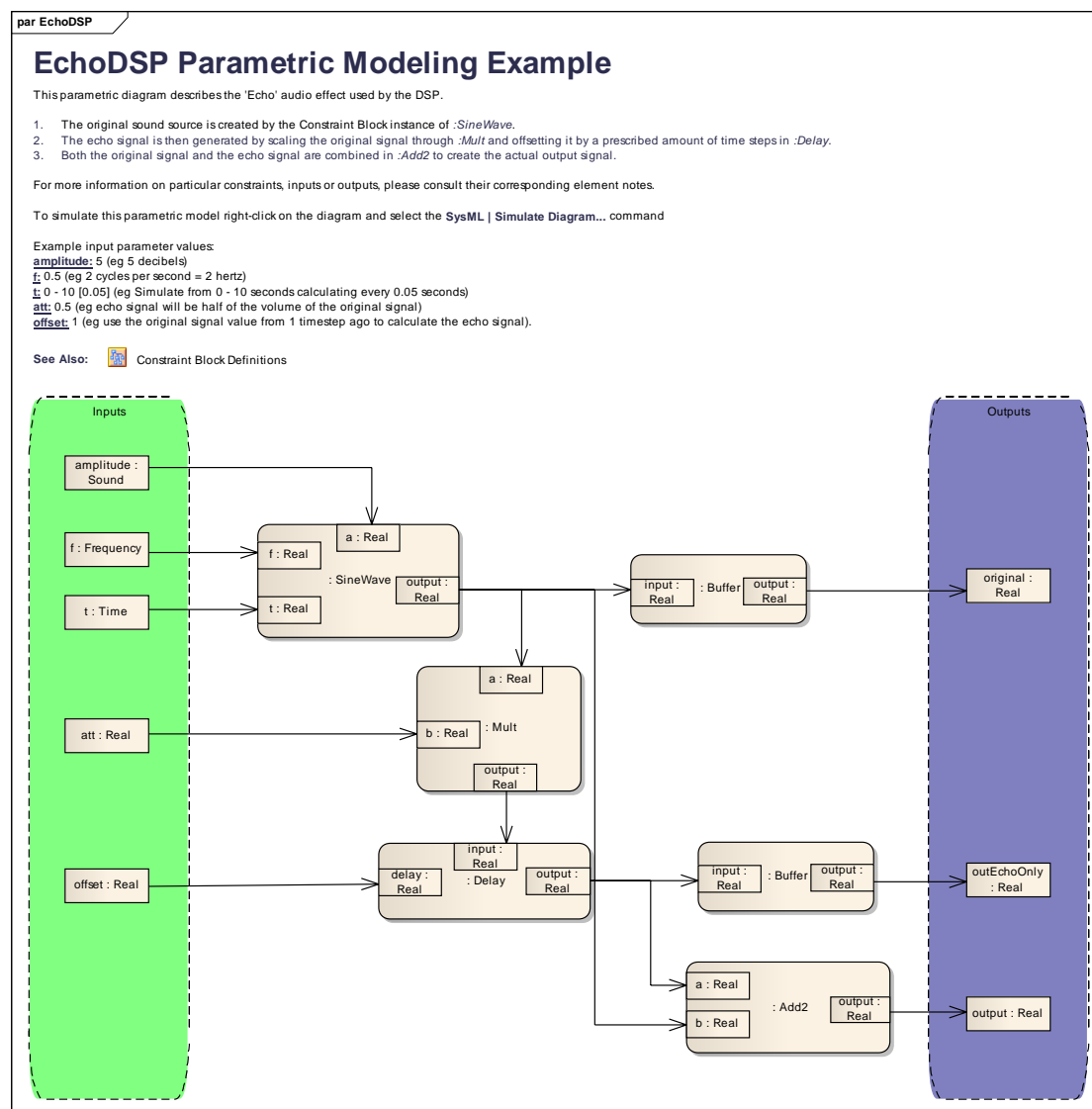


図 6 - Echo DSP のためのパラメトリック図

パラメトリック図では基礎的な制約ブロックのインスタンスについて、入力値を出力値を結びつけます。図 6 では入力値として SineWave・遅延値・減衰値を受け取り、エコー効果を出すために編集前の SineWave に信号を追加します。減衰やオフセットなどのパラメータを調整し、希望する効果が得られるまでシミュレーションを実行します。

このような方法で、ロードマップの 2 番目の作業であり、シミュレーションの設定と実行を行います。

シミュレーションの設定

ここまでの作業で、制約ブロック・スクリプト・パラメトリックが定義できました。これでシミュレーションを実行するための準備ができました。パラメトリック図の背景で右クリックして、「SysML」→「ダイアグラムのシミュレーション」を選択して下さい。シミュレーションの設定ダイアログが表示されます。

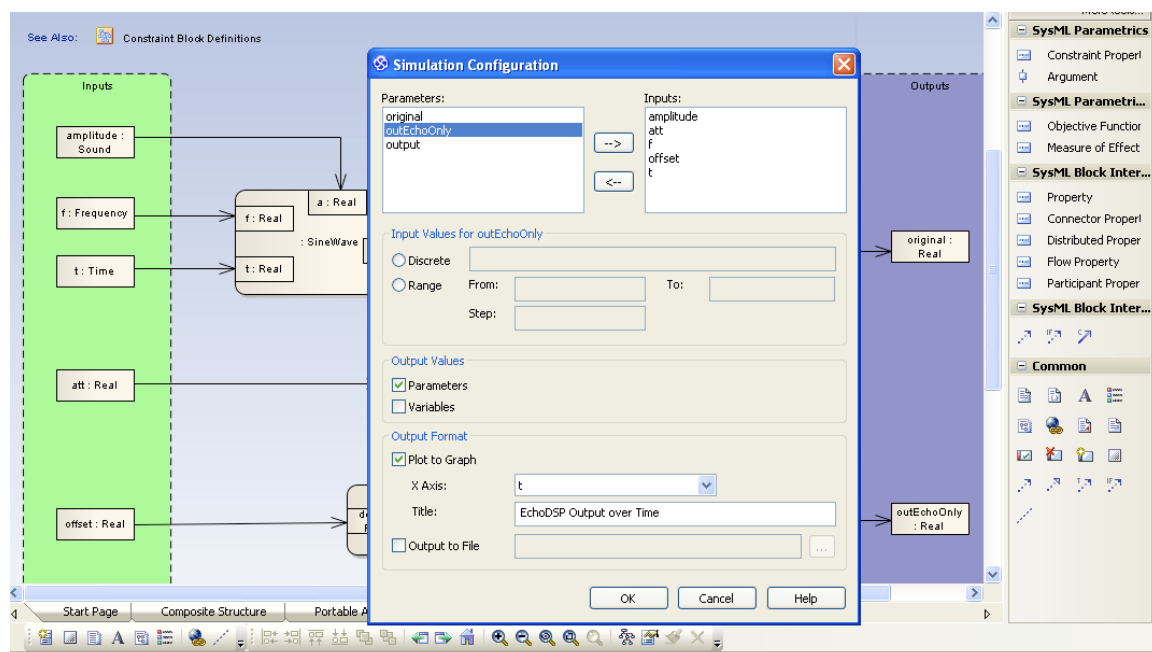


図 6 - シミュレーションの設定

以下に従ってシミュレーションの設定を行ってください。

- **入力値の割り当て**「パラメータ」の一覧では、入力値として割り当て可能な全てのパラメータが表示されています。入力値として指定する項目を選択し、「→」ボタンを押して入力値として指定して下さい。入力値として指定されたパラメータは、「入力」の一覧に表示されます。少なくとも 1 つのパラメータを入力値に指定しなければなりません。
- **入力値に値を割り当てる**それぞれの入力パラメータについて、値を割り当てます。値は、固定値か、あるいは変化する値の上限・下限・増分を指定します。
- **出力値の種類を指定する** パラメータや内部変数を出力するかどうかを指定します。
- **シミュレーションの結果の出力方法を指定する**出力形式のパネルでは、シミュレーションの結果のデータを出力する方法を指定することができます。設定により、CSV 形式のファイルを出力するか、2 次元グラフを表示するかを指定することができます。

シミュレーションの設定が完了したら、実行することができます。

シミュレーションの実行

SysML のシミュレーションを実行するには、ダイアログの OK ボタンを押してください。

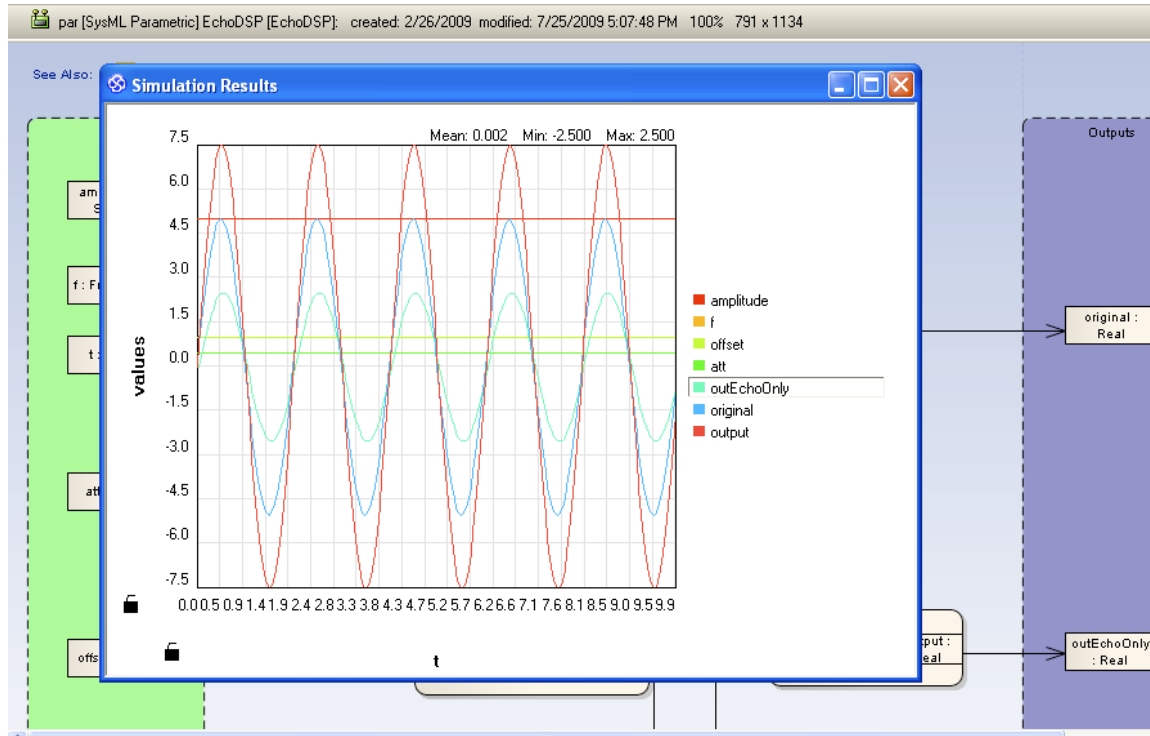


図 7 - シミュレーション結果を直接 Enterprise Architect に表示

CSV ファイルとして外部に結果を出力する機能もありますが、Enterprise Architect 内部で結果をグラフとして表示する機能は、他の SysML 対応ツールにはない特徴です。1 つのツール内で完結することで、パラメータの調整を素早く簡単に行い、システムが要求される性能を満たすようにすることができます。

次の 2 つの章では、SysML のモデルをハードウェアあるいはソフトウェアに変換する方法について説明します。

第 6 章 音楽プレーヤーのハードウェアの実装

第 3 章では、ステートマシン図を利用した振る舞いモデリングについて説明しました。この章では、そのステートマシン図からどのようにハードウェア記述言語 (HDL) のソースコードを生成するかについて、音楽プレーヤーの例で説明します。ソフトウェアの実装については第 7 章で説明します。

