Rational. SDL

IBM

SDL Suite クィックスタート

Getting Started

IBM Rational SDL Suite 6.3

Getting Started

日本語版

本書は、IBM Rational SDL Suite 6.3 および IBM Rational TTCN Suite 6.3 および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

著作権表示

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、または サービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の 製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの 評価および検証は、お客様の責任で行っていただきます。

IBM特許権

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について 実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒 106-8711
東京都港区六本木 3-2-12
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムと その他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、製造元に連絡してください。

```
Intellectual Property Dept. for Rational Software|
IBM Corporation
1 Rogers Street
Cambridge, Massachusetts 02142
U.S.A.
```

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

保証の不適用

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示 もしくは黙示の保証責任を負わないものとします。国または地域

によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

機密情報

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

追加の法的通知が、本書で説明するライセンス付きプログラムに付随する「プログラムのご使用条件」に含まれている場合があります。

サンプルコードの著作権

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに 準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下におけ

る完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性が あることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

IBM の商標

IBM および関連の商標については、**www.ibm.com/legal/copytrade.html**. をご覧ください。これは、IBM が現在所有する米国における商標の最新リストです。以下は、International Business Machines Corporation の米国およびその他の国における商標です。

このページには、IBM が使用しているすべてのコモン・ロー商標は掲載されていません。IBM が販売している製品は多数あるため、コモン・ロー商標のうち、最も重要な商標のみを掲載しております。このページに商標が掲載されていなくても、それは IBM がその商標を使用していないということではなく、その製品が現在販売されていない、または関連する市場で、その製品が重要ではないということを意味するものではありません。

他社の商標

Adobe、Adobe ロゴ、PostScript は、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows 2003、Windows XP、Windows Vista および / またはその他の Microsoft 製品は、Microsoft Corporation の米国およびその他の国における商標または登録商標です。

Pentium は、Intel Corporation の商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

はじめに

このマニュアルについて

このマニュアル、*Getting Started*では、SDL Suite環境の学習に役立つさまざまな内容を紹介します。

最初に、SDL Suite、およびサポートされている言語についての概要を説明します。次に、後続の章で、SDL Suiteに収められている各種ツールの使用方法を学習するためのチュートリアルを示します。また、それらのチュートリアルには、オブジェクト指向分析と設計に主眼をおいた演習があります。各チュートリアルは、DemonGameという比較的簡単なシステムを共通の題材として取り上げています。さらに、SDL Suiteの事前知識がなくても読み通すことができるように設計されています。

残りの章では、ASN.1データ式、Cmicroターゲティングなどのより高度な機能について説明します。

すべてのチュートリアルは、以前のチュートリアルの学習内容をベースに説明していますので、順序にしたがってお読みいただくことが必要です。

ドキュメント構成

ドキュメント構成に関する情報は、Release Guide日本語版のviiiページ、「ドキュメント」を参照してください。

表記規則

ドキュメントで使用している表記規則については、Release Guide日本語版のxページ、「表記規則」を参照してください。

カスタマー サポートへのお問い合わせ

IBM Rationalカスタマー サポートへの問い合わせに関する情報は、Release Guide日本語版のivページ、「IBM Rationalソフトウェア・サポートへの問い合わせ」を参照してください。

第 1 章 言語と表記方法の概要

この章では、最初にSDLの概要について説明し、SDLの特長や、背景、使用目的、応用分野などに触れます。

次に、MSC言語とハイレベルMSC言語の概要について説明します。具体的には、オブジェクト モデルの表記や、ステート チャート表記、ASN.1表記などを紹介します。

予備知識として、TTCNの表記法についても簡単に説明します。TTCNに関するより詳しい情報は、『TTCN Suite 6.2 Getting Started』の第1章、「Introduction to Languages and Notations」を参照してください。

この章を読まれた後に、さらにSDLへの理解を深めたい場合は、『Methodology Guidelines』の第1章、「SDLを使用したオブジェクト指向設計」を参照してください。このドキュメントにはSDL Suite環境でSDL-92言語を利用する方法の説明があります。

また、この章の最後には、言語に関するさまざまな文献の一覧が掲載されています。19ページの「参考文献」を参照してください。

仕様記述言語の利点

システムの開発を適切に進めるには、システムの完全な仕様を作成することと設計することが重要です。そして、この段階の作業には、以下の要求を満たす適切な仕様記述言語が求められます。

- 概念を明確に定義できること。
- 明瞭、正確、簡潔に仕様を記述できること。
- 仕様の完全性と正確性を検証する基準があること。
- 実装された技術が仕様に適合しているかどうかを確認する基準があること。
- 仕様間の一貫性を確認する基準があること。
- 仕様の作成、保守、検証、シミュレーション、バリデーションに対してコンピュータ ベースのツールを利用できること。
- コンピュータを使ったアプリケーション生成がサポートされ、従来要求されていたコーディング工程を省略できること。

SDL Suiteは、上記の要求をすべて満たしています。

SDL言語に関する一般知識

SDL （仕様記述言語）は、システムの仕様を設計し、記述する[1] ための標準言語です。SDLは、ITU-Tによって作成され、Z.100勧告として標準化されました。

SDLの作成は、研究作業が終了した後、1972年に開始されました。SDLの初版は1976年に公開され、その後、4年ごとに改定されています。最新版では言語体系が大幅に拡張されており、現在のSDLはあらゆる点で「完全」な言語になっています。

SDL Suiteでは、SDLが完全サポートされ、SDL-96の一部の概念も含まれています。SDL Suiteにおける SDLのサポート状況については、『Release Guide』の第1章「互換性について」の2ページ、「ITU SDLとの互換性」を参照してください。

モジュール性

SDLで記述された仕様と設計（システム）は相互に結合されたモジュール（ブロック）で構成されます。ブロックは、より多くのブロックに分割することができ、さらにこれを繰り返して階層構造を構成することができます。チャネルは、ブロック間や、ブロックと外部環境との通信経路を定義します。通常、各チャネルには無制限のFIFOによる待ち行列があり、チャネル経由で転送される信号が待ち行列内に入ります。各末端のブロックの振る舞いは、1つ以上の通信プロセスによって記述されます。また、プロセスは、拡張有限状態マシンによって記述されます。

オブジェクト指向の設計

さらにSDLでは、タイプ[2] という概念の導入によってオブジェクト指向の設計がサポートされます。タイプを使用することで、ブロック、プロセス、データ型などのほとんどのSDLコンセプトで特殊化や継承が使用できます。タイプを使う利点には、コンパクトなシステムを設計できることや、コンポーネントを再使用してシステム保守作業を軽減できることなどがあります。

1. SDLアプリケーションでは、「仕様」と「記述」は一般的に意味が異なりますが、SDLでは「仕様」と「記述」は区別されません。
2. 「タイプ」というSDL用語は、多くのオブジェクト指向の表記法やプログラミング言語で使われる「クラス」という用語に対応します

図式表現とテキスト表現

SDLでは、図式表現（SDL/GR）とテキスト表現（SDL/PR）の2つの等価な構文形式を選択できます。SDL Suiteは、双方の表現方式をサポートします。

用途

現在、SDLは主に電気通信業界で知られていますが、SDLの応用分野は広く、リアルタイム ソフトウエア業界でも次第に認知されるようになっています。SDLの用途をまとめると以下のようになります。

- SDLによって記述できるシステムの種類：リアルタイム システム、インタラクティブ システム、分散システム。
- SDLによって記述できる情報の種類：振る舞いと構造についての情報。
- SDLがサポートする抽象化のレベル：システム全体から機能の詳細まで。

SDLの詳細

理論モデル

SDLシステムの基本理論モデルは、並列に実行される拡張有限状態マシン（FSM）で構成されます。これらのマシンは、互いに独立しており、個別の信号で通信します。

SDLシステムは、以下のコンポーネントで構成されます。

- 構造
  - システム、ブロック、プロセスおよびプロシージャなどの主要ブロックによる階層的な分離状態
- 通信
  - オプションの信号パラメータを持つ非同期信号
  - リモート プロシージャ コールによる同期通信
- 振る舞い
  - プロセス
- データ
  - 継承、一般化、および特殊化が可能な抽象データ型
  - Z.105に準拠したASN.1データ型
- タイプの概念
  - 継承、一般化、および特殊化が可能なタイプ階層の記述

構造

図1には、SDLの4つの主要な階層レベルであるシステム、ブロック、プロセス、プロシージャが示されています。

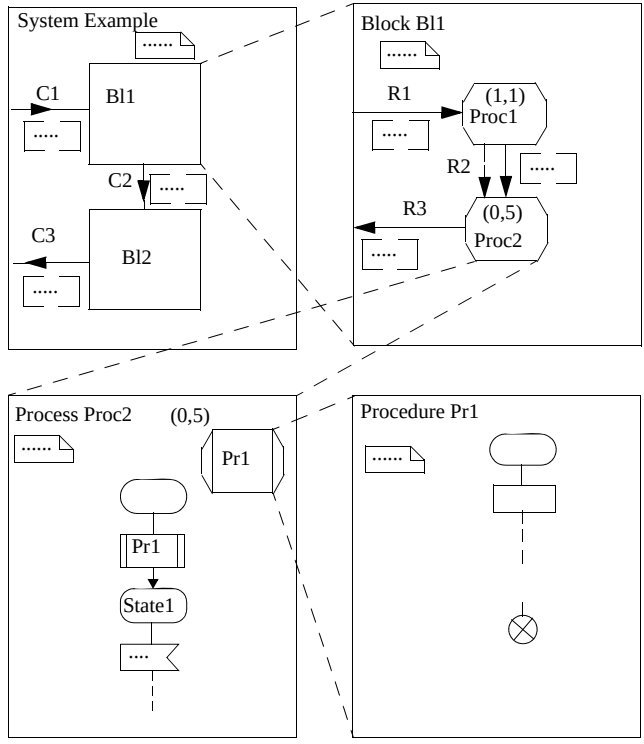


図1: SDL システムの構造

さらにプロセスでは、サービスという概念を使用できます。プロシージャは、プロセス内とサービス内の双方で使用できます。

通信

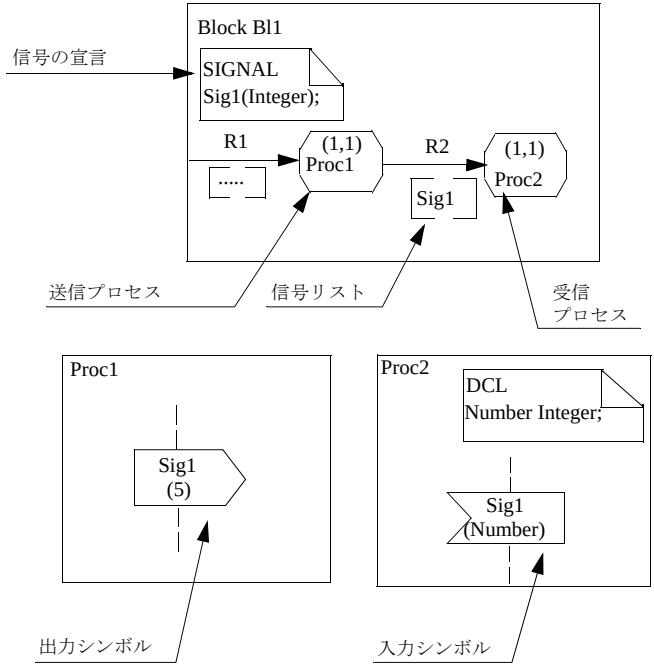


図2: プロセス間の信号送信

SDLでは、システム全体で使用できるデータは存在しません。この制限によって、プロセス間およびプロセス-外部環境間での情報のやり取りは、信号や信号パラメータのオプションを使用して行わなければなりません。信号は非同期で送信できるため、送信プロセスは受信プロセスからの確認信号を待たずに動作を継続できます。

同期通信は、簡略記法を使用したリモート プロシージャ コールによって可能です。略記は、信号に変換され、確認用の追加信号と共に送信されます。

振る舞い

SDLシステム内の動的な振る舞いは、プロセスとして記述します。システムおよびブロックの階層は、システム構造の中で唯一の静的な記述です。SDLのプロセスは、システムの起動時に生成することができ、実行時に動的に生成および終了することもできます。また、プロセスのインスタンスは複数個生成することができます。各インスタンスには、ユニークなプロセス識別子（PId）を割り当てます。したがって、プロセスから生成された複数のインスタンスのそれぞれに対して、信号を送ることができます。SDLは、独立して同時に動作できるプロセスやプロセス インスタンスの概念を取り入れているため、本格的なリアルタイム制御言語として位置付けられています。

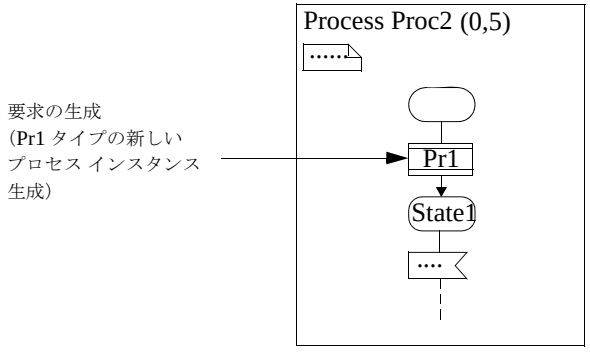


図3: 実行時での新しいプロセス インスタンスの生成

データ

SDLで使用している抽象データ型の概念は、仕様記述言語に非常に適しています。抽象データ型は、特定のデータ構造を持たないデータ型です。その代わりに、抽象データ型には、値の集合や、値のデータ型が許容する操作の集合、操作を定義する等式の集合などを定義できます。この方法によって、SDLのデータ型に対して別の高級言語で使われているデータ型を簡単に割り当てることができます。

さらに、SDLでは、ASN.1のデータ型を使うことができます。これは、ASN.1をすでに採用している電気通信アプリケーションの仕様を記述する際や、実装する際に便利です。ITU-TのZ.105勧告には、ASN.1をSDLと組み合わせて使用する方法が定義されています。ASN.1の詳細については、17ページの「ASN.1 - Abstract Syntax Notation One」を参照してください。

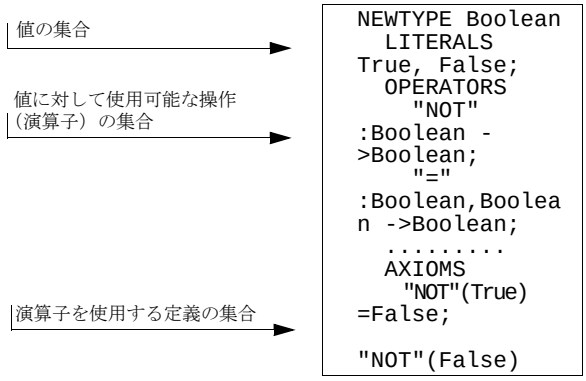


図4: 抽象データ型の記述例

タイプの概念

SDLで採用されているオブジェクト指向の概念によって、構造化や再使用が可能になります。この概念はタイプの定義によって成り立ちます。構造で使用するブロックはすべて、タイプとして定義できます。タイプの定義はシステムの任意の場所やシステム外のパッケージにも配置できます。

オブジェクト指向言語を使用する利点には、既存オブジェクトへの新しいプロパティの追加が可能なことや、既存オブジェクトのプロパティを再定義した新しいオブジェクトの生成が可能なことを挙げることができます。この操作は、通常、特殊化と呼ばれます。

SDLでは、タイプの特殊化が以下の2種類の方法で可能です。

- サブタイプに、スーパータイプで定義されていないプロパティを追加できます。たとえば、プロセス タイプに新しい遷移を追加したり、ブロック タイプに新しいプロセスを追加することができます。
- サブタイプに、スーパータイプで定義されている仮想タイプや仮想遷移を再定義できます。たとえば、プロセス タイプに定義されている遷移の概念やブロック タイプに定義されている概念/構造などを再定義できます。

メッセージ シーケンス チャート言語

背景

近年、ITUは、メッセージ シーケンス チャート（MSC）を定義する形式言語の標準化に注力しており、1992年夏にMSCの勧告Z.120の初版を公開しています。

Z.120勧告に定義されているとおり、SDLシステムの動的な振る舞いを記述する際は、MSC言語によってSDLの記述を補うことができます。MSC言語の図式表現は、複雑な動的振る舞いを表現するのに適した、明確であいまいさのない、理解しやすい表現です。

現在、MSC'96というMSC標準の拡張仕様がZ.120で制定されています。SDL Suiteでは、MSC'96の最も重要な拡張仕様をサポートしています。詳細は、『Release Guide』の第1章「互換性について」の5ページ、「ITU MSCとの互換性」を参照してください。

MSC

MSCには、SDL仕様を基にして作成された状態遷移ツリーの、1つのノードから別のノードへのトレースを記述します。

基本的に、情報のやり取りは、メッセージを1つのインスタンスから別のインスタンスに送信することで実行されます（図5参照）。メッセージは、SDLの仕様において1つのプロセスから送信され別のプロセスで取得される信号に相当します。インスタンスには、SDLシステム、ブロック、またはプロセスなど、仕様のあらゆる部分が該当します。

MSCでは、*MSC参照*シンボルを使って別のMSCを参照することができます。たとえば、初期化シーケンスを記述する1つのMSCを作成し、MSC参照を使って、その他複数のMSCからこのMSCを参照することができます。

参照シンボルは、MSCを単に参照するだけではなく、1つ以上のMSCを参照するMSC参照式を内容として持つことができます。この構造によって、簡潔なMSCの表現が可能になります。また、特定のMSCを再使用するための優れた手段になります。

*操作のインライン表現*を使用することで、複数のMSCシナリオを1つのダイアグラムで作成できます。同じ構造を持つイベントは、MSC参照シンボルを使って表現できます。

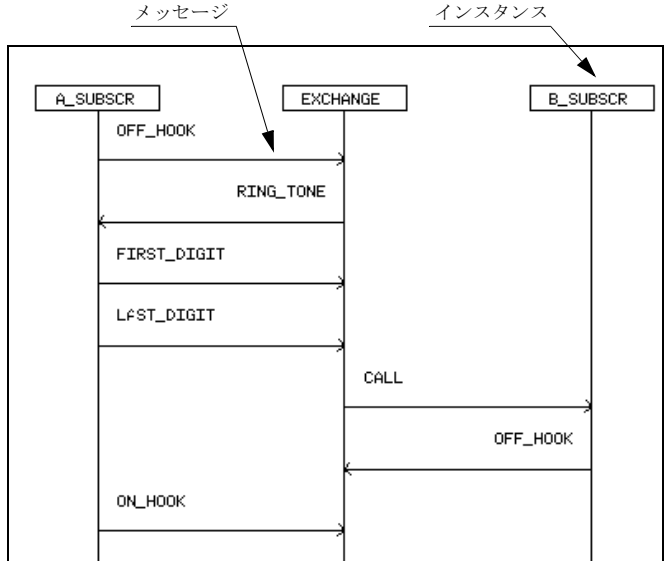


図5: 簡単なメッセージ シーケンス チャートの例

ハイレベルMSC

ハイレベルMSC（HMSC）によって、複数のMSCの結合を図式的に定義できます。通常のMSCとは異なり、HMSCの内部にはインスタンスやメッセージは記述せず、各MSCの構成のみを記述します。HMSCは入れ子構造にすることができます。つまり、HMSCの詳細部分を他のHMSCで記述することができます。MSC言語の機能は、新しいHMSCの概念の導入によって大幅に強化されます。たとえば、メインのシナリオと付随する例外処理のすべてを、一括して記述することが容易になります。

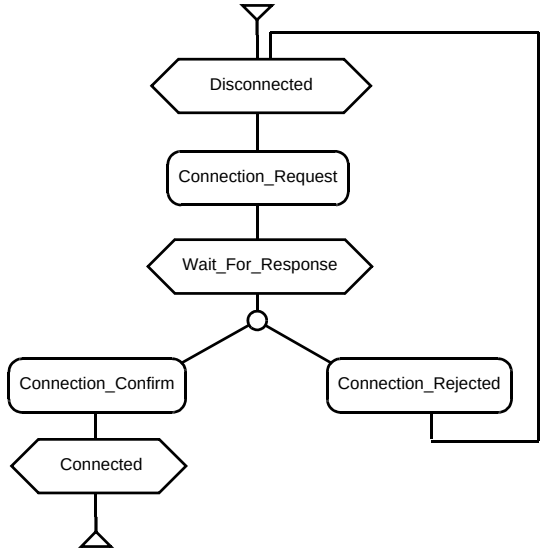


図6: HMSCの例

図式表現とテキスト表現

MSC言語では、2種類の等価な表記法がサポートされています。図式表現（MSC-GR）に加えて、テキスト表現（MSC-PR）が1994年秋に標準化されています。

用途

さまざまな用途の中から、代表的な使用例を以下に示します。

- システムの要求条件を定義するドキュメントの作成に使用できます。
- SDLで設計を開始する前の設計段階において、多数の動的な動作の定義やドキュメントの作成に使用できます。
- シミュレーションの実行結果をグラフィカルに表現できます。これにより、シミュレーション結果を容易に理解でき、また、後から仕様を検証することができます。SDL Suiteでは、SDLシステムに対するメッセージ シーケンス チャートの検証が可能です。
- 対話型シミュレーションの実行やレポートの作成の際に、SDLシステムの実行トレースを表示するために使用できます。

オブジェクト モデルの表記法

SDL Suiteで使用されるオブジェクト モデルの表記法は、OMT（Object Modeling Technique）とUML（Unified Modeling Language）で使用される表記法を採用したものです。OMTとUMLの表記法は、広く受け入れられた図式表現であり、オブジェクトやオブジェクト同士の関係を示すダイアグラムの記述に使います。

クラス

オブジェクト モデルにおいて最も重要な概念はクラスの定義です。クラスは、類似したオブジェクトの集合を記述するものです。これらのオブジェクトはクラスの特性を共有します。クラスの特性は、属性や演算によって定義します。クラスにおけるオブジェクト モデルの表記例を図7に示します。図7の右側のクラス定義には、属性と操作の定義も表示されています。

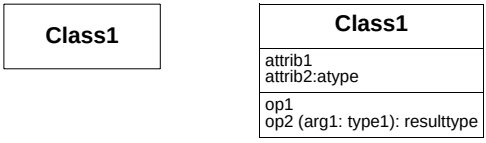


図7: 折りたたまれたクラス シンボルと
属性と操作を表示したクラス シンボル

クラスは、属性や操作を別のクラスから継承することもできます。これらの方法は、特殊化および一般化と呼ばれます。継承に関するオブジェクト モデルの表記を図8に示します。

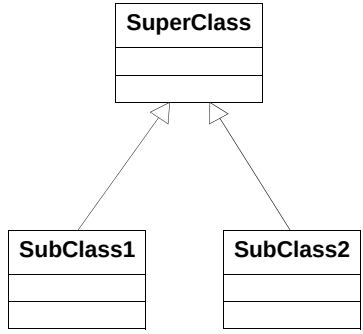


図8: クラス間の継承

リレーションと多重性

クラス同士は、物理的または論理的に結び付けることができます。図9のように、オブジェクト モデルでは、これを関連という概念によって表現します。関連には、名前を付けることができます。また、関連のエンドポイントには、各エンドポイントでの役割り名をラベルとして割り当てることができます。

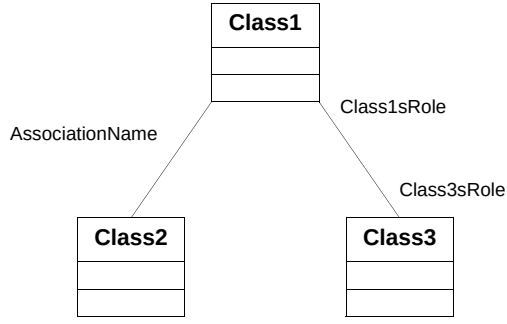


図9: クラス間の関連

アグリゲーションは、関連の一種であり、「~で構成された」というようなリレーションを表します。アグリゲーションには、図10のような専用の表記を使用します。

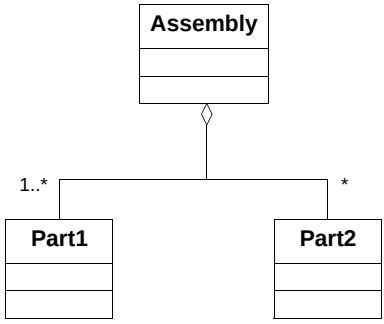


図10: アグリゲーション

関連とアグリゲーションのエンドポイントには、以下のような多重度を持たせることができます。

- 多重度なし（ちょうど1つ）
- `*`（0またはそれ以上）
- `0,1`（0または1）
- `1..*`（1またはそれ以上）
- `1..3,6,10..*`（特定個数：1、2、3、6、10またはそれ以上）

オブジェクト

クラスの定義以外に、オブジェクト モデルには、オブジェクト（インスタンス）とその関係情報を格納することができます。オブジェクト間の関係情報は、リンクという概念によって表現され、クラス間での関連付けに相当します。オブジェクト シンボルには、オブジェクト名とクラスへの参照を記述するフィールドと（「名前:クラス」 で表記）、オブジェクトの属性に対して定数やデフォルト値を割り当てることができる属性フィールドがあります。図11を参照してください。

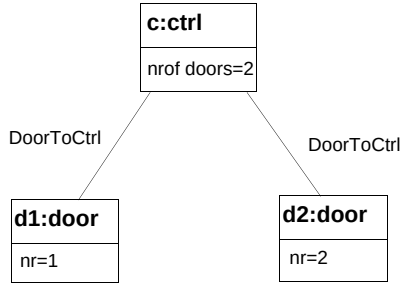


図11: リンクによって関連付けられたオブジェクト

ステート チャートの表記法

SDL Suiteで使用しているステート チャートの表記法は、OMT（Object Modeling Technique）とUML（Unified Modeling Language）で使用されている表記法のサブセットです。

ステート チャート モデルは、クラスやオブジェクト モデルと共に使用するのに適しています。クラスのダイアグラムに記述されたクラスの振る舞いは、ステート チャートにまとめることができます。ステート チャートには、*状態*や状態間の*遷移*を使って、モデルの動的な振る舞いを示すことができます。

状態

状態シンボルには、クラスの名前、状態変数、および内部動作を記述します。内部動作は、状態に入るとき、状態内にいるとき、および状態から出るときに発生します。動作は、イベントやイベントに関連付けられた処理を定義することによって記述します。図12は、折りたたまれた状態と、イベントを表示している状態を示しています。

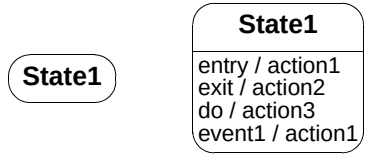


図12: 折りたたまれた状態シンボルと
イベントを表示した状態シンボル

遷移

遷移シンボルは矢印で表現され、一般的に2つの状態シンボルを接続するために使用します。遷移は、イベントの発生と条件の成立がトリガになります。そして、遷移によって処理が実行されます。図13を参照してください。

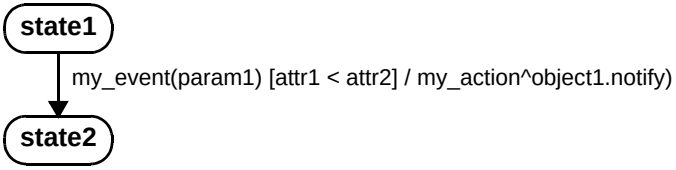


図13: state1からstate2への遷移は、my_eventイベントの発生とattr1がattr2より小さいという条件で実行されます

スタート シンボルと終了シンボル

スタート シンボルは、ステート チャートで記述される状態マシンの始点を示し、終了シンボルは、状態マシンの終点を示します。図14に、ドアの振る舞いを記述した簡単な状態マシンの図を示します。図14には、スタート シンボルと終了シンボルも記述されています。

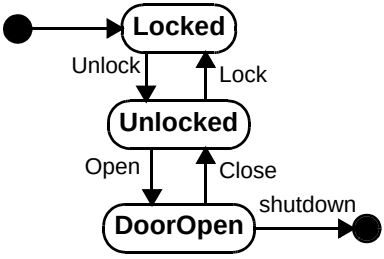


図14: スタート シンボルと終了シンボルが記述された簡単なステート チャート

サブ状態

状態は、サブ状態として埋め込まれたダイアグラムや、下位階層として定義された状態のダイアグラムを使って詳細を記述することができます。この記述方法によって、複雑な振る舞いを下位のダイアグラムに記述し、状態を単純化して表現することができます。

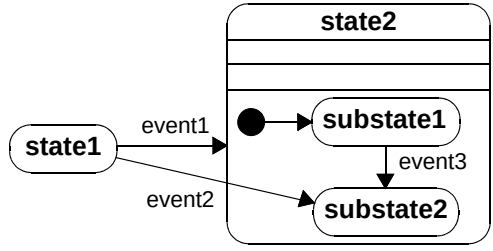


図15: サブ状態を持つ状態

ASN.1 - Abstract Syntax Notation One

ASN.1（ITU-T 勧告X.680-683）は、ISOとITUによって標準化された表記法であり、データ型や値を規定するために使用します。ASN.1の基本理念は、転送形式に依存しないデータ型の情報を記述することにあります。

ASN.1は当初、FTAM、CMIP、MHS、DS、VTなどの高レベル プロトコルにおいて情報記述に使用されていましたが、現在では、その他のさまざまな電気通信プロトコルやアプリケーションにも使用されています。

ASN.1のデータ型と値は、TTCNやSDLで使用するモジュール内に定義できます。これにより、アプリケーションのデータ型が、SDL仕様とTTCNテスト スイートのどちらででも使用できるようになり、システム仕様とテスト仕様で転送される情報間の一貫性を確保できます。

TTCN表記法

情報技術と電気通信の分野では、この10年間に標準規格を使用する割合が大幅に増加しています。これに伴って、標準規格と実際のインプリメンテーションとの整合性を検証するための、手法やツールへの要求も増加しています。

この要求は、ISOとITUによって「Framework and Methodology for Conformance Testing of Implementations of OSI and ITU Protocols」の中で提起され、近年、国際標準であるISO/IEC 9646 (X.290)として制定されています。

この標準には、抽象テスト ケースによって構成される抽象テスト スイートという概念が取り入れられています。抽象テスト スイートは、システムに対して実施すべきテストの集合を記述したものです。この標準では、テストはブラックボックス モデルを使用して記述しなければなりません。つまり、制御と観測にのみ、インターフェイスを使用できます。

抽象テストでは、非形式的自然言語ではなく形式言語を使用して記述する必要があります。この標準の一部は、抽象テストを記述するためのTTCN言語の定義に割かれています。

TTCN - Tree and Tabular Combined Notation

テスト スイートは、TTCNによって定義されます。テスト スイートは、さまざまなテスト ケースや、テスト スイートに必要な宣言などがまとめられたものです。

各テスト ケースは、イベント ツリーとして記述します。このツリー内に、「最初にAを送信。次にBまたはCを受信。受信がBならばDを送信...」のような形式で振る舞いを記述します。最新版のTTCNの規定では、いくつかのイベント ツリーを同時に実行することができます。

TTCNは、実際のテスト システムから独立しているという点で抽象的であるといえます。つまり、あるアプリケーション（プロトコル、システムまたはその他の用途）用のTTCNテスト スイートは、そのアプリケーションに対するさまざまなテスト環境で使用できます。

この数年でTTCNの使用が大幅に増加しています。これは多数のTTCNテスト スイートがさまざまな標準化団体から発表されたためです。しかし、TTCNは標準化作業での使用だけでなく、通信システムに対する、すべての機能試験で利用できます。そのため、現在では通信業界でも広く利用されるようになっています。

送受信するメッセージの仕様は、TTCNのビルトイン フォームかASN.1を使用して定義できます。

ツールのサポート

IBM Rationalは、SDLの開発環境を長年にわたってサポートしてきました。さらにITUと連携してこの言語分野の改良にも関与しているほか、ETSIと連携してSDLを使用したプロトコル標準の確立に努めています。IBM Rationalは、これら言語を各種事業で応用するための国際的な研究プログラムを発足させると同時に、それを推進しています。研究プログラムには、EC（欧州共同体）のRACE、ESPRIT、EUREKAなどがあります。このようなIBM Rationalの経験とノウハウは、ソフトウエア エンジニアリング分野での、言語をサポートするツールの開発に活かされています。

仕様記述言語用のツールは、仕様を作成、保守、分析できなければなりません。同時に、シミュレーション機能や、バリデーション機能を持ち、他の高級言語でのアプリケーション コード生成する機能が必須となります。

SDL Suiteでは、このような機能がすべて実現されています。

参考文献

[1] ITU Recommendation Z.100:
Specification and Description Language (SDL)
1994, ITU, General Secretariat- Sales Section,
Places des Nations, CH-1211 Geneva 20

[2] Annex A, B, C1, C2 D, E, F1, F2 and F3 to Z.100, as above

[3] ITU Recommendation Z.120:
Message Sequence Charts (MSC)
1992, ITU General Secretariat - Sales Section
Place des Nations, CH-1211 Geneva 20

[4] Jan Ellsberger, Dieter Hogrefe, Amardeo Sarma:
SDL – Formal Object-oriented Language for Communicating Systems.
Prentice Hall Europe (1997)
ISBN 0-13-63288-5

[5] A. Olsen, O. Færgemand, B. Møller-Pedersen, R. Reed, J.R.W. Smith:
Systems Engineering Using SDL-92.
Elsevier (1994)
ISBN 0-444-8987-7

[6] Ferenc Belina, Dieter Hogrefe, Amardeo Sarma:
SDL with Applications from Protocol Specification.
Prentice Hall International (UK) Ltd. (1991)
ISBN 0-13-785890-6

[7] Belina, Hogrefe:
The CCITT Specification and Description Language SDL
Computer Networks and ISDN System.
North-Holland, Amsterdam (1988/1989)

[8] Bræk, Gorman, Haugen, Melby, Møller-Pedersen, Sanders:
TIMe - The Integrated Method
SINTEF 1998
http://www.sintef.no/time

[9] Færgemand, Marques (editors):
SDL 89: The language at work.
Proceedings of the Fourth SDL Forum,
North Holland, Amsterdam (1989)

[10] Færgemand, Reed (editors):
SDL 91: Evolving Methods.
Proceedings of the Fifth SDL Forum,
North-Holland, Amsterdam (1991)

[11] Færgemand, Sarma (editors):
SDL 93: Using Objects.
Proceedings of the Sixth SDL Forum,
North-Holland, Amsterdam (1993)

[12] Haugen, Møller-Pedersen:
Tutorial on object-oriented SDL.
SISU Project Report 91002
Norwegian Computer Center
PO Box 114, N-0314 Oslo 3, Norway

[13] Behcet Sarikaya:
Principles of Protocol Engineering and Conformance Testing.
Simon & Schuster International (1992)

[14] Sarraco, Smith, Reed:
Telecommunications system engineering using SDL.
North-Holland, Amsterdam (1989)

[15] K.J. Turner (editor):
Using Formal Description Techniques -
An Introduction to Estelle, LOTOS and SDL.
John Wiley & Sons (1992)

[16] ITU Recommendation X.680-683
Abstract Syntax Notation One (ASN.1)
1994, ITU, General Secretariat- Sales Section,
Places des Nations, CH-1211 Geneva 20

[17] ITU Recommendation Z.105
SDL Combined with ASN.1 (SDL/ASN.1)
1995, ITU, General Secretariat- Sales Section,
Places des Nations, CH-1211 Geneva 20

第 2 章 SDL Suiteの概要

この章では、SDL Suiteとそのコンポーネントの機能について簡単に説明します。

まずこの章をお読みいただき、内容を十分理解されたうえで、SDL Suiteの操作の学習に進んでください。その段階では、簡単な題材を使用してツールの起動方法や操作方法を実習します。これらの学習にあたっては、以下のドキュメントを参照してください。

- SDL Suiteの基本操作:本書の第3章「チュートリアル: SDLエディタとアナライザ」
- 新機能について:『Release Guide』の第2章、「リリース ノート」

SDL Suiteについて

IBM Rationalとは

SDL Suiteは、IBM Rationalが開発、販売している製品です。IBM Rationalでは、SDL、MSC、UMLといった言語や開発環境を長年にわたってサポートしてきました。さらに、ITUやOMGと連携し、これらの言語の改良に関与しているほか、ETSIと協力して通信プロトコル分野での国際標準の確立に努めています。

IBM Rationalは、これら言語を各種事業で応用するための国際的な研究プログラムを発足させると同時に、それを推進しています。研究プログラムには、EC（欧州共同体）のRACE、ESPRIT、EUREKAや、スウェーデンの国家レベルでのITプログラムなどがあります。

このような経験とノウハウは、ソフトウェア エンジニアリング分野での、言語をサポートするツールの開発に活かされています。

SDL Suite

設計や仕様記述言語用のツールは、言語で規定されている構文や意味の規則に合致する仕様を作成、保守、検証できなければなりません。同時に、シミュレーション機能やバリデーション機能を持ち、他の高級言語でのアプリケーション コードを生成する機能が必須となります。

開発サイクルのすべてをカバーするためには、初期の分析フェーズをサポートし、さらにオブジェクト指向分析からSDL設計へ移行できなければなりません。

また、他のSDLツールとの情報のエクスポート、インポート機能も必要です。主要な標準規格やデファクト スタンダードをサポートしていることも重要な条件です。

さらに、視覚的に内容が把握でき、一貫性を持ったグラフィカル ユーザー インターフェイスが必要です。これはツールの学習時間を短縮し、ツールを使いやすいものにします。この他、ユーザーが手を入れることなく大量の情報を処理できる、バッチ処理機能も必要です。

強力なコンテキスト対応ヘルプ機能も重要です。コンテキスト対応ヘルプ機能により、目的のトピックを探すためにユーザー ドキュメントを調べる手間が省けます。

SDL Suiteはこのような機能をすべて持っているばかりでなく、それ以外にも数々の機能を備えています。

SDL Suiteの概要

アーキテクチャ

SDL Suiteは、TTCN SuiteやLogiscope.などをはじめとする幅広いツールファミリのメンバーです。

SDL Suiteは、いくつかの独立したツールによって構成されており、それぞれのツールで情報が処理されます。個々のツールは、特殊な*通信メカニズム*によって統合されており、高度に統合されたシステムを独立したツール群を使って設計することができます。また、この方法によって、既存のツールと競合することなく新しいツールを加えることができます。さらに、2つの独立したツールを結び付けて、機能を強化することができます。ツールをネットワーク環境で使い、ネットワークを介して互いに通信させることもできます。

SDL Suiteは、ポストマスタ（*The PostMaster*）と呼ばれるインターフェイスをツールとエディタの統合に使用しています。このインターフェイスの一部は、仕様がマニュアル化されています。これはユーザーにとって便利な点であり、独自のツールをSDL Suiteに統合する際に利用できます。

SDL Suite ツールの起動

通常、SDL Suiteのコンポーネントは、インストールしたディレクトリ内のbinディレクトリで、*sdt*開始スクリプトを実行することによって起動します。このスクリプトは、以下のように数個のオプションを取ります。

- `sdt -reuse`
  稼動中のSDL Suiteセッションがある場合、そのセッションのオーガナイザを表示します。ない場合は新しいセッションを起動します。
- `sdt <system file>`
  SDL Suiteのコンポーネントを起動して、指定されシステム ファイルをロードします。
- `sdt <`ダイアグラム ファイル`>`
  SDL スイートのコンポーネントを起動して、エディタで指定されたダイアグラム ファイルをロードします。
- `std <`アーカイブ ファイル`> [ <`展開用ディレクトリ`> ]`
  SDLスイートのコンポーネントを起動し、指定されたアーカイブ ファイルを展開して、その中のシステム ファイルをロードします。アーカイブ ファイルは、指定された展開用ディレクトリに展開されます。展開用ディレクトリを指定しなければ、オーガナイザのダイアログ ボックスが表示され、指定するよう求められます。

- `sdt -fg`（UNIXの場合）
  通常は、*sdt*開始スクリプトを実行するとすぐに、新しいコマンド ライン プロンプトが表示され次のコマンドを入力できるようになります。*sdt -fg*を使用した場合、SDL Suiteが終了するまで新しいコマンド ライン プロンプトは表示されません。
- `sdt -noclients`
  このオプションをつけると、オーガナイザなどのクライアント/アプリケーションを起動することなく、ポストマスタが開始されます。そしてポストマスタ クライアントを後から起動して、ポストマスタに接続することができます。実行中のポストマスタに接続するには、クライアントを「-post」オプションを付けて起動します。この方法の詳細については、第10章「The PostMaster」507ページの「Run-Time Considerations」をご参照ください。
- `sdt -grdiff [-notmoved] v1.ssy v2.ssy [ [ -original o.ssy ] -mer-geto r.ssy ]`
  SDL Suiteのコンポーネントと起動し、その後すぐにダイアグラムの比較の処理を呼び出します（第43章「Using the SDL Editor」2004ページの「Compare Diagrams」および第39章「Using Diagram Editors」1674ページの「Compare Diagrams」を参照）。これにより、SDLダイアグラム ファイルのv1.ssyとv2.ssyが比較されます。HMSCまたはMSCダイアグラムを比較するコマンドも使用できます。-notmoved オプションは、[オブジェクト IDを使用して内容比較グループを検索]オプションをオフに設定した場合に対応します。(UNIX で -n オプションを使用した場合には、X Window システムで処理されます)。通常、このオプションは、移動されたシンボルや、サイズ変更されたシンボルが異なるシンボルとして検出されないことを意味します。処理が終了すると、SDL Suiteは終了して閉じます。*-mergeto*オプションを使用した場合は、ダイアグラムのマージの処理が実行されます。第43章「Using the SDL Editor」2004ページの「Merge Diagrams」および第39章「Using Diagram Editors」1674ページの「Merge Diagrams」を参照してください。マージ処理によって出力されたダイアグラムはr.ssyに保存されます。また、-originalオプションを使用した場合には、o.ssyを参照した相違点の自動マージ処理が実行されます。o.ssyファイルは、v1.ssyとv2.ssyをマージする手引きとして参照されます。設定管理用ツールなど、ほかのツールからマージ処理を起動するときには、-grdiffオプションを使用します。-grdiffに-fgを付けると、マージ処理が終了するまで、起動したツールでの処理は待ち状態になります。
- `sdt -sdtdiff v1.sdt v2.sdt [ -mergeto r.sdt ]`
  SDL Suiteのコンポーネントを起動し、その後すぐにシステムの比較処理を呼び出します（第2章「The Organizer」74ページの「Compare System」を参

照)。これにより、システム ファイルのv1.sdtとv2.sdtが比較されます。処理が終了すると、SDL Suiteは終了して閉じます。*-mergeto*オプションを使用すると、処理が終了したときにマージ結果を保存するか尋ねるメッセージが表示されます。また、このときの既定のファイル名としてr.sdtが表示されます。

バッチ機能

バッチ機能は、OSプロンプトから入力して実行します。バッチ機能はポストマスタを利用してツールにメッセージを渡し、各ツールが要求に従って情報を処理するように命令します。バッチ機能でサポートされている処理には、以下のようなものがあります。

- 印刷（ファイルまたはプリンタに出力）
- 分析（通常はSDLシステムの構文と意味のチェック）
- 実装（ターゲット環境に対応するアプリケーションの構築など）
- SDL、MSC、または、HMSCダイアグラムの比較（テキスト形式のレポートに出力）

ライセンス メカニズム

ソフトウェア ライセンス サーバーは、SDL Suiteに含まれるツールのライセンスを管理します。ライセンス管理には、サード パーティのソフトウェアFLEXnet Publisher™をベースにしたフローティング ライセンス メカニズムが取り入れられています。現在使用しているライセンス番号とキーは、ソフトウェアのインストールの際に作成されるテキスト ファイルに記録されています。したがって、ライセンスのアップグレードや新しいライセンス契約の追加を柔軟に管理できます。また、購入したツールの実際の使用状況を把握できます。

FLEXnetはもともと、製造元が異なる複数のツールをサポートするシステムです。また、サポートされるツールは単一のライセンス サーバーを共有することができます。したがって、コンピュータ環境にインストールした際に、IBM Rational によって問題が引き起こされることはありません。

オプションのタイムアウト機能を有効にすると、ライセンスを使う際の自由度を高めることができ、また起動後、使用されていないツールによってライセンスが消費されるのを避けることができます。この機能によって、ユーザーが指定した期間、ツールがアイドル状態になっている場合に、自動的にライセンスが開放されます。

ライセンス メカニズムの詳細については、『Installation Guide』の第6章、「IBM Rationalライセンスの手引き」を参照してください。

共有するツール

SDL Suiteは、SDL Suite and TTCN Suiteに共通のツールを使用します。

- オーガナイザ（*The Organizer*）は、システムを構成するすべてのダイアグラムとドキュメントを図式によって表示できます。表示できるものには、SDL階層や、メッセージ シーケンス チャート、オブジェクト モデル ダイアグラム、ステート チャート、ハイレベルMSC、TTCNドキュメント、およびテキスト ドキュメントなどがあります。これらの表示は、オーガナイザのチャプタやモジュール内に自由にまとめることができます。

  さらに、オーガナイザは他のツールを管理し、必要に応じてそれらの機能を起動します。これによりユーザーは、完全に一体化されたツール セットのような使用感が得られます。
- リンク マネージャ（*The Link Manager*）とエンティティ辞書（*The Entity Dictionary*）によって、実装リンクとエンドポイント（*Implinks and Endpoints*）の表示や管理が可能になります。実装リンク（Implementation Link）は、実装や設計決定をトレースする際に使います。実装リンクによって、開発プロセスの異なるフェーズに定義されている、概念とオブジェクトの間をトレースすることができます。リンク エンドポイントは、テキスト情報とオブジェクトで構成されており、次の項で説明する、すべてのエディタで作成できます。また、リンク マネージャーとエンティティ辞書は、すべてのエディタから起動できます。
- 環境設定マネージャ（*The Preference Manager*）は、環境パラメータの値を設定するために使います。環境パラメーターの値をカスタマイズすることで、ツールのデフォルトの動作の設定や変更が可能です。動作を設定する際は、プロジェクト全体に反映させるか、または、会社全体に反映させるかを指定できます。また、ユーザーごとのカスタマイズもできます。
- ドキュメントとダイアグラムの印刷（*Printing Documents and Diagrams*）を行う際には、印刷出力をカスタマイズできるさまざまなオプションが用意されており、自分のドキュメント環境に合わせた印刷ができます。TTCN以外のドキュメントを印刷する際には、PostScriptやEncapsulated PostScript、FrameMaker™、Interleaf™、Webファイルを生成することができます。また、Windowsの環境では、Microsoft Windows（MSW Print）用に設定されたすべてのプリンタに印刷出力が可能です。
- *オンライン ヘルプ*を使うことにより、ツールや、ウィンドウ、ダイアログ ボックス、コマンドからヘルプにアクセスすることができます。オンライン ヘルプは、HTML形式で記述されているため、ハイパーテキスト リンクやナ

ビゲーションの機能を使うことができます。また、このマニュアルを印刷するために作成されたPostScriptファイルも、製品内に収められています。必要なマニュアルのページは自由に印刷することができます。

- ポストマスタ（*The PostMaster*）によって、各ツールの統合が実現されます。ポストマスタのインターフェイスで公開されている部分は、ドキュメントになっています。これを参照することによって、他のツールをSDL Suite and TTCN Suiteツールやツールが管理する情報に結合することができます。

SDL Suite グラフィカル ツール

SDL Suiteには、以下のようなグラフィカル ツールがあります。

- *ダイアグラム エディタ*は、オブジェクト モデル、ステート チャート、メッセージ シーケンス チャート、ハイレベル メッセージ シーケンス チャートなどのダイアグラムを作成、編集、印刷するために使います。

  OMエディタでは、OMTやUMLの完全な図式表記を使うことができます。また、OMダイアグラムのスコープ内にある同じ名前のすべてのクラスとオブジェクト定義をトレースすることができ、これらの定義を併合して、クラスを一括管理することができます。SCエディタやHMSCエディタにも同様の機能があります。

  MSCエディタは、Z.120標準で定義されている図式表記法に準拠しています。また、SDLで記述されたシステムをシミュレーションおよびバリデーションする際に、強力な図式トレース ツールとして利用できます。MSCとSDLシステムとの間の一貫性は、SDLエクスプローラによって検証できます。

- SDLエディタは、Z.100標準で定義されているSDLのGR形式によって仕様や記述を作成、編集、印刷する際に使います。また、SDLエディタによって編集時にさまざまな構文チェックが実行されます。

  – SDLエディタでは、状態依存の機能を持った構文ヘルプ や信号辞書などがサポートされています。信号辞書には、SDLダイアグラムの編集時にシステムに記述したすべてのSDL信号が自動的に追加されます。これにより、定義済みの記号をただちに参照することができます。

  – SDLエディタは、SDLシステムの全ダイアグラムをローカル形式で表示したダイアグラムのオーバービュー（*Overview Diagrams*）を表示できます。そして、下位の階層のダイアグラムが各シンボルに入れ子のように関連付けられているため、各ダイアグラムを表示できます。

- SDLタイプ ビューワー（*The SDL Type Viewer*）は、SDL-92システム内で継承と特殊化が利用されている部分を表示します。タイプ ビューワーは、ツリーを図式的に表示するため、SDLシステムで定義したSDLタイプ[1] を把握し、利用する際に役立ちます。

- SDLインデックス ビューワー（*The SDL Index Viewer*）は、定義とクロスリファレンスのリストを明確でわかりやすい図式表記によって表示します。インデックス ビューワーにはフィルタリングとナビゲーション機能があり、元のSDLダイアグラムやMSCダイアグラムへの逆参照ができるようになります。

- カバレジ ビューワー（*The SDL Coverage Viewer*）は、テスト カバレジの算出と観測を行うツールです。シミュレーションやバリデーションの結果をグラフィカルな遷移ツリーやシンボル ツリーの形式で表示できます。そして、システムの全体的な、あるいは部分のカバレジを表示できます。

その他のSDL Suiteツールとバックエンド機能

その他にも以下のようなツールと機能を利用できます。

- テキスト エディタ（*The Text Editor*）は、ASCIIテキストのドキュメントを作成、編集、印刷するときに使います。テキスト ドキュメントには、開発工程で使用する、テキスト形式の要求条件記述やユース ケースなどがあります。また、テキスト エディタは、SDLシステムとリンクするASN.1およびCコードの記述にも使用できます。

- ADTライブラリ（*The ADT Library*）（抽象データ タイプのライブラリ）には、SDLシステムの設計の際に、しばしば必要となる基本的なサービスを提供する、ADTの一般記述が収めれられいます。ADTライブラリは、ソースコードで提供されているので、必要に応じてADTを修正し特定の仕様に合わせることができます。

- *SDL アナライザ*には、さまざまな機能があります。まず、SDLの記述に対する構文分析と意味分析を実行し、必要時にエラー レポートや警告を発生します。また、SDLアナライザには、SDLシステムの定義とクロス リファレンスに関する情報を生成する機能があります。さらに、SDL情報を図式形式の表現（SDL/GR）からテキスト形式の表現（SDL/PR）に変換する機能や、その逆変換の機能があります。たとえばSDLをサポートするほかのツールからPRファイルをインポートすることもできます。

1. 「タイプ」というSDL用語は、多くのオブジェクト指向の表記法で使われる「クラス」という用語に対応します。

- *Cadvanced/Cbasic SDL to C コンパイラ*）は、SDLシステムをいくつかのCソース ファイルに変換します。ソース ファイルはコンパイルされ、SDL Suiteランタイム ライブラリとリンクされます。以下で説明する、各種ライブラリを開発環境に配備すれば、生成されたCコードをさまざまな用途に使うことができます。SDL to Cコンパイラには、シミュレーションや検証に使用するCbasicと、各種アプリケーションの構築に使用するCadvancedがあります。

- *SDL シミュレータ*ライブラリによって、*シミュレーション*用の実行プログラムであるシミュレータを作成できます。シミュレータを使用すると、システム仕様の振る舞いの理解やデバッグが可能になります。シミュレータは、グラフィック ユーザー インターフェイス（SimUI）を使って制御することができます。

  シミュレーションでは、システム仕様と外部環境とのインターフェイスを解析するために、システム全体に対して実行することができ、さらに、内部の振る舞いを確認するために、一部分に対して実行することもできます。シミュレータの実行状況は、ソースのSDLダイアグラム上でグラフィカルにトレースすることができます。また、メッセージ シーケンス チャートとして記録することもできます。さらに、ターゲット シミュレーションも、サポートされています。

- SDLエクスプローラ（*The SDL Explorer*）ライブラリによって、エクスプローラの実行ファイルを作成できます。エクスプローラは、高度な「自己探索型」のシミュレータです。エクスプローラによって、SDLシステム内でのエラーや不整合の発見、システムとメッセージ シーケンス チャートとの整合の検証などが可能になります。エクスプローラは、グラフィック ユーザー インターフェイス（ValUI）によって制御することができます。

- パフォーマンス ライブラリ（*The Performance Library*）を使用すると、ホスト コンピュータで動作するSDLシステムのパフォーマンス モデルを作成できます。ライブラリは、パフォーマンスに対して最適化されており、大量の統計データを短い実行時間で生成できます。

- Cadvanced SDL to Cコンパイラを使用すると、ホスト環境とターゲット環境の双方に対するアプリケーションの作成（*アプリケーションの構築*）が可能です。定義済みのアプリケーション ライブラリは特定のホスト環境で使用できます。マスタ ライブラリ（*The Master Library*）は、SDL Suiteランタイム ライブラリのソース コードであり、さまざまな要求仕様やオペレーティング システムに合わせてカスタマイズできます。

  オペレーティング システムとの統合機能（*Integration with Operating Systems*）によって、ほとんどの市販のリアルタイム オペレーティング シス

テムがサポートされます。また、ランタイム ライブラリによってシステムがスケジューリングされ、リアルタイムのペースが設定されるアプリケーションを構築できます。

- Cmicro SDL to Cコンパイラ（*The Cmicro SDL to C Compiler*）は、マイクロコンピュータを使って制御する小、中規模アプリケーションの要求を満たすように、設計されています。Cmicroコード ジェネレータによって、SDLシステムはコンパクトで、最適化されたCコードに変換され、メモリに対する負荷が大幅に削減されます。Cmicro SDL to Cコンパイラは、*Cmicro*パッケージのコンポーネントであり、CmicroライブラリやSDLターゲット テスタと共にCmicroパッケージを構成しています。Cmicroパッケージは、別途購入する必要があります。
  - Cmicroライブラリ（*The Cmicro Library*）は、「仮想SDLマシン」であり、生成されたCmicroコードから実行可能ファイルを構築するために必要です。
  - SDLターゲット テスタ（*The SDL Target Tester*）を使用すると、生成されたSDLシステムのテストとデバックを、ターゲット環境で実行できます。ただし、その場合はホスト システムとの通信リンクがあらかじめ必要になります。SDLターゲットテスタは、Cmicroパッケージのオプション コンポーネントです。
- TTCNテスト スイート ジェネレーション（*TTCN Test Suite Generation*）には、TTCNリンクとオートリンクの2つの機能があります。これらの機能を使用することにより、SDL Suiteで管理するSDLシステムと、TTCN[1] で表現しTTCN Suiteで管理するテスト仕様との整合性をチェックできます。TTCNリンクはテスト仕様の宣言を自動的に生成します。TTCN Suite内のテスト ケース エディタからのSDLシステム仕様に直接アクセスし、テスト ケースを対話形式で作成することができます。オートリンクは、エクスプローラの機能の1つでありSDLの仕様から完全なテスト スイートを生成することができます。

1. TTCN（Tree and Tabular Combined Notation）は、ISOの標準であり、テスト仕様の記述に使用されます。

情報管理

SDL Suiteを適切に使うには、各情報の基本構成を理解することが必要です。

SDL ダイアグラム

SDL Suiteでは、主にSDL情報を図式表現で記述するSDL/GRを扱います。この方法の主な利点は、シンボルや外形線を自由に配置できるため、ダイアグラムの記述ルールを任意に決めることができます。

各SDLダイアグラムは、複数のダイアグラム ページで構成されます。SDLダイアグラム ページには、別のSDLダイアグラムへの参照を記述することができます。これにより、SDL構文規則に準拠した方法で階層構造を構築できます。図16を参照してください。

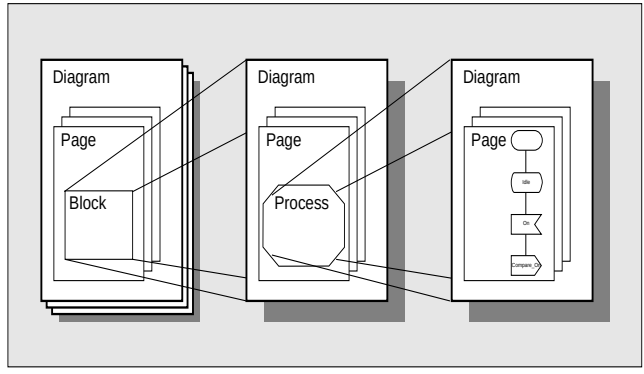


図16: SDL情報の構成

各ダイアグラムは、それぞれ個別のファイルに格納されます。SDL構造は、多数のSDLファイルで構成され、単一のSDL構造を構築するように、SDL Suiteのコンポーネントによって論理的に結合されます。そして、これらの処理は、オーガナイザによって管理されます。SDL Suiteでは、複数の独立したSDL構造を同時に管理することもできます。

また、SDL/PRファイルをSDL/GR構造内へ組み込むことができます。SDTでは、SDL/GRからSDL/PRへの変換やその逆の変換がサポートされています。

MSCダイアグラム

メッセージ シーケンス チャートは、主に図式表現であるMSC/GRで記述します。

SDLダイアグラムとは対照的に、MSCではページ分割ができません。また、階層構造を構成することもできません。ただし、MSCダイアグラムから別のMSC（または下記のHMSC）を参照することができますが、この場合も両者の間に構造情報を定義することはできません。

MSCダイアグラムは、それ自体を独立して管理することができます。また、オーガナイザでMSCを*関連ドキュメント*として定義することにより、SDL構造に組み込むことができます。

SDL Suiteでは、テキスト表現のMSC/PRで記述されたMCSに対して、読み取りや書き込みが可能です。また、勧告に準拠した、*インスタンス指向*と*イベント指向*の両形式がサポートされています。

ハイレベルMSC

通常のMSCとは異なり、ハイレベルMSC（HMSC）ではページ分割がサポートされます。ただし、オーガナイザで表示可能な、階層構造を構築することはできません。MSCダイアグラムから、別のHMSCやMSCを参照することができますが、両者の間に構造情報を定義することはできません。

SDLとMSCのテキスト形式

SDL Suiteでは、SDLとMSCのテキスト形式のファイルであるSDL/PRとMSC-PRの読み取りと書き込みが可能です。この機能の主要な目的は、SDLやMSC情報のインポートやエクスポートを可能にすることであり、代替となる保存形式をサポートすることではありません。ダイアグラムを、PR形式で保存して再度読み取ると、レイアウトや正確な表示状態が失われてしまいます。

なお、SDL Suiteでは、情報を処理する際に一時的な保存形式として、PR形式を使用する場合もあります。

SDL Suiteに付属されているCIFコンバータを使用すると、SDL/GRダイアグラムとCIF（Common Inter-change Format）ファイルとの間の双方向の変換が可能です。CIFはSDL-PRを拡張した表現方法であり、図式のレイアウト情報も格納できます。ただし、CIFファイルはSDL Suiteで直接管理することができません。

オブジェクト モデル ダイアグラム

オブジェクト モデル（OM）ダイアグラムは、それ自体を独立して管理することができます。また、オーガナイザ内で*モジュール*として複数のダイアグラムをまとめることもできます。

OMダイアグラムは、ページ分割ができますが、構造情報を定義することはできません。ただし、*スコープ*の概念を適用すれば1つ以上のOMページやダイアグラムを同じオブジェクト クラス内にまとめて定義できます。クラスに対する完全な定義が必要な場合は、スコープ内のすべてのダイアグラムとページを考慮に入れなければなりません。オーガナイザの同じ*モジュール*内で管理されているOMダイアグラムは、同じスコープ内にあることを示します。

ステート チャート ダイアグラム

ステート チャート（SC）ダイアグラムは、ページ分割することができますが、構造情報や別のダイアグアムへの参照を作成することはできません。ステート チャートは、それ自体を独立して管理することができます。

テキスト ドキュメント

SDL Suite and TTCN Suiteでは、標準的なASCIIファイル形式のテキスト ドキュメントを使います。SDL Suite and TTCN Suiteは、テキスト ファイルの拡張子によって種類を識別し、Cヘッダー、ASN.1、および通常のテキスト ファイルとして認識します。

テキスト ドキュメントは、それ自体を独立して管理することができます。また、CヘッダーおよびASN.1は、*依存リンク*を使用してシステム内の別の部分とリンクすることができます。この方法によって、テキスト ドキュメントの分析や、SDL/PR形式への変換が可能です。

また、テキスト ドキュメントは、ビルド スクリプトとして使用される場合もあります。ビルド スクリプトには、分析やコード生成プロセスを詳細に制御するためのコマンドが記述されます。

システム ファイル

SDL Suiteを起動して作業を始めると、個々のドキュメントを独立したオブジェクトとして使用できます。ただし、そのためには個々のファイルを追跡し続けなければなりません。

ドキュメントが増えると、トレースはかなり複雑になり、特にSDLダイアグラム間の継承や特殊化を導入した場合や、さまざまなドキュメント間の依存リングがある場合はいっそう難しくなります。

この問題に対処し、ドキュメント構造の一貫性を維持する手段として、システム ファイルを使用する方法があります。*システム ファイル*は、オーガナイザで管理できます。

ドキュメント構造

システム ファイルには、SDL構造や他のドキュメントの情報がすべて保持されます。また、ファイルの結合情報も記録され、特定のドキュメントを保存しているファイルの情報や、ドキュメント間の依存関係を表わすリンク情報などが格納されます。ドキュメントの編集時に加えた変更は、すべてオーガナイザによって監視され、変更内容に応じてシステム ファイルが更新されます。

オーガナイザは、システム ファイルの内容を図式表示します。ドキュメントは、自由に*チャプタ*やチャプタ内の*モジュール*に入れて分類することができ、関連するドキュメントをまとめておくことができます。

異なるチャプタ内や、モジュール内、SDL構造内に存在するドキュメントの関係は、*関連付け*や*依存リンク*の形式で記録することができます。

オプション

上記の他に、システム ファイルが管理するドキュメント構造に設定した*オプション*情報も、そのシステム ファイルに格納されます。通常、*分析*オプションや*コード ジェネレーション*オプションが、システム ファイルに格納されます。

リンク ファイル

リンク マネージャーは、システム内のすべてのリンク エンドポイントとインプリメンテーション リンクを記録します。リンク データベースは別のリンク ファイルに格納され、リンク ファイルはシステムファイルから参照されます。

コントロール ユニット ファイル

コントロール ユニット ファイルは、SDLシステムを複数のユーザーで使う場合に利用します。コントロール ユニット ファイルには、ドキュメント システム内のサブセットの構造情報が格納されます。また、コントロール ユニット ファイルは、コンフィグレーション管理（リビジョン管理）への対応が考慮されています。シ

ステム ファイルは、コントロール ユニット ファイルが存在する場合、このファイルを参照します。

ソース管理

SDL Suiteの処理はファイル単位で行われるため、任意のリビジョン処理システムを使用して、SDLダイアグラムの作業中ファイルと、作業用ディレクトリ内に保存されているファイルとの整合を取ることができます（この処理は、SDL Suiteの外で実行する必要があります）。

ドキュメントをファイル システム内の適当なファイルに結合することで、SDL Suiteがソース ドキュメントの複数の版数を管理するように設定できます。

SDL Suiteには、ドキュメントを簡単に再結合する機能があります。このファイルの結合機能を使用すると、ソース ドキュメントの複数の版数を最小の労力で管理できます。

ターゲット管理

SDL Suiteによって生成される情報は、事実上すべてがファイルに出力されます。そして、それらのファイルのほとんどがテキスト形式のファイルです（たとえば、SDL/PRファイルやCファイル）。

生成されるファイルの出力場所の設定は変更が可能です。また、出力されるファイルの大きさを指定して、複数のファイルを生成したり、1つのファイルだけを生成したりできます。

さらに、SDL Suiteには*SDL-実装*という機能があり、ソース ダイアグラムの修正に対応して実行されるツールへの経路が計算され、応答時間が最小に抑えられます。

パーソナル コンピュータとワークステーション

ユーザー インターフェイス

UNIXワークステーションで使用するSDL Suiteは、Motifウィジェット セットを使ったX Windowアプリケーションとして実装されます。また、パーソナル コンピュータに対応するSDL Suiteは、Microsoft Windowsアプリケーション（Windows 2000とWindows XP）として設計されています。現在、パーソナル コンピュータ プラットフォームでは、利用できない機能があります。

SDL Suiteは、さまざまなシステムに対してサポートされているため、環境によってはツールの表示がわずかに異なる場合があります。しかし、基本的なシステムのレベルの機能や、OSがサポートする機能においては同一です。

すべてのSDL Suite and TTCN Suiteグラフィカル アプリケーションは、『User´s Manual』の第1章、「User Interface and Basic Operations」に記載されている同一のスタイル ガイドにしたがっています。

UNIXシステムのサポート

パーソナル コンピュータのSDL Suiteと、UNIXワークステーションのSDL Suiteとの間には完全な互換性があります。したがって、コンピュータをワークステーションにアップグレードした際にも、既存データをそのまま利用できます。また、UNIXベースのファイル サーバーにパーソナル コンピュータを接続しているような、異機種のコンピュータで構成されたネットワーク ソリューションにも対応できます。

ワークステーションの環境に対するアーキテクチャとオペレーティングシステムのサポートを以下に示します。

- Sun SPARCstation (Solaris)

プラットフォームのサポートに関する詳細は、『Installation Guide』の第1章、「プラットフォームおよび製品」を参照してください。

第 3 章

チュートリアル: SDLエディタとアナライザ

SDL Suiteは、リアルタイム システムなどのシステムに対する設計と仕様記述に使います。SDL Suiteでは、ITUのZ.100勧告が提唱する仕様記述言語(SDL)がサポートさています。さらに、メッセージ シーケンス チャート（MSC）とUML（Unified Modeling Language: 統一モデリング言語）がサポートされています。ただし、UMLが完全にサポートされているツールが必要な場合には、IBM Rational Tau/Developerを使用してください。

このチュートリアルは、SDL言語の使用経験があり、メッセージ シーケンス チャートに関する基本的な知識のある方を対象としています。

チュートリアルでは、簡単なSDLの仕様を事例に挙げて、SDLエディタの編集機能や分析機能を説明します。そして、SDLアナライザやSDLエディタ、MSCエディタの操作に慣れることができる、さまざまな演習が用意されています。

これらのツールの使用方法を正しく理解するために、このチュートリアルの内容をすべて読んでください。また、記載されている手順にしたがって、お使いのコンピュータ システムで実際に演習してください。

このチュートリアルの目的

このチュートリアルでは、SDL Suiteのユーザー インターフェイスや、基本的な編集機能の操作に慣れていただくことを目的としています。また、SDL Suiteの機能を理解するためのガイダンスとなるように配慮されています。チュートリアルを読む際は、説明されている各演習をコンピュータで実際に操作して確認してください。

このチュートリアルでは、内容を容易に理解できるように比較的簡単な題材を選んでいます。ただしSDL言語の学習を目的として**いない**ため、説明はすべてSDL言語に関する基礎知識があることを前提に記述されています。

このチュートリアルはSDL Suiteを使った経験が、ほとんど、またはまったくない方でも理解できるように構成されています。

この章での学習が終了したら、以下のチュートリアルの章に進んでください。

- 第4章「チュートリアル: SDLシミュレータ」
- 第5章「チュートリアル: SDLエクスプローラ」
- 第6章「チュートリアル: SDL-92のDemonGameへの適用」

> メモ：プラットフォーム間の相違点
>
> このチュートリアルは、UNIXとWindowsプラットフォームで同じ手順で実施できます。プラットフォームの間で相違点がある場合は、「UNIXのみ」または「Windows」などの記述を必ず表記します。また、画面表示についても、プラットフォーム特有の表示内容がある場合には同じような記述を示します。
>
> このようなプラットフォームについての記述がある場合は、ご使用のプラットフォームに関する記述のみを参照してください。

Demon Game

このチュートリアルで使用する例題は、「Demon Game」を簡略化したものです。Demon Gameは、SDLの勧告に事例として取り上げられているため、SDLの業界ではよく知られているものです。

Demon GameのSDL仕様は、SDL/GRの形式でこの章の最後に記載します(128ページの「付録A: SDL-88 DemonGameの定義」を参照してください)。Demon Gameの振る舞いの定義は、ゲームを説明する手段としては最も簡単な方法とはいえませんが、シミュレーションとバリデーションを実行するのに適しているため採用しています。

Demon Gameの振る舞い

Demon Gameのシステムを外部から見た振る舞いは次のようになります。SDLシステムは、外部環境からNewgame、Endgame、Probe、および、Resultという4種類の信号を受信します。最初の2つの信号はゲームの開始時と終了時に使用します。一度に実行できるゲームは1つです。つまり、ゲームの進行中にはNewgame信号は無視され、ゲームの進行中でない場合にはEndgame信号は無視されます。

ゲームのルールは非常に簡単です。SDLシステム内でDemonプロセスとして表現される、いわゆる「デーモン」は、たびたびシステムの状態を「勝ち」と「負け」の間で変更します。これらの状態は、GameプロセスのWinning状態とLosing状態として表現されます。ユーザーは、Gameプロセスの状態がいつWinningになるかを推測しなければなりません。ユーザーが、Probe信号を送信して探査した際に、Winningの状態になっていれば1ポイント得点できます。状態がLosingの時にProbe信号を送信した場合には1ポイント失います。システムはWinまたはLose信号でProbe信号に応答し、ユーザーの勝ち負けを通知します。また、現在の得点を確認するために、Result信号を送信することができます。システムは、Result信号に対してScore信号で応答します。Score信号には、現在の得点を示す整数型のパラメータがあります。