ハードウェア実装のロードマップ

HDL でのソースコード生成を通してハードウェアを実装する際のロードマップは 3 つの場合に分かれます。つまり、VHDL, Verilog, SystemC です。この 3 つの場合のいずれの場合でも、Enterprise Architect の機能を利用してステートマシン図からソースコードを生成します。テンプレートのカスタマイズ機能により、強力なソースコード生成機能を実現しています。

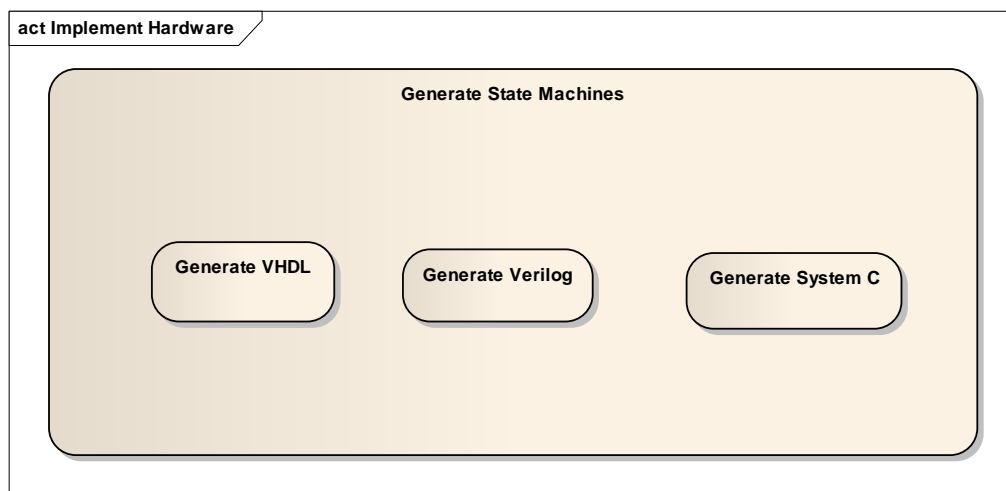


図 1 - 代表的な3つのハードウェア記述言語(HDL)をサポートしたハードウェアの実装のロードマップ

音楽プレーヤーのハードウェアの実装

今までと同様に、音楽プレーヤーの例を利用してロードマップを説明します。今回の説明では、Playback の操作について、VHDL, Verilog, SystemC の実装方法について説明します。

図 2 は音楽プレーヤーのモデルの「実装」の部分の最上位のパッケージ構成を示しています。なお、次の章ではソフトウェアについての同様の構成を示します。これから、ステートマシン図からソースコードを生成する方法について説明し、Playback についての 3 つの HDL ソースコードの結果を示します。

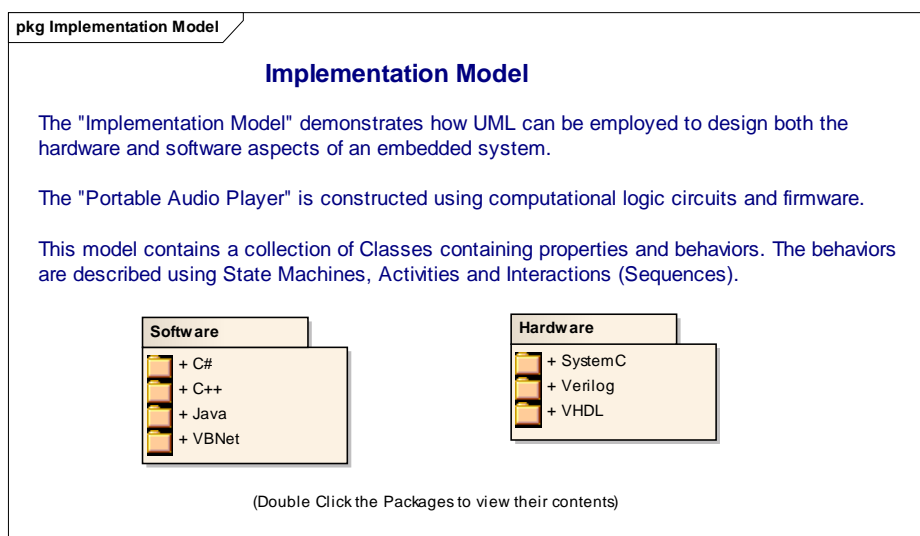


図 2 - 最上位の実装パッケージ

それでは、ハードウェアのパッケージについて詳細を見ていきます。

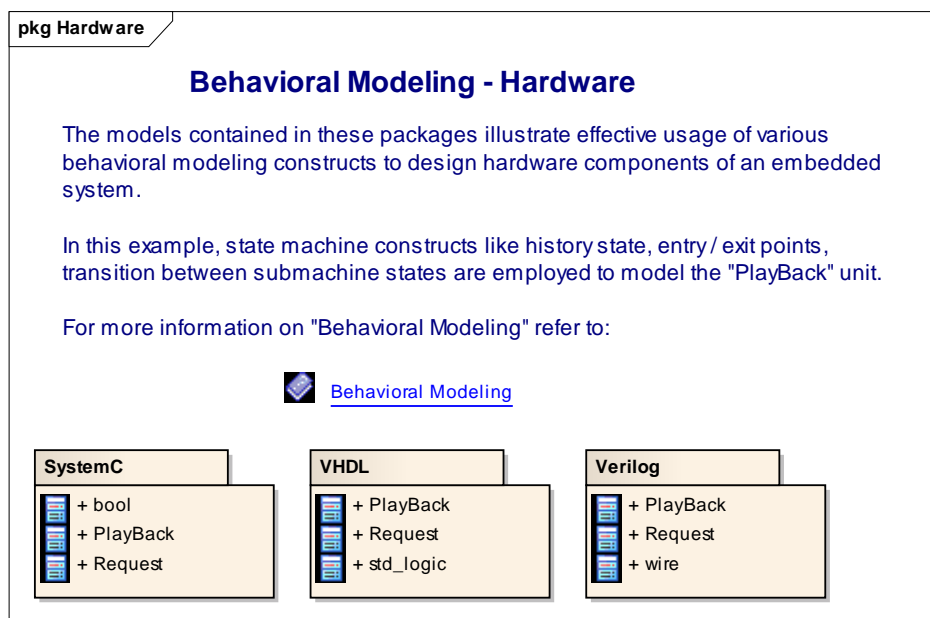


図 3 - 複数言語に対する HDL コード生成に対応した Enterprise Architect

この図を見るとわかるように、ステートマシン図の実装について、3 つとも共通の設計構成になっています。それぞれの場合で、Playback クラスには On と Off の状態を含むステートマシン図が定義されています。On の状態には子ダイアグラム（サブステートマシン図）があり、実際の設計情報が定義されています。

HDL のステートマシン図を作成する手順は 3 つのステップから構成されます。

1. 関係する（影響を受ける）トリガを明示する
2. ポートとトリガの関連付けをする
3. アクティブ状態の状態遷移を定義する

それぞれのステップを順に見ていきましょう。

1.関係する（影響を受ける）トリガを明示する

最上位のステートマシン図は、ハードウェアコンポーネントが持つモード（状態）を定義するために利用しなければなりません。図 4 のように、モードを変化させる、関係するトリガも定義します。

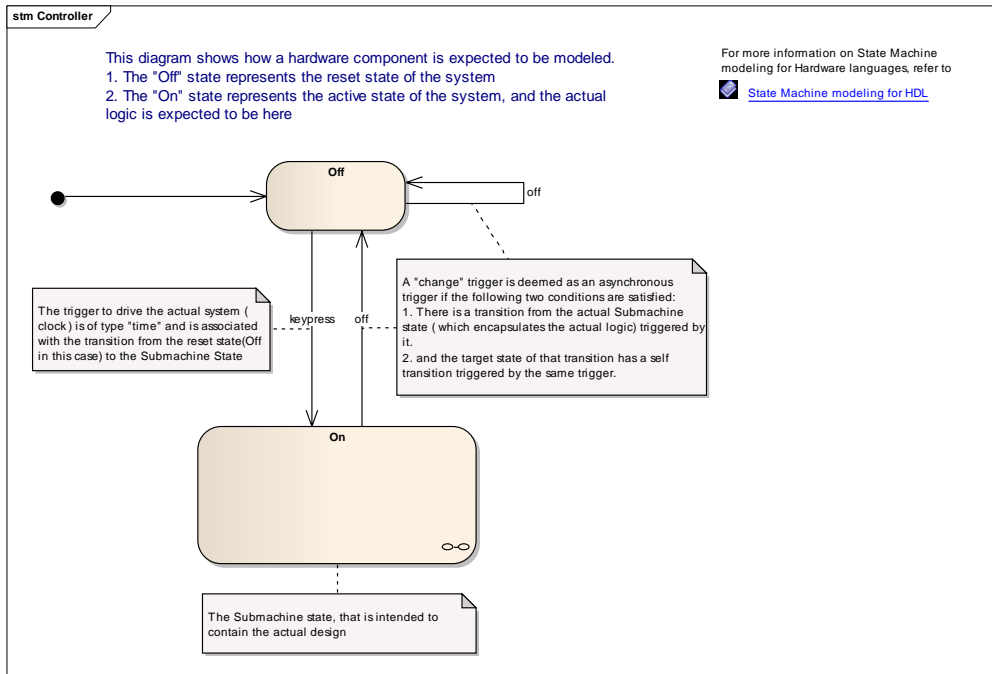


図 4 - 操作モードと作動トリガ指定に使用される最上位ステートマシン図

トリガにはいくつかの種類があります。

非同期トリガ

非同期トリガは次の条件を満たす場合のみ利用されます。

- 「変更 (Change)」のトリガ(取り得る値: true / false)
- そのトリガによってアクティブ状態（サブ状態）へ遷移する
- そのトリガで、そのアクティブ状態自身に自己遷移する

クロック

「時間 (time)」の種類トリガは、アクティブ状態（サブ状態）への遷移がクロックであると見なせる場合に利用します。このトリガの取り得る値は対象の言語により異なります。

Trigger Type	Language	Specification	
		Positive Edge Triggered	Negative Edge Triggered
Time	VHDL	rising_edge	falling_edge
	Verilog	posedge	negedge
	SystemC	positive	negative

図 5 - クロックのトリガー仕様

2.ポートの定義とトリガとの関連付け

コンポーネントの異なる動作モードを定義した後は、トリガとポートを関連づけ、トリガを受け取ることを明示します。図 6 のようにコンポーネントのポートとトリガを結びつけます。

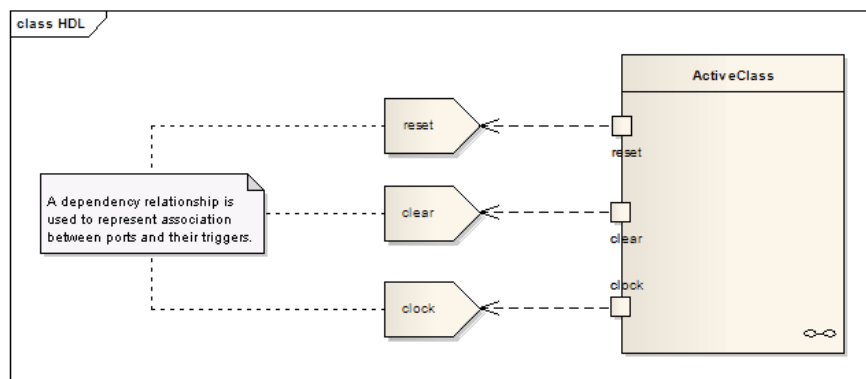


図 6 - ポートをトリガーに関連付けるために依存関係を使用

3.アクティブ状態の処理を定義する

上記の 2 つの作業は、ハードウェアコンポーネントの効率的な実装のために必要な準備段階です。実際の状態遷移を示すステートマシン図は、アクティブ状態（サブ状態）の子ダイアグラムとしてモデリングします。