SDL Suiteの起動

事前準備

このチュートリアルでは、SDL Suiteが『*Installation Guide*』の手順に従って正しくインストールされていることを前提にしています。

> メモ：インストール ディレクトリ
>
> UNIXにおけるインストール ディレクトリの環境変数は、`$telelogic`になります。UNIX環境内にこの変数が設定されていない場合は、システムの管理者やSDL Suite環境の管理担当者に正しい環境変数の設定方法を確認してください。
>
> Windowsにおけるインストールディレクトリとして、このチュートリアルでは`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J`を想定しています。PCにこのディレクトリが設定されていない場合は、システムの管理者やSDL Suite環境の管理担当者に正しいパスの設定方法を確認してください。

インストールの際、UNIXでは`$telelogic/sdt/examples/demongame`ディレクトリが、Windowsでは`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥sdt¥examples¥demongame`ディレクトリがそれぞれ生成されます。そして、これらのディレクトリ内には模範解答のサンプル ファイルが保存され、正しい解答がわからない場合や、演習を飛ばして先に進む場合などにこのファイルを利用することができます。

完成しているサンプル ファイルを誤って変更しないように、チュートリアル専用の作業用ディレクトリを作成してください。

UNIXの場合は以下の手順で作業用ディレクトリを作成します。

1. 以下のコマンドを入力してホームディレクトリ内に、新しいサブディレクトリを作成します。

   ```
   mkdir ~/demongame
   ```

   このチュートリアルでは、作業ディレクトリ名にこの名前を使用します。

Windowsの場合は以下の手順で作業用ディレクトリを作成します。

1. パーソナル コンピュータに`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥work¥demongame`を作

成します。このチュートリアルでは、作業ディレクトリ名にこの名前を使用します。

> メモ：Windowsでは、スペースを名前に使うことはできません。Windowsで使用するSDL Suiteはスペースを含むファイル名やディレクトリ名を認識できません。スペースを使用しないように注意してください。

SDL Suiteの起動

UNIXでは、以下の手順でSDL Suite環境を起動します。

1. 次のコマンドを実行して現在のディレクトリからdemongameディレクトリへ移動します。

   ```
   cd ~/demongame
   ```

2. 次のコマンドを実行してSDTを起動します。

   ```
   sdt
   ```

> メモ：
>
> `sdt`コマンドが実行できないときには、まず`$path`変数を正しく設定する必要があります。設定方法がわからない場合はシステム管理者、またはSDL Suite環境の管理担当者に確認してください。

Windowsでは、チュートリアルの学習に適した方法でSDL Suiteを起動するために、以下のようにショートカット アイコンを作成し、使用します。

1. `C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥bin¥wini386`内でSDL Suiteの実行可能ファイル`sdt.exe`または`sdt`を探します（このディレクトリが見つからない場合は、42ページの「インストール ディレクトリ」を参照してください）。
2. このファイルへのショートカット アイコンをWindows デスクトップ上に作成します。
3. 新しいアイコンをマウスの右ボタンでクリックし、ポップアップメニューから[プロパティ]を選択します。ダイアログ ボックスの一番上に表示されている[ショートカット]タブを選択します。
4. [作業フォルダ]フィールドに、新しいディレクトリのパスとして「`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥work¥demongame`」を入力します。
5. [OK]をクリックしてダイアログ ボックスを閉じます。
6. ショートカット アイコンをダブルクリックします。

オーガナイザ ウィンドウ

SDL Suiteを起動すると、*オーガナイザ ウィンドウ*が表示されます（図17と図18を参照）。オーガナイザは、SDL Suiteの各種ツールへアクセスするために使用する主要なツールです。

また、オーガナイザにはようこそウィンドウが表示され、SDL Suite and TTCN Suiteのライセンス契約の内容を確認することができます。このウィンドウは、オーガナイザで最初に表示され、何かを実行するとすぐに消えます（[継続]ボタンをクリックした場合も同様です）。

これで、作業の準備が完了しました。

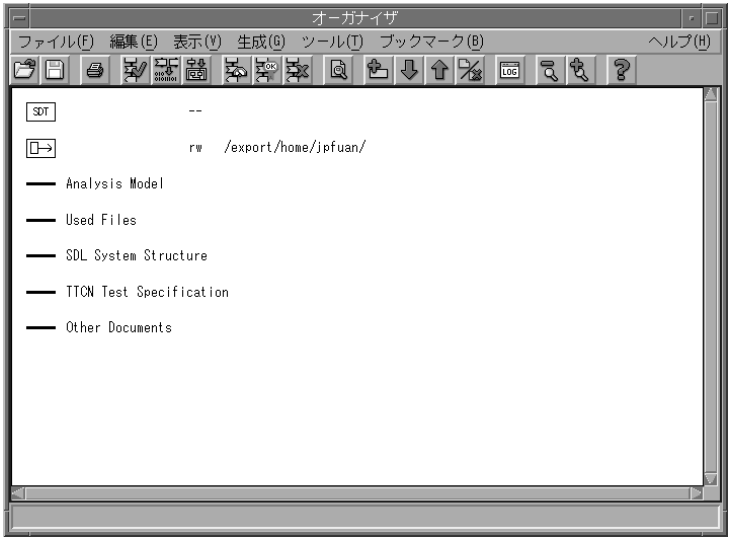


図17: IBM Rational Organizer ウィンドウ（UNIX）

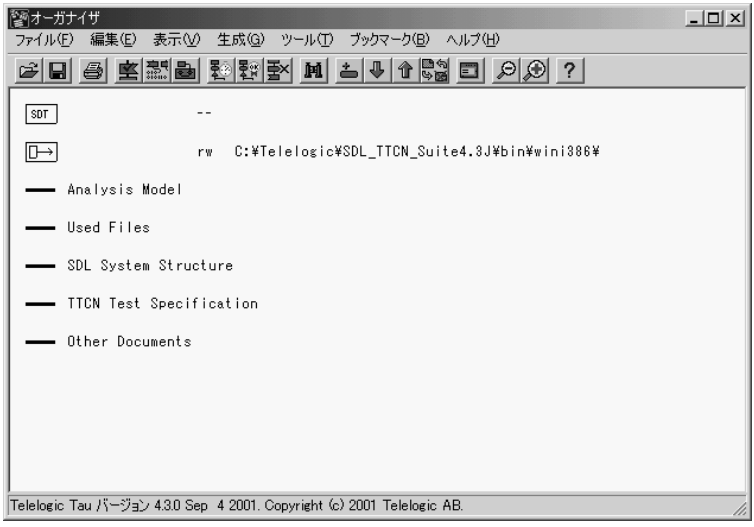


図18: オーガナイザ ウィンドウ（Windows）

メモ：画面表示

図18と図19には、オーガナイザ ウィンドウの画面表示としてUNIXとWindowsの双方が示されています。これ以降の説明では、両方のプラットフォームで画面に表示される情報が等しい場合、どちらか一方の画面表示のみを示します。したがって、図のレイアウトや概観は、実際にコンピュータに表示される画像と異なることがあります。

重要な部分がプラットフォーム間で異なる場合に限って、それぞれの画面表示を示します。

環境設定

学習内容

- ユーザー環境の設定と保存
- 以下のSDL Suiteの基本的なグラフィカル ユーザー インターフェイスに対するマウスとキーボードからの操作
  - グラフィカル リストの使い方
  - プルダウン メニューの使い方
  - ポップアップ メニューの使い方
  - クイック ボタンの使い方
  - ステータス バーの使い方
  - オプション メニューの使い方
  - テキスト フィールドの使い方
  - スライド バーの使い方
  - キーボードからのショートカット キーの入力

ユーザー環境設定の目的

SDLダイアグラムを作成する前に、いくつかの環境設定パラメータをコンピュータの環境に合わせて設定する必要があります。これらの環境設定によって、SDL Suiteツールのデフォルトの動作が決定されます。また、SDL Suiteを正しく機能させるために使用環境に適合した値を設定すべきです（ほとんどのオプションは、この環境パラメータで設定可能です）。SDL Suiteをインストールした直後の環境パラメータは、工場出荷時の設定になっています。システム管理者によって環境パラメータがすでに設定されている場合も考えられますが、いずれにしても環境設定の内容を確認することをお勧めします。

少なくとも以下の項目については確認すべきです。

- ヘルプ環境の設定
- プリンタ環境の設定
- ドローイング エリアのサイズ
- プラットフォームのモード

環境設定の表示と変更

環境設定パラメータの表示と変更の手順を以下に示します。

1. オーガナイザの[ツール]メニューから、[環境設定マネージャ]を選択します。[環境設定マネージャ]ウィンドウが表示されます。

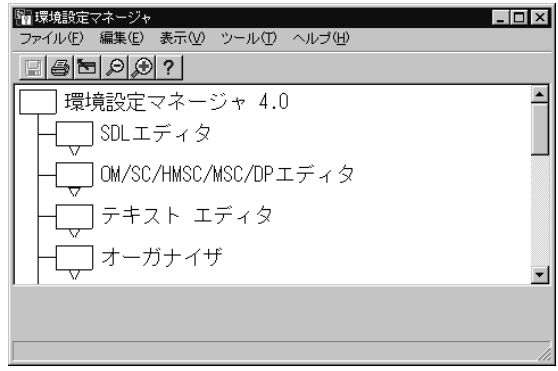


図19: [環境設定マネージャ] ウィンドウ

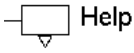
次に、環境パラメータを確認し、必要に応じて設定内容を変更します。

ヘルプ環境の設定

SDL Suiteでは、コンテキスト対応のオンライン ヘルプ機能がサポートされ、ウィンドウや、コマンド、ダイアログ ボックスなどからヘルプにアクセスすることができます。

UNIX上ではオンラン ヘルプはHTMLで記述されており、ヘルプ ビューワーとして、Firefox、Netscape Navigator、Internet Explorerを使用できます。ヘルプ ビューワーは環境設定の修正によって変更できます。

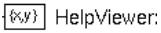
ヘルプ環境の設定手順を以下に示します。



1. [ヘルプ]と表示されたアイコンを見つけて、ダブルクリックします。

   – または、[ヘルプ]アイコンを右クリックして、ポップアップ メニューから[展開]を選択します。

   これによりリスト構造が展開されて、ヘルプの環境設定が見えるようになります。

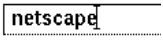


2. [ヘルプ ビューワー]環境パラメータを見つけます。

   環境パラメータのアイコンの右側には、現在の値、保存されている値、および説明文が表示されています。

3. 環境設定を変更するには、[ヘルプ ビューワー]アイコンをクリックして、ウィンドウの下部に表示されるオプション メニューから、新しいビューワーを選択します。使用できるビューワーがわからない場合は、システム管理者に確認してください。[ヘルプ ビューワー]アイコンが灰色表示され、パラメータが変更されたことと、設定を保存しなければならないことが示されます。

4. 選択するビューワーの種類によっては、ヘルプ ビューワーの起動時に実行するコマンドが実際のコンピュータ環境に適合しているかを確認する必要があります。
   - [NetscapeCommand]または[InternetExplorerCommand]と表示されたアイコンを見つけて選択します。現在の値が右側に表示されます。値が正しくない場合には、ウィンドウの一番下にあるテキスト フィールドにテキストを入力します。正しい値が分からない場合には、システム管理者に確認してください。



5. [ヘルプ]アイコンをダブルクリックして、折り畳みます。

ここでは、グラフィカル リストの使用方法を学習しました。グラフィカル リストはあらゆるツールに対して設定が可能です。また、すべての階層に対して設定することができます。たとえば、環境設定マネージャには、画面に表示されているとおり3つの階層があります。

いくつかのツールでは、グラフィカル リストの代わりに垂直ツリーがサポートされています。これらは表示方法が異るだけで機能は同じです。グラフィカル ツリーについては、このチュートリアルで後述します。

デフォルト プリンタの設定

ここでは、ダイアグラムの印刷方法を学習します。印刷を始める前に、プリンタ環境を確認し、必要に応じてコンピュータの環境に合わせて設定を変更してください。

プリンタ環境の設定手順を以下に示します。

1. [印刷]アイコンをクリックし、パラメータを展開します。
2. [PrinterCommand]環境パラメータを選択します。印刷に必要なコマンド文字列を設定します（わからない場合はシステム管理者に確認してください）。ここでは、いかなるオペレーティング システムのコマンドをも設定することができます。たとえば、出力された印刷データを`lpr`コマンドによって印刷

キューへ送信したり、PostScript ファイルのプレビューをGhostview[1] などで参照したりすることができます。

3. [PaperFormat]環境パラメータを選択します。SDL Suiteでは、オプション メニューでさまざまな用紙サイズを選択することができます（A4や、A3、米国レター、米国リーガルなど)。また、[UserDefined]の値を設定して任意のサイズを指定できます。その場合は[UserDefinedWidth]や [UserDefinedHeight]環境パラメータを指定します。これらの値はミリメートル単位で表します。
4. 必要に応じて、[MarginUpper]、[MarginLower]、[MarginLeft]および [MarginRight]環境パラメータを調整します。これらの環境パラメータによって、プリント ページの余白にどれくらいのスペースを割り当てるかをミリ単位で調整できます。また、この領域には印刷出力のヘッダとフッターを入れることができます。

   – これらの環境パラメータを調整する際には、環境パラメータを選択し、スライダをドラッグして粗い調整を行います。そして、スライド バーの右または左をクリックするか、矢印をクリックして微調整します。
5. 必要ならば[Landscape]環境パラメータをオンまたはオフにします。この環境パラメータで紙の方向のランドスケープ（横）またはポートレート（縦）を選択します。

ドローイング エリア サイズの設定

SDLダイアグラムを編集する際には、あらかじめ決められたサイズのページが割り当てられます。ページのデフォルトのサイズは、前項で設定したプリント ページとプリント マージンのサイズに合わせる必要があります。

1. 一番上の[SDLエディタ]アイコンをクリックして展開表示します。 [PageWidth]と、[PageHeight]パラメータを確認します。
2. 必要に応じて、これらのパラメータを適切な値に変更します。

環境パラメータの保存

設定した環境パラメータを保存して後の作業で使用できるようにします。

1. [ファイル]メニューで、[保存]を選択します。

   – この操作の代わりに、[保存]クイック ボタンをクリックすることもできます。クイック ボタンは、メニュー バーのすぐ下のツール バーに表示さ

1. Ghostviewはゴーストスクリプトのユーザー インターフェイスです。1992 Timothy O. Theisen.

れています。クイック ボタンは、頻繁に使うコマンドに割り当てられたマウス ボタンであり、[環境設定マネージャ]ウィンドウばかりでなく、すべてのSDL Suite and TTCN Suiteツールでサポートされています。

– クイック ボタンにマウスのカーソルを合わせるとそのボタンの機能を確認することができます。ボタンにカーソルを合わせることで、ウィンドウの最下部にあるステータス バーに説明文が表示されます。また、マウス ポインタをクイック ボタン上に一定時間合わせると、ボタンのすぐ下に簡単な説明が表示されます。
– `Ctrl+S`キーを押すことによって、[保存]コマンドを実行することもできます。このショートカット キーは、メニューの[保存]コマンドの真横に表示されています。

2. 変更した環境設定が各ツールを再起動するまで適用されないことを通知するメッセージが表示されます。[OK]をクリックしてこのメッセージを了解します。

   環境設定がファイルに保存され、現在および今後の作業で利用できるようになります。

3. [ファイル]メニューから[終了]コマンドを選択して、[環境設定マネージャ]ウィンドウを閉じます。

これで環境設定は終了です。ただし、[環境設定マネージャ]に戻れば、いつでも他のパラメータを調整することができます。

SDL構造の作成

ここではSDLダイアグラムを作成します。

学習内容

- オーガナイザ チャプタのカスタマイズ
- SDL構造の作成
- システム ルート ノードの追加
- システム ダイアグラムの作成
- ページの追加
- システム ダイアグラムの作成
- ファイルへのダイアグラムの保存
- システム ファイルへのダイアグラム構造の保存
- ダイアログの使用（モーダルとモードレス）
- ツリー構造の使用

オーガナイザ チャプタのカスタマイズ

SDL Suiteを起動すると、オーガナイザには2種類のアイコンが表示されます。それらのアイコンは、システム ファイルとダイアグラムのソース ディレクトリをシンボル化したものです。システム ファイルについては後ほど説明します。ソース ディレクトリは、SDL Suiteのコンポーネントが既存ダイアグラムの保存場所として参照するディレクトリであり、また、新規ダイアグラムの保存場所として使用します（ソース ディレクトリの場所は変更可能です)。

ソース ディレクトリは、SDL Suiteを起動した場所にすでに設定されているはずです（UNIXの場合は、`~/demongame`、Windowsの場合は、`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥work¥demongame`）。

デフォルト設定では、SDL Suiteを起動するとオーガナイザに5つの領域が表示されます。

- *Analysis Model*
- *Used Files*
- *SDL System Structure*
- *TTCN Test Specification*
- *Other Documents*.

これらの領域はチャプタと呼びます。チャプタは複数のダイアグラムやドキュメントを保存するために使用します。チャプタのデフォルト設定はあくまでも推奨する構成の1つであり自由に変更できます。このチュートリアルでは、単純なシステムを設計するので、[Analysis Model]や[Used File]、[TTCN Test Specification]などのチャプタを削除してから作業を始めます。

チャプタは以下の手順で削除します。

1. [Analysis Model]チャプタを選択します。
2. [編集]メニューから[削除]を選択します。またはキーボードの`Del`キーを押します。
   - ダイアログ ボックスが表示されます。ダイアログ ボックスの[Remove]ボタンをクリックして削除します。

図20: チャプタの削除

3. 上記の手順を繰り返して、[Used File]チャプタと[TTCN Test Specification]チャプタを削除します。

また、残った2つのチャプタに新しい名前を付けることもできます。

1. [SDL System Structure]チャプタを選択します。
2. [編集]メニューから[編集]を選択するか、チャプタをダブルクリックします。
3. ダイアログ ボックスが表示されたら、[チャプタ シンボルの編集]が選択されていることを確認して、[編集]ボタンをクリックします。

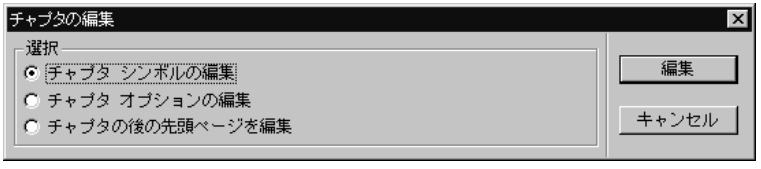


図21: チャプタ シンボルの編集

4. [編集]ダイアログ ボックスでドキュメント名を変更します。例えば、「My first SDL system」などの名前を入力します。ここで、[オーガナイザ]ボタンと、オプション メニューの値[Chapter]によって設定されているドキュメントタイプを変更しないでください。

図22: チャプタ名の入力

   – UNIXでは、オーガナイザの親ウィンドウの上にカーソルを置くと疑問符の形に変わります。この規則は、ほかの操作を行う場合に現在開いているダイアログ ボックスを閉じなければならないことを表します。ほかの操作に移る前に、操作しなければならないダイアログ ボックスをモーダル ダイアログ ボックスと呼びます。

5. [エディタで表示] オプションを**オフ**にします。

6. [OK]ボタンをクリックして終了します。

   – [Other Documents]チャプタの名前も、同じように変更することができます。このチャプタは、SDL以外のダイアグラムを保持するために、このチュートリアルで後から使用します。

システム ダイアグラムの作成

ルート ノードの追加

ここでは、SDLシステムをトップダウン方式で作成します。

1. オーガナイザで、[My first SDL system]チャプタが選択されていることを確認します。
2. [編集]メニューで[新規追加]をクリックします。[新規追加]ダイアログ ボックスが表示され、追加するダイアグラムの名前とタイプを指定するように促されます。

図23: ダイアグラムの新規追加

3. 上図のように、[新しいドキュメントのタイプ]を、[SDL]に設定し、SDLダイアグラムのタイプを[System]に設定します。
   - SDLダイアグラムのタイプにSystem以外の値が表示されている場合は、オプション メニューをクリックして変更します。
4. [新しいドキュメント]に「`DemonGame`」と入力します（デフォルトの[無題]の表示が消えます）。
   - テキスト フィールドを変更する際は、カーソルをテキスト フィールドに移動するかテキスト フィールドでクリックする必要あります。
5. [エディタで表示]ボタンが**オフ**になっていることを確認します。

6. [OK]をクリックします。

   – ダイアログが閉じます。オーガナイザ ウィンドウが更新され、ルート ノードとして、DemonGameシステム ダイアグラムが新たに表示されます。ダイアグラムの右側に表示されている[未接続]は、まだファイルとの接続がない（ファイルに保存されていない）ことを意味します。

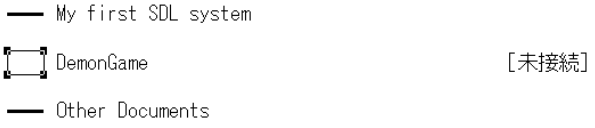


図24: 新しいルート ノード

システム ダイアグラムの作成

SDLシステム *ダイアグラムへの参照*が1つ存在する、オーガナイザの *ダイアグラム構造*の作成が完了しました。ただし、参照するダイアグラム自体はまだ存在していません。

次に、以下の手順でシステム ダイアグラムを作成します。

1. DemonGame SDLシステム ダイアグラムのアイコンを選択します。図24を参照してください。
2. [編集]メニューから[編集]を選択します。

   – または、アイコンをマウスの**右**ボタンでクリックします。ポップアップメニューが表示されたら、[編集]を選択し、サブメニューで[編集]をクリックします。

   – また、マウスの左ボタンでアイコンをダブルクリックしてダイアグラムを編集する方法もあります。

3. 新しいダイアグラムの作成と[エディタで表示]が有効になった状態で、[編集] ダイアログ ボックスが開きます（このダイアログ ボックスの表示内容は、先ほど設定した[新規追加]ダイアログ ボックスとほとんど同じです）。[OK]をクリックしてダイアグラムの作成とエディタでの表示を了解します。

図25: 新規ダイアグラムの作成要求

*SDLエディタ ウィンドウ*が表示され、左上の角にDemon Gameシステム ダイアログの1ページであることが表示されます。

SDLエディタは、ダイアグラムの内容を記述する際に使用するツールです。また、SDLエディタを使用してオーガナイザ ウィンドウで表示するダイアグラム構造を作成することができます。

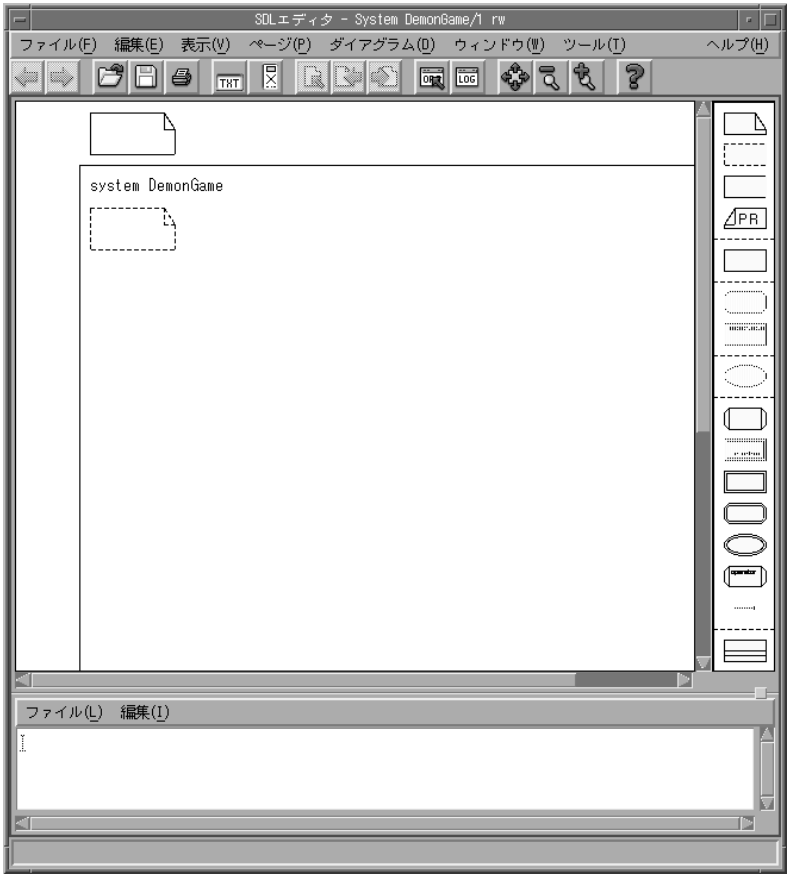


図26: SDLエディタ ウィンドウ（UNIX）

次に、ダイアグラムの内容を記述します。図27は、完成したダイアグラムの印刷イメージを表しています。このシステム ダイアグラムは、GameBlockとDemonBlockという2つのブロック参照シンボルで構成されています。また、2つのブロック間で信号を伝送するC3チャネルと、ブロックと外部環境との間で信号を伝送するC1およびC2チャネルがあります。さらに、信号の宣言が記述されているテキスト シンボルがあります。

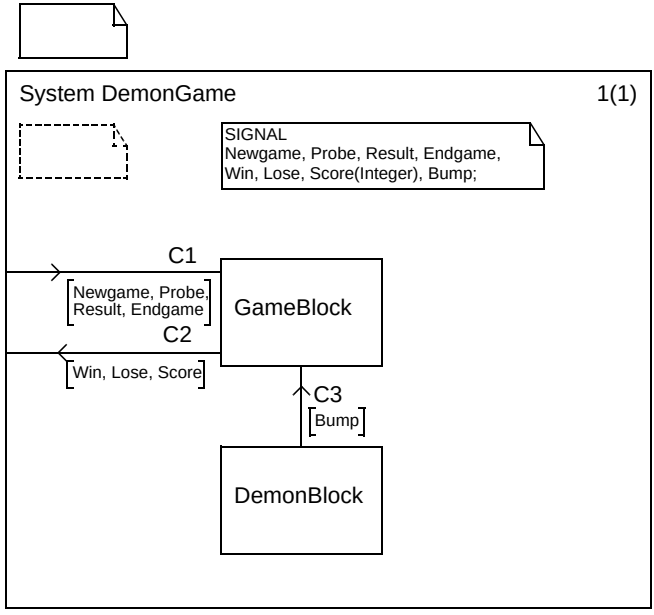


図27: システム ダイアグラム

次のページでは、シンボルとテキストをダイアグラムに追加する方法について詳細に説明します。

SDLエディタ ウィンドウのカスタマイズ

編集作業を始める前に、エディタ ウィンドウのサイズを調整することができます。また、[表示]メニューの[ウィンドウ オプション]を選択して、サブウィンドウの表示または非表示を設定することができます。

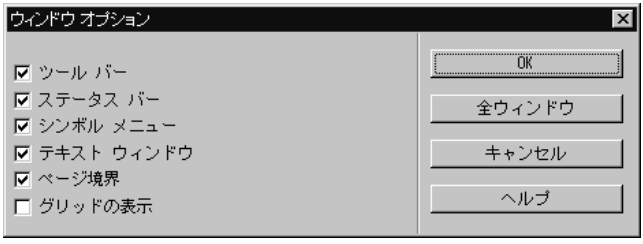
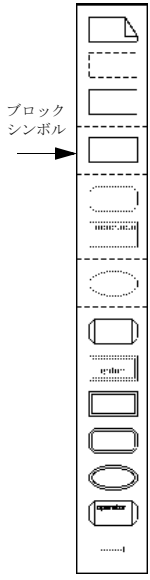


図28: ウィンドウ オプション

- エディタ ウィンドウ内の、ツール バー（[ツール バー]）やステータス バー（[ステータス バー]）、ページ境界（[ページ境界]）、グリッド（[グリッドの表示]）などを表示したり隠したりすることができます。

- また、テキスト ウィンドウと、シンボル メニューはクイック ボタンを使用して表示、非表示を設定できます。ただし、この2つのサブウィンドウは、この後すぐに使用します。

ブロック参照シンボルの設定



1. まずGameBlockとDemonBlockの2つのブロック参照シンボルを配置します。

   ブロック参照シンボルは以下の方法で配置します。

   – シンボル メニューでブロック シンボルをクリックします。UNIXの場合、シンボル メニューはウィンドウの右端に表示されています。Windowsの場合、シンボル メニューは別のウィンドウに表示され、SDLエディタ ウィンドウの手前に常に表示されます。

   どのシンボルを使用すべきかわからない場合は、シンボル メニューでシンボルをポイントまたは選択します。シンボルのタイプがSDLエディタ ウィンドウ最下部のステータス バーに表示されます。

   – シンボルをクリックしたまま、ドローイング エリアへマウスをドラッグします。シンボルがマウス カーソルについていきます。シンボルを配置する場所でマウスを再度クリックします。シンボルは 重ねて配置することはできません（シンボルを重ねて配置すると、警告音が鳴り、シンボルの配置を再度行わなければなりません）。

> メモ：操作を取り消して元に戻す
>
> - マウスをドローイング エリアに移動した後に、シンボルの配置を取り消す場合は、Escキーを押します。
> - 間違ったコマンドを選んだり、誤った操作をした場合には、すぐに[編集]メニューの[元に戻す]を選択してください。

2. シンボルを配置したら、DemonBlockおよびGameBlockというブロック名を付けます。

   – シンボルのテキストをクリックすると、そのカーソルの位置からテキストを直接入力できます。ただし、シンボルに表示されているテキストを直接選択する（反転表示させる）ことはできません。

– テキストの編集を始める前の時点では、テキスト カーソルは点滅しません。この状態で`Delete`キーを押すと、選択しているシンボル自体が削除されます。テキストの編集を始めると、テキスト カーソルが点滅し、`Delete`キーを押すと1文字だけ削除されます。

– SDLの名前付け規則に適合していない名前を入力した場合は、テキストに赤い下線が表示されることがあります。通常SDLエディタが構文エラーを通知するとこの赤い下線を表示します。

– テキストを編集する際は、*テキスト ウィンドウ*を利用すると便利です。テキスト ウィンドウを使用すると、テキストをドラッグによって選択することができます。UNIXでは、テキスト ウィンドウはドローイング エリアの下部にあります。Windowsでは、テキスト ウィンドウは別のウィンドウに表示され、SDLエディタ ウィンドウの手前に常に表示されます(シンボル メニューと同じです)。

– どの方法で入力した場合でも、編集中のテキストは、シンボル上のテキスト枠とテキスト ウィンドウの双方に常時表示されます。

– ブロック参照シンボルが挿入されると、オーガナイザのダイアグラム構造も、自動的に更新されます(シンボルの選択が解除された時に更新されます)。

シンボルの移動とサイズ変更

ブロックは次の方法で移動します。

- マウスでブロックを選択し*ドラッグ*して適当な位置へ移動します(他のシンボルの上に重ねて配置することはできません)。

適当な場所にブロックを配置できたら、ブロックのサイズを以下の手順で変更することが可能です。

- シンボルの角の1か所をポイントして、ドラッグします。この際、ポインタはシンボルの角と十分に接近させることが必要です。うまくできない場合は、まず、シンボルをクリックして選択し、シンボルの四隅に表示される四角形をクリックしてから、同じ手順を繰り返します。

ブロック間のチャネル作成

以下の方法で、DemonBlockブロックからGameBlockブロックへ向かうチャネルを作成します。

1. DemonBlockシンボルを選択します。ハンドルが表示されます。

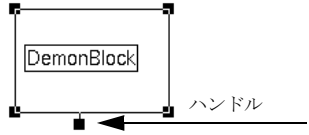


図29: ブロック シンボルのハンドル

2. ハンドルをドラッグします（ハンドルをクリックした状態でマウスのボタンを押しながらマウスを移動します)。
3. マウスの動きに合わせてエディタ上にチャネルを表すラインが引かれます。一度ハンドルを指定したら、マウスを動かす間、ボタンを放しても構いません。
4. GameBlockシンボルまでマウスを動かして、マウスのボタンをクリックします。チャネルがブロックの両端に接続されます。
   - チャネルの末端をドラッグして、末端を個々に移動させることができます。チャネルの末端をマウスで選択することが難しい場合は、まずチャネルを選択します。
   - SDLエディタ上に引かれたラインの中央に小さな四角形が表示されます。この四角形の部分で線を異なる線分に分割することができ、ラインを折り曲げる際に使用します。チュートリアルでは、この機能を使いません。

SDLエディタによって、チャネルに関連付けられた2種類の*テキスト属性*が生成されます。これらの属性のテキスト フィールドは*四角形の枠*として表示され、チャネル名の入力と、チャネルによって転送される信号リストの入力のために使用します。初期の状態ではテキスト属性は空欄になっており、SDLの構文規則に合っていないため赤い下線が表示されます。

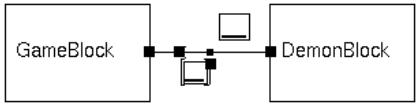


図30: チャネルのテキスト属性と赤い下線

2つのテキスト属性を区別しやすいように、2つのブロックは水平に配置されています。

以下の方法でチャネルの名前C3を入力します。

- チャネルを作成した直後に、直接チャネル名を入力します（チャネルの選択を解除した場合は、再度チャネルを選択します）。

以下の方法で信号リストのテキスト フィールドにBump信号を入力します。

1. 「[ ]」（角かっこ）内に表示されているテキスト フィールドをクリックします（角かっこ内をクリックします）。
2. 信号名を入力します。角かっこはテキストのサイズに合うように自動調整されます。
3. これらのテキスト属性は、マウスでドラッグして移動することができます。

ブロックから外部環境へ接続するチャネルの作成

以下の手順でブロックから外部環境へ接続するC2チャネルを作成します。

1. ブロックを選択します。
2. ハンドルをドラッグし、フレーム シンボル上でクリックしてドラッグを終了します（フレーム シンボルとは、ダイアグラムを囲む枠です。64ページの図32を参照）。
3. チャネルの名前と信号を入力します。
   - 信号リストの入力時には、SDLの構文規則に適合していないテキストの部分に赤い下線が表示されます。信号名の間にカンマを入力するとすぐに赤い下線は消え、テキストが正しい構文で記述されていることを示します。
   - SDLエディタでは、テキストの構文エラーをそのままにしておくことができます。ただし、構文エラーが残っている場合、SDLシステムを作成することはできません。

外部環境からブロックへ接続するチャネルの作成

以下の手順でブロックから外部環境へ接続するC1チャネルを作成します。

1. 前述の手順で、ブロックから外部環境へ接続するチャネルを作成します。
2. チャネルが選択されていることを確認します。
3. 矢印の方向を変えるために、[編集]メニューから[リダイレクト]を選択します。前述の方法でチャネル名と信号名を入力します。

– 信号名が長い場合は、信号リストの中でReturnキーを押し、改行することができます。

テキスト シンボルの作成

ダイアグラムに、信号の宣言を定義するテキスト シンボルを追加します。

1. シンボル メニューから、一番上のテキスト シンボルを選び、ドローイング エリアに追加します。図27にしたがって内容を入力します。入力に反応して、組み込みの構文チェックが動作します。

2. テキスト シンボルの内容が変更されると、エディタは、テキストに合うようにテキスト シンボルのサイズを自動変更します。また、テキスト シンボルは、右下の角をドラッグすることによってサイズを変更することができます。またシンボルをダブルクリックするとサイズを最大または最小にすることができます。これらの操作を実際に試してみてください。

   – テキスト シンボルの右下以外の3つの角はグレイで表示されます。これは、右下の角以外ではドラッグによるサイズが変更できないことを意味しています。

テキスト ウィンドウのサイズ変更

テキスト ウィンドウが小さすぎてすべてのテキストを画面上に表示できない場合は、サイズを変更することができます。Windowsでは、通常のウィンドウ操作と同じ方法で変更できます。UNIXでは、テキスト ウィンドウのメニュー バーの左上にある小四角形の*サッシュ*を上下にドラッグしてサイズを変更します。

図31: SDLエディタのサッシュ（UNIXのみ）

システム ダイアグラム内のそのほかのアイテム

ダイアグラムには、SDLシンボル以外に以下のようなアイテムがあります。

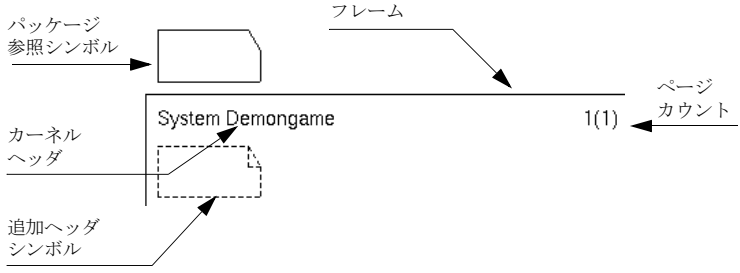


図32: その他のシンボル

パッケージ 参照シンボル

パッケージ参照は、システムに含まれるSDLパッケージを参照するために使用します。今回使用している事例では、パッケージを使用しません。したがって、パッケージ参照は空のままにしておきます。

カーネル ヘッダ

カーネル ヘッダには、編集対象となっているダイアグラムのタイプと名前が、エディタによって自動的に表示されます。このシンボルは編集可能ですが、ここでは編集方法については触れません。

追加ヘッダ シンボル

Z.100では、追加ヘッダ シンボルに関する詳細な規定はありません。SDLエディタによって、追加ヘッダ シンボルは点線で表示したテキスト シンボルのように表示されます。このシンボルは編集可能で、テキスト シンボルと同じ方法でサイズを変更することができますが、移動することはできません。SDLエディタにおける主な使い方には、ダイアグラムの継承と特殊化の定義、および仮パラメータの指定などがあります。このシンボルも、最初のチュートリアルでは使用しません。

フレーム

フレーム は、ダイアグラム内のオブジェクトを囲むように配置されます。よりコンパクトなダイアグラムを作成するためにフレーム シンボルのサイズを変更することもできます。フレームのサイズは角をドラッグするだけで変更できます。

メモ：

フレームは用紙の印刷範囲を表すものではありません。

ページ カウント
ページ カウントにはページ名と総ページ数が表示され、エディタによって自動更新されます。編集することはできません。

新しく作成したシステム ダイアグラムの保存

ここでは、SDLダイアグラムをファイルに保存するコマンドを使用します。

1. 現在、オーガナイザ ウィンドウとSDLエディタ ウィンドウの2つのウィンドウがディスプレイに表示されているはずです。

   – オーガナイザ ウィンドウは、SDLエディタの[ツール]メニューで[オーガナイザの表示]を選ぶか、[オーガナイザの表示]クイック ボタンをクリックすることによっていつでも表示可能です。

2. 保存する前に、オーガナイザの[表示]メニューから[表示オプション]コマンドを選択し、[表示オプション]ダイアログを開きます。

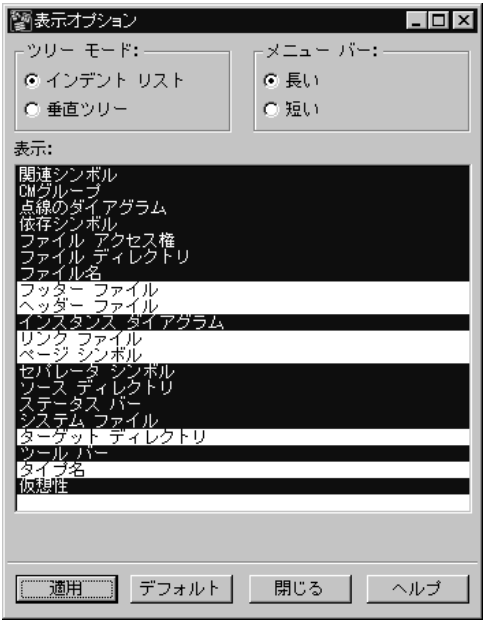


図33 オーガナイザの[表示オプション]

3. 図33に従ってオプションを設定し[適用]ボタンをクリックします。この操作によって、オーガナイザ内にファイル名やディレクトリ名が表示されます。

   – ダイアログに表示されているリストの項目は複数選択が可能です。各項目をクリックすることにより、他の選択内容に影響せずに、選択または非選択ができます。(Windowsの場合は、`Ctrl`キーを押しながらクリックします。`Ctrl`キーを押さずに選択してしまった場合は、[デフォルト]ボタンを押してデフォルト設定に戻すことができます。)

   – [表示オプション]ダイアログ ボックスを閉じる場合は、[閉じる]ボタンをクリックします。ただし、[表示オプション]ダイアログ ボックスのようなモードレス ダイアログは、開いたままにしておいても構いません。ツールを継続して利用できるように、モードレス ダイアログでは終了を強制されません。以前に新しいシステムを作成する際に使用した[新規追加]ダイアログ ボックス（54ページの図23）のようなモーダル ダイアログとはこの点で異なります。

4. オーガナイザの表示を確認してください。灰色になっているシステム ダイアグラム アイコンはダイアグラムが変更されたにも関わらず、まだ保存されていないことを表します。また、ダイアグラム名、すなわちDemon Gameが太字で表示されていますが、これは現在エディタでダイアグラムが開かれていることを示します。さらに、アイコンの右側に表示されている[未接続]は表記規則であり、このダイアグラムに現在関連付けられたファイルがない（保存ファイルがない）ことを示します。

   [未接続] と表示されているダイアグラム アイコンが依然として2つあります。これらは、システム ダイアグラムの編集の際に追加した、ブロック ダイアグラムへの参照です。

図34: 変更された未接続のダイアグラム（Demon Game）

5. SDLエディタに戻り、SDLエディタの[ファイル]メニューで[保存]を選択し、SDLダイアグラムを保存します。
   – オーガナイザからSDLエディタを起動するには、システム ダイアグラムのアイコンをダブルクリックします。この方法によって、簡単にSDLエディタ ウィンドウを表示できます。

6. ファイルの選択ダイアログが表示されます。このダイアログは、ファイルを開いたり、保存したりする際に表示される共通ダイアログです。ダイアログのタイトルには操作の種類が表示されます。この場合は[保存]と表示されます。

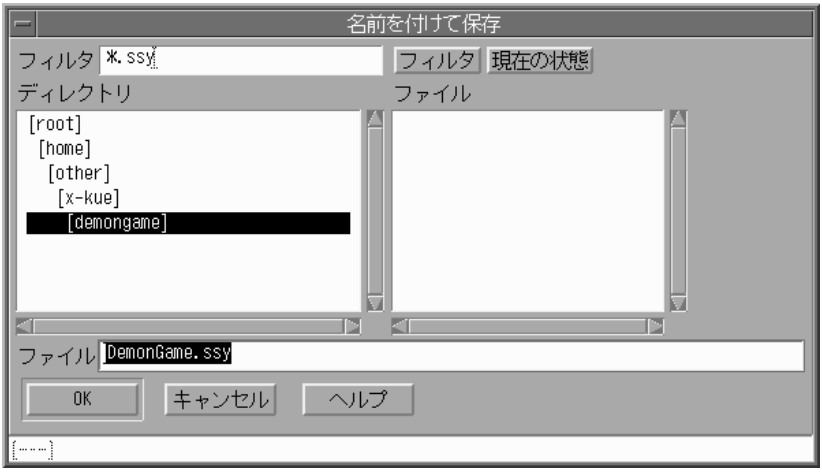


図35: ファイルの選択ダイアログ ボックス（UNIX）

UNIXのファイルの選択ダイアログ ボックスには、以下のような機能があります。

– [フィルタ]フィールドに表示されている`*.ssy`は、SDLシステム ダイアグラムを保存するファイルのデフォルトの拡張子です。他のファイルを表示するには、[フィルタ]フィールドの内容を変更し、[フィルタ]ボタンをクリックします（ただし、今回はこの操作をしないでください）。
– ダイアログの右側には、[フィルタ]フィールドに対応するファイルの一覧が表示されます。まだダイアグラムを保存していないので、現在は何も表示されていないはずです。
– 左側のファイルの一覧には、ファイル システムのルート ノードから現在のディレクトリまでのディレクトリ構造が表示されます。このリストをダブルクリックしてディレクトリ構造内を移動することができます（ただし、今回はこの操作をしないでください）。

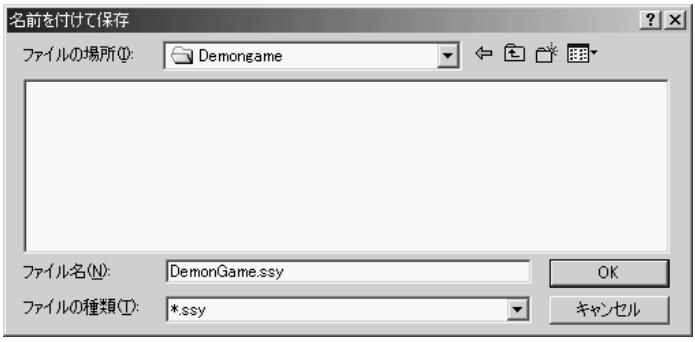


図36: ファイルの選択ダイアログ ボックス（Windows）

Windowsのファイルの選択ダイアログ ボックスには、以下のような機能があります。

- [ファイルの種類]フィールドに表示されている`*.ssy`は、SDLシステム ダイアグラムを保存するファイルのデフォルトの拡張子です。他のファイルを表示するには、[ファイル名]フィールドの内容を変更し、[OK]ボタンをクリックします（ただし、今回はこの操作をしないでください）。
- リスト ボックスには[ファイルの種類]フィールドに対応するファイルの一覧が表示されます。まだダイアグラムを保存していないので、現在は何も表示されていないはずです。
- [ファイルの場所]リスト ボックスには、ファイル システムのルート ノードから現在のディレクトリまでのディレクトリ構造が表示されます。このリスト ボックスをクリックしてディレクトリ構造内を移動することができます（ただし、今回はこの操作をしないでください）。

7. SDLエディタによって、`DemonGame.ssy`というファイル名が自動的に入力されるため、このファイル名でダイアグラムを保存できます。ファイル名を変更しても構いませんが、チュートリアルではこのファイル名での保存を前提として説明を進めます。

   ファイル名を確定するために[OK]ボタンをクリックします。これでダイアグラムの保存が完了しました。SDLエディタ ウィンドウのタイトルを参照すれば、ダイアグラムがファイルに保存されたことを確認できます。

図37: SDLエディタ ウィンドウのタイトル

この情報はオーガナイザ構造においても利用可能です。また、オーガナイザのダイアグラム構造の表示も、[未接続]からDemonGame.ssyに変わります。

図38: ダイアグラムを保存した後のダイアグラム構造

ダイアグラム構造の保存

ここまでの学習でシステム ダイアグラムを保存しましたが、今後の作業で使用するオーガナイザ ビュー オプションとダイアグラム構造も保存します。オーガナイザ ウィンドウのタイトルには、末尾に「*」（アスタリスク）が付いています。このアスタリスクは、オーガナイザ ビューか構造情報が変更されたため、保存する必要があることを示しています。

図39: オーガナイザ ウィンドウのタイトル

システム ファイル

オーガナイザは、多くのオプションと共にそのビューをシステム ファイル[1] と呼ばれる専用のファイルに保存します。システム ファイルは、SDL構造の一貫性を維持するためのもので、SDL構造に定義されたダイアグラムへ即時にアクセスすることを可能にします。

システム ファイルは、オーガナイザ ビューの一番上にある専用のアイコンで表示されます。このアイコンは長方形の中に「SDT」と表示されたものです。システム ファイルが保存されていない場合も、オーガナイザによってシステム ファイル用のファイル名が割り当てられます。

1. システム ファイルはSDL構造に関するあらゆる情報が記録されます。その内容は必ずしも、SDL**システム**に関わるものではありません。なお、システム ファイルという用語は一般用語です。

以下の手順でシステム ファイルを保存します。

1. オーガナイザの[ファイル]メニューから[保存]コマンドを選択します。オーガナイザに、[保存]ダイアログ ボックスが表示されます。

図40: オーガナイザの[保存]ダイアログ ボックス

2. ダイアログ ボックスには、保存するファイル名`demongame.sdt`が自動的に入力されます（システム ファイルはデフォルトで、`.sdt`の拡張子が割り当てられます）。[保存]ボタンをクリックして保存するファイル名を了解します。

システム ファイルが生成されると、ダイアグラム構造とオーガナイザ オプションが保存され今後の作業で利用できるようになります。システム設計を再開する際には、オーガナイザの[ファイル]メニューから[開く]コマンドを使って、保存したシステム ファイルを開きます。

保存に関する補足事項

ここまでの学習で、個々のダイアグラムとシステム ファイルの保存方法を学びましたが、1つのコマンドでこれらのすべてを保存する簡単な方法もあります。1つのコマンドですべての要素をファイルに保存するには、以下の2種類の方法があります。

- オーガナイザの[保存]ダイアログ ボックスにある[すべて保存]ボタンをクリックします（図40参照）。

- オーガナイザのツール バーにある[保存]クイック ボタンをクリックします。このボタンを使用すると、特にユーザーへ通知する事項がない場合、ダイアログ ボックスが表示されないまま、ダイアグラム構造を含むすべてのダイアグラムが保存されます（SDLエディタの[保存]クイック ボタンは、現在操作しているダイアグラムのみを保存するときに使用します）。

システム ダイアグラムの印刷

学習内容

- 学習内容 SDLダイアグラムの印刷
- プリント オプションの調整
- 印刷出力の拡大と縮小

印刷方法

最初のSDLダイアグラムの作成が完了しました。ここで、後続の演習に進む前にダイアグラムを印刷すると便利です。UNIXの場合は、コンピュータ環境内にPostScriptプリンタがあることが前提となります。PostScriptプリンタがない場合は、この演習を飛ばしても構いません。

以下の方法でダイアグラムを印刷します。

1. SDLエディタ ウィンドウを起動します。
2. [ファイル]メニューから[印刷]を選択します。[印刷]ダイアログ ボックスが表示されます。

– または、[印刷]クイック ボタンをクリックします。

図41: [印刷] ダイアログ ボックス

3. ユーザー環境が正しく設定されていれば、適切なオプションが設定されたダイアログが表示されます。正しい設定内容が表示されない場合は、少なくとも以下の項目を確認し、必要に応じて変更します。

   – [用紙]オプション メニュー

   – [出力先]-[形式]。WindowsにおいてPostScriptプリンタを利用していない場合は、[MSW印刷]を選択してください。

   – [実行]コマンド（ユーザー環境設定で設定したパラメータ。*PrinterCommand*の値が入っています）

4. 設定が正しい様であれば、[印刷]ボタンをクリックして印刷を始めます。UNIXの場合は、SDTによってPostScriptファイルが`/tmp`ディレクトリに生成され、指定した[実行]コマンドに送られます。Windowsの場合は、印刷用に指定した[実行]コマンドによってプリント ファイルが生成されます。また、[MSW印刷]フォーマットを選択した場合は組み込みのドライバによってプリントファイルが生成されます。

   ファイルがデフォルトプリンタの待ち行列へ送られると、たいていの場合はすぐにプリンタが印刷を始めます。印刷されたサイズが適当でない場合は、以下の手順によって印刷の縮尺を変更します。

   – [設定]ボタンをクリックします。[印刷設定]ダイアログ ボックスが表示されます。

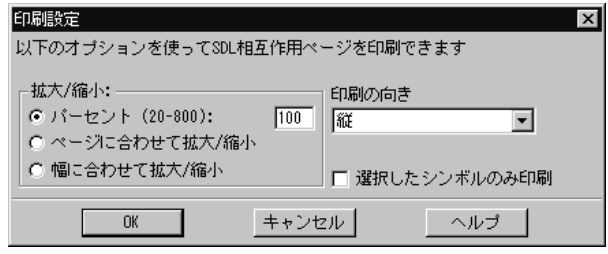


図42: [印刷設定]ダイアログ ボックス

   – 倍率の値を調節して、[OK]をクリックします。

   – [印刷]をクリックして、再度印刷します。

システム ダイアグラムのチェック

学習内容

- アナライザの起動
- アナライズ オプションの設定
- オーガナイザ ログ ウィンドウの使用
- 構文エラーの検索と訂正

アナライザの起動

残りのダイアグラムを作成する前に、これまでに生成したシステム ダイアグラムの構文をチェックします。構文をチェックするには、オーガナイザで *アナライザ* ツールを使用します。アナライザ ツールはバックエンド ツールの一種であり、オーガナイザに統合されています。

1. オーガナイザに表示されているダイアグラム構造で、SDLシステム ダイアグラム アイコンを選択し、オーガナイザの[生成]メニューから[分析]を選択します。

2. ダイアグラムに何らかの変更を加えている場合は、オーガナイザから変更したダイアグラムを保存するように促します（[保存]ダイアログ ボックスが表示されます。71ページの図40参照）。その場合、[すべて保存]をクリックしてすべてを保存します。

3. [保存]ダイアログ ボックスを閉じると、[分析]ダイアログ ボックスが開きます。

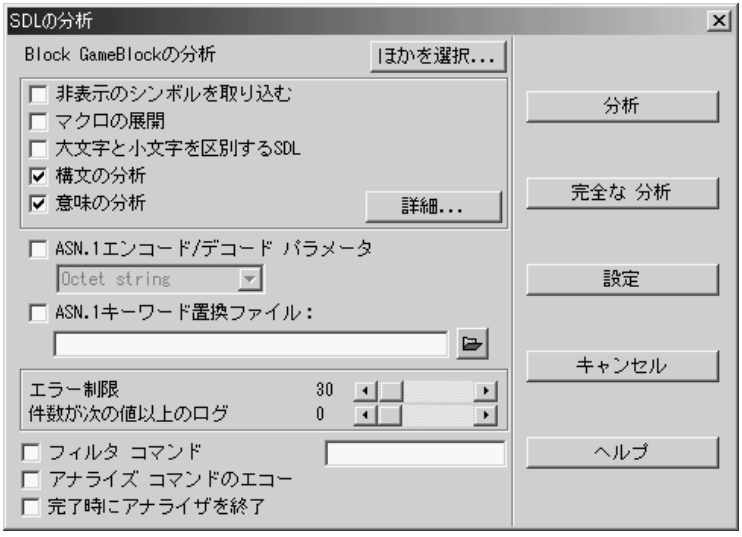


図43: アナライザのオプション設定用ダイアログ ボックス

4. 図43にしたがってオプションを設定します。設定内容をまとめると以下のようになります。

   – [マクロの展開]をオフ
   – [構文の分析]をオン
   – [意味の分析]をオフ
   – 必要に応じて、[エラー制限]スライドバーを適当な値に変更します。このパラメータによって、アナライザが何個のエラーや警告を検出したら分析を中止するかを設定できます。

5. [分析]ボタンをクリックします。アナライザが、オプション ダイアログに設定されたオプションにしたがって動作を開始します。処理が終了すると、オーガナイザのステータス バーに、「分析が完了しました」と表示されます。また、これに続いて追加の情報が表示されることもあります。

6. オーガナイザには重要情報を表示するテキスト表示用ウィンドウがあります。警告やエラー レベルの情報が発生すると、オーガナイザ ログ ウィンドウが自動的に表示されます。分析が完全に終了してもウィンドウが自動的に表示されない場合は、手動でウィンドウを開いてください。オーガナイザ ログ ウィンドウを表示するには、オーガナイザの[ツール]メニューから[オーガナイザ ログ]を選択するか、クイック ボタンをクリックします。

分析エラーの検出

アナライザによって検出されたエラーは、その他の重要情報と共にオーガナイザ ログに記録されます。オーガナイザ ログの文末を確認すると、以下のような記述があるはずです。

```
-------------------------------------------------

警告の数 : <diagram dependent>
+ Analysis completed
```

（文末を表示させるためにオーガナイザ ログ ウィンドウを下にスクロールしなければならない場合があります。）

「警告の数」あるいは「エラーの数」の右に表示される数値は、ダイアグラムで検出された構文のエラーと警告の数を表します（エラーや警告が検出されなかった場合は、この行は表示されません）。

1. 練習のために、ここでは、故意に構文エラーのあるダイアグラムを作成します。たとえば、チャネルC1の信号リストからカンマを削除してみます。この際、信号名の間はスペースで区切ります。

   – この構文エラーはSDLエディタ内で検出され赤い下線が表示されますが、アナライザによってどのようにエラーが通知されるかを確認するのが目的であるため、そのままにしておきます。

2. すべてを保存して、アナライザを再度実行します。

分析エラーの訂正

オーガナイザ ログには以下のようなメッセージが表示されます。

```
#SDTREF(SDL,/opt/home/tmi/demongame/DemonGame.ssy(1)
,131(25,50),3,8)（UNIX の場合）
#SDTREF(SDL,C:¥Telelogic¥SDL_TTCN_Suite6.2J¥work¥dem
ongame¥DemonGame.ssy(1),131(25,50),3,8)（Windows の場合）
ERROR 312 Syntax error in rule SIGNALLIST, symbol
Name found but one of the following expected:
,   ;  comment
Result Endgame;
       ?
```

エラーメッセージの解釈方法

ここで、エラー メッセージの内容について説明します。

- 最初の部分（#SDTREF...）はエラー箇所への参照であり、ソース ダイアグラム、ファイル名、ページ、シンボル、行番号、および行内での位置を表わ

しています。アナライザによって生成されるすべての参照は、この書式の全部あるいはその一部で表現されます。内容は、アナライザがエラーの発生元をどこまで詳細に追跡するかによって異なり、上記の例ほど厳密に表示されない場合もあります。

- 2番目の部分（`ERROR 312...`）には、エラー番号と説明文が記述されます。上記の例は、カンマ、セミコロン、あるいはコメントの記述が必要であることを示しています。
- 「?」シンボルとともに表示されている最後の部分（`Result Endgame`）は、エラーが起こった部分をさらに明確に表示します。上記の例は、Resultと Endgame信号の間にカンマを挿入すべきであることを示しています。

以下の方法によって、エラーが検出されたダイアグラムとシンボルを表示させることができます。

1. マウスをドラッグして、エラー メッセージを含んでいるテキストの部分を選択します。

図44: エラー メッセージの選択

2. [ツール]メニューから[エラーの表示]を選択します。
   - または、[エラーの表示]クイック ボタンをクリックします。
3. SDLエディタ内で、エラーが発生したシンボルが選択されます。参照の記述内容が詳細なほど、選択される部分も詳細になります。では、エラーを訂正してください（カンマを挿入します）。
4. 次のエラーを訂正するには、[エラーの表示]ボタンを再度クリックします。
5. ダイアグラムを保存しアナライザがエラーを検出しなくなるまで分析を繰り返します。エラー メッセージの解読方法がわからない場合や、訂正方法がわからない場合は、前出のシステム ダイアグラムを確認してください（58ページの図27参照）。

   - オーガナイザ ログ ウィンドウの内容は、いつでも消去することができます。例えば、ログの内容を読みやすくするために、部分的に消去するこ

ともできます。[編集]メニューから[ログのクリア]コマンドを選ぶか、クイック ボタンを実行します。

– 同じオプションの設定で分析を再度実行する際は、オーガナイザの[分析]クイック ボタンを使用します。

これで、SDL Suiteを使った最初のSDLダイアグラムが完成しました。また、ダイアグラムの構文が、Z.100勧告に適合していることを確認できました。

新しいブロック ダイアグラムの作成

学習内容

- ブロック ダイアグラムの作成と記述
- 信号辞書の利用
- SDLエディタでの複数のダイアグラムの使用
- ページ内での新しいウィンドウの起動
- 複数のSDLエディタ ウィンドウの使用
- ブロック ダイアグラムの構文チェック

オーガナイザでのブロック ダイアグラムの作成

ここでは、オーガナイザでブロック ダイアグラムを作成します。

1. オーガナイザ ウィンドウで、GameBlockアイコンをダブルクリックします。システム ダイアグラムを作成した時と同じ手順で、[編集]ダイアログ（ボックス）を設定します（56ページの図25参照）。[エディタで表示]オプションがオンになっていることを確認して、[OK]ボタンをクリックします。

   [ページの追加]ダイアログ ボックスが表示されます。このダイアログ ボックスでは、プロセスやブロックの相互作用を示すページの形式を定義します。また、ページ名を指定することもできます。ページ番号は自動的に付加されるように指定できます（1, 2,... N）。デフォルトでは、この機能は有効に設定されています。

2. [プロセス相互作用ページ]ボタンをオンにして、[OK]をクリックします。

図45: [プロセス相互作用ページ]の設定

SDLエディタによって、新しく作成したブロック ダイアグラムの1ページ目のウィンドウが表示されます。ブロック ダイアグラムのエディタ ウィンドウは、システム ダイアグラムのエディタ ウィンドウと似ていますが、シンボル メニューの内容が異なります。

図46にGameBlockブロックの完成例を示します。GameBlock内には、2つのプロセス参照シンボルと、5つの信号ルート、3つの接続ポイント、そして、信号の宣言が書き込まれたテキスト シンボルがあります。

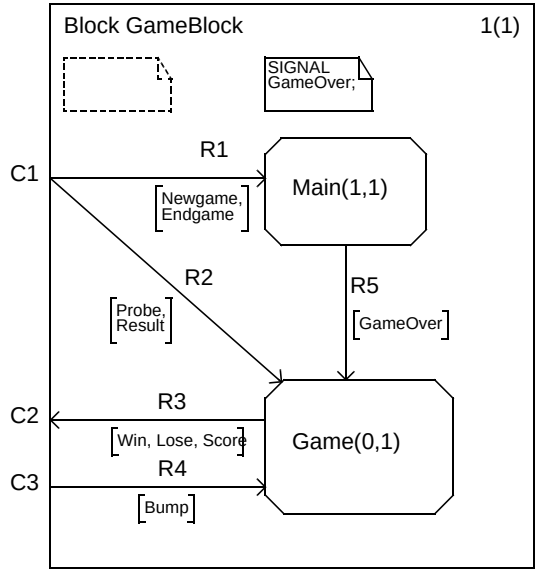


図46: GameBlockブロック

以後、図46にしたがってブロック ダイアグラムを記述します。ブロック ダイアグラムは、システム ダイアグラムと同じような方法で、シンボルやラインを追加できます。ダイアグラムを作成する前に、新しく導入する概念とその管理方法を、以下の3つの項で学習します。

プロセス名とインスタンスの数

プロセス参照シンボルを追加したら、プロセス参照シンボル名のすぐ後にテキストを追加してインスタンスの数を指定します。インスタンスの数は「( )」（かっこ）内にテキストで入力します。

信号ルート

信号ルートはチャネルと同じ方法で作成します。プロセス参照シンボルを選択すると2つの「ハンドル」が表示されます。

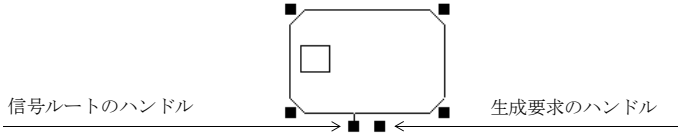


図47: プロセス参照シンボルの2つのハンドル

- 左のハンドルは、チャネルと同じように信号ルートを作成する際に使用します。
- 右のハンドルは、生成要求を作成する際に使用します。ただし、このチュートリアルでは使用しません。

接続ポイント

信号ルートをフレーム シンボルから引き出すか、フレーム シンボルへ向かって引く際には、信号名と信号リストを入力するだけでなく、図式上の*接続ポイント*を作成して、信号ルートと上位のシステム ダイアグラムを接続しなければなりません。つまり、信号ルートをチャネルと接続しなくてはなりません。信号ルートをフレームに向かって引くと、テキスト領域がフレーム シンボルの近くに現れます。このテキスト領域に、対応するチャネル名を入力します。図48を参照してください。

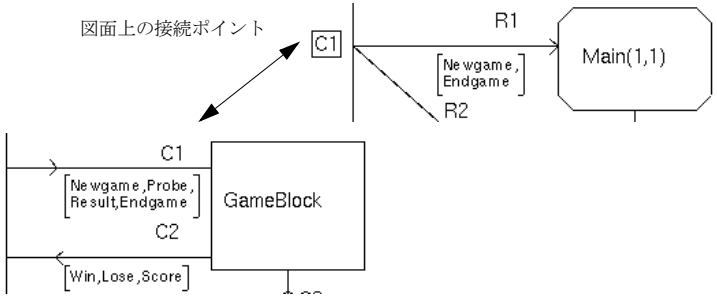


図48: 接続ポイント

この図では、ブロック ダイアグラム内の信号ルート(R1)と、これを参照するシステム ダイアグラム内のチャネル(C1)が接続されます。

接続ポイントは次の方法で編集します。

- 接続ポイントを選び、テキストの内容を入力します。

ブロック ダイアグラムの編集

以下の手順でダイアグラムを作成します。

1. まず、Main(1, 1)プロセス参照シンボルを追加します。
   - システム ダイアグラムを作成したときと同様に、入力したすべてのテキストには構文分析が実行され、エラーには、赤い下線が表示されます。赤い下線は、SDLの構文に適合したテキストを入力すればすぐに消えます。
   - テキストの編集を開始する前の時点では、テキスト カーソルは点滅していません。この状態で`Delete`キーを押すと、入力した文字ではなく、選択しているすべてのシンボルが削除されることに注意してください。
2. 外部環境からプロセスへ信号ルートを引きます（矢印の逆転には[リダイレクト]コマンドを使います）。信号ルート名としてR1を入力します。

信号辞書を使った各信号の定義

ここで、信号ルート（R1）で伝達される信号名を指定します。SDLエディタでは、信号辞書のサポートによってSDL構造ですでに定義した信号を参照できます。トップダウン方式で設計を進めているため、信号はすでにシステム ダイアグラムに定義されています。以下の手順によって、信号辞書のサポートを有効にします。

1. 信号リストのテキスト フィールドを選択します。
2. [ウィンドウ]メニューで[信号辞書]を選択します。[信号辞書]ウィンドウが表示されます。必要に応じて、ウィンドウの位置を変え、エディタ ウィンドウの信号リストが見えるようにします。

> メモ：
>
> 信号辞書の機能が正常に動作するためには、対象となるSDLダイアグラム内の記述が構文的に正しくなければなりません。SDLダイアグラム内に構文エラーがある場合は、前の演習に戻り、エラーを修正するためにアナライザを実行してください（76ページの「分析エラーの訂正」を参照）。

図49: [Signal Dictionary] ウィンドウ（UNIX）

左側に表示されているリストの画像は、ディスプレイの解像度によって異なった外観になる場合があります。

3. 信号辞書ウィンドウの左側の一覧を参照すると、最初のセクションは*Up*というセパレータで始まっていることを確認できます。このセクションは、ダイアグラム構造内の1つ上のレベルにある親ダイアグラムの信号を表示します（ここでの親ダイアグラムはDemonGameシステム ダイアグラムになります）。

図50: Up セパレータ（UNIX）

図51: Up セパレータ（Windows）

4. ゆえに、このセクションの最初の項目には、*System DemonGame*と表示されるはずです。

図52: DemonGame システムを表示した項目（UNIX）

図53: DemonGame システムを表示した項目（Windows）

5. 現在、システム ダイアグラムからブロック ダイアグラムへ向う信号ルートを作成しているため、親ダイアグラムからのチャネル、つまりチャネルの入力を意味するアイコンとラインを探します。

図54: 親ダイアグラムからカレントダイアグラムへのチャネルを表すアイコン（UNIX）

図55: 親ダイアグラムからカレントダイアグラムへのチャネルを表すアイコン（Windows）

– 条件に合うチャネルC1を見つけることができるはずです。他のチャネルは、ダイアグラムの内部で使用されているC3チャネルと、現在のダイアグラムに向かっていないC2チャネルです。

6. 各チャネルの下には、そのチャネルで伝達されるすべての信号が表示されています。[Newgame]という信号をクリックします。

図56: Newgame信号（UNIX）

図57: Newgame信号（Windows）

7. [編集]メニューで[挿入]を選択します。SDLエディタの信号リストが直ちに更新されます。

– または一覧に表示されている信号をダブルクリックします。

メモ：操作を元に戻す

誤って、信号リスト以外のテキスト フィールド（信号ルート名など）に信号を挿入してしまった場合は、信号辞書の[編集]メニューから[元に戻す]コマンドを選択して元に戻すことができます（SDLエディタのテキスト ウィンドウには[元に戻す]の機能がありません）。

8. エディタ ウィンドウで、カンマを挿入し、必要ならば改行します（必要に応じて信号リストを移動します）。
9. 信号辞書で[Endgame]信号をダブルクリックします。
10. 信号辞書によって接続ポイントC1のテキストが自動入力されます。ダイアグラムで接続ポイントテキスト フィールドを選択し、信号辞書に表示されているチャネルをダブルクリックします。

信号辞書を使った多信号の定義

1. SDLエディタ ウィンドウで、Game(0, 1)プロセス参照シンボルを追加します。
2. Gameからフレーム シンボルに向かって信号ルートR3を引きます。

   R3の信号リストを入力する際には、R1の時のように1つずつ信号を入力する必要はありません。R1とR2に分割されているC1チャネルとは異なり、C2チャネルは複数の信号ルートに分けられていないためです（印刷したダイアグラムの図か、58ページの図27を参照してください）。したがって、1回の操作ですべての信号を挿入できます。
3. 信号辞書でチャネルC2を選択します。右側の一覧が更新され、チャネルを経由するすべての信号が表示されます。[挿入]コマンドを選択するかチャネルC2をダブルクリックして、それらの信号を挿入します。

図58: [Signal Dictionary] ウィンドウでチャネルを選択することにより すべての信号を表示（UNIX）

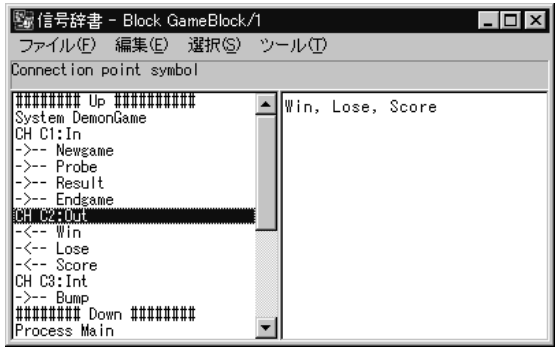


図59: [信号辞書] ウィンドウでチャネルを選択することにより すべての信号を表示（Windows）

信号辞書ユーティリティの操作方法は、ご理解いただけましたでしょうか。

ダイアグラムの完成

1. ダイアグラムの残りの信号ルートにも任意の方法で入力します。入力が終了したら、[信号辞書]ウィンドウを閉じます（[ファイル]メニューから[閉じる]を選択します）。