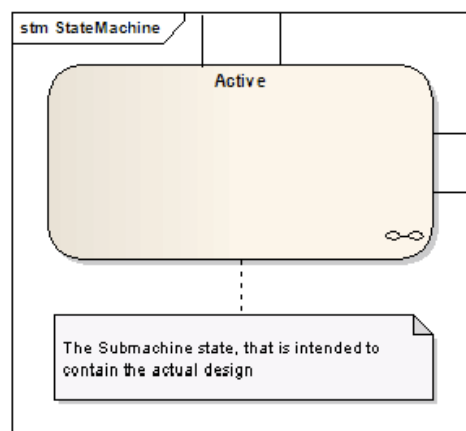


図 7 - アクティブ状態に使用する子サブマシンに特定されたアクティブロジック

ステップ 3 の VHDL、Verilog、System C を詳しく見てみましょう。

VHDL での実装

図 8 は Playback のクラス図です。入出力のポートが定義されています。Playback 機能の設計には、Playback クラスの子要素として複数の階層に渡る状態マシン図が含まれています。

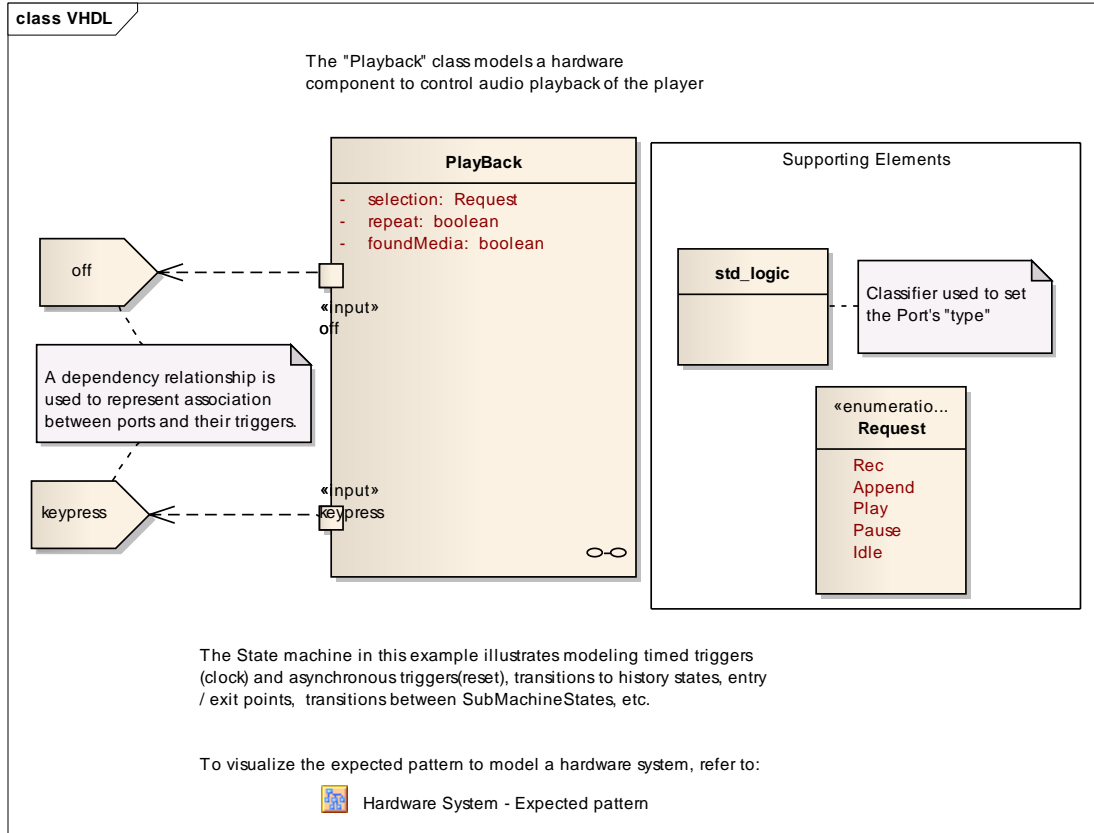


図 8 - Playback クラス内に入れ子になったサブステートから生成される VHDL ソースコード

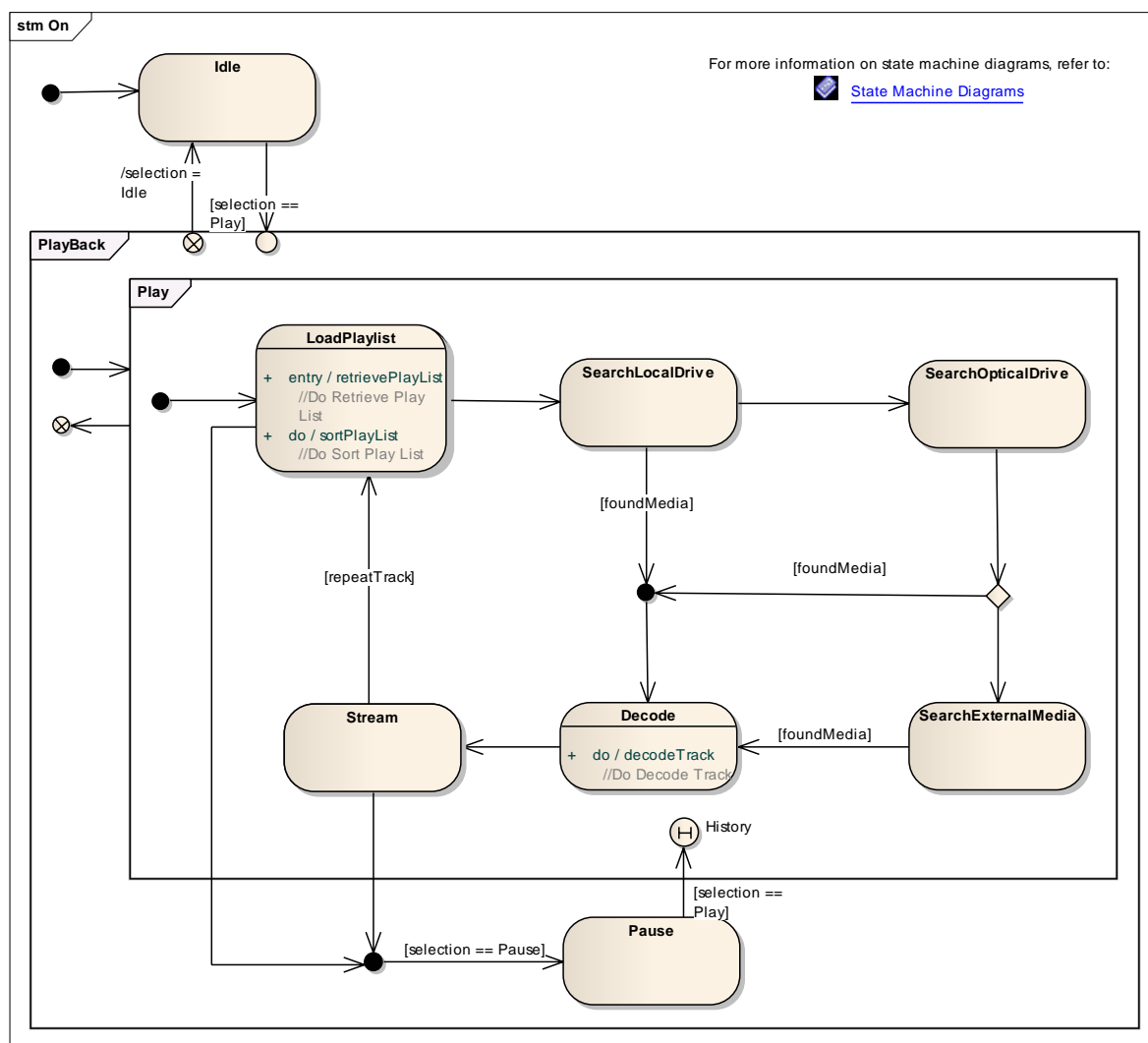


図 9 - Playback のステートマシン図

図 9 のステートマシン図は Verilog や SystemC の実装の場合でも基本的に同じです。両者の違いは非常に小さいので、ここではあえて説明しません。

VHDL のソースコード生成とリバースエンジニアリング

Enterprise Architect では VHDL コードのラウンドトリップ開発にも対応しています。次のステレオタイプやタグ付き値を利用して、情報の同期を行っています。

図 12 と図 13 は Enterprise Architect でソースコード生成した結果の一部分です。

```

1  -----
2  -- Playback.vhdl
3  -- Implementation of the Class Playback
4  -- Created on:    11-Aug-2009 6:29:06 PM
5  -- Original author: mnizam
6  -----
7  library IEEE;
8  use IEEE.std_logic_1164.all;
9  use IEEE.std_logic_arith.all;
10 use IEEE.std_logic_textio.all;
11 use STD.textio.all;
12 entity Playback is
13 Port
14 (
15     off : in std_logic;
16     keypress : in std_logic
17 );
18 end Playback;
19 architecture Controller of Playback is
20 signal selection : Request;
21 signal foundMedia : boolean;
22 signal repeat : boolean;
23
24 type StateType is
25 (
26     Controller_Off,
27     Controller_On,
28     Controller_On_PlayBack,
29     Controller_On_PlayBack_Pause,
30     Controller_On_PlayBack_Play,
31     Controller_On_PlayBack_Play_Decode

```

図 12 - Enterprise Architect によるステートマシン図からの VHDL ソースコードの生成

なお、生成された VHDL のソースコードは非常に詳細であり、堅牢なコードになっています。

```

74     end process CLOCK_DIV2;
75
76 CLOCK_DIV4 : process(clockDiv2 , off)
77 begin
78     if(off = '1') then
79         clockDiv4 <= '1';
80     elsif(rising_edge(clockDiv2)) then
81         clockDiv4 <= not clockDiv4;
82     end if;
83 end process CLOCK_DIV4;
84
85 TRANSITION_LOGIC: process(currState , off)
86 variable bFlag : boolean; begin
87     currTransition <= TT_NOTTRANSITION;
88     if(off = '1') then
89         nextState <= Controller_Off;
90         currTransition <= TT_NOTTRANSITION;
91     else
92     case currState is
93         when Controller_Off =>
94             bFlag := false;
95             nextState <= Controller_On_Idle;
96             transcend <= '1';
97             Controller_history <= currState;
98         when Controller_On =>
99             nextState <= Controller_On_Idle;
100            currTransition <= Controller_On_PlayBack_to_Controller_On_Idle;
101            transcend <= '1';
102            Controller_On_PlayBack_history <= currState;
103         when Controller_On_PlayBack =>
104            nextState <= Controller_On_Idle;

```

図 13 - 完璧な実装を実現する SysML パート/ポート、VHDL、ステレオタイプ、タグ値と結合した Enterprise Architect のステートマシン図のコード生成機能