2. GameBlockブロックの編集を終了したら、すべてを保存します。オーガナイザの[保存]クイック ボタンをクリックして、[保存]ダイアログ ボックスに表示されているファイル名をそのまま使用してください。

複数のダイアグラムの使用

ここまでの演習で2つのSDLダイアグラムを作成しました。SDLエディタは、2つのダイアグラムを両方同時に開くことができます。ただし、ウィンドウ上に表示し編集できるのは、一方のダイアグラムだけです。

SDLエディタには、現在エディタで開いているダイアグラムとページの一覧を表示する、[ダイアグラム]というメニューがあります。

1. [ダイアグラム]メニューをクリックします。現在2つのダイアグラムが一覧表示されるはずです。

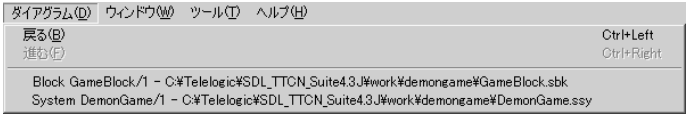
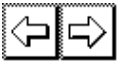


図60: [ダイアグラム] メニュー

メニューに表示されている項目は、現在SDLエディタで開いているダイアグラムとページに対応します。また、ダイアグラムを保存しているファイル名がダイアグラム名の右に表示されます。

では、DemonGameシステムのダイアグラムを表示させてみましょう。

2. 以下のメニュー項目を選択します。
[*System DemonGame/1 ..... DemonGame.ssy*[1]]
システム ダイアグラムがすぐに表示されます（ブロック ダイアグラムが非表示になります）。



– メニュー項目の[戻る]と[進む]を使ってもダイアグラムやページを移動することができます。SDLエディタでは、作成したページの検索が可能で

1. メニュー項目の実際の表示内容は、現在使用しているディレクトリ構造によって異なります。

あり、Web ブラウザでWebページを表示しているときのようにリスト内を前後に移動できます。

複数ウィンドウの使用

ここまでの演習では1ページを1つのウィンドウに表示して使用してきましたが、SDLエディタでは、同じダイアグラムに対して新しいウィンドウを開くことができます。すなわち、1つのページを、複数のウィンドウに表示することができます。この方法は、ウィンドウの*インスタンス化*とも呼びます。

以下の手順でページから新しいウィンドウを開きます。

1. 編集するページがSDLエディタに表示されていることを確認します。表示されていなければ、[ダイアグラム]メニューを使って表示させます。
2. [ウィンドウ]メニューから、[新しいウィンドウを開く]を選択します。現在のページを表示する新しいウィンドウがすぐに現れます（図61参照）。
3. ページの編集には、どのウィンドウを使用しても構いません。片方のウィンドウに加えられた修正は、もう一つのウィンドウにすぐに反映されます。シンボルの一つを移動すれば、これが確認できます。
4. このチュートリアルで使用するダイアグラムは小規模なので、1つのウィンドウで十分編集できます。[ウィンドウ]メニューから[ウィンドウを閉じる]を選択して、2つのウィンドウのいずれかを閉じます。

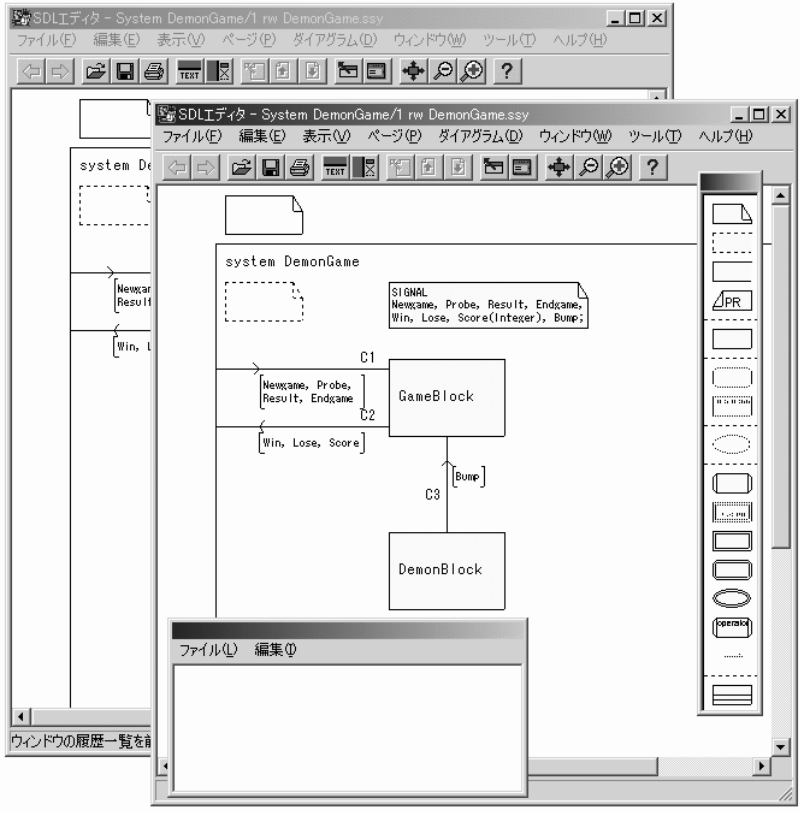


図61: 同ページを表示する2つのウィンドウ

オーガナイザの状況

すべての変更を保存します。現在のダイアグラム構成は、オーガナイザで以下のように表示されるはずです。

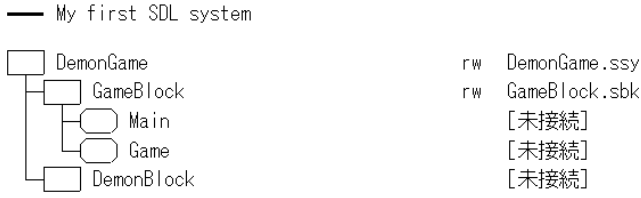


図62: 現在のオーガナイザの表示

ブロック ダイアグラムの構文チェック

アナライザを使用して、作成したブロック ダイアグラムの構文をチェックします。

GameBlockブロックのチェック方法を以下に示します。

1. アナライザの実行対象を指定するために、オーガナイザでGameBlockブロックのアイコンを選択します。
2. [分析]コマンドを選択し、ブロック ダイアグラムの分析を実行します。

図63: [アナライザ]への入力の指定

   – アナライズ ダイアログが表示され、システムのいずれの部分を分析するか指定できます。[Analyze Block GameBlock]が選択されていることを確認して、[分析]ボタンをクリックします。

3. システム ダイアグラムに対して以下の操作を行います。

   – オーガナイザ ログに構文エラーがないか確認します。
   – 検出されたエラーを訂正します。
   – 必要に応じて、操作を繰り返します（訂正方法を忘れた場合は、76ページの「分析エラーの訂正」を参照してください）。

コピーによるブロック ダイアグラムの作成

学習内容

- 既存ファイルのコピーによるダイアグラムの作成
- 作成したダイアグラムの新しいファイルへの保存

DemonBlockブロックの作成

以前の演習では、オーガナイザでGameBlockブロックのシンボルをダブルクリックして、ブロックを作成しました。また、ブロック ダイアグラムは、SDLエディタ内でダイアグラム参照シンボルをダブルクリックしても同じように作成できます。

以下の手順でDemonBlockブロックを作成します。

1. SDLエディタでDemonBlockブロック参照シンボルをダブルクリックします。ダイアログ ボックスが表示されます。

図64: DemonBlockブロックの作成時におけるダイアログ ボックスの表示

ここでは、既存ファイルをコピーしてダイアグラムを作成します。インストール ディレクトリには、数多くのSDLダイアグラムのサンプルがインストールされています。その中には、完成したDemon Gameのサンプルもあります。これらのダイアグラムは、デフォルトでインストール ディレクトリ内のサブディレクトリに保存されています。ディレクトリパスは、UNIXの場合

は `$telelogic46/sdt/examples/demongame`、Windowsの場合は `C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥sdt¥examples¥demongame`です。

次に、DemonBlockブロックのサンプル ファイルを見つけます。このファイルの名前は、`DemonBlock.sbk`です。

メモ：

上述のディレクトリの正確な場所がわからない場合は、システム管理者に確認してください。DemonGameのサンプルが存在するディレクトリを見つけられない場合でも、諦めないでください。最悪の場合は、これまでの演習で学習した方法を使って、SDLエディタを起動し、Newオプションで残りのダイアグラムをすべて手書きで作成することもできます。

2. [既存のファイルをコピー]ボタンがオンになっているか確認します。
3. 以下のいずれかの操作を行います。
   - ディレクトリ パスを含めたファイル名を入力します（上記パスを参照）。

     または、
   - テキスト フィールドの右にある[フォルダ]ボタンをクリックします。[Block DemonBlock用にコピーするファイルの選択]というタイトルのファイル選択ダイアログ ボックスが表示されます。
   - ダイアログ ボックスで、サンプル ダイアグラムが保存されているディレクトリが見つかるまで、ディレクトリ内を移動します（上述のディレクトリ パスを参照）。UNIXの場合は、左側の一覧内に表示されているディレクトリをダブルクリックして、右側のリスト ボックスに表示されるブロック ダイアグラム ファイルを確認します。Windowsの場合は、[ファイルの場所]ボックスでディレクトリを選択するか、リスト ボックス内のディレクトリをダブルクリックします。
   - 一覧から `DemonBlock.sbk`というファイルを選択して[OK]をクリックします。

メモ：別のディスクにあるファイルにアクセスする

Windowsの場合は、UNCと呼ばれるパスでネットワークへアクセスします。UNCパスでファイル名を記述する際は、「¥¥ディスク名¥ディレクトリ名」の構文で入力します。

UNIXの場合、ファイル選択ダイアログから[root]ディレクトリへ移動できるかどうかは、コンピュータ システムやネットワーク ファイル システムの設定によります。名前の先頭に「/」（スラッシュ）を入力します。そして、現在アクセスしているディスクではなく、別のディスクに保存されているファイルにアクセスするために*OK*をクリックします（ただし、まず始めに[root]ディレクトリをダブルクリックしてみてください）。

4. [OK]ボタンをクリックして[編集]ダイアログを閉じます。SDLエディタのウィンドウにダイアグラムが表示されます。ダイアグラムを印刷した際のイメージは、図65のようになります。

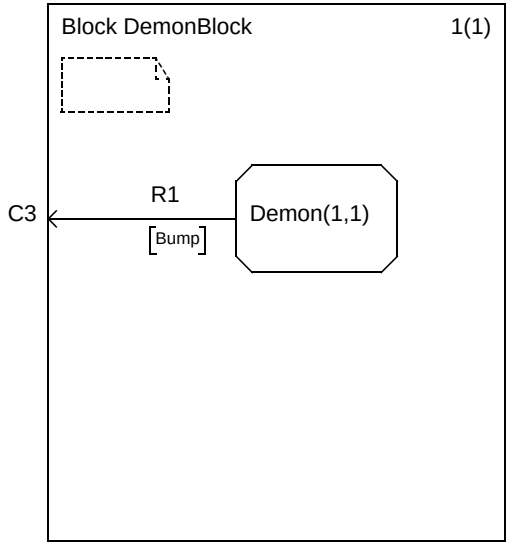


図65: DemonBlockブロック

5. では、SDLエディタで[保存] クイック ボタンをクリックして、ダイアグラムを保存します。DemonBlock.sbkというファイル名がすでに設定された、ファイル選択ダイアログ ボックスが表示されます。

6. [OK]ボタンをクリックします。現在オーガナイザに表示されているダイアグラム構造は以下のようになるはずです。

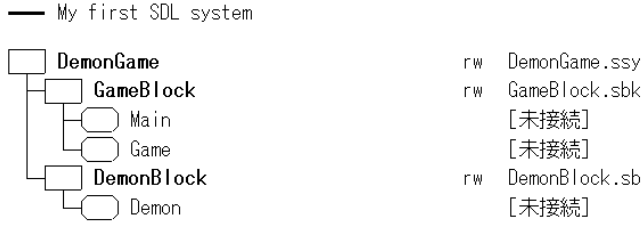


図66: 現在のオーガナイザの表示

既存ダイアグラムをコピーする方法によって、チュートリアルの残りのダイアグラムを作成することができます。ただし、SDLエディタのさまざまな機能に慣れるために、新規にダイアグラムを作成することをお薦めします。次の項で、説明するプロセス ダイアグラムの作成方法は、ブロック ダイアグラムの作成方法と多少異なります。

プロセス ダイアグラムの作成

SDLシステムの基本的な構造を作成できましたので、次に、構造に機能を実装します。すなわち、システムの振る舞いを記述するプロセス フローチャートを作成します。

これまでの演習で、オーガナイザとSDLエディタの双方から新しいダイアグラムを作成する方法を学習しましたので、ここでは、これらのツールの詳細な使い方を説明しません。オーガナイザやSDLエディタでアイコンをダブルクリックして作成するか、オーガナイザの[編集]コマンドを使用して作成するかなどは、任意に判断してください。

次の演習では、プロセス ダイアグラムを記述する際の、SDLエディタの使い方を学びます。まず、図67に示すDemonプロセスの記述から始めます。

Demonプロセスの編集

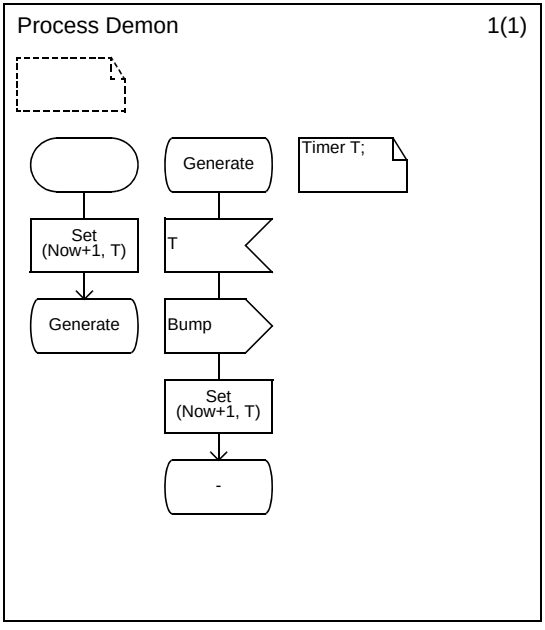


図67: Demonプロセス

ここでは、SDLエディタを使用したプロセス ダイアグラムの作成方法を説明します。

学習内容

- ダブルクリックによるシンボルの追加
- クリップボード機能の使用
- フローチャートへのシンボルの挿入
- 文法ヘルプの呼び出し

ダイアグラムの生成

1. Demonダイアグラムを編集します。ページを追加する際は、ページ形式に[グラフページ]を指定してください。

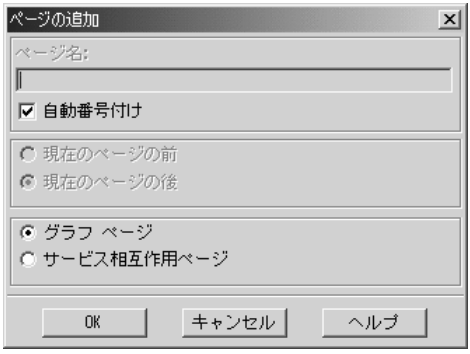


図68: ページの形式を[グラフページ]に指定

2. SDLエディタによって新しいダイアグラムが表示されると、シンボル メニューがブロック ダイアグラムと異なっていることがわかります。このメニューには、フロー ダイアグラムで利用できる状態シンボルや入力シンボルなどが表示されています。

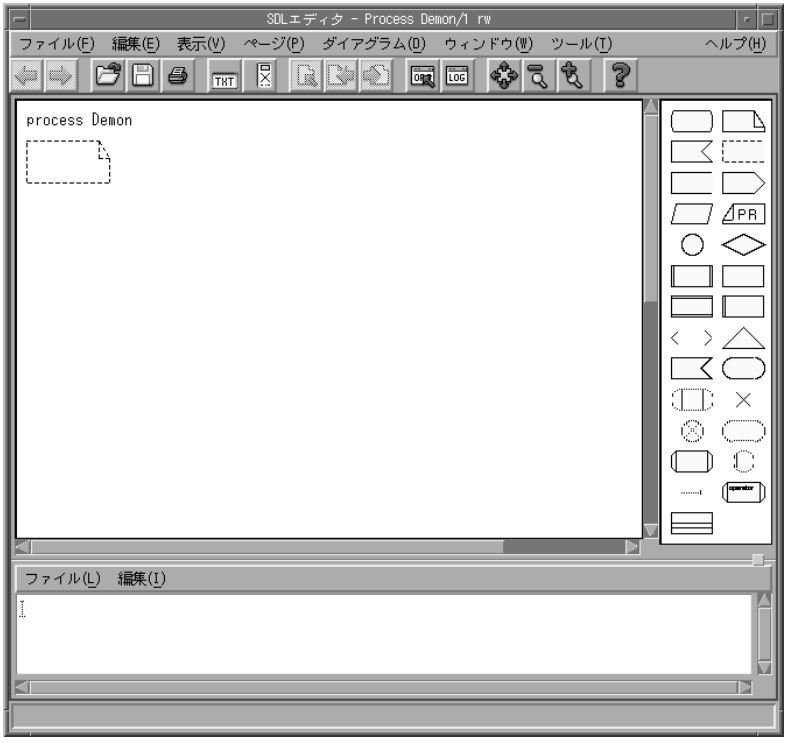


図69: フロー ダイアグラム用SDLエディタ ウィンドウ（UNIX）

このダイアグラムは、2つのフローで構成されています（96ページの図67参照）。シンボルをフローに追加すると、SDLエディタが、シンボルとフロー ラインを自動的に接続します。

シンボルを挿入するたびに、そのシンボルを選択してテキストを入力することができます。また、すべてのシンボルを挿入してから、テキストを入力することもできます。さらに、それらの手順を組み合わせて入力することもできます。

文法ヘルプを使った左側のフローの作成

以下の手順で左側のフローを作成します。

1. シンボル メニューで開始シンボルを選び、ドローイング エリアの適当な位置へ配置します。

   – シンボル メニューでシンボルをポイントまたは選択すると、SDLエディタ ウィンドウの最下部にあるステータス バーにシンボルのタイプが表示されます。どのシンボルを選択すべきかわからない場合は、ステータス バーを参照してください。

2. シンボル メニューでタスク シンボルをダブルクリックします。何も記述されていないタスク シンボルが開始シンボルの下につながります。

   タイマー設定用のSetステートメントが記述されたタスク シンボルを作成する際、*Set* ステートメントの構文に関する知識がなくても演習を進めることができるように、この演習は配慮されています。

   SDLエディタには、*文法ヘルプ* ([文法ヘルプ]) *ウィンドウ*というステートメント入力補助機能があり、正しいSDLの式を入力するために役立ちます。Setステートメントを正しく記述するために、文法ヘルプウィンドウを使用します。

3. [ウィンドウ]メニューから[文法ヘルプ]を選択します。SDL エディタから、*文法ヘルプウィンドウ*が表示されます。

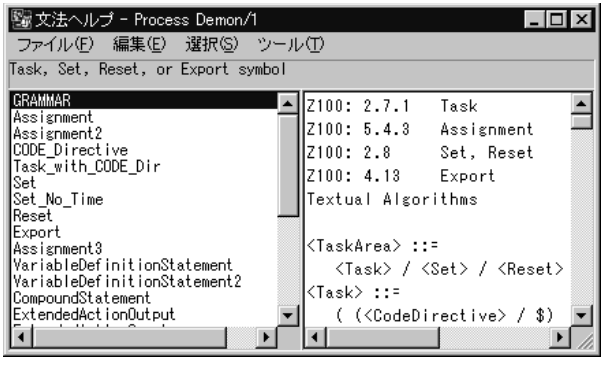


図70: [文法ヘルプ] ウィンドウ

   – 文法ヘルプウィンドウの左側のリスト ボックスには、様々な使用パターンが表示されます。各使用パターンは、それらの名前によって識別できま

す。左側のリスト ボックスの一番上には、現在GRAMMARが選択されています。

– 右側のリストの先頭には、Z.100勧告内の定義へのリファレンスが表示されます。

– Z.100勧告へのリファレンスの下には、シンボルに追加することができるテキスト表現の一般式（SDL/PR）が表示されます（この一般式は、当然、記述する仕様に合わせて、実際の値に変更しなければなりません）。

4. ここで使用するパターンはタイマーの設定であるため、左のリストから*Set*と表示された項目を見つけます。*Set*をクリックします。

– 右側のリスト ボックスに、Setに対応する文法「SET(Now+Expr, TimerName)」が表示されます。

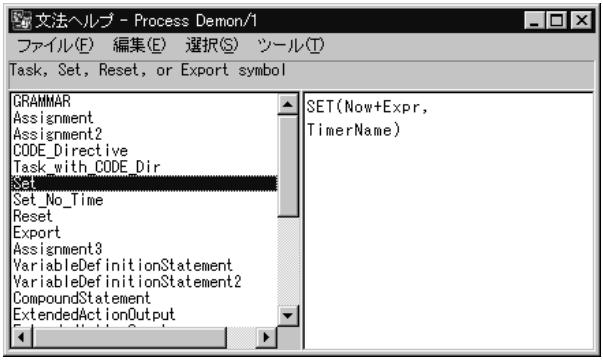


図71: タイマー設定に対応する文法

5. 文法ヘルプウィンドウの[編集]メニューから[挿入]コマンドを選び、Setに対応する文法「SET(Now+Expr, TimerName)」を、タスク シンボルに入力します。

– または文法ヘルプウィンドウの一覧で[設定]をダブルクリックします。

6. 一般名の*Expr*と*TimerName*を、実際の値に変更します（*1*と*T*に変更）。

– この変更はSDLエディタのテキスト ウィンドウで行います。例えば、変更したい箇所にマウスのポインタを移動して、新しいテキストを入力します。

図72: テキスト ウィンドウでのテキストの編集（UNIX）

ここでは文法ヘルプの基本的な使用方法を学びました。

7. 状態シンボルをダブルクリックして、「`Generate`」と入力します。これで、左側のフローは完成です。

右側のフローの作成

以下の手順で、右側のフローを作成します。

1. 新たに追加した状態シンボルをクリップボードにコピーします。クリップボードへのコピーには、[編集]メニューの[コピー]を選択するか、マウスの右ボタンでクリックして[コピー]を選択します。
2. 状態シンボルを貼り付けます。[貼り付け]をクリックしたら、新しいシンボルの位置を指定します。適当な位置までマウスを移動し、左のマウスボタンをクリックして貼り付けます。
3. シンボルメニューでダブルクリックして入力シンボルを追加します。入力シンボルに、テキスト「`T`」を記入します。
4. ダブルクリックして出力シンボルを追加し、出力シンボルに、テキスト「`Bump`」を入力します。
5. 左側のフローより、「SET (Now+1, T)」タスク シンボルをコピーします。ただし、すぐには貼り付けません。
6. Bump出力シンボルを選択します。マウスの右ボタンをクリックして、[挿入貼り付け]を選択します。この操作によって、出力シンボルとタスク シンボルが接続されます。
7. 状態シンボルをダブルクリックして、右側のフローに接続します。状態名称として、ハイフン（-）を入力します。この操作で、右側のフローは完成です。

8. 最後に、テキスト シンボルを追加して、タイマー Tの宣言を入力します。
9. 必要に応じて、フレーム シンボルのサイズ変更します。
10. 「Demon.spr」というファイル名で、ダイアグラムを保存します。

これで、Demonプロセスの編集は完了です。

Gameプロセスの編集

ここまでに学習した方法で、Gameプロセス ダイアグラム作成します。この演習では、いくつかの新しい編集機能を学習します。

学習内容

- 並列のフロー チャートの編集
- シンボル間の接続

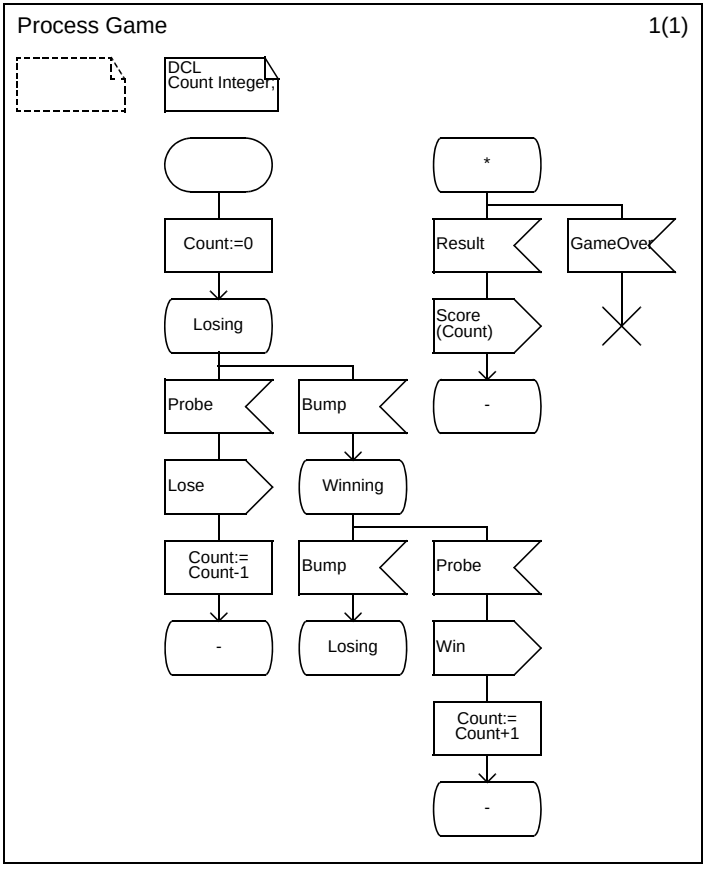


図73: Game プロセス

図73のプロセス ダイアグラムの作成方法を以下に示します。

スタート遷移の編集

1. 開始シンボルを配置します。次に、タスク シンボルとLosing状態シンボルを追加します。

2. 次に、2つの入力シンボルを並列に並べて追加します。この操作を行うには、まず状態シンボルを選択します。そして、`Shift`キーを押しながら、2つの入力シンボルを追加する位置でダブルクリックします（この間`Shift`キーは押したままです）。

3. Shiftキーを放し、左側の入力シンボルを選択します。
4. 入力シンボル名として「Probe」を入力し、これに続く左側のフローを完成させます。
5. 右側の入力シンボルを選択して、「Bump」と入力し、同様にこの下に続くフローを完成させます。この時、Winning状態シンボルから、Probe入力シンボルへ枝別れするフローは以下に示すやり方で作成してください。
6. 左側のフローでProbe入力シンボルを選択します。このシンボルから、フローの末尾までを選択するために、[編集]メニューの[最後の選択]を使用します。
7. 選択した部分をコピーして、貼り付けます。選択したフロー（シンボルの集合として表示されます）を適切な位置（図81参照）に移動し、クリックして貼り付けます。余白が不足したときや、他のシンボルと重なってしまったときなど、貼り付けに失敗すると警告音が鳴ります。この場合は、再度同じ操作を行ってください。
8. 入力シンボルのテキストをLoseから「Win」へ変更します。
9. タスク シンボルのテキストを「Count:=Count+1」に変更します。
10. Winning状態シンボルと、Probe入力シンボルを接続します。状態シンボルを選ぶと、ハンドルが表示されます。

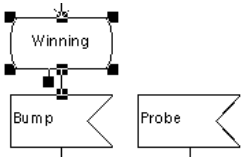


図74: 選択された状態シンボルとそのハンドル

マウスをクリックしたままハンドルをドラッグし、Probe入力シンボルをポイントしたところでマウスを放します。シンボル間にラインが引かれます。

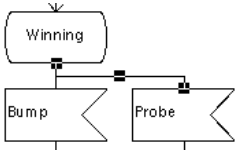


図75: 2つのフローの接続

11. 残りの部分も記述し、ダイアグラムを保存して終了します。

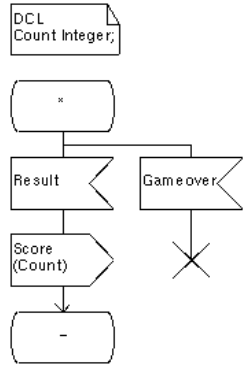


図76: 編集すべき残りの部分

Mainプロセスの編集

最後にMainプロセスのダイアグラムを生成および編集してください。この作業が面倒であれば、この演習は省略してもかまいません。その場合は、収めれらているサンプル ファイルからコピーしてダイアグラムを作成します（92ページの「コピーによるブロック ダイアグラムの作成」に、この操作方法が記述されています）。図77に、作成するダイアグラムの図を示しておきます。

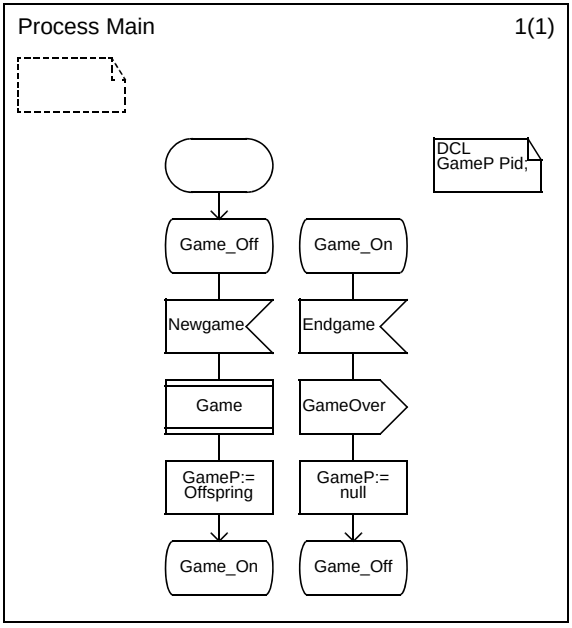


図77: Mainプロセス

オーガナイザについての補足

ダイアグラムが完成したら、すべての変更内容を*保存*します。現在 オーガナイザ ウィンドウに表示されているダイアグラム構造は以下のようになります。

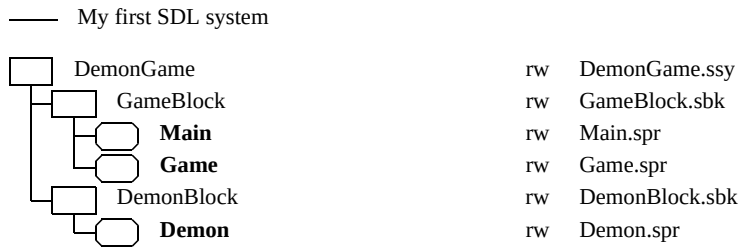


図78: 完成したダイアグラム構造

このチュートリアルで学習した機能は、使用可能な機能の一部分にしかすぎません。たとえば、異なる表示オプションを使用して、オーガナイザの表示方法をカスタマイズすることもできます。

学習内容

- 垂直ツリーの使用
- ダイアグラム構造の展開と折りたたみ
- オーガナイザ構造内のダイアグラムの並べ替え
- ディレクトリとページの表示
- システム全体の印刷

ツリー構造

1. [表示オプション]ダイアログ ボックスを起動します（66ページの図33参照）。[垂直ツリー]ボタンをクリックし、[適用]をクリックします。オーガナイザ ウィンドウの画面表示モードが変更されます。

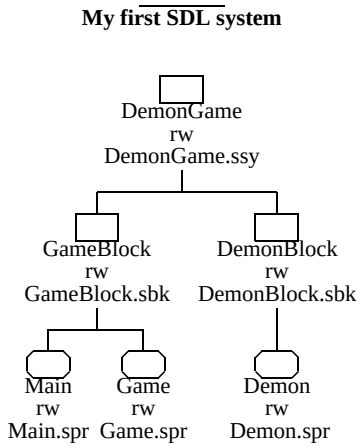


図79: 垂直ツリー構造

2. 再度[インデント リスト]モードに戻して[適用]をクリックします。

展開と折りたたみ

[表示]メニューの[展開]と[折りたたみ]を選択して、ダイアグラム構造の一部を展開したり、折りたたんだりできます。[展開]は折りたたまれている状態のみで

使用可能です（小さな三角形が表示された状態です）。また、[折りたたみ]はサブストラクチャが存在するノードが展開表示されている場合にのみ使用可能です。

1. GameBlockブロックを*折りたたみ*ます。GameBlockブロックのサブツリーはアイコンの下にある小さな三角形のノードによって表示されます。

図80: 折りたたまれたノード

2. 再度サブツリーを展開します（[サブストラクチャの展開]を使用します）。

ダイアグラムの並べ替え

ダイアグラムの並べ替え

オーガナイザに表示されている、シンボルの順序を並べ替えることが可能です。矢印キー（↑、↓、←、→）、または[上に移動]/[下に移動]クイック ボタンによって順序を並べ替えます。

DemonBlockブロックとGameBlockブロックの順序を入れ替えます。

1. GameBlockブロックを選択します。

2. [下に移動]クイック ボタンを一回クリックするか、`Shift`キーを押しながら↓矢印キーを押すと、以下のような表示結果になります。

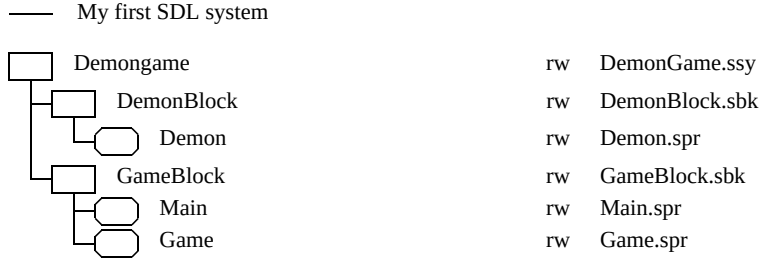


図81: 並べ替えられたGameBlockとDemonBlock

3. 最初のダイアグラム構造へ戻します。

ダイアグラム ページ

1. [表示オプション]ダイアログ ボックスで、[ページ シンボル]をオンにします。
2. [適用]ボタンをクリックします。SDLダイアグラムの各ページが表示されます。

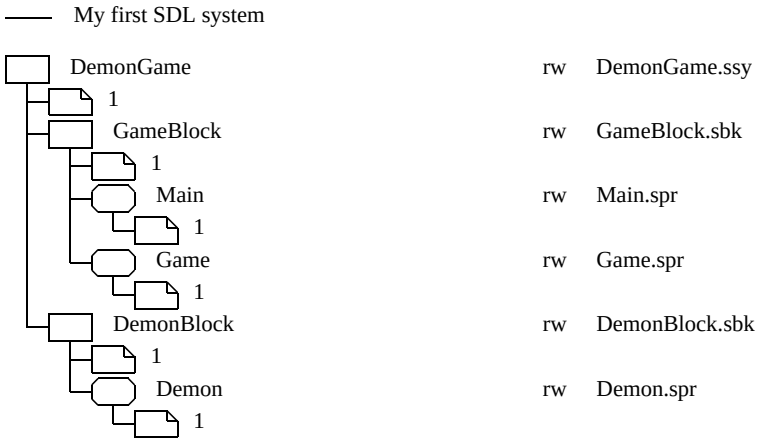


図82: ダイアグラム ページの表示

システムの印刷

オーガナイザでは、1つのコマンドで、システムに含まれる全てのダイアグラムを印刷することができます。また、目次も同時に印刷することができます。

1. すべてのダイアグラム シンボルの選択を解除するか、システム ダイアグラムを選択します。

2. [印刷]クイック ボタンをクリックします。[印刷]ダイアログ ボックスが表示されます。
3. [目次]トグル ボタンをオンにして、目次を含むすべてのSDLダイアグラムを印刷するように設定し、[印刷] ボタンをクリックします（図83 参照）。

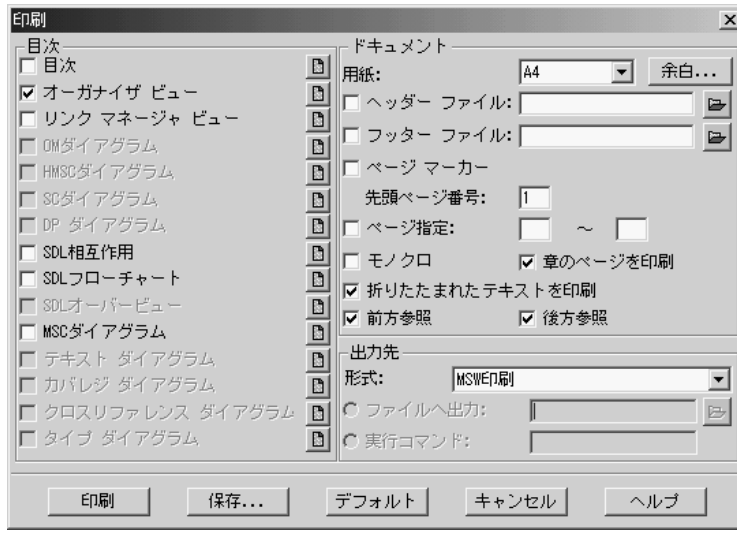


図83 : 目次（[目次]）の印刷を有効にする

SDL Suiteを使用した最初の、SDLシステムが完成し、印刷も完了しました。次に、完成したシステムに対してSDLの構文と意味のチェックを実行しましょう。

システムの分析

学習内容

- システム全体に対する構文分析と意味分析の実行
- 定義とクロス リファレンスを含んだファイルの生成

意味分析の実行

システムを分析する際は、意味的なチェックも実行します。以下の手順で意味分析を有効にします。

1. オーガナイザでシステム ダイアグラムのアイコンを選択します。
2. [生成]メニューで[分析]コマンドを選択します。
3. 下図のとおりにアナライザ オプションを設定します。

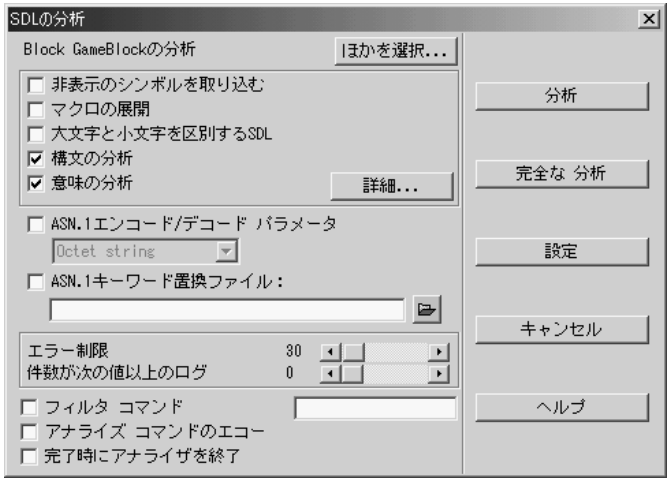


図84 : 意味分析の設定

– [クロスリファレンス ファイルの生成]ボタンをオンにして、アナライザがクロス リファレンスを記述したファイルを生成するように指定します。チュートリアルの今後の演習でこのファイルが必要になります。

– アナライザの意味分析には、他にもいくつかのオプションがありますます。それぞれのオプションは、トグル ボタンによって個々に設定できますが、このチュートリアルでは使用しません。

4. [分析]ボタンをクリックします。

5. オーガナイザのステータス バーに、「分析が完了しました」と表示されたら、アナライザによってエラーが検出されていないか*オーガナイザ ログ*で確認します。

6. 必要に応じてエラーを訂正し、同じ手順を繰り返します。SDLダイアグラム内のどこにエラーが発生しているかを見つける方法については、76ページの「分析エラーの検出」と76ページの「分析エラーの訂正」を参照してください。

7. システムが構文的および意味的に正しければ、オーガナイザログの末尾に以下のようなメッセージが出力されるはずです。

```
+ 分析が開始しました
SDL アナライザ
SDL の PR への変換を開始しました
PR への変換が完了しました
構文分析を開始しました
構文分析が完了しました
意味分析を開始しました
意味分析が完了しました
+ 分析が完了しました
```

すべてを保存してこの演習を終了します。ここで再度ダイアグラムを印刷することもできます（印刷方法については、110ページの「システムの印刷」を参照してください）。

メッセージ シーケンス チャートの管理

SDL Suite and TTCN Suiteでは、メッセージ シーケンス チャート（MSC）で知られている、ITU-T Z.120勧告もサポートされています。この演習を完全に理解するためには、基本的なMSCシンボルについての知識が必要です。

ここでは、MSCを利用できるいくつかの事例について説明します。

- 第一に、MSCはシステムの振る舞いに関する要求条件を記述するために使用できます。この場合、システムは外部からの信号（外部環境から送信されたSDL信号）を受信し、SDL信号で応答する「ブラック ボックス」として表示されます。
- MSCでは、SDLで設計する前の段階において、システムで発生する様々な状況をグラフィカルな形態で表示できるため、問題の把握に役立ちます。
- システムのシミュレーション結果をMSCとして生成・表示させることにより、システムの振る舞いを理解するために利用でき、また予測した振る舞いとの違いを確認できます。
- さらに、MSCは、*エクスプローラ*に入力することができます。エクスプローラによって、MSCで記述されたシーケンスを、指定した条件下で実際に発生させて確認することができます。

学習内容

- MSCをダイアグラム構造に追加
- SDLダイアグラムとMSCの接続
- MSCの生成
- MSCの編集

MSCをオーガナイザへ追加

オーガナイザを使用して、MSCを作成し、*Other Document*として管理することができます。この演習では、DemonGameシステムの振る舞いを記述するためにMSCを作成します。また、MSCをシステムのシミュレーションやバリデーションを実行するために使用します（この演習は後から実施します）。以下の手順でMSCを生成します。

以下の手順でMSCを生成します。

1. オーガナイザで[Other Documents]チャプタを選択します。

2. [編集] メニューから、[新規追加]を選択するか、[新規追加]クイック ボタンをクリックします。

3. [新規追加]ダイアログ ボックスが表示され、ダイアグラム名とタイプを指定するように促されます。

図85: 追加するダイアグラムの名前とタイプの指定

図85にしたがってダイアログ オプションを設定します。設定内容をまとめると以下のようになります。

– [新しいドキュメントのタイプ]を[MSC]に設定します。

– 名前を「`DemonGame`」に変更します。

– [エディタで表示]ボタンをオンにします。

4. [OK]をクリックします。MSCエディタが起動されますが、オーガナイザのウィンドウがMSCエディタ ウィンドウの後ろに隠れた場合は、そのウィンドウを表示させます。オーガナイザの*Other Document*チャプタに、新しく追加された*MSC*のアイコンが表示されます。ウィンドウの下側には図86のように表示されるはずです。

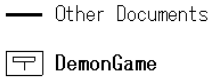


図86: MSCが追加されたオーガナイザ

オーガナイザに追加したMSCは、SDLシステム全体の振る舞いを表現するために使用するつもりですので、システム ダイアグラムに関連付けます。オーガナイザでは、この様な関係を表示することができます。

5. オーガナイザ上でMSCアイコンが選択されていることを確認し、[編集]メニューから[関連付け]コマンドを選択します。[関連付け]ダイアログ ボックスが表示されます。

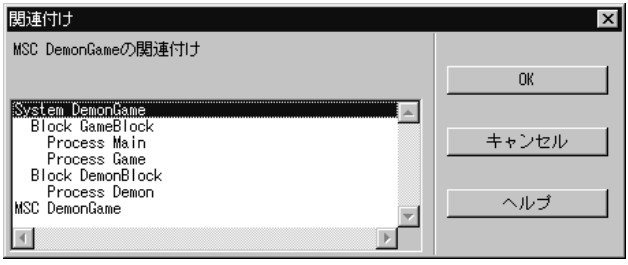


図87: MSCとSDLダイアグラムの関連付け

6. 一覧から、System DemonGameを選択します。

7. オーガナイザの内容を確認します。*Other Documents*チャプタに表示されているMSCアイコンの他に、システム ダイアグラムに接続されたMSCリンク アイコンが表示されます。MSCリンク アイコンを選択すると、オーガナイザのステータス バーに、実際にリンクしているMSCの情報が表示されます。

   – MSCリンク アイコンが表示されない場合はオーガナイザの[表示オプション]を確認します。必要に応じて、[関連シンボル]オプションをオンにして[適用]ボタンをクリックします。

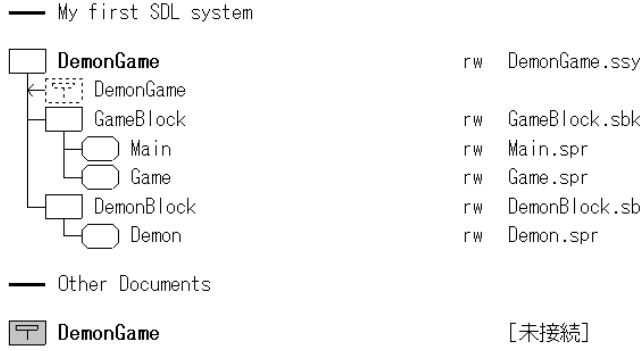


図88: システム ダイアグラムと関連したMSCの接続

MSCの編集

1. 新たに追加したMSCシンボルに対して*MSC*エディタが起動されています。MSCエディタのウィンドウは、SDLエディタ ウィンドウとよく似ていますが、シンボル メニューやコマンドおよびクイック ボタンの種類が異なります。

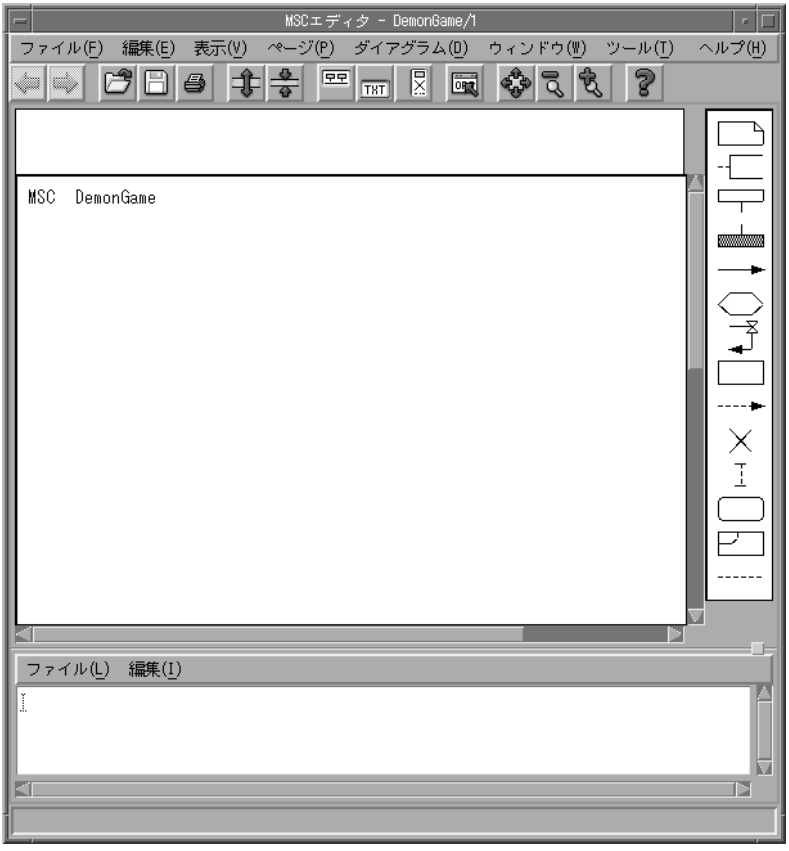


図89: MSCエディタ ウィンドウ（UNIX）

次に、MSCエディタを使用してダイアグラムを完成させていきます。

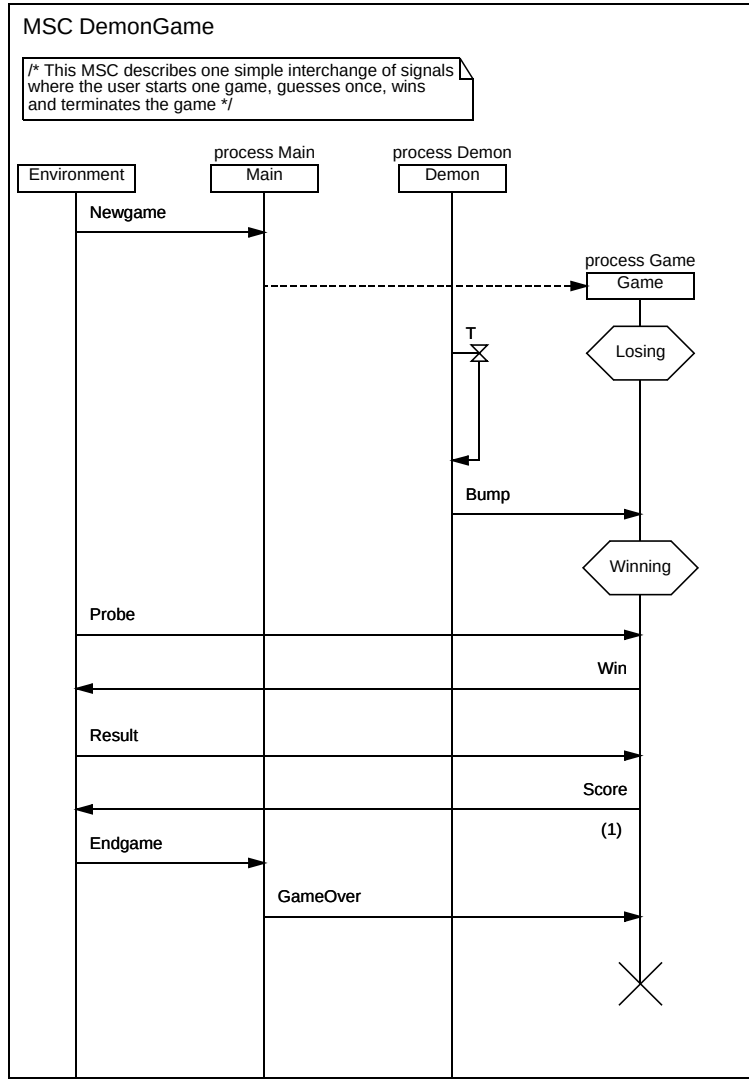


図90: DemonGame システムに対応するMSC

MSCは、4つのインスタンス（長方形の名称枠で始まる垂直線）と、多数のメッセージ（水平の矢印）、プロセス生成（水平の点線）、タイマ（砂時計で始まり矢印で終わるシンボル）、および、2つの条件シンボル（六角形）で構成されています。また、テキスト形式のコメントが記述されたテキスト シンボルも存在します。

MSCの編集方法

MSCの編集方法を以下に示します。シンボル メニュー内のシンボルの名称がわからない場合は、シンボルを選択またはポイントして、ステータス バーに表示されている説明を参照します。

1. まず、テキスト シンボルを追加し、図97の内容にしたがってテキストを入力します（SDLエディタと操作方法は同じです）。

2. 次に、3つのインスタンスを配置し、各インスタンスの名前として「`Environment`」、「`Main`」、「`Demon`」を入力します。

   – MSCエディタにインスタンスを配置するには、まず、シンボル メニューからインスタンス ヘッド シンボルを選び、図90で示しているドローイング エリアの位置へドラッグします。インスタンスヘッドを配置すると、MSCエディタによって、ただちにインスタンスの縦軸が自動生成されます（無限長）。

   – インスタンス名を割り当てるために、テキスト「`Environment`」、「`Main`」、「`Demon`」を入力します。

   – インスタンスの種別（`process Main`、`process Demon`）を指定するために、各インスタンス ヘッド シンボルのすぐ上にある小さい長方形を選択し、「`process Main`」、「`process Demon`」と入力します。

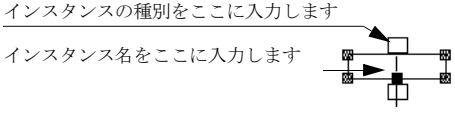


図91: インスタンス ヘッドへのテキスト属性の結合

   – インスタンス ヘッドの位置を変えたければ、新しい場所へドラッグすることができます。

3. 3つのインスタンスを追加したら、Newgameメッセージを追加します。
   - シンボル メニューから、メッセージ シンボルを選択します。
   - ダイアグラムにマウスのポインタを移動します。インスタンス軸の外に始点を表す円形のシンボルが表示されます。
   - メッセージの始点を指定するために、Environmentインスタンスの軸をクリックします。
   - Mainインスタンスの軸に向かってマウスのポインタを移動します。メッセージを示す矢印がポインタによって引き出され、インスタンス軸の外に終点を表す黒い円形のシンボルが表示されます。
   - メッセージの終点を指定するために、インスタンスMainの軸をクリックします。
   - メッセージ名「`Newgame`」を入力します。
   - メッセージの位置を変えたければ、マウスをドラッグしてメッセージを上下に動かすことができます。また、インスタンスの縦軸に沿って、メッセージの始点または終点だけを動かすことができます。
4. Gameインスタンスは動的に生成されるプロセスです。Gameを追加するには、プロセス生成シンボルを使用します。プロセス生成シンボルの追加はメッセージと同じ方法で追加できます。
   - シンボル メニューから、プロセス生成シンボルを選択します。
   - プロセス生成シンボルの元になるインスタンスを指定するために、Mainインスタンスの軸を1回クリックします。
   - インスタンス ヘッドを配置する位置で再度クリックして、配置位置を指定します。プロセスが生成され、インスタンス ヘッドと縦軸が追加されます。
   - インスタンスの種別とインスタンス名を入力します。
   - 必要であれば、インスタンス ヘッド シンボルを移動することができます。
5. Gameインスタンスの軸に、最初の条件シンボルを追加します。
   - シンボル メニューで条件シンボルをクリックして選択し、マウスをインスタンスの軸まで移動します。再度クリックして条件シンボルを挿入します。条件名「`Losing`」を入力します。

– 条件シンボルは、インスタンスの軸に沿って垂直に動かすことができます。

6. タイマをDemonインスタンスの軸に追加します。

   – シンボル メニューでタイマ シンボルを選択します。

   – タイマシンボルの始点を指定するために、Demonインスタンスの軸をクリックします。

   – タイマシンボルの終点を指定するために、同じ軸を再度クリックします(終点は、始点よりも下にしなければなりません)。

   – タイマ名として`T`を入力し、終了します。

   – 必要であれば、タイマの始点、または終点を、軸に沿って上下にドラッグし、タイマ シンボルのサイズを変更することができます。また、シンボルを上下にドラックして移動することができます。

7. Bumpのメッセージを追加します。

8. Gameインスタンスの軸に、2番目の条件シンボル、`Winning`を追加します。

9. 残りのメッセージを追加します。Scoreメッセージには、パラメータ値として「1」を定義します。パラメータの値を入力するには、メッセージを選択した際に表示される2つの長方形の下側にテキスト`1`を入力します。

図92: メッセージとテキスト属性の結合

10. プロセス停止シンボルを追加してMSCの編集を終了します。

    – シンボル メニューでプロセス停止シンボルを選択します。

    – Gameインスタンスの軸をクリックして、最後のメッセージの下にこのシンボルを配置します。

11. MSCエディタを終了する前に、MSCを保存します。保存する際、MSCエディタにはファイル名として`DemonGame.msc`が自動的に入力されます。[OK]をクリックして、このファイル名で保存します。

インデックス ビューワーの使用

この演習では、インデックス ビューワーの使用方法を学びます。インデックス ビューワーは、SDLシステムを構成するSDLエンティティの定義と参照をグラフィカルに表示するツールです。このツールによって、SDLシステムと関連する事実上すべてのSDL情報を管理することができます。また、SDLダイアグラムを逆参照する際に役立つ多くの機能が備えられています。

この演習を始めるには、事前にシステムの全SDL要素の定義と参照を含んだ最新のクロスリファレンス ファイルを用意する必要があります。そのファイルは、システムに対する意味分析の実行結果として生成されているはずです。

システムを分析した後にSDLダイアグラムを変更している場合は、クロスリファレンス ファイルを再度作成する必要があります。[クロスリファレンス ファイルの生成]オプションがONになっていることを確認して、再度、意味分析を実行してください（112ページの図84参照）。

学習内容

- インデックス ビューワーの起動
- 定義の検索
- 参照の検索

インデックス ビューワーの起動

1. オーガナイザの[ツール]メニューでサブメニューの[SDL]を選択し、サブメニューから[インデックス ビューワー]を選択します。

   – または、[インデックス ビューワー]クイック ボタンをクリックします。保存していないダイアグラムを保存するように促されます。SDLシステムの分析が始まり、新しいクロスリファレンス ファイルが自動的に生成されます。

2. [インデックス ビューワー] ウィンドウが表示されます。生成されたクロスリファレンス ファイル`DemonGame.xrf`を開きます（クイック ボタンを使用していない場合は自動的にファイルが開きます）。

3. インデックス ビューワーは、クロスリファレンス ファイルを読み込むと内容を変換し、それらをグラフィカルに表示します。

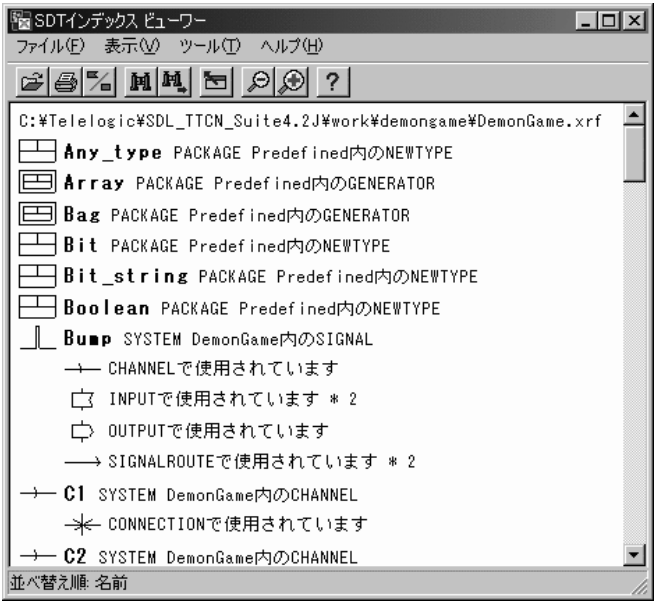


図93: インデックス ビューワー ウィンドウ

– いくつかの定義はSDL Suite環境にあらかじめ組み込まれています (*PACKAGE Predfined*もその1つです)。ただし、このチュートリアルではそれらを取り扱いません。

次の演習では、ある信号を送受信することのできるすべての状態を検出するために、インデックス ビューワーを使用するでしょう。また、その信号が定義されている箇所を検索することもできます。

定義の検索

Probe信号が定義されている箇所を検索してみましょう。デフォルトの設定では、ウィンドウ内に表示される定義はアルファベット順に表示されます。ただし、その定義を見つけるために手動でウィンドウをスクロールする必要はありません。

[検索]クイック ボタンを使用すれば、ウィンドウ内のあらゆるテキストを検索できます。ただし、定義名を検索する場合には、他にも高速に検索する方法があります。

1. インデックス ビューワーに信号名「Probe」を入力します。入力を始めると、太字で表示された名前に対してすぐに検索が開始されます。キーボードから文字を入力するたびに、ウィンドウ最下部のステータス バーに検索しているテキストが表示され、最初に適合する名前までウィンドウがスクロールされます。何文字か入力すると、インデックス ビューワーに、Probe信号が選択された状態で表示されます。

図94: Probe信号の検索

選択された行には、信号のアイコン、定義名と定義のタイプ（Probe SIGNAL）、および信号を定義しているダイアグラム（SYSTEM DeomonGame）が表示されます。
次に、信号の定義されている場所を確認します。

2. Probeアイコンが選択されていることを確認します。

3. [ツール]メニューで[定義の表示]を選択します。

   – または、Probeアイコンをダブルクリックします。

   Demon Gameシステムのダイアグラムを表示するSDLエディタ ウィンドウが現れます。信号の宣言（定義）が存在するテキスト シンボルが自動的に選択されます。

参照の探索

インデックス ビューワー内の[Probe]アイコンの下には、Probe信号が参照しているSDLエンティティのアイコンなど、Probe信号で使用しているすべての要素（参照）が一覧表示されます。図94に表示された情報は、以下のように解釈できます。

- この信号は1つのチャネルによって伝達されます。
- この信号は2つの状態に入力される可能性があります。
- この信号は1つの信号ルートによって伝達されます。

最後に、この信号が入力される場所を検索します。

1. 入力アイコンを選択します。
2. [ツール]メニューを選択すると、[使用状態1の表示]と[使用状態2の表示]の、2つのメニュー項目が表示されます。
3. メニュー項目[使用状態1の表示]を選択します。Gameプロセスのダイアグラムを表示するSDLエディタ ウィンドウ内で入力シンボルが選択されます。
4. メニュー項目[使用状態2の表示]を選択します。Gameプロセス内の2番目の入力シンボルが選択されます。したがって、この信号が入力される可能性のある場所は2箇所あります。
5. 信号ルート アイコンをダブルクリックします。Probe信号が定義された信号ルートがSDLエディタ内で選択されます。

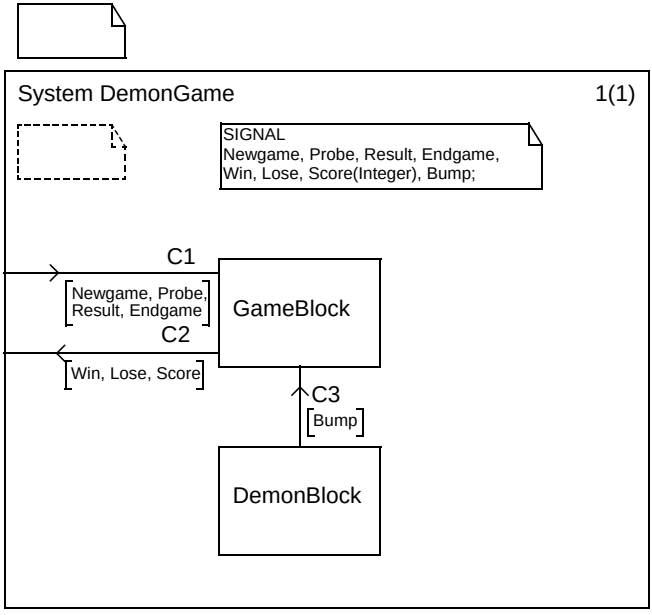
   – 複数の参照があるアイコンをダブルクリックすると、SDLエディタ内での選択は次に発生する要素に移動します。インプット アイコンをダブルクリックしてこれを確認してください。

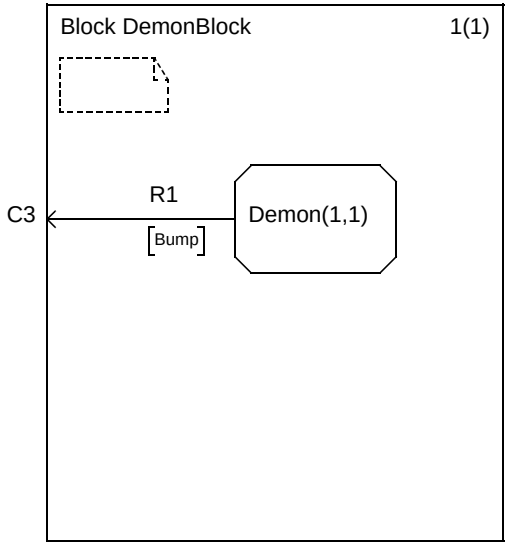
まとめ

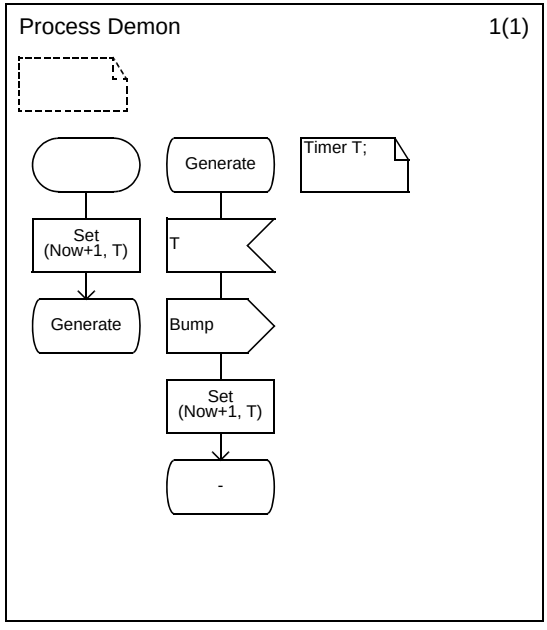
この章では、SDL Suiteでサポートされている基本機能について学習しました。すなわち、SDLダイアグラムの作成、管理、編集、および印刷の実施や、MSCエディタでのメッセージ シーケンス チャートの作成を行いました。また、SDLダイアグラムに対する意味的、構文的なチェックの実行方法や、インデックス ビューワーの使用方法を学習しました。

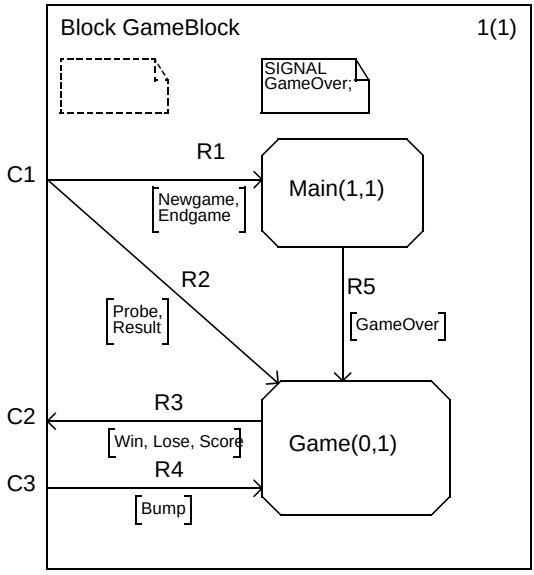
次のチュートリアルでは、シミュレーションを実行してこの章で作成したSDLシステムを動作させます。さまざまな演習が用意されている次のチュートリアルは、136ページの「このチュートリアルの目的」から開始します。

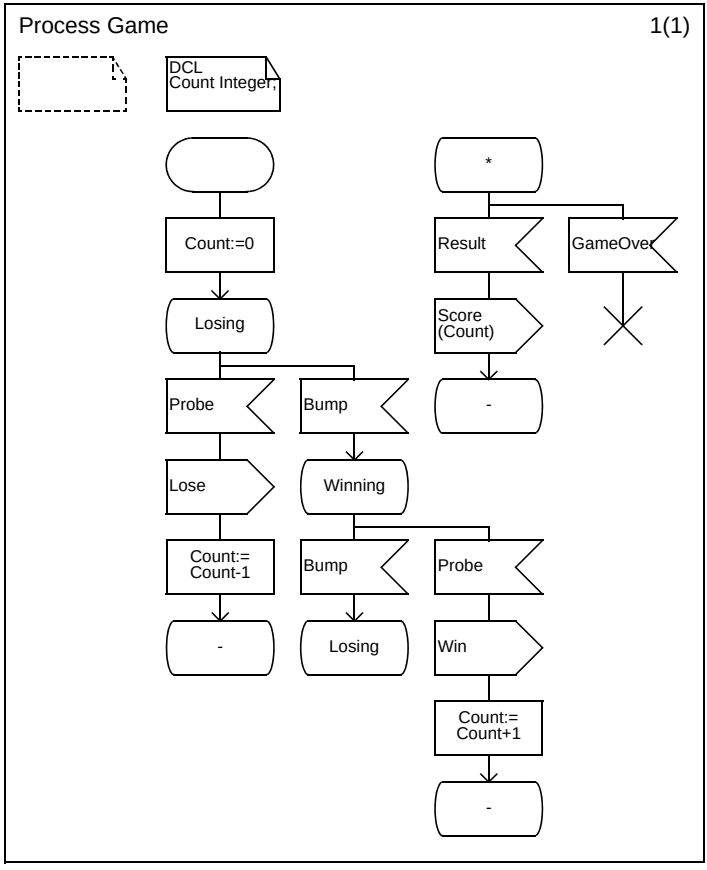
付録A: SDL-88 DemonGameの定義

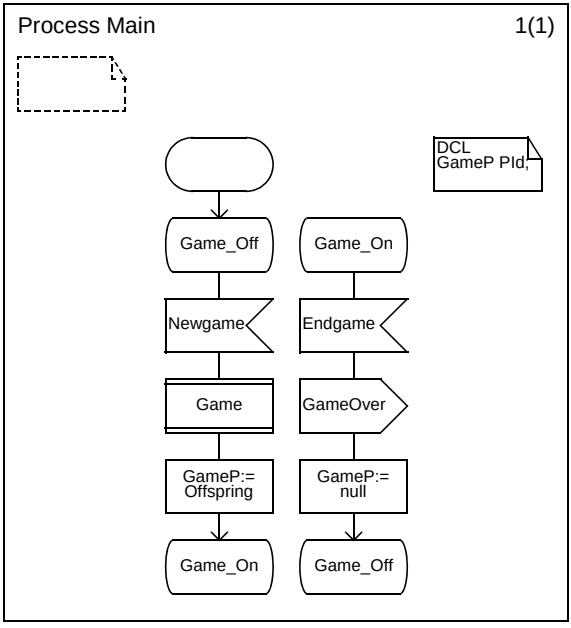



Block DemonBlock
1(1)
C3
R1
Bump
Demon(1,1)