Verilog の実装

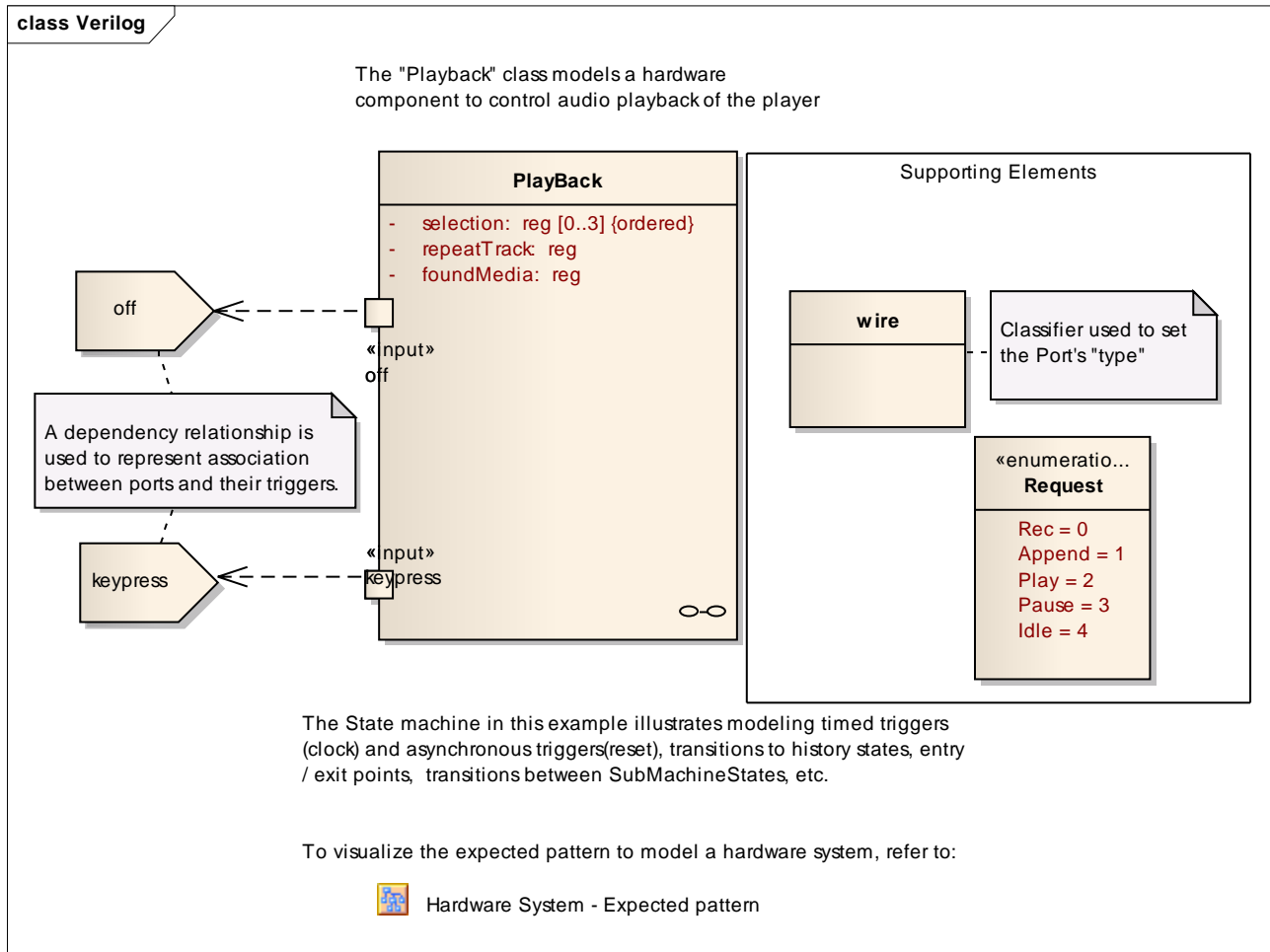


図 14 - Verilog 実装の Playback クラス図

Enterprise Architect では Verilog コードのラウンドトリップ開発にも対応しています。次のステレオタイプやタグ付き値を利用して、情報の同期を行っています。

Embedded Systems Development using SysML

```

1 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2 // Playback.v
3 // Implementation of the Class Playback
4 // Created on: 11-Aug-2009 6:32:44 PM
5 // Original author: mnizam
6 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
7
8
9 module Playback
10 {
11     keypress,
12     off
13 };
14 parameter SIZE = 4;
15 input wire keypress;
16 input wire off;
17
18 parameter /*{StateType}*/
19     Controller_Off = 0,
20     Controller_On = 1,
21     Controller_On_PlayBack = 2,
22     Controller_On_PlayBack_Pause = 3,
23     Controller_On_PlayBack_Play = 4,
24     Controller_On_PlayBack_Play_SearchLocalDrive = 5,
25     Controller_On_PlayBack_Play_LoadPlaylist = 6,
26     Controller_On_PlayBack_Play_Decode = 7,
27     Controller_On_PlayBack_Play_SearchExternalMedia = 8,
28     Controller_On_PlayBack_Play_SearchOpticalDrive = 9,
29     Controller_On_PlayBack_Play_Stream = 10,
30     Controller_On_Idle = 11,
31     ST_MOSTATE = 12;

```

図 17 - Enterprise Architect による Verilog コード生成

SystemC の実装

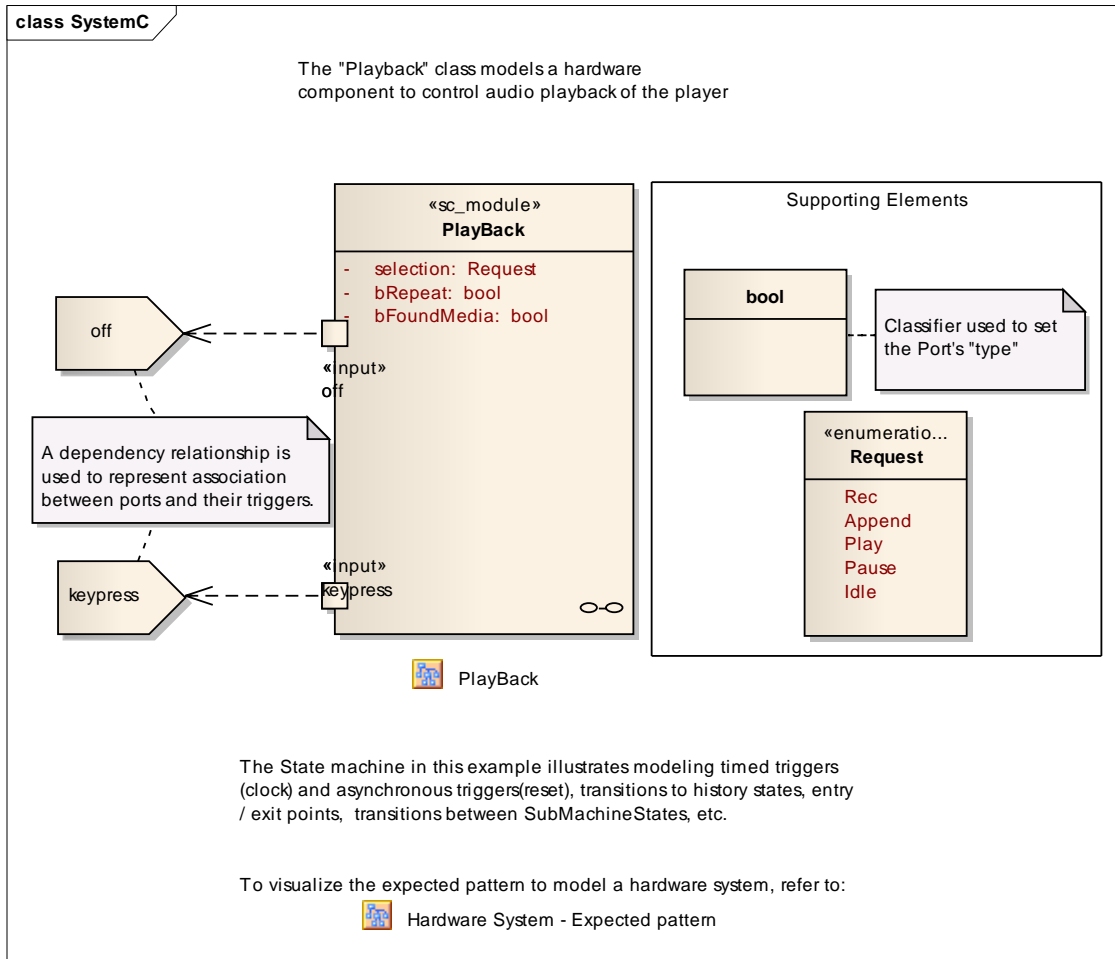


図 18 – System C 実装の Playback クラス図

Enterprise Architect では SystemC コードのラウンドトリップ開発にも対応しています。次のステレオタイプやタグ付き値を利用して、情報の同期を行っています。

```

1  ////////////////////////////////////////////////////////////////////
2  // Playback.sc
3  // Implementation of the Class Playback
4  // Created on: 11-Aug-2009 6:31:54 PM
5  // Original author: mnizam
6  ////////////////////////////////////////////////////////////////////
7
8  #include "systemc.h"
9  SC_MODULE ( Playback )
10 {
11     Request selection;
12     bool bRepeat;
13     bool bFoundMedia;
14
15     sc_in_clk keypress;
16     sc_in <bool> off;
17     /* -----Begin - Code rendered by EA for the underlying Behavioral Model-----
18
19     enum StateType
20     {
21         Controller_On,
22         Controller_On_PlayBack,
23         Controller_On_PlayBack_Pause,
24         Controller_On_PlayBack_Play,
25         Controller_On_PlayBack_Play_SearchLocalDrive,
26         Controller_On_PlayBack_Play_SearchExternalMedia,
27         Controller_On_PlayBack_Play_SearchOpticalDrive,
28         Controller_On_PlayBack_Play_Decode,
29         Controller_On_PlayBack_Play_LoadPlaylist,
30         Controller_On_PlayBack_Play_Stream,
31         Controller_On_Idle
    
```

図 21 – Enterprise Architect による System C コード生成

第7章 音楽プレーヤーのソフトウェアの実装

Enterprise Architect には数多くのソースコード生成・読み込みに関する機能が搭載されています。また、MDGテクノロジーを通して Visual Studio や Eclipse の開発環境と密接に連携します。フォワード、リバースエンジニアリングを含む、多くの Enterprise Architect のソースコードエンジニアリング機能と、Enterprise Architect の強力なコード生成テンプレートのフレームワークは、ヘルプファイルなどで説明しています。この章では、Sparx System 特有の 振る舞い図からのソースコード生成機能と、IDEとの連携機能について説明します。図 1 は、ソフトウェアの実装に関するロードマップです。

ソフトウェアの実装のロードマップ

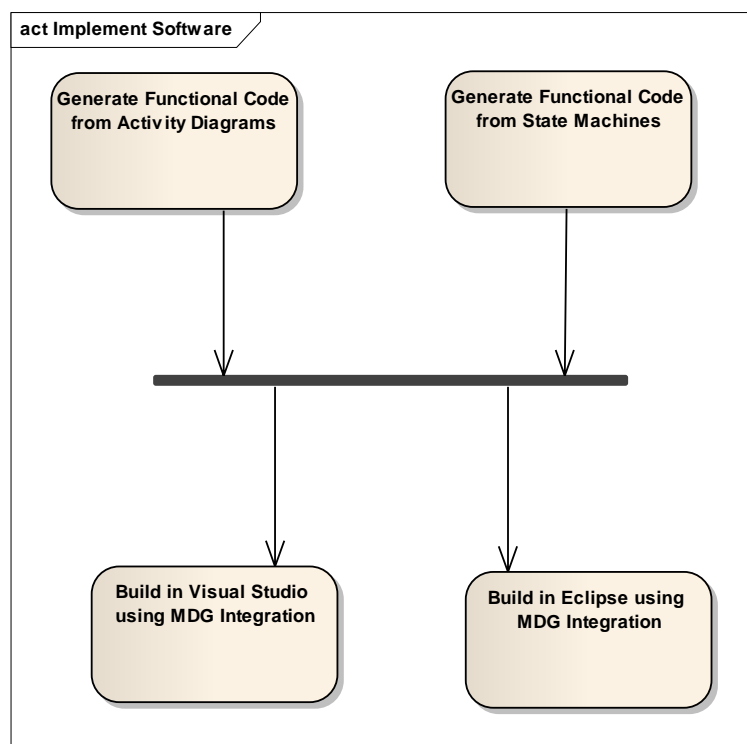


図 1 - ソフトウェアの実装に関するロードマップ

ソースコード生成可能な振る舞い図

Enterprise Architect は、クラスの操作として要素の振る舞いを定義することができます。具体的には、アクティビティ・相互作用・アクション・相互作用の発生などの振る舞い要素を利用して振る舞いを定義します。

この章では、第 3 章で説明したような振る舞いモデルをどのように C#, C++, Java, VisualBasic.NET のソースコードに変換するのかを説明します。図 2 は音楽プレーヤーの最上位のパッケージを示すダイアグラムです。この図を利用して、振る舞い図からのソースコード生成について説明します。

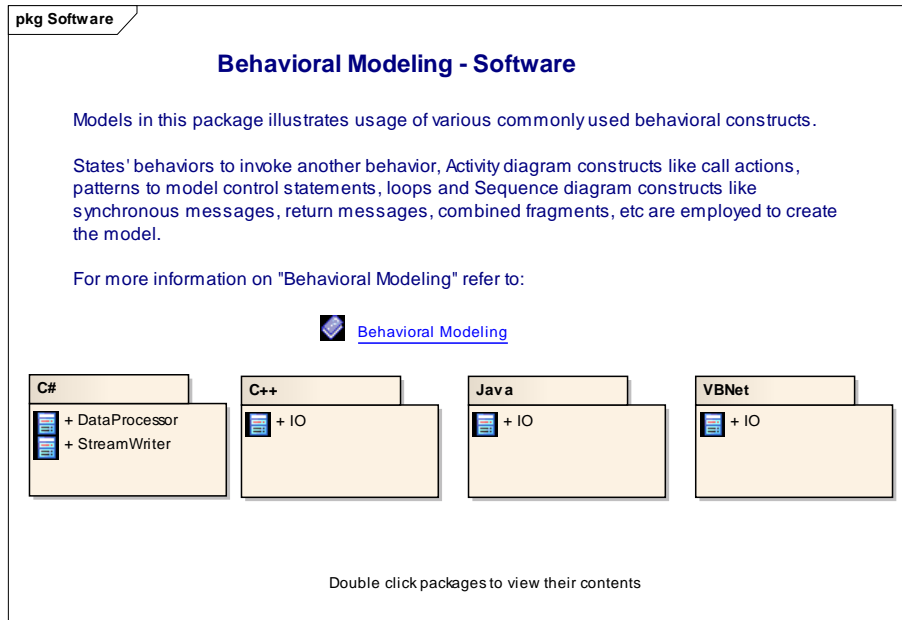


図 2 - 振る舞いソースコード生成のための音楽プレーヤーサンプル組織図

Enterprise Architect では、クラス図からのソースコード生成に加えて、以下のような振る舞い図からのソースコード生成も可能になっています。

- ステートマシン図
- シーケンス図
- アクティビティ図

この章では、この振る舞い図からのソースコード生成について詳細に確認します。最初に、音楽プレーヤーの DataProcessor クラスに定義されているステートマシン図とアクティビティ図からの C# のソースコード生成について見ていきます。

図 3 は DataProcessor クラスのクラス図です。このクラスには、子要素として Searching External Media というステートマシンと Appending to a Buffer というアクティビティを持っています。図 4 はコンポジット構造図を利用して、振る舞いの概要を示しています。

データプロセッサ: ステートマシン図とアクティビティ図からの C# ソースコード生成

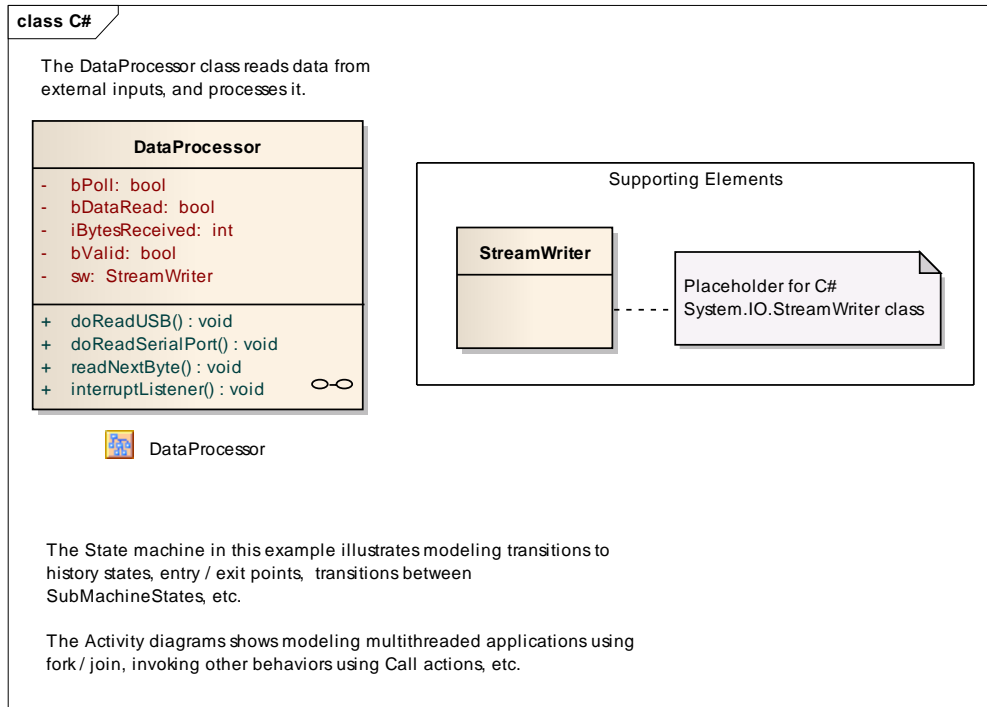


図 3 - DataProcessor のクラス図

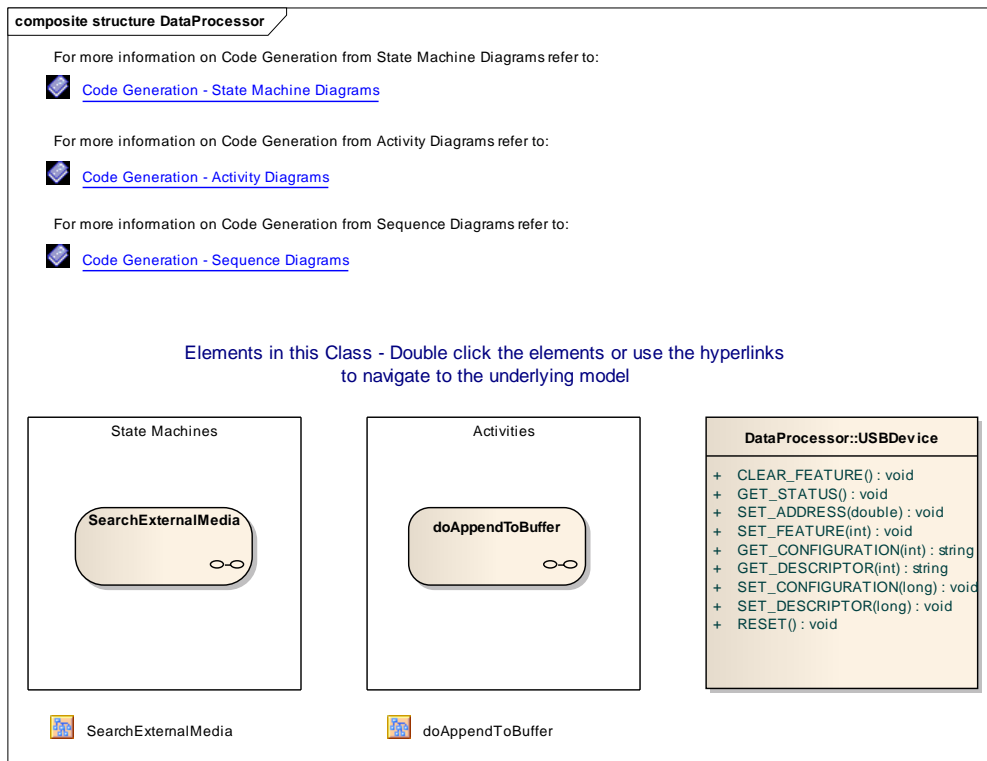


図 4 - コンポジット構造図を利用した DataProcessor の入れ子になった振る舞い図

Enterprise Architect での振る舞い図からのソースコード生成は、対象のクラスの子要素として振るまいが定義されなければなりません。図 5 はプロジェクトブラウザの構成を示しています。

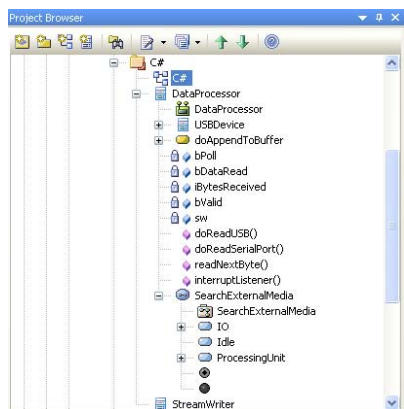


図 5 - 親クラス内に入れ子になった、ソースコード生成される振る舞い

図 6 にある状態マシン図は、SearchExternalMediaに必要です。また、図 7 にある Do、Entry、Exit の状態にはソースコードが自動生成されているのがご覧になれると思います。状態マシン図の振る舞いは、すべて自動生成されたものです。