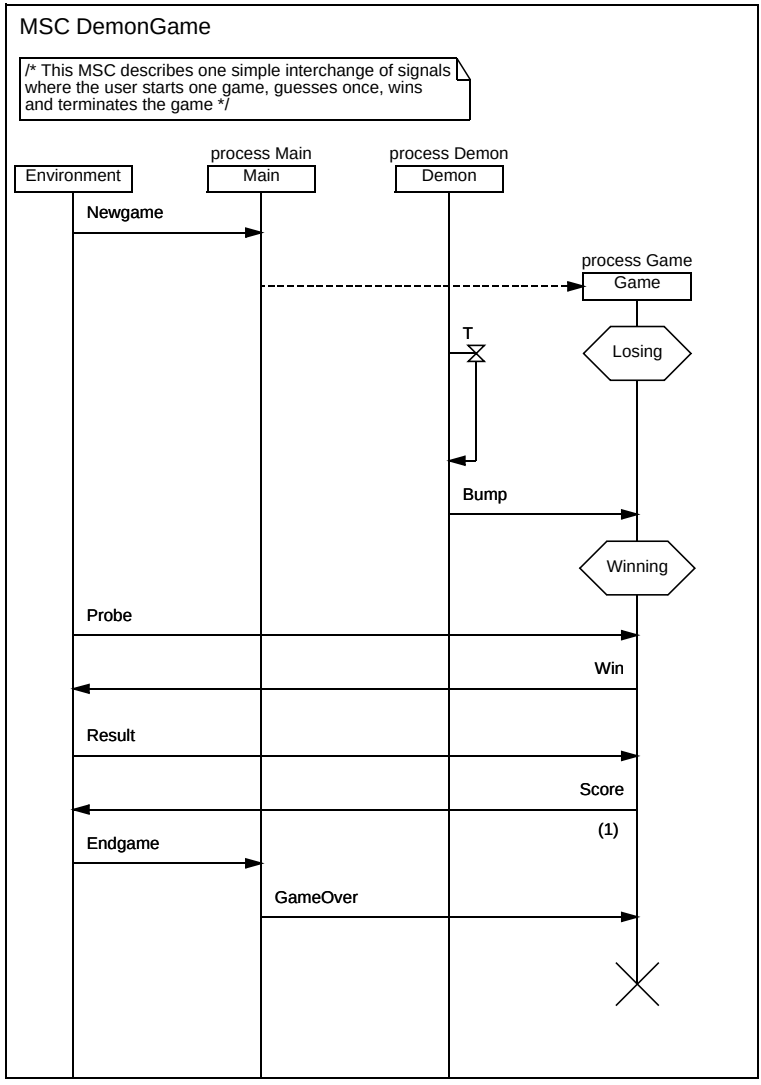

Process Demon
1(1)
Generate
Timer T;
Set
(Now+1, T)
T
Generate
Bump
Set
(Now+1, T)
-


Block GameBlock
1(1)
SIGNAL
GameOver;
R1
C1
Main(1,1)
Newgame,
Endgame
R2
R5
Probe,
Result
GameOver
C2
R3
Win, Lose, Score
Game(0,1)
C3
R4
Bump


Process Game
1(1)
DCL
Count Integer;
Count:=0
Losing
Probe
Bump
Lose
Winning
Count:=
Count-1
Bump
Probe
-
Losing
Win
Count:=
Count+1
-
*
Result
GameOver
Score
(Count)
-


Process Main
1(1)
DCL
GameP PId,
Game_Off
Game_On
Newgame
Endgame
Game
GameOver
GameP:=
Offspring
GameP:=
null
Game_On
Game_Off

付録B: DemonGameのMSC



第 4 章 チュートリアル: SDLシミュレータ

SDLシミュレータは、SDLシステムの振る舞いをテストするためのツールです。このチュートリアルでは、DemonGameシステムを実際に動作させてみます。

このチュートリアルを正しく理解するには、第3章「チュートリアル: SDLエディタとアナライザ」の演習を終了しておくことが必要です。

シミュレータの使用方法を理解するために、このチュートリアルの内容をすべて読んでください。また、記載されている手順にしたがって、お使いのコンピュータ システムで実際に演習してください。

このチュートリアルの目的

このチュートリアルでは、SDL Suiteにおけるシミュレーションの基本的な機能について学習します。一般的にシミュレーションとは、ユーザーの制御下でシステムを実行することを意味します。ユーザの制御は、通常はデバッガを使用したステップ実行や、ブレークポイントの設定、システムやプロセスの状態チェック、変数の参照、信号送信、実行状況のトレースなどを言いますが、シミュレータではこれらをSDLのレベルで実行できます。

このチュートリアルは、SDL Suiteの機能を理解するためのガイダンスとなるように配慮されています。チュートリアルを読む際は、説明されている各演習をコンピュータで実際に操作して確認してください。

このチュートリアルでは、内容を容易に理解できるように比較的簡単な題材を選んでいます。ただしSDL言語の学習を目的としていないため、説明はすべてSDL言語に関する基礎知識があることを前提に記述されています。

このチュートリアルでシミュレータを使用するためには、第3章「チュートリアル: SDLエディタとアナライザ」の演習を終了しておくことが必要です。

> メモ：Cコンパイラ
>
> SDLシステムをシミュレーションするには、使用するコンピュータ システムにCコンパイラがインストールされていなければなりません。このチュートリアルを始めるまえに、どのようなCコンパイラが使えるか確認しておいてください。

> メモ：プラットフォーム間の相違点
>
> この章のチュートリアルやその他の記述は、UNIXとWindowsの両方のプラットフォームで実行可能な項目に対して1種類の手順のみを記述します。プラットフォームによって操作が異なる場合は、「UNIX」または「Windows」などの指示を記述します。
>
> このようなプラットフォームを特定した記述がある場合は、ご使用のプラットフォームに関する記述のみを参照してください。通常は、両方のプラットフォームで画面に表示される情報が等しい場合、どちらか一方の画面表示のみを示します。したがって、図のレイアウトや概観は、SDL Suiteを使用環境で実行したときに表示される画像と異なることがあります。重要な部分がプラットフォーム間で異なる場合に限って、それぞれの画面を示します。

シミュレータの生成と起動

SDLシステムの設計とアナライザによるチェックを完了すれば、このシステムをシミュレーションすることができます。すなわち、SDLシステムに対する対話型の動作確認が可能になります。DemonGameシステムをシミュレーションできるようにするためには、まずシミュレータ実行形式ファイルを作成します。次に、対応するユーザー インターフェイスを使用してシミュレータを起動します。

メモ：

この章の説明通りに動作するシミュレータ実行形式ファイルを生成するには、作成したダイアグラムではなく製品に収められているSDLダイアグラムを使います。製品に収められているSDLダイアグラムは、以下の方法で使用できるようにします。

- UNIX の場合は、`$telelogic/sdt/examples/demongame`にあるすべてのファイルを作業用ディレクトリ`~/demongame`にコピーします。
- Windowsの場合は、`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥sdt¥examples¥demongame`にあるすべてのファイルを作業用ディレクトリ`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥work¥demongame`にコピーします。

これまでに作成したダイアグラムを利用する場合は、シミュレーションの実行手順が多少異なります。

製品に収められたダイアグラムを使う場合は、オーガナイザで`demongame.sdt`システム ファイルを再度開くことが必要です。

学習内容

- シミュレータ実行形式ファイルの作成
- シミュレータのユーザー インターフェイスの起動
- ユーザー インターフェイスからのシミュレータ実行形式ファイルの起動

シミュレータの作成

以下の手順でシミュレータの実行形式ファイルを作成します。

1. オーガナイザで、システム ダイアグラム アイコンが選択されていることを確認します。
2. [生成]メニューから[実装]コマンドを選択します。[実装]ダイアログ ボックスが開きます。

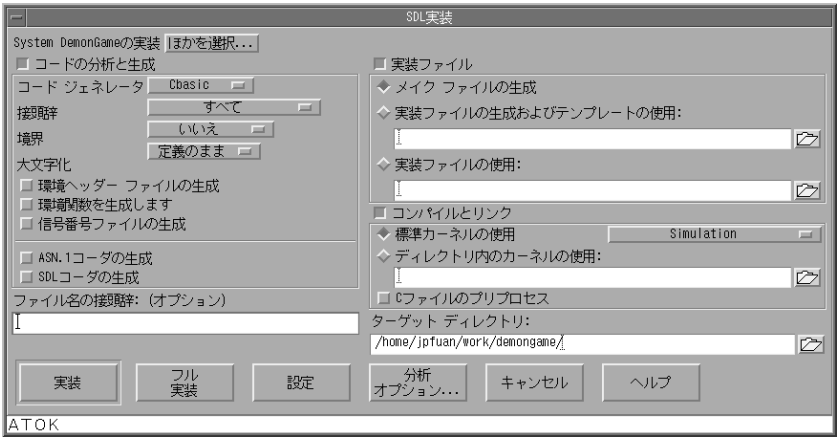


図95: [実装] ダイアログ ボックス (UNIX)

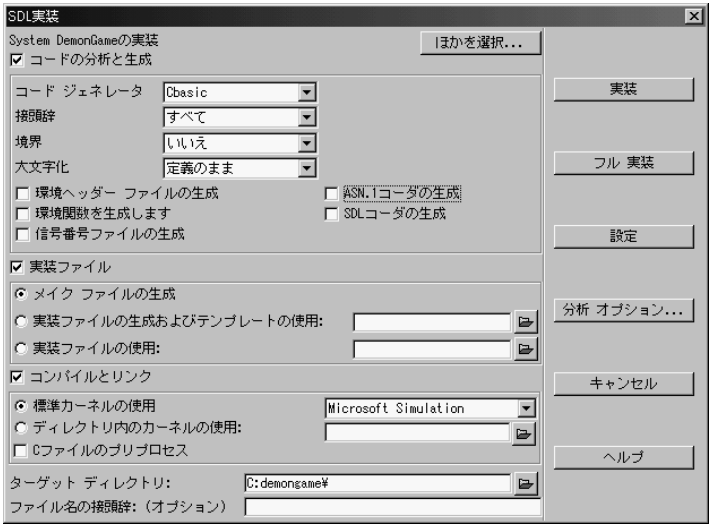


図96: [実装] ダイアログ ボックス (Windows)

3. [実装]ダイアログ ボックスの図にしたがって、オプションを設定します。
   – [コードの分析と生成]をオンに設定します。
   – [メイク ファイル]と[メイク ファイルの生成]をオンに設定します。
   – [コンパイルとリンク]をオンに設定します。
   – [標準カーネルの使用]をオンに設定します。右のオプション メニューで正しいシミュレーション カーネルが指定されていることを確認します。正しくない場合は、メニューから適切なシミュレーション カーネルを選択します。
4. [実装]ボタンをクリックします。
5. [ツール]メニューから[オーガナイザ ログ]を選択します。エラーが発生していないかをチェックします。エラーが発生していなければ、オーガナイザのステータス バーに、「アナライザが完了しました」と表示され、オーガナイザ ログの「実装が開始しました」メッセージから「実装が完了しました」メッセージの間に、エラーの通知が表示されていないはずです。
6. エラーが通知された場合は、[実装]ダイアログ ボックスを開き、[フル 実装]ボタンをクリックします。今度はエラーが通知されないはずです。
   – ほかにも、使用しているCコンパイラが問題の原因になることがあります。依然としてエラーがなくならない場合は、[gcc-シミュレーション]や[Microsoftシミュレーション]などの、Cコンパイラに対応するシミュレーション カーネルを選択して、シミュレータ実行形式ファイルの[実装]の操作を再度行ってください。

シミュレータの起動

生成されたシミュレータの実行形式ファイルは、SDL Suiteを起動したディレクトリに、UNIXでは、`demongame_xxx.sct`というファイル名で、また、Windowsでは`demongame_xxx.exe`というファイル名で保存されます。`_xxx`の部分は、使用されるプラットフォームや、カーネル、コンパイラによって異なります。シミュレータの実行形式ファイルには*モニタ システム*が盛り込まれ、シミュレータの実行を制御およびモニタする際に使用する、各種コマンドが利用可能になります。

シミュレータは、OSのプロンプトから直接実行することができます。その場合は、コマンド ライン インターフェイスを使用して、すべてのコマンドをテキストでモニタ システムに入力します。

また、SDL Suiteでは、オーガナイザから起動できる使いやすいグラフィカル ユーザー インターフェイスがサポートされています。

1. [ツール]メニューの[SDL]サブメニューから[シミュレータUI]コマンドを選択します。[シミュレータUI]ウィンドウが表示されます。

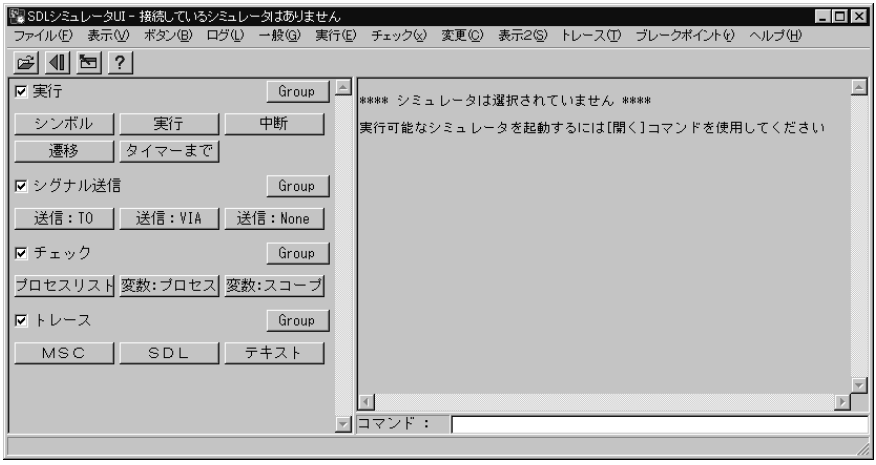


図97: [シミュレータUI]のメイン ウィンドウ（Windows）

右側のテキスト エリアは、現在シミュレータが実行されていないことを表示しています。テキスト エリアには、入力されたコマンドや、コマンドの実行結果、エラー メッセージなどのモニタ システムの入出力がテキストで表示されます。

2. シミュレータを起動するには、[ファイル]メニューから[開く]を選択するか、クイック ボタンの[開く]をクリックします。

3. ファイル選択ダイアログ ボックスが表示され、作成したシミュレータ実行形式ファイルが一覧表示されます。これを選択して、[OK]をクリックします。

4. [シミュレータUI]のテキスト エリアには、シミュレータが起動されたことを示すウェルカム メッセージが表示されます。

```
Welcome to SDL SIMULATOR. Simulating system Demongame.
```

シミュレータが起動されると、SDLシステム内の静的生成プロセスのインスタンスが作成されます（ここでは、MainとDemon）。ただし、それらの最初の状態遷移はまだ実行されません。なお、最初に実行されるのはMainプロセスです。

現在、シミュレータUIは、モニタ システムでコマンド入力を受け付けることができます。コマンドを受け付けることができる時はいつでも、コマンド：がテキスト エリアに表示されます。

状態遷移の実行

学習内容

- テキストによるコマンドの入力
- 次に実行されるシンボルの表示
- テキスト トレースと図式トレースの解釈
- 次の遷移の実行
- ボタンによるコマンドの実行
- 外部環境からの信号の送信

スタート遷移の実行

この演習では、システムのプロセス インスタンス内でスタート遷移を実行します。まず、実行時に表示するトレース レベルを設定します。テキスト トレースのレベルを設定するために、[Set-Trace]コマンドを使用します。

[シミュレータUI]ウィンドウ右側のテキスト エリアの下にある、[コマンド]テキスト フィールドにテキストでコマンドを入力します。このフィールドを、*入力行*と呼びます。

1. 入力行をクリックして、カーソルを移動します。「`set-trace 6`」と入力し、`Enter`キーを押します。「6」の値はトレース レベルの指定であり、状態遷移の間に実行される動作をすべて通知するように指定します。入力したコマンドは、テキスト エリアに表示され、シミュレータのモニタ システムによってトレース設定の内容も表示されます。現在テキスト エリアは次のように表示されているはずです。

```
Welcome to SDL SIMULATOR. Simulating system Demongame.

コマンド : set-trace 6
Default trace set to 6

コマンド :
```

   また、図式トレース（*GR トレース*）を指定することができます。図式トレースでは、次に実行されるシンボルが選択されることで、SDLダイアグラムにおける実行状況がトレースされます。GRトレースのシミュレーション起動時のデフォルト設定は、オフです。

2. GRトレースを有効にするには、コマンドとして「`set-gr-trace 1`」を入力します。「1」の値を指定すると、モニタ システムへコマンドが入力されるたびに、次に実行されるシンボルがSDLエディタに表示されます。

3. コマンド「show-next-symbol」を入力してSDLエディタ ウィンドウを起動させます。SDL エディタが現れて、スタート シンボルが選択された状態でMainプロセスのSDLダイアグラムが表示されます（選択されたシンボルは、次に実行されることを表します）。SDLエディタ ウィンドウは、シミュレーション実行時にGRトレースのビューワーとして使用します。

4. 必要ならば[シミュレータUI] ウィンドウとSDLエディタ ウィンドウの移動やサイズ変更を行い、双方のウィンドウを見えるようにし、ディスプレイの大きさに合わせます。以下にウィンドウを操作するための、いくつかのヒントを示します。

   – SDLエディタのサイズを変更する前に、クイック ボタンを使ってシンボル メニューとテキスト ウィンドウを閉じた方がディスプレイを広く使えます。ここでは、エディタ ウィンドウは、図式トレースを参照するためだけに使用し、編集作業には使用しません。

   – SDLエディタの[常に新しいウィンドウを開く]オプションが、オフになっていることを確認します。[表示]メニューの[エディタ オプション]コマンドを使用すると、このオプションの参照と変更ができます。

   – 必要に応じて、シミュレータUIウィンドウの幅を狭くすることができます。このウィンドウを狭くしても、テキスト エリアの幅が狭くなるだけです。また、高さを縮めることもできますが、ウィンドウの左側にあるボタン類は表示させておいてください。

   – シミュレータのチュートリアルを学習している間は、オーガナイザを参照しないため、ウィンドウ システムのコマンドでオーガナイザ ウィンドウを隠すか、最小化しても構いません。

   次に実行される遷移を参照するには、エディタ ウィンドウの表示内容を確認します。現在Mainプロセスのスタート遷移が選択されています。また、次の状態シンボルはGame_Offです。この（空の）遷移を実行するために「Next-Transition」と入力します。

5. Next-Transitionコマンドは、単純に入力行に「n-t」と入力しても実行できます。このように、短縮表現が全コマンドの中で固有になるものに限り、短縮表現で入力することができます。

   Mainプロセスのスタート遷移は以下の2種類の方法でトレースされます。

   – 第1の方法では、テキスト エリア上に、遷移に関する情報がテキストで表示されます。すなわち、プロセス インスタンスIDや、初期状態の名称、

シミュレーション時間の現在の値などが表示されます。最後に、遷移後の状態名が表示されます。

```
*** TRANSITION START
*       PId     : Main:1
*       State   : start state
*       Now     : 0.0000
*** NEXTSTATE  Game_Off
```

– 第2の方法ではSDLエディタ上で、GRトレースによって次に実行されるシンボルが選択されます。次に実行されるのはDemonプロセスのスタート状態なので、Demonプロセスのダイアグラムがエディタにロードされます。

次の遷移はタイマの設定が含まれたDemonプロセスのスタート遷移です。この実行には、コマンドインターフェイスの別の機能を使用します。

6. 入力行にポインタを置き、矢印キーの「↑」を押します。前に入力したコマンド`n-t`が表示されます。`Enter`キーを押してこのコマンドを実行します。

– すでに実行したコマンドを表示するために、入力行で「↑」と「↓」を繰り返して使用することができます。この機能は、*コマンドヒストリ*と呼ばれています。また、パラメータの値を変更するなど、実行する前にコマンドを修正することもできます。

Demonプロセスのスタート遷移が実行されます。SDLエディタでは、Tタイマの宣言が記述されたテキスト シンボルが選択されます。これはSDL Suiteの表示規則であり、外部環境から信号が送信されない場合、システムで次に起こるイベントはタイマの終了であることを示しています。

なお、テキスト トレースに表示された情報にも、Tタイマを指定する同様の記述があります。

外部環境から信号の送信

SDLシステムに対して何らかの操作を行なう場合、外部環境からシステムへ向かって信号を送信しなければなりません。ここでは、まずNewgame信号をMainプロセスへ送信します。入力行に「`Output-Via`」と入力します。このOutput-Viaコマンドのパラメータには、信号名と、信号パラメータ（ここでは使用しません）、およびチャネル名があります。

ここでは、入力行にテキストで実行コマンドを入力する代わりに、コマンド ボタンを使用します。さまざまなコマンド ボタンが、シミュレータUIの左側のモジュールに表示されています。ボタンを選択し、マウスのボタンを放す前に、マ

ウスのポインタをボタンから遠ざけると、実行するコマンドの内容を前もって参照することができます。対応するコマンドがウィンドウの下のステータス バーに表示されます。

1. [シグナル送信]ボタン モジュールにある、[送信：VIA]ボタンをクリックします。

   このボタンをクリックすると、Output-Viaコマンドが実行され、実行内容がテキスト エリアに表示されます。ダイアログ ボックスが表示され、最初のパラメータである信号名の入力を促されます。ダイアログ ボックスの一覧には外部環境から送信できるすべて信号が表示されます。

図98: Newgame信号の送信

2. Newgame信号を選択して、[OK]をクリックします。
3. 別のダイアログ ボックスが表示され、チャネル名の入力を促されます。

図99: [Send Via] コマンドを実行する際のチャネルの選択

4. 外部環境からシステムへ送信するチャネルは1つだけなので、明示的に選択する必要はありません。単純に[OK]ボタンをクリックします。

   テキスト エリアで信号が送信されたことを確認できます。GRトレースでは、次に実行されるシンボルとしてMainプロセスにあるNewgameの入力が選択されます。

5. [実行]モジュールの[遷移]ボタンをクリックして、次の遷移を実行します。(この操作は、Next-Transitionコマンドの実行と等価です)。テキスト トレースの情報を参照すると、遷移はGame_On状態まで進んだことがわかります。また、遷移の開始時には、状態がGame_Offで、入力信号がNewgameであったことがわかります。

```
*** TRANSITION START
*      PId    : Main:1
*      State  : Game_Off
*      Input  : Newgame
*      Sender : env:1
*      Now    : 0.0000
*   CREATE Game:1
*   ASSIGN  GameP := Game:1
*** NEXTSTATE  Game_On
```

   遷移の実行中にGameプロセスが生成されたので、GRトレースでは、次に実行されるシンボルとしてGameのスタート状態が選択されます。これは、テキスト トレースとGRトレースの内容が異なる顕著な例です。

   – テキスト トレースには、遷移前後のプロセスの状態を含む、直前の遷移で起こった内容が表示されます。

– GRトレースでは、システムが放置された場合に次に何が起こるかが表示されます。したがって、次の遷移として、これまでとは別のプロセス ダイアグラムのスタート シンボルが選択されることもあります。

6. [遷移]ボタンをクリックして、Gameのスタート遷移を実行します。Gameプロセスは、Losing状態に移行し、GRトレースはDemonプロセスに戻ります。SDLエディタでは再び、Tタイマの宣言が記述されたテキスト シンボルが選択されます。

7. タイムアウトを発行させるために、次の遷移を実行します。次の遷移はタイマの出力です。つまり、タイムアウトを通知する信号を、タイマを起動したプロセスへ送信することを意味します。タイマ出力も、状態遷移と考えられます。シミュレーション時間が1に更新されたか確認します。

```
*** TIMER signal was sent
*   Timer    : T
*   Receiver : Demon:1
*** Now      : 1,0000
```

8. もう一度、遷移を実行します。この遷移は、Generate状態におけるTタイマの入力として記述されます。また、Bump信号がGameプロセスに送信されたことも表示されます。

```
*** TRANSITION START
*      PId    : Demon:1
*      State  : Generate
*      Input  : T
*      Sender : Demon:1
*      Now    : 1,0000
*   OUTPUT of Bump to Game:1
*   SET on timer T at 2.0000
*** NEXTSTATE  Generate
```

GRトレースでは、次に実行するシンボルとして、Bump入力が選択されます。プロセス ダイアグラムにおいて、Bump信号が入力されると、GameプロセスはWinning状態に移行しBumpまたはProbe信号の入力待ち状態になります。

9. GameプロセスをWinning状態へ移行する次の遷移を実行します。GRトレースの表示はDemonプロセスに切り換わり、次のデフォルト動作としてTタイマのタイムアウトが表示されます。ここでは代わりに、外部環境からProbe信号を以下の手順によって送信します。

10. [送信：VIA]ボタンを使って前の様にProbe信号を送信します。GRトレースはGameプロセスに切り換わります。

11. [遷移]ボタンで次の遷移を実行します。Probe信号が受信され、その結果としてWin信号が外部環境へ出力されます。プロセスはWinning状態に戻り、次のBump（またはProbe）信号を待機する状態となります。

```
*** TRANSITION START
*      PId    : Game:1
*      State  : Winning
*      Input  : Probe
*      Sender : env:1
*      Now    : 1,0000
*   OUTPUT of Win to env:1
*   ASSIGN  Count := 1
*** NEXTSTATE  Winning
```

ここでは、シミュレーションにおいて特定の状況または状態に遷移させるために、いかにNext-TransitionとOutput-Viaコマンドを使用するかについて学習しました。

内部状態の調査

この演習では、システムの内部状態を参照するために使用するコマンドのいくつかについて学習します。グラフィカル ユーザー インターフェイスを使用することにより、手動でコマンドを実行せずに内部状態を継続的に参照することが可能です。

前の演習で、トレースの解釈方法をすでに学習しているので、ここでは特に必要でない限りこれらの詳細については説明しません。

学習内容

- シミュレータUIを終了せずにシミュレータを再起動する方法
- コマンド ウィンドウとウォッチ ウィンドウの使用
- プロセス待ち行列の表示と解釈
- プロセスの信号入力ポートの表示
- さまざまな値の表示
- プロセス インスタンスの調査
- アクティブ タイマの表示

シミュレータUIの再起動

作業を継続する前に、シミュレータを再起動し、トレース レベルを設定します。

1. [ファイル]メニューから[再開]コマンドを選択します。
2. 現在起動しているシミュレータの終了を通知する表示があります。[OK]をクリックします。シミュレータがリセットされ、テキスト エリアがクリアされます。
3. 前と同じ方法で、トレース レベルを6に、GRトレース レベルを1に設定します。入力行にカーソルを置き、「↑」キーを使って以前入力した「Set-Trace 6」や、「Set-GR-Trace 1」コマンドを表示させて`Enter`キーを押せば、簡単にこれらのコマンドを実行できます。
   - 新しいシミュレータを起動した場合でも、シミュレータUIには、同一セッションで入力したコマンドがすべて記録されています。

プロセスと信号待ち行列の表示

SDL システムの内部状態を継続的に表示させるために、[コマンド]ウィンドウを使用します。ユーザー環境が正しく設定されていれば、このウィンドウにすべてのプロセスのリストと待ち行列が表示されます。

これらの情報は、List-Ready-Queueコマンドと List-Processコマンドを実行することにより、[コマンド]ウィンドウの、別々の「モジュール」に表示されます。これらのモジュールはメイン ウィンドウ内のボタンが配置されたモジュールと類似しています。

1. [表示]メニューから[コマンド ウィンドウ]を選択し、[コマンド]ウィンドウを開きます。ウィンドウのサイズを調節して双方のコマンド モジュールが表示されるようにします（図100を参照）。またウィンドウを移動して、SDL エディタ ウィンドウの参照と、[シミュレータUI]メイン ウィンドウの操作ができるようにします。

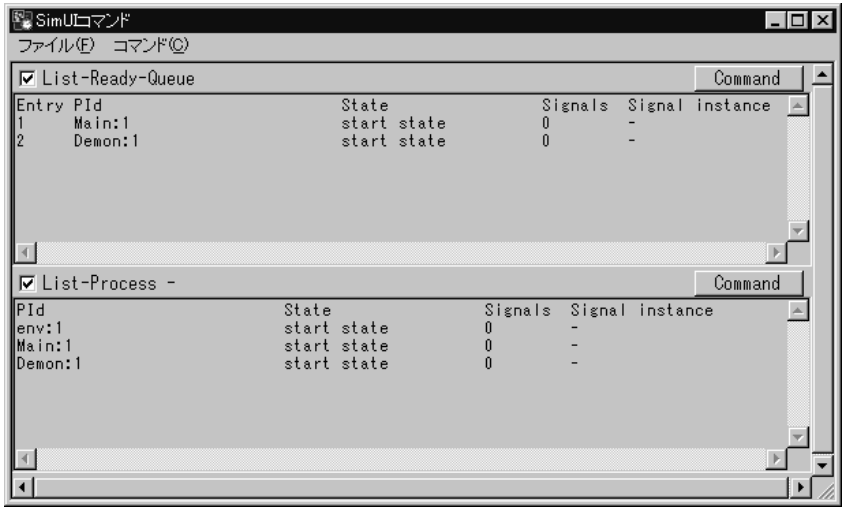


図100: [コマンド] ウィンドウ

メモ：

[コマンド]ウィンドウが初期状態で各コマンドモジュールを表示するかどうかは、ユーザー環境の設定によります。起動時にウィンドウがこれらのコマンドモジュールを表示しない場合は、[コマンド]メニューから[コマンドの追加]メニューを選択してコマンドを追加することができます（図101参照）。

図101: 追加するコマンドの指定

各コマンドモジュールには以下の情報が表示されます。

- [List-Ready-Queue]は、待ち状態にあるプロセスを表示します。つまり、次の遷移で実行が可能なプロセスが順番に並べられてリストに表示されます。このリストに表示される項目には、待ち行列内での順位を表す番号、プロセス インスタンス、プロセスの現在の状態、プロセスの入力ポートに存在する信号の数、および次の遷移を発生させる信号名などがあります。現在、Mainプロセスと Demonプロセスはスタート遷移の実行を待つ状態にあるため、信号は何も表示されていません。
- [List-Process]には、システム内でアクティブになっているプロセスがすべて表示されます。リストには、外部環境を表すenvプロセスが表示されていることを除いて、上記List-Ready-Queueと同じ情報が表示されます。現在、2つのリストで表示されている情報は等価です。

2. 次の遷移を実行し、[コマンド]ウィンドウの表示内容の変化を確認します。Mainプロセスは、待ち行列のリストからは、消去されます。これは、Mainプロセスが次の遷移を実行するためにNewgame信号を必要としますが、この信号がまだ送信されていないためです。一方、プロセスのリストにはMainプロセスの最新状態であるGame_Offが表示されます。

3. さらに次の遷移を実行します。現在待ち行列は空です。これは、Demonプロセスが Tタイマ信号の入力を必要としていますが、タイマがタイムアウトになっていないためです。

4. 外部環境からNewgame信号を送信します。[コマンド] ウィンドウには、Newgame信号がMainプロセスの信号入力ポートに入力されたことが表示され、これによりMainプロセスは実行可能となり、待ち行列リストに表示されます。

5. 必要ならばプロセスの入力ポートに存在し、次の遷移で実行されるすべての信号の一覧を表示することもできます。この表示を実行するコマンドには、対応するコマンド ボタンがないため、メニュー バーの[チェック]メニューから、[入力ポート]を選択します。各信号に対して、待ち行列内の順位を表す番号と、信号名、および信号の送信元が表示されます。Newgameの番号の前に付いている「*」（アスタリスク）は、この信号が次の遷移で消費されることを示しています。

変数とプロセス インスタンスの表示

[コマンド]ウィンドウの他に、[ウォッチ]ウィンドウを使用して変数の値を継続的に表示することができます。ここでは、MainプロセスのGameP変数を表示させて、Gameプロセスの開始から終了までの間に、どのようにその値が変化するかを観察します。

1. [表示]メニューから[ウォッチ ウィンドウ]を選択し[ウォッチ]ウィンドウを開きます。必要に応じて、ウィンドウを移動し、コマンド ウィンドウの内容も見えるようにします。

2. [ウォッチ]ウィンドウで、[ウォッチ]メニューから[追加]を選択し、変数リストに変数を追加します。

3. ダイアログ ボックスに、かっこで囲んだプロセス名と変数名を入力します。ここでは「`(Main) GameP`」と入力して、[OK]をクリックします。

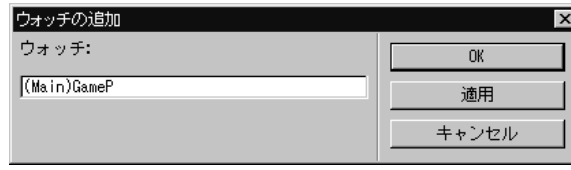


図102: 表示する変数を追加

4. 変数`null`が[ウォッチ]ウィンドウに表示されます。

図103: GamePを[ウォッチ]ウィンドウに追加

必要に応じて、値が見えるように、ウィンドウのサイズを変更します。

5. 次の遷移を実行します。[ウォッチ]ウィンドウのGamePの値が、Gameプロセスが生成されたことによってGame: 1に変更されることを確認します。Gameプロセスは、[コマンド]ウィンドウのリストにも追加されます。

6. ここで、新しく生成されたGameプロセスの確認ができます。Examine-PId（またはex-PId）コマンドを入力行に入力して、この情報を表示します。Parent、Offspring、Senderの現在の値が表示されます。Parentの値はMain:1と表示されるはずです

7. 外部環境からEndgame信号を送信します。待ち行列リスト内のGameプロセスの後に、Mainが追加されることを確認します。

8. 次の遷移を実行すると、Gameのスタート遷移が実行されます。

9. 次の遷移でMainプロセスが実行されるので、ここでMainプロセスを調べることができます。Examine-PIdコマンドを再度使って値を表示し、Gameプロセスの時の値と比較します。

10. Gameプロセスを終了するために、次の遷移を2回実行します。GamePの値がnullにリセットされます。また、[コマンド]ウィンドウからはGameプロセスが表示されなくなります。

その他のビューオプション

[チェック]メニューには、他にも多くの表示用コマンドがあります。アクティブなタイマの一覧を表示したり、信号やタイマ インスタンスなどのパラメータをチェックしたりできます。ここでの最後の演習として、待ち行列が空であるにも関わらず、システムがアイドル状態ではないことを表示させます。

1. [チェック]メニューで[タイマーのリスト]を選択して、Tタイマがまだ動作していることを確認します。タイマ名、対応するプロセス インスタンス、およびタイムアウト時間が表示されます。
2. 次の遷移を実行します。Examine-Timer-Instanceコマンドを実行して（`ex-tim-ins`を入力行に入力）、タイマ インスタンスを表示します。タイマの待ち行列が空になっていることが確認できます。つまり、Tタイマは、すでに動作していません。

動的エラー

学習内容

- 動的エラーの検出と解釈
- 最後に実行されたSDLシンボルの検索
- 中断した状態遷移の続行

動的エラーの検出

この演習では、Demongameシステムで発生する動的エラーを検出します。システム レベルで最初の4つの遷移を実行すれば、簡単に検出できます。

1. [ファイル]メニューから[再開]コマンドを選択します。
2. 「Set-trace 6」を実行して、トレース レベルを「6」に設定します。
3. この演習では、グラフィカル トレースは使用しません。[ファイル]メニューから[終了]を選択して、現在開いているSDLエディタを終了させてください。
4. [コマンド]ウィンドウと[ウォッチ]ウィンドウも必要ないので、[表示]メニューから[すべて閉じる]を選択してそれらを閉じてください。

5. テキスト エリアに警告メッセージが表示されるまで、Next-Transitionを4回実行します。

```
******************** WARNING ********************
Warning in SDL Output of signal Bump
Signal sent to NULL, signal discarded
Sender: Demon:1
TRANSITION
  Process      : Demon:1
  State        : Generate
  Input        : T
  Symbol       :
#SDTREF(SDL,c:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥work¥...
TRACE BACK
  Process      : Demon
  Block        : DemonBlock
  System       : Demongame
************************************************
```

このメッセージは、Demonプロセスから送信されたBump信号を受信するプロセスが存在しなかったことを示しています。この現象は、実はGameプロセスのインスタンスが生成されていないので当然なのです。本来、Bump信号が送信される際には、ユーザーは常にGameプロセスを動作させる必要があるため、現在のDemongameの定義は正しくありません。この問題の正しい修正方法は、Bump信号をDemonからMainへ送信させることです。そして、Gameプロセスが動作しているときに、MainプロセスによってBump信号が転送されるように変更します。