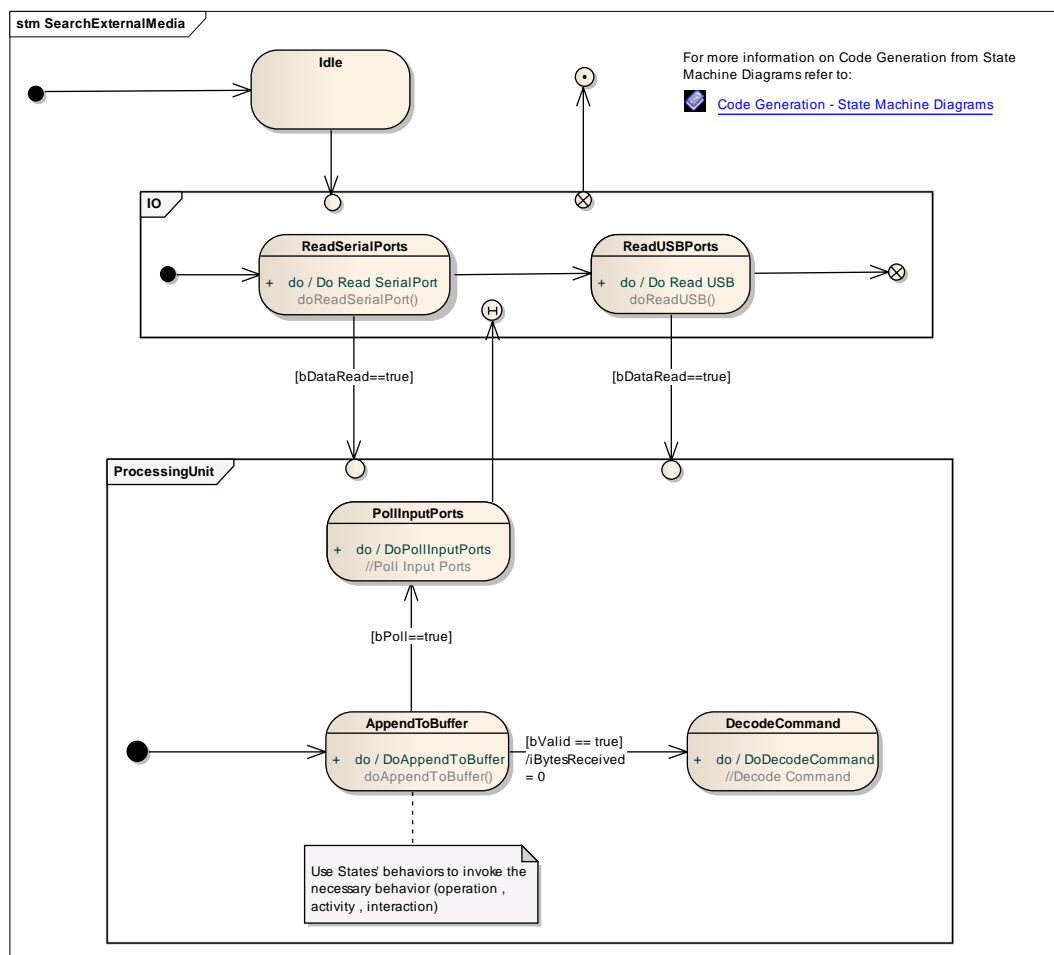


図 6 - 外部メディア検索の状態マシン図

なお、ProcessingUnit には、サブ状態として PollInputPorts, AppendToBuffer, DecodeCommand の 3 つの状態を持ちます。AppendToBuffer について、ネストしたアクティビティ図からの生成は図 8 と図 9 に示しています。図 7 は、この状態マシン図から生成したソースコードの一部です。

```

DataProcessor.cs
155
156 /* Begin - EA generated code for StateMachine */
157
158
159 private enum StateType : int
160 {
161     SearchExternalMedia,
162     SearchExternalMedia_ProcessingUnit,
163     SearchExternalMedia_ProcessingUnit_PollInputPorts,
164     SearchExternalMedia_ProcessingUnit_AppendToBuffer,
165     SearchExternalMedia_ProcessingUnit_DecodeCommand,
166     SearchExternalMedia_IO,
167     SearchExternalMedia_IO_ReadUSBPorts,
168     SearchExternalMedia_IO_ReadSerialPorts,
169     SearchExternalMedia_Idle,
170     ST_NOSTATE
171 }
172 private enum TransitionType : int
173 {
174     SearchExternalMedia_ProcessingUnit_AppendToBuffer_to_SearchExternalM
175     TT_NOTTRANSITION
176 }
177 private enum CommandType : int
178 {
179     Do,
180     Entry,
181     Exit
182 }
183 private StateType currState;
    
```

図 7 - Enterprise Architect によるステートマシン図の振る舞いソースコード生成

図 8 は AppendToBuffer についてのアクティビティ図です。図 9 はこのアクティビティ図から出力した C# のソースコードの一部です。

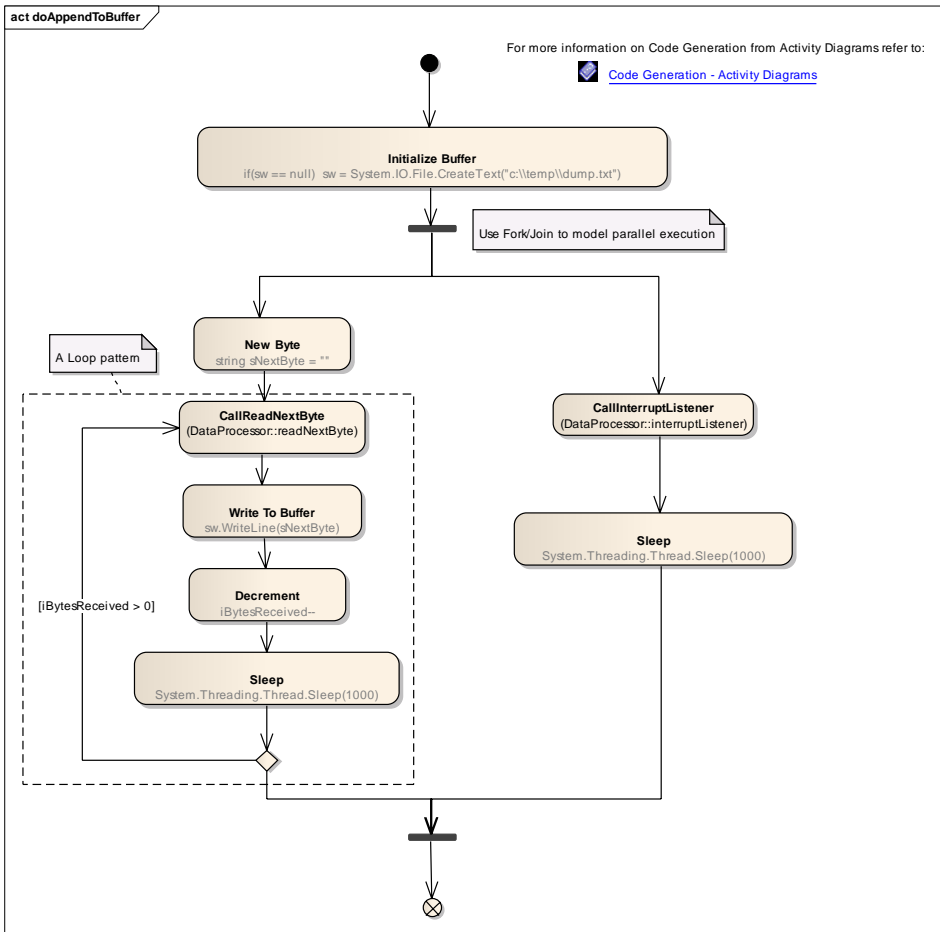
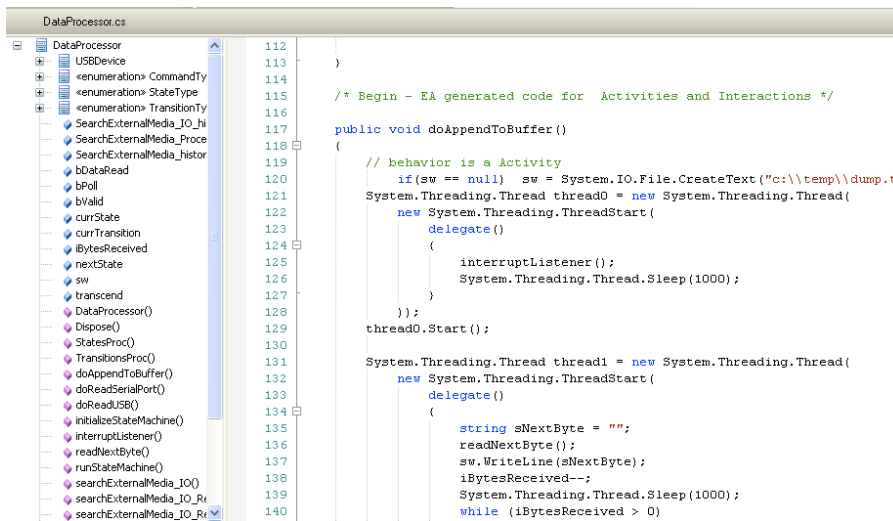


図 8 - AppendToBuffer のためのアクティビティ図

Embedded Systems Development using SysML



```
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140

/* Begin - EA generated code for Activities and Interactions */

public void doAppendToBuffer()
{
    // behavior is a Activity
    if (sw == null) sw = System.IO.File.CreateText("c:\\temp\\dump.tx");
    System.Threading.Thread thread0 = new System.Threading.Thread(
        new System.Threading.ThreadStart(
            delegate()
            {
                interruptListener();
                System.Threading.Thread.Sleep(1000);
            }
        ));
    thread0.Start();

    System.Threading.Thread thread1 = new System.Threading.Thread(
        new System.Threading.ThreadStart(
            delegate()
            {
                string sNextByte = "";
                readNextByte();
                sw.WriteLine(sNextByte);
                iBytesReceived--;
                System.Threading.Thread.Sleep(1000);
                while (iBytesReceived > 0)
            }
        ));
}
```

図 9 - AppendToBuffer のための生成 C# ソースコード

繰り返しになりますが、ダイアグラムで詳細化された振る舞いのすべてについて、自動的にソースコードに出力されます。

IO -C++, Java, VB.Net でのソースコード生成

次に、C++, Java, VB.NET でのソースコード生成について、IO クラスを例に確認します。図 10 は C++ の場合のクラス図の例です。なお、ここには掲載していませんが、Java および VB.NET についても同様のクラス図になっています。

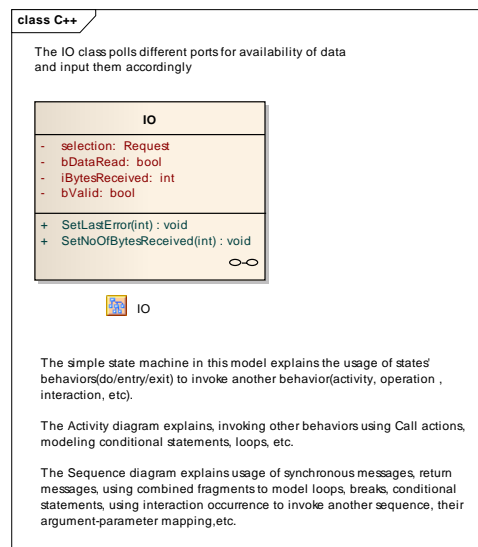


図 10 - IO のクラス図

DataProcessor の例では、ソースコード生成の対象となる全ての振る舞いはこの IO クラスの子要素として定義されています。

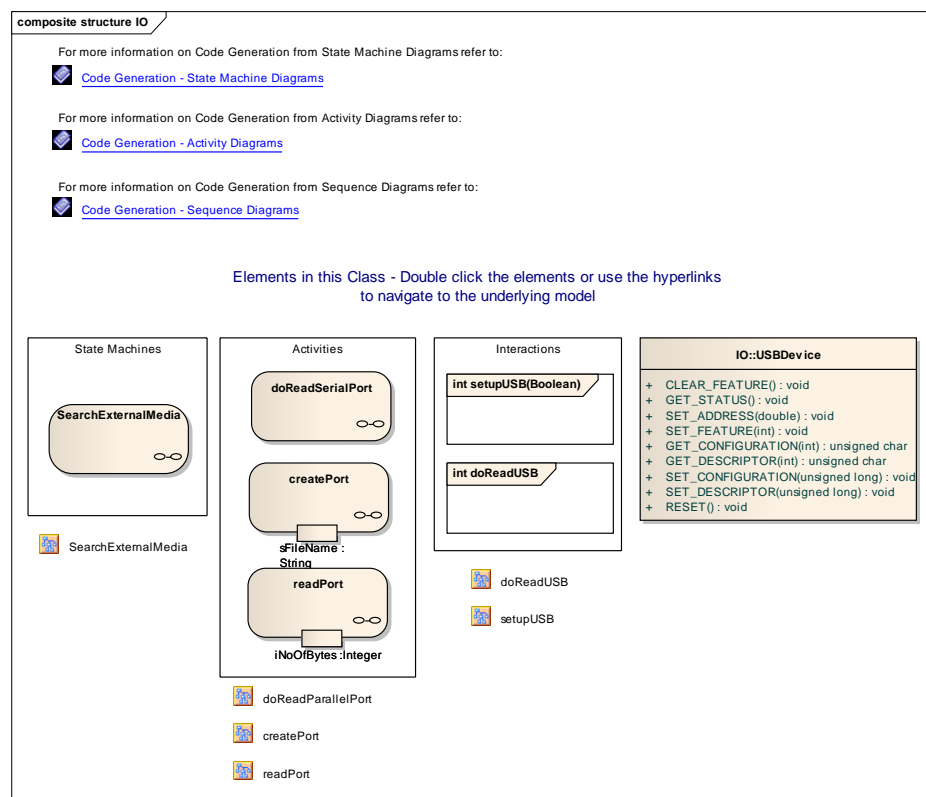


図 11 - IO の入れ子になった振る舞い

Embedded Systems Development using SysML

ここでは、シーケンス図・状態マシン図・アクティビティ図からのソースコード生成の結果を確認します。図 12 は USB ポートの設定に関するシーケンス図です。図 13 は USB ポートからの読み込みについての定義です。図 14 は自動的に生成された C++ のソースコードです。

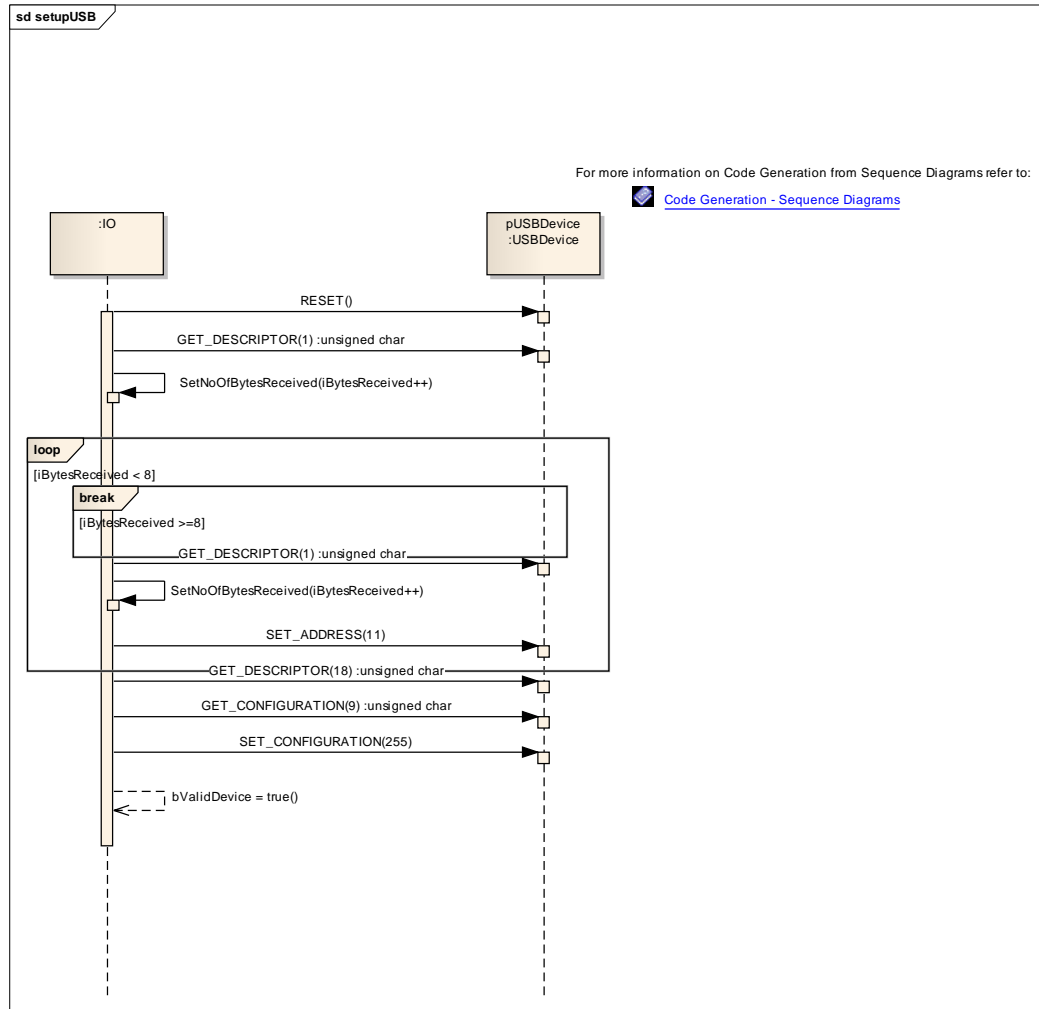


図 12 - ループフラグメントを表した USB 設定のためのシーケンス図

Embedded Systems Development using SysML

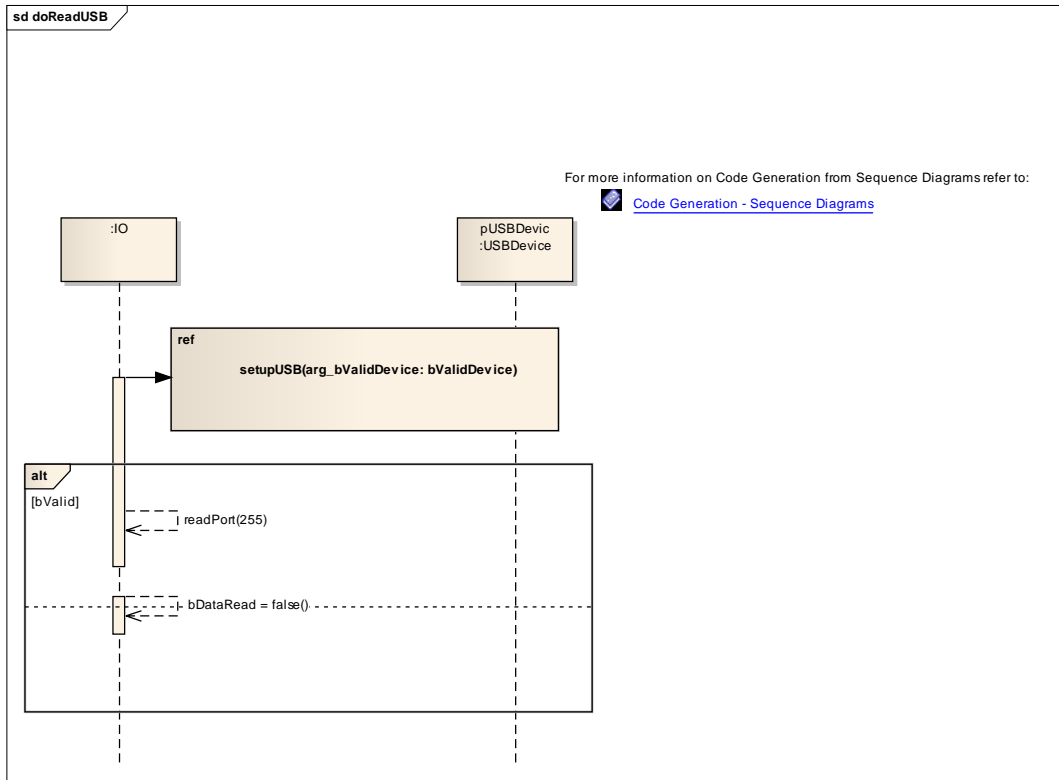


図 13 - USB ポートの読み込み

```

IO.h
73 {
74     TT_NOTTRANSITION
75 };
76 enum CommandType
77 {
78     Do,
79     Entry,
80     Exit
81 };
82 private:
83     StateType currState;
84     StateType nextState;
85     TransitionType currTransition;
86     bool transcend;
87     StateType SearchExternalMedia_history;
88     void searchExternalMedia_ReadSerialPorts(CommandType command);
89
90     void searchExternalMedia_ReadUSBPorts(CommandType command);
91     void StatesProc(StateType currState, CommandType command);
92     void TransitionsProc(TransitionType transition);
93     void initializeStateMachine();
94     void runStateMachine();
95
96     /* End - EA generated code for StateMachine */
97 };
100 #endif // !defined(EA_789E55B0_BBD8_4c3d_B0B3_39C99FB6BAB2_INCLUDED_)
101

```

図 14 - IO のための生成 C++ ソースコード

ステートマシン図・アクティビティ図から Java, VB.NET のコードの生成

ここまでの説明で、ステートマシン図・アクティビティ図・シーケンス図について、言語を問わずソースコード生成を行うという考え方を理解できたのではないかと思います。図 16 は、図 15 のステートマシン図から生成した VB.NET のソースコードを示しています。

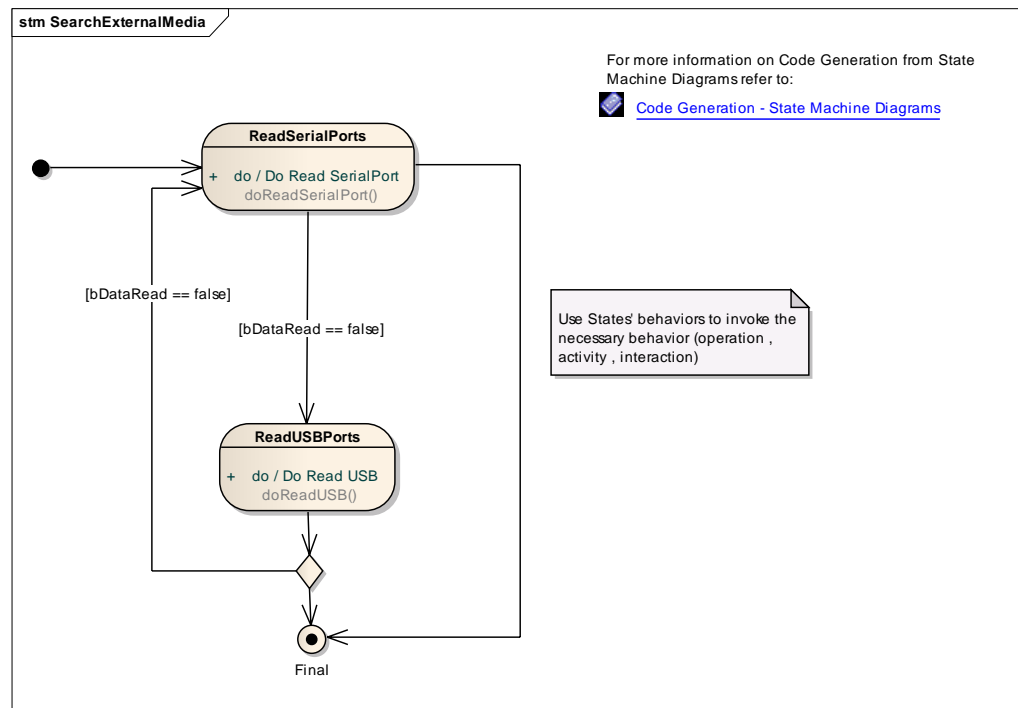


図 15 - 外部メディア検索のステートマシン図

```

IO.vb
160
161     TT_NOTTRANSITION
162 End Enum
163 Private Enum CommandType
164     BehDo
165     BehEntry
166     BehExit
167 End Enum
168 private currState As StateType
169 private nextState As StateType
170 private currTransition As TransitionType
171 private transcend as Boolean
172 private SearchExternalMedia_history As StateType
173 Private Sub searchExternalMedia_ReadSerialPorts(ByVal comma
174     Select Case command
175         case CommandType.BehDo
176             'Do Behaviors..
177             doReadSerialPort()
178             'State's Transitions
179             Dim bFlag as Boolean
180             If (bDataRead = false) Then
181                 bFlag = true
182                 nextState = StateType.SearchExternalMedia_R
183             End If
184             If ( bFlag = False ) Then
185                 nextState = StateType.ST_NOSTATE'Final Stat
186             End If
187         End Select
188     End Sub
    
```

図 16 - 上記ステートマシン図からの VB.Net 振る舞いソースコードの自動生成

最後に、アクティビティ図と、この図から生成された Java のソースコードを図 17/18 で示します。

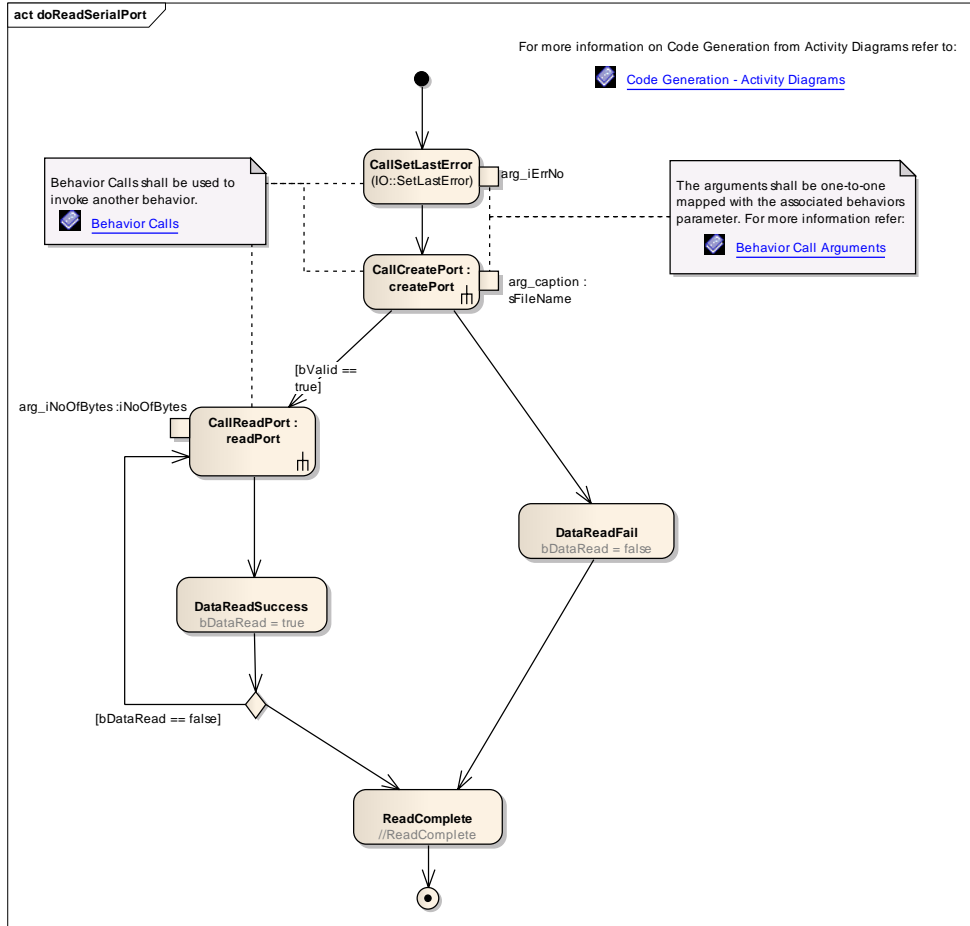


図 17 - シリアルポート読み込みのアクティビティ図

```

IO.java
123 public void createPort(String sFileName)
124 {
125     // behavior is a Activity
126     //Create a handle to port;
127 }
128
129 public void doReadSerialPort ()
130 {
131     // behavior is a Activity
132     SetLastError(0);
133     createPort("LPT1");
134     if (bValidPort)
135     {
136         readPort(1);
137         bDataRead = true;
138         while (bDataRead == false)
139         {
140             readPort(1);
141             bDataRead = true;
142         }
143     }
144     else
145     {
146         bDataRead = false;
147     }
148     //ReadComplete;
149 }
150
151 public void setupUSB(boolean bValidDevice)
    
```

図 18 - シリアルポート読み込みの振る舞い生成 Java ソースコード

ソースコード生成のカスタマイズ

Enterprise Architect ではソースコード生成においてテンプレートを利用しています。また、テンプレートを編集するためのエディタも利用できます。

ソースコード生成のテンプレートは、UML の要素から指定された言語のソースコードの一部に変換する方法を指定するものです。テンプレートはテキストで定義され、その定義に使用した構文はマークアップ言語とスクリプト言語、それぞれの一部の側面を持っています。

図 20 は、C# ソースコードの生成方法を追跡するのに使用するコードテンプレートエディタを示しています。

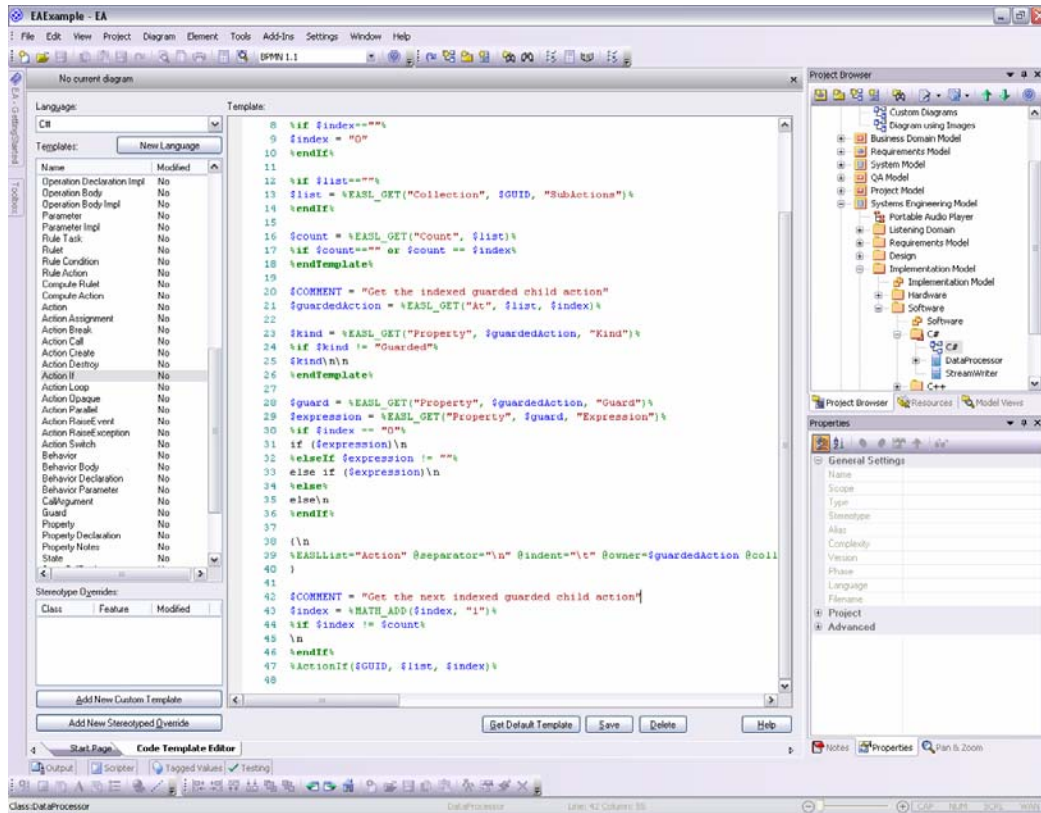


図 20 - C# のソースコード生成のテンプレートのアクション

ソースコード生成のテンプレートは、プレーンテキストで書かれています。テンプレートの構文は、次の 3 つの基本構造で成り立っています。

- 文字テキスト
- マクロ
- 変数

テンプレートはこの 3 つのいずれか、もしくはすべてを含むことができます。

IDE にモデルとソースコードを連携する

モデルの初期から、モデルとソースコード間のギャップ（時にそれは大変深い）にはいつも問題がありました。ソースコードには現実味がありましたが、議論が進んでもモデルには現実味が欠けていたためにその価値が落ち、モデル化を飛ばしてソースコード化が進められていました。

その頃は、リバースエンジニアリングや同期エンジニアリングをスムーズに行うことがほぼ不可能だったため、この議論を盾にモデル化を避けていた人々は、非モデル化に何の疑念も持たなかったでしょう。しかし、まさにそれがスパークスシステムズが解決した MDG テクノロジー（Visual Studio、Eclipse 共に利用可能）なのです。

ここで非常に重要な質問です。プロジェクト開発の間、どうやってモデルとソースコードの同期を維持するのでしょうか。その答えは、図 19 にあります。

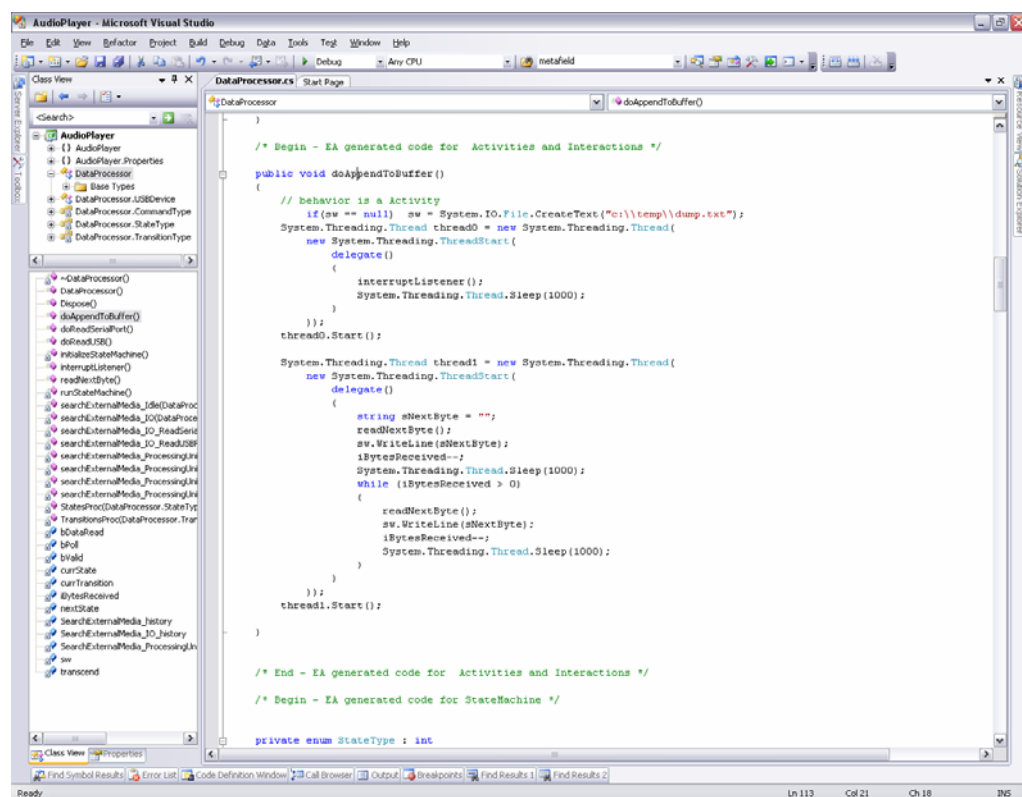


図 19 - Microsoft Visual Studio 2008 の DataProcessor のために 生成された C# ソースコード

以下のような流れになります。

- 1) UML モデルを Visual Studio または Eclipse（以下 IDE）と関連づけます
- 2) モデルのパッケージを IDE のプロジェクトにリンクします。
- 3) クラス上にある操作(メソッド)からソースコード内の定義位置に移動
- 4) IDE でソースコードを編集

IDE で変更した内容は、自動的にモデルに更新されます。また、モデルの変更内容は自動的にソースコードに反映されます。MDG 連携機能があなたに代わってモデルとソースコードの同期を維持します。これで問題は解決です。

まとめ

以上で、SysML と Enterprise Architect Suite システムエンジニアリング版を利用した組み込みシステムの設計のロードマップの概要説明を終わります。ロードマップでは、要求の定義・割り当て・追跡やハードウェアとソフトウェアの設計・制約やパラメトリックの追加・振る舞い図からのソースコード生成までを含みます。

このような方法で、皆さんの設計開発が成功することを祈っています！