6. GRトレースが起動していない場合も、エラーの発生場所を確認することができます。[表示2]メニューで[1つ前のシンボル]を選択します。この操作によって、SDLエディタが開き、最後に実行されたシンボルが選択されます。ここでは、DemonプロセスのBump出力シンボルが選択されます。

   動的エラーが発生した場合も、さらに遷移を実行するために、また、システムの状態を調査するために、シミュレーションを継続することができます。動的エラーが発生したシンボルの直後で、実行状態が停止していることに注意してください。つまり、エラーが発生すると状態遷移は中断するわけです。

7. 中断された遷移を終了するまで続行します。通常のNext-Transitionコマンドを使用します。テキスト トレースを参照すると、エラーとなった出力シンボルで実際には、Bump信号が送信されなかったことを確認できます。

異なるトレース レベルの使用

状態遷移の間に表示されるテクスチュアルなトレース情報の量は、Set-Traceコマンドによって設定できます。これまで、トレース レベルを6に設定してこのコマンドを使用してきました。トレース レベルが大きい程、より詳しいトレースが表示されます。

また、トレース レベルはシステムの異なる部位ごとに設定可能です。つまり、ブロック、プロセス タイプ、およびプロセス インスタンスに対して異なるトレース レベルを設定することができます。あるプロセスが、設定されたトレース レベルを持っていない場合は、そのプロセスが属するブロックの値が適用されます。ブロックにもトレース レベルが設定されていない場合は、さらにブロックが属するより上位の構造の値が適用されます。システムに対しては常にトレース レベルが定義されており、初期のトレース レベルとして「4」が与えられています。

この演習では、DemonGameを実行するうえで、このトレース機能を使用します。この機能を使って、MainプロセスとGameプロセスの遷移に対してのみトレース情報を表示させ、Demonプロセスはトレースしないようにします。このように設定するには、システムのトレース レベルを「0」に設定し、GameBlockブロックの値を「6」に設定します。

GRトレースの値はこの演習では「1」に設定します。

学習内容

- システムのさまざまな要素に対するトレース レベルの設定
- 現在のトレース レベルの一覧表示
- 次のSDL シンボルのみの実行
- トレースが表示されるまで、遷移を連続して実行させる方法

トレース レベルの設定

Set-Traceコマンドは、ユニット名とトレース レベルの2つのパラメータを取ります。そして、このユニットに対してトレース レベルを割り当てます。ユニットは、入力行でユニット名を入力する代わりに、オプション メニューによって簡単に設定できます。

1. シミュレータを再起動します。必要に応じて、現在開いているSDLエディタ ウィンドウのサイズを変更、移動します。

2. [Trace]メニューから[Text Level: Set]を選択します。最初に表示されるダイアログ ボックスで、[System Demongame]を選択して[OK]をクリックします。次のダイアログ ボックスでトレース レベルを「0」に設定します。各トレース レベル（0 - 6）には簡単な説明が記述されています。

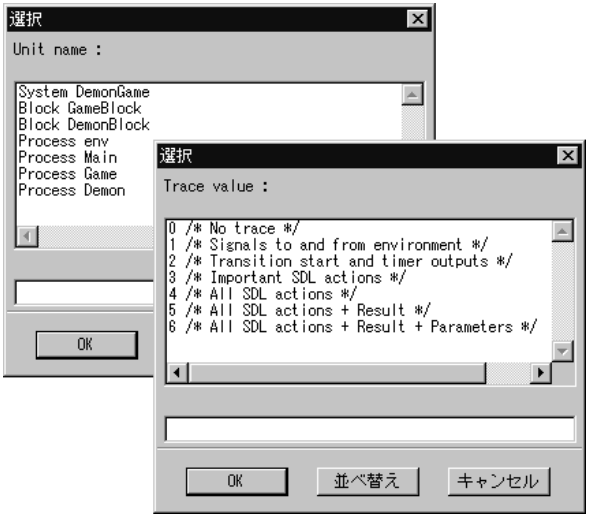


図104: DemonGame システムのトレース レベルを0に設定

3. 同じ手順で、GameBlockブロックのトレース レベルを6に設定します。

4. 同様に、[トレース]メニューの[SDL：設定]によってGRトレースの値も設定できます。DemonGameシステムのGRトレースの値を「1」に設定します。

5. [トレース]メニューの[テキスト：表示]と、[SDL：表示]を使って、正しいトレース レベルが設定されたか確認します。次のような情報が表示されるでしょう。

```
Default    4 = All SDL actions
System  Demongame  : 0 = No trace
Block   GameBlock  : 6 = All SDL actions + Result +
Parameters

Default    0 = GR trace off
System  Demongame 1 = Show next symbol when entering
monitor
```

シンボルごとの実行

Demonプロセスの情報がテキスト エリアに表示されていないことを確認するために、実行可能な最小単位のSDLのシンボルごとに遷移を実行してみましょう。この実行には[Step-Symbol]コマンドを使用します。

1. 外部環境からNewgame信号を送信します。
2. [実行]モジュールの[シンボル]ボタンをクリックして、Mainプロセスのスタート シンボルに対して遷移を実行します。次のシンボルに対する処理は、まだ実行されていないので、テキスト トレースには次の状態であるGame_Offの情報は表示されません。

   ```
   *** TRANSITION START
   *       PId    : Main:1
   *       State  : start state
   *       Now    : 0.0000
   ```

3. [Symbol]ボタンで次のシンボルを実行します。テキスト トレースには、Game_Offについての情報が表示されます。

   ```
   *** NEXTSTATE  Game_Off
   ```

4. Demonプロセス内で遷移の実行を継続します。Demonのスタート遷移から、3番目のシンボルまで実行します。DemonプロセスはGameBlockブロックの一部ではないので、テキスト エリアには何もトレースされません。
5. MainプロセスとGameプロセスでシンボル単位での実行を繰り返し、再びDemonプロセスに戻るまで実行します。この操作を行うには、SLDエディタ ウィンドウで実行されている遷移の位置を確認しながら、[シンボル]ボタンを何度も押し続け、SDLエディタ ウィンドウで、実行されている遷移の位置を確認します。個々のシンボルに対するトレース結果が表示されることに注意してください。
6. Demonプロセスに戻ったら、シンボル単位での実行を継続するか、[遷移]ボタンを使って通常のやり方で状態遷移を行います。この間、トレースは表示されません。
7. Gameプロセスに戻ったら、遷移の実行を終了します。

興味のない状態遷移を隠ぺいする

現在の設定で状態遷移の実行を継続すると、Gameプロセスで遷移が実行される時にだけトレースが表示されます。しかし、Demonプロセスにおいても、トレースに「表示されない」遷移を手動で実行しなければなりません。これを避けるには、Next-Visible-Transitionコマンドを使用します。このコマンドによってトレースの表示されないプロセスの遷移は連続して実行されます。そして、トレース レベルが0以上のプロセスに到達したら実行が停止します。つまり、トレースが「表示される」最初のプロセスで実行が停止します。このように、システム内の見る必要のない部分の遷移を完全に隠ぺいすることができます。

1. [実行]メニューで[（トレース値）>0まで]を選択してコマンドを実行します。実行はGameプロセス内で、WinningまたはLosing状態に到達するまで停止しません。そして、最後に実行された遷移に対するトレースが表示されます。

2. [（トレース値）>0まで]の実行を何回か繰り返します。GameプロセスがLosingとWinningの間で遷移を繰り返していることがトレースに表示されます。そして、Demonプロセスの実行状態はトレースに表示されません。

   なお、GRトレースでは、Demonプロセスのみが表示されます。これは、GRトレースが次に実行されるシンボルを選択するためです。GameプロセスがWinningかLosingの状態にある時は、次に実行されるシンボルはDemonプロセスになります。Gameプロセスの現在の状態を確認するには、次の操作を行います。

3. [表示2]メニューで[1つ前のシンボル]をクリックし、最後に実行されたシンボルを表示させます。GameプロセスのWinningまたはLosingの状態シンボルが選択されるはずです。

外部から見た振る舞いの表示

学習内容

- 信号のログ機能の使用
- Simulator UIへのコマンド ボタンの追加
- 所定の時間までの遷移の実行

トレースの設定と信号のログの記録

この演習では、システムを外から見た振る舞いをトレースすることになるでしょう。この結果は、人がDemonGameで実際に遊ぶ際に観察される振る舞いを示すはずです。これを見るには、システム トレース レベルを「1」に設定します。つまり、外部へ送信される信号のみをトレースし、システム内部の状態遷移は表示しません。また、ファイルにこの振る舞いのトレースを保存するために、*signal log*コマンドを使用します。

1. シミュレータを再起動します。
2. システムのトレースの値を「1」に設定します。
3. 外部環境から送信される信号や、外部環境へ送信する信号を記録するために、入力行に「`signal-log`」と入力して、`Enter`キーを押します（このコマンドに対応するボタンやメニュー項目はありません）。

   Signal-logコマンドは、ダイアログ ボックスに表示されている2つのパラメータを取ります。最初のパラメータはユニット名です。指定したユニットへ送信される信号やユニットから送信される信号のすべてが、ファイルに記録されます。
4. 一覧からユニット名を選択する代わりに、ダイアログ ボックスのテキスト フィールドに「env」と入力し、［OK］をクリックします。この値により、システムの外部環境をユニットとして指定することができます。

図105: 外部環境の指定

2番目のパラメータは、ファイル名であり、信号に関する情報を書き込むファイル名を設定します。ファイル選択ダイアログ ボックスが開きます。

5. ［ファイル］フィールドに「signal.log」というファイル名を入力し、[OK]をクリックします。

使用頻度の高いコマンドのボタンを追加

DemonGameで遊ぶ際は、外部環境からシステムへ信号を送信します。最初にNewgame信号を送信してゲームを開始します。この操作は、このシステムでシミュレーションを実行する際に頻繁に行うので、必要なコマンドを実行する新しいボタンを追加します。ボタンを追加すれば、信号の送信はボタンのクリックのみで済みます。

1. [シグナル送信]モジュールの右側にある[Group]メニューから、[追加]を選択します。

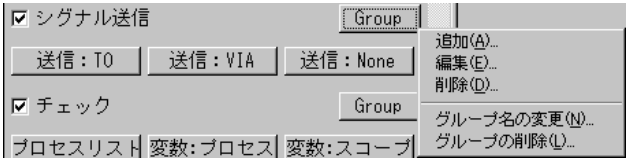


図106: 新しいボタンをモジュールに追加

2. ダイアログ ボックスの、[ラベル]フィールドに「Newgame」と入力します。まだ[OK]キーは押さないでください。Definitionフィールドに「output-via Newgame -」と入力します。

図107: ボタンの追加

   – Output-Viaパラメータの末尾の「-」は、デフォルトの値であることを示します。この場合は、チャネル パラメータに対するデフォルト値になります。「-」によって、パラメータのダイアログ ボックスで[OK]をクリックしたことと同じ操作を指定できます。

3. [適用]をクリックします。新しいボタンがモジュールに表示され、ダイアログ ボックスは他のボタンを設定できる状態になります。

4. Probe信号を送信するために、同じ手順で[Probe]ボタンを追加します。

5. 必要ならば、［Result］と［Endgame］ボタンを同様の方法で追加します。最後に、［OK］ボタンでダイアログ ボックスを閉じます。
［ボタンのラベルは空にはできません］というエラー メッセージが表示された場合には、無視してください。

ゲームの実行

ゲームを始める準備が整いました。新しいボタンを使用してゲームに信号を送信します。**Proceed-Until**コマンドを使用して、次に信号を送信する時間まで、遷移の実行を継続させます。

1. [Newgame]ボタンをクリックして、Newgame信号を送信します。

2. [実行]メニューの[指定時刻まで]を使って、遷移を実行する時間を5.5に設定しましょう。[指定時刻まで]の実行時に表示されるダイアログ ボックスに「5.5」と入力します。この操作によって、`Proceed-Until 5.5`コマンドが実行されます。シミュレーション時間が、指定された時間の値と等しくなるまで、すべての遷移が実行されます。

3. 同様にProbe信号を送信します。

4. 時間が10.3になるまで遷移を実行します。Lose信号またはWin信号が外部環境に出力されていることを確認してください。

5. 再度Probe信号を送信します。さらにもう一度Probe信号を送信します。この2つの信号はGameプロセスの入力ポートに入力されます。[チェック]メニューの[入力ポート]を選択してこれを確認します。

```
Input port of Game:1
Entry   Signal name          Sender
*1      Probe                env:1
 2      Probe                env:1
```

6. 同様にResult信号を送信します。Resultボタンをあらかじめ定義している場合は、これを使用します。あるいは、[送信：VIA]ボタンを使用するか、入力行にコマンドを入力することもできます。

7. 時間が13.5になるまで遷移を実行します。各Probe信号に対応してLose信号またはWin信号が外部環境に出力され、さらにScore信号が出力されていることを確認します。

信号ログ ファイルの確認

1. [実行]メニューの[中止]を選択してシミュレーションを終了します。この操作は、信号の記録を終了するために必要です。シミュレータUI自体はこのコマンドでは閉じません。

2. シミュレータUIの外部で、`signal.log`ファイルを参照します。以下のようにファイルには、外部環境から送信された信号と、外部環境へ送信した信号のすべての情報が記録されています。

```
Signal log for system Demongame with unit Process
env on file ...
0.0000 Newgame from env:1 to Main:1
5,5000 Probe from env:1 to Game:1
5.5000 Win from Game:1 to env:1
10.3000 Probe from env:1 to Game:1
10.3000 Probe from env:1 to Game:1
10.3000 Result from env:1 to Game:1
10.3000 Lose from Game:1 to env:1
10.3000 Lose from Game:1 to env:1
10.3000 Score from Game:1 to env:1
Parameter(s) : -1
```

ブレークポイントの使用

この演習では、シミュレータの機能の1つである*ブレークポイント*について説明します。ブレークポイントによって遷移の実行を停止し、指定したポイントでモニタ システムを作動させることができます。ブレークの条件としては、シンボル、状態遷移、出力、および変数の4つが使用できます。ここでは最初の2つを説明します。

学習内容

- SDLシンボルに対するブレークポイントの設定
- 特定の遷移に対するブレークポイントの設定
- 定義されているすべてのブレークポイントの一覧表示
- 実行された各SDLシンボルのグラフィカル トレース
- SDLシンボルに対するテクスチュアルなSDT参照の取得
- 継続的なシステムの実行

システムのセットアップ

ブレークポイントで停止した状態を見るため、まずGoコマンドを使ってシステムの遷移を実行します。Goコマンドは、エラーが発生するか、ブレークポイントに達するか、またはシステムが完全にアイドル状態になるまで、状態遷移を連続的に実行します。したがって、システムを継続的に実行できるように設定することが必要です。

1. シミュレータを再起動します。
2. システムのトレース情報が、実行中に記録されないように、システムのトレース レベルを「0」に設定します（[Trace]メニューの[Text level : Set]を使います)。
3. システムのGRトレース レベルを「2」に設定します。遷移が実行されると、SDLエディタ内でSDL シンボルが選択されます。この設定によって、テキスト トレースを表示しない場合も、シミュレーションの実行を再開することが可能になります。

シンボル ブレークポイントの設定

*シンボル ブレークポイント*によって、プロセス ダイアグラム内の特定のSDLシンボルにブレークポイントを設定することができます。シンボル ブレークポイントは、シンボルが実行される前にチェックされます。つまり、ブレークポイントに到達した時点では、まだシンボルは実行されていません。ここではGameプロセスの最初のタスク シンボルすなわち「Count:=0」というタスク シンボルにブレークポイントを設定します。

1. まず、追加したボタンを使ってNewgame信号を送信します。この操作は重要です。この操作を行なわないとGameプロセスが生成されず、ブレークポイントに到達できません。

2. SDLエディタで、[ダイアグラム]メニューを使ってGameプロセス ダイアグラムを開きます。

   – Gameプロセス ダイアグラムがメニューに表示されていない場合は、Gameプロセス ダイアグラムを明示的に開きます。[開く]クイック ボタンをクリックして`Game.spr`ファイルを選ぶか、オーガナイザ ウィンドウのGameプロセス ダイアグラムのアイコンをダブルクリックします。

3. シミュレータUIで、[ブレークポイント]メニューの[SDLエディタと連動]を選択します（必要に応じて、メニューを表示するようにウィンドウのサイズを変更します）。シミュレータとSDLエディタが関連付けられます。新たに[Breakpoints]メニューがSDLエディタのメニュー バーに表示されます（[Breakpoints]メニューが見えない場合は、必要に応じて、ウィンドウの横幅を変更します）。

4. SDLダイアグラムに戻り、「Count:=0」というタスク シンボルを選択します。次に、SDL エディタの新しい[Breakpoints]メニューから2番目のコマンド、[Set Breakpoint]を選択します（コマンド名の後に点線がないコマンドを選択します）。シンボル ブレークポイントが設定されます。

   ブレークポイントが設定されたことを示す赤い「stop」のマークがタスク シンボルに追加されます。シミュレータUIに戻ると、シンボル ブレークポイントの設定が表示されています。

5. [実行]モジュールの[実行]ボタンを押して、システムを実行します。実行が開始されると対応するSDLシンボルが連続的に選択されます。最終的に、ブレークポイントに到達し、実行が停止します。この時点で、ブレークポイントが設定されたシンボルは、次に実行されるシンボルに相当します。

状態遷移ブレークポイントの設定

*状態遷移ブレークポイント*によって、システムの特定の遷移にブレークポイントを設定できます。状態遷移ブレークポイントは遷移が実行される前に、チェックされます。つまり、ブレークポイントに到達した時点では、その遷移は実行されていません。ここでは、DemonプロセスがGenerate状態において、Tタイマを受信した場合の遷移にブレークポイントを設定します。

1. ブレークポイントを定義するために、[ブレークポイント]メニューの[遷移]を選択します。このコマンドは、表示されるダイアログ ボックスにおいて、多くのパラメータを設定します。

2. Demonプロセスを選び、[OK]をクリックします。

図108: プロセスの指定

3. [Instance number]のフィールドに何も入力せずに、[OK]をクリックします。

図109: [Instance number]のフィールドを空欄のままにする

4. [Service name]にも何も入力せずに、[OK]をクリックします。

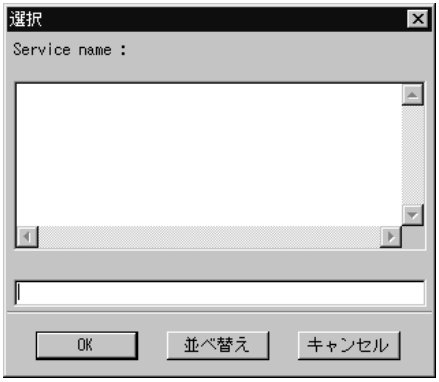


図110: [Service name]を空欄のままにする

5. Generate状態を選択して、[OK]をクリックします。

図111: Generate状態を指定

6. Tタイマを選択して、[OK]をクリックします。

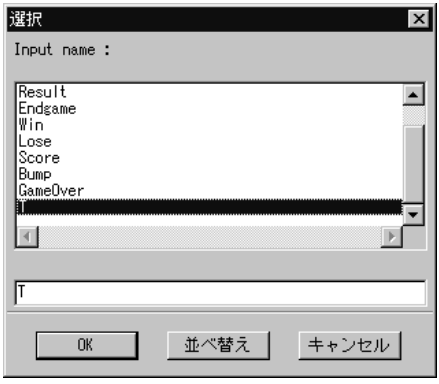


図112: Tタイマを指定

7. 後続のダイアログ ボックスでは、すべて[OK]をクリックします。

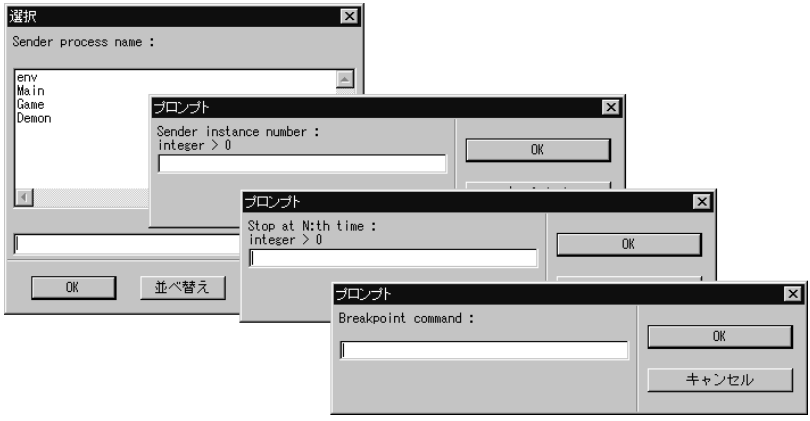


図113: 後続のダイアログ ボックスを空欄のままにする

パラメータの値を設定しない場合は、名前や番号など、省略したパラメータでは、全ての値がブレークの対象となります。ここでは、Demonのあらゆるインスタンスや送信元がブレークポイントの対象として処理されます。

8. 新しいブレークポイントの定義内容を確認するには、[ブレークポイント]メニューの[ブレークポイントの表示]ボタンですべてのブレークポイントを一覧表示させます。

```
1
Process name     : Demon
Instance         : any
Service name     : any
State  : Generate
Input  : T
Sender name      : any
Sender instance  : any
Stop each time
```

9. [実行]ボタンをクリックして、遷移の実行を再開します。ブレークポイントに到達した際、現在のシステム状態がブレークポイントの定義と合っていることを確認できます。

```
Breakpoint matched by transition
PId    : Demon:1
State  : Generate
Input  : T
Sender : Demon:1
Now    : 1.0000
```

システムの変更

シミュレータのモニタには、システムの振る舞いを変更する多くのコマンドがあります。これらのコマンドを実行した後にシミュレートされるシステムは、元のシステムとは異なるので、コマンドを使用する際には注意が必要です。これらのコマンドは、特にデバッグの作業において有益です。なぜなら、デバッグの作業時に簡単な変更を加えることができるため、エラーが検出された時にもシミュレーションを継続することができます。また、コマンドを使用して、システムを強制的に、特定の条件下に置いてシミュレーションすることもできます。これらのコマンドを利用しない場合は、条件を再現するために多くの遷移の実行が必要になってしまいます。

学習内容

- [コマンド]ウィンドウでのコマンドの変更と追加
- プロセスを手動で生成
- プロセス スコープの変更
- プロセス変数の値の変更
- プロセスの状態の変更
- 手動によるタイマのセットとリセット

各種準備

この後の演習では、プロセスとタイマを変更します。シミュレーションを再開する前に、シミュレーションとシミュレータUIの設定を変更します。

1. シミュレータを再起動します。
2. [表示]メニューから[コマンド]ウィンドウを開きます。[コマンド]ウィンドウで、実行されるコマンドを変更します。List-Processコマンドを Examine-PIdに変更し、さらに新しいコマンドとしてList-Timerを追加します。
3. [List-Process]コマンド モジュールの右端の[コマンド]メニューから[編集]を選択します（このメニューはボタンモジュールの[Group]メニューと同じ方法で操作できます)。
4. ダイアログ ボックスで、`List-Process`コマンドを「`examine-PId`」に変更して、[OK]をクリックします。
5. [コマンド]ウィンドウのメニュー バーに移り、[コマンド]メニューから[コマンドの追加]を選択します。ダイアログ ボックスに、「`list-timer`」と入力し、[OK]をクリックします。
6. 新しいコマンドモジュールがウィンドウに追加されます。3つのモジュールが見えるように、ウィンドウのサイズを変更します。
   – モジュールの[コマンド]メニューから[サイズ]を選択して、モジュールに表示するテキスト行を減らすこともできます。ダイアログ ボックスでスライダーを使って設定します。

図114: テキストの行を設定する

7. すべて情報を参照できるように、システムのテクスチュアルなトレースの値を6に設定します。
8. [遷移]ボタンを2回クリックしてMainプロセスとDemonプロセスの最初の遷移を実行させます。
9. SDLエディタ ウィンドウのGRトレースが邪魔で、[コマンド]ウィンドウの出力表示が見えにくい場合は、GRトレースを「0」に設定します。

プロセスの生成

この演習では、Newgame信号を受信した後の状態にシステムを設定します。この設定は、実際にNewgame信号を送信しなくても可能です。代わりに、Createコマンドを使用して、Gameプロセス タイプのインスタンスを手動で生成します。

1. [変更]メニューの[プロセスの生成]ボタンで、Gameプロセスを生成します。Gameプロセスを選択して[OK]をクリックします。

2. 次のダイアログ ボックスで親プロセスとして[Main]を選択して[OK]をクリックします。この操作によって、2つのプロセス インスタンス間にParent-Offspringのリンクが設定されます。

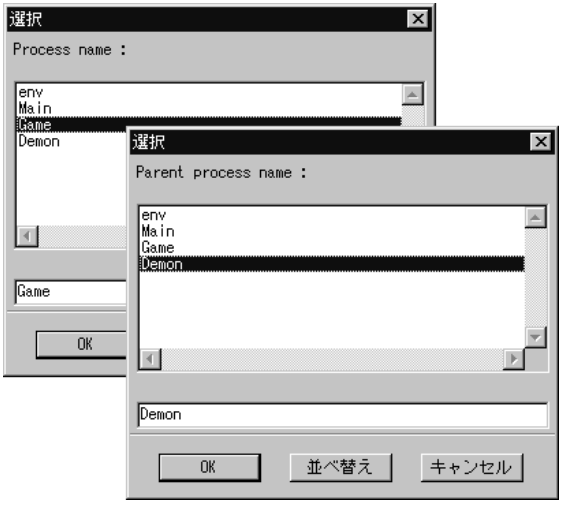


図115: Mainプロセスから Gameプロセスを作成

Newgame信号を受信したときの状態を完成させるには、Mainプロセスで GameP変数に値を設定し、MainプロセスをGame_On状態に設定する必要があります。これらの設定は、Assign-ValueコマンドとNextstateコマンドによって可能です。ただし、これらのコマンドは次の実行プロセスに対して適用され、ここではMainではなくGameプロセスがそれに相当します。これは、[コマンド]ウィンドウの[List-Ready-Queue]で確認できます。

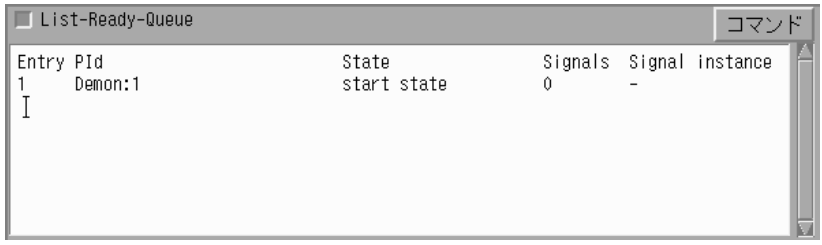


図116: 待ち行列

次に実行されるプロセスはGameプロセスです。

したがって、まず操作対象となるカレント プロセスを変更しなければなりません。カレント プロセスは、*process scope*とも呼ばれます。

3. [Examine]メニューの[Set Scope]を選択し、スコープを変更します。ダイアログ ボックスでMainプロセスを選択して、[OK]をクリックします。

   [コマンド]ウィンドウの[Examine-PId]には、Mainが現在のプロセスであることが表示されます。変数GamePには、MainのOffspringの値を設定しなければならないので、[コマンド]ウィンドウで、この値が`Game:1`であることを確認します。

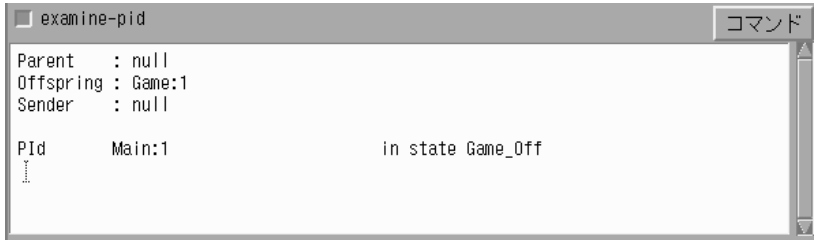


図117: カレント プロセスであるMainプロセス

現在のプロセスのOffspringはGame:1です。

4. [Change]メニューの[Variable]を選択し、GameP変数に値を割り当てます。表示されたダイアログ ボックスでGamePを選び、PId値に「`Game:1`」と入力します。
5. ［Change］メニューの［State］を選択して、Mainプロセスを正しい状態に設定します。Game_On状態を選択し、［OK］をクリックします。

   これで、システムは、Newgame信号を受信した後の状態に設定されました。プロセス スコープを変更しましたが、次に実行される遷移はGameの最初の遷移になっています（この状態はプロセス待ち行列で確認できます。171ページの図116を参照してください）。

6. 次の遷移を実行し、Gameプロセスが実行されることを確認します。

タイマの状態を変更

この演習では、モニタ システムで直接タイマのセットとリセットを操作します。Tタイマは、Demonプロセスによってセットされたので、現在Tタイマはアクティブになっています。この状態は[コマンド]ウィンドウのList-Timerモジュールで確認できます。

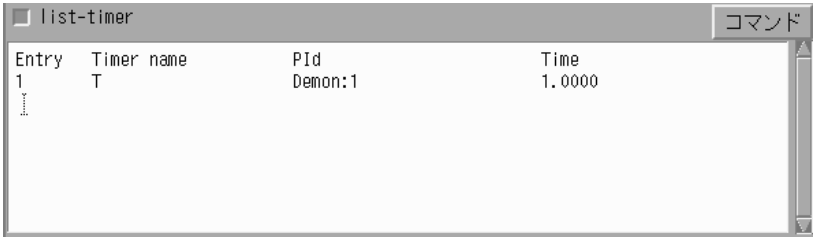


図118: Tタイマは現在アクティブ

1. [変更]メニューの[タイマーのリセット]で、タイマをリセットします。Tタイマを選び、[OK]をクリックします。

2. 次の遷移を実行し、以下のメッセージが表示されることを確認します。

   ```
   No process instance scheduled for a transition
   ```

   システムは現在、完全なアイドル状態です。つまり、システムには実行できる状態遷移が存在しません。[コマンド]ウィンドウを参照すると、待ち行列とタイマ行列に何もないことがわかります。システムを再起動するためには、DemonプロセスでTタイマをセットすることが必要です。

3. [変更]メニューで[タイマーのセット]をクリックし、Tタイマを選択します。時間の値に「10」を入力します。

4. 次の遷移を実行し、タイマが「10」に設定されていることを確認します（メイン ウィンドウのトレースを参照します)。

   ```
   *** TIMER signal was sent
   *   Timer    : T
   *   Receiver : Demon:1
   *** Now      : 10.0000
   ```

メッセージ シーケンス チャートの生成

この演習では、メッセージ シーケンス チャートの機能を使用します。メッセージ シーケンス チャートによって、システムの動的な振る舞いがグラフィック表示されます。この表示は、システムのシミュレーションの際に*MSC トレース*を使用することにより簡単に実行できます。MSCトレースは、信号の送信やプロセスの動的な作成など、一部のSDLイベントを、MSCイベントに変換します。MSCトレースは、シミュレーションの実行に伴い、MSCエディタ上にグラフィカルに記録されます。

以前の演習では、簡単なメッセージのシーケンスを記述したメッセージ シーケンス チャートを作成しました。ここではシミュレータを動作させて、実際に発生するイベントのMSCトレースを作成します。

学習内容

- MSCイベント ログの取得開始と終了
- MSCからSDLダイアグラムへのトレース バック

MSCトレースの初期設定

1. シミュレーションを再実行します。
2. SDLエディタ ウィンドウの更新や起動を避けるために、GRトレースが無効になっていることを確認します（[トレース]メニューの[SDL：表示]を選択し、システムのGRトレースが0になっていることを確認します）。

   デフォルトでは、MSCトレースは、システムに全体に対して有効になっています。ただし、以下の手順で、MSCエディタを明示的に起動しなければなりません。
3. [トレース]メニューの[MSCトレース：開始]を選択します。ダイアログ ボックスが表示され、MSCトレースに含める情報量の指定を促されます。ここでは状態をMSCトレースに記述させるので、「1」を選択して[OK]をクリックします。

図119: MSCトレースに含める状態を指定

4. MSCエディタが起動し、「SimulatorTrace」という名前のMSCダイアグラムが表示されます。任意にウィンドウの移動や、サイズ変更を行いシミュレータUIと共に表示できるようにします。

– MSCエディタのサイズを変更する前に、クイック ボタンで、シンボル メニューとテキスト ウィンドウを閉じることもできます。ここではMSCの編集は行わないので、これらのウィンドウは使用しません。

初期の状態で、システムには、3つのアクティブ インスタンスがあります。これらはプロセスを表す`Main_1_1`と`Demon_1_2`、および外部環境をシステムで表示するために導入した`env_0`です。

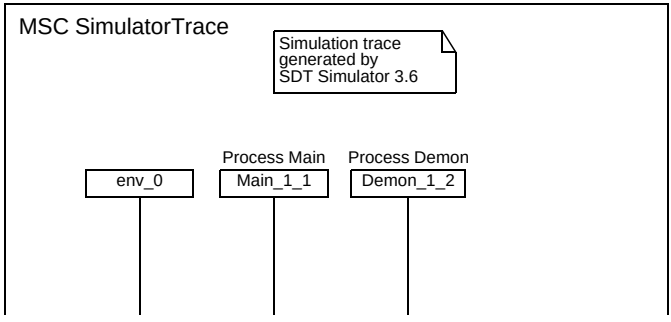


図120: MSCの初期状態

MSCにおける実行のトレース

1. 外部環境からNewgame信号を送信します。env_0インスタンスから送信されたメッセージとしてMSCエディタに表示されます。

   この時点では、メッセージはまだ消費されていないためMain_1_1インスタンスに接続されています。そして、不明なメッセージとして黒い点で一時的に表示されます。メッセージを表す黒い点には「Main_1_1」というテキストが関連付けられ、メッセージの送信先が表示されます。

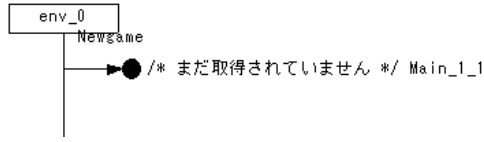


図121: 送信されたNewgame信号

2. 次の遷移を実行します。Mainプロセスのインスタンス軸上にGame_Offと表示された条件シンボルが現れます。このシンボルは、プロセスが実行を開始したことと、対応するSDLの状態がGame_Offに達したことを示しています。

図122: 条件シンボル

3. 次の遷移を実行します。TタイマがDemonプロセスでセットされます。MSCの表示画面が、あるいは下にスクロールし、イベントの順序関係を表すMSC図が展開されます。また、インスタンスの軸にGenerateと表示された条件シンボルが現れます。

4. 次の遷移をシンボル単位で実行します（[シンボル]ボタンを使用します）。MainプロセスはNewgameメッセージを取得します。黒い点が消えることでこの信号が受信されたことを確認できます。メッセージが送信された後にTタイマが設定されたので、メッセージの始点と終点が異なる高さに位置しています。

5. 次のシンボルを実行します。Gameプロセスのインスタンスが生成されます。新しいインスタンス ヘッドとインスタンスの軸が追加されます。MSCの現在の表示は以下のようになっているはずです。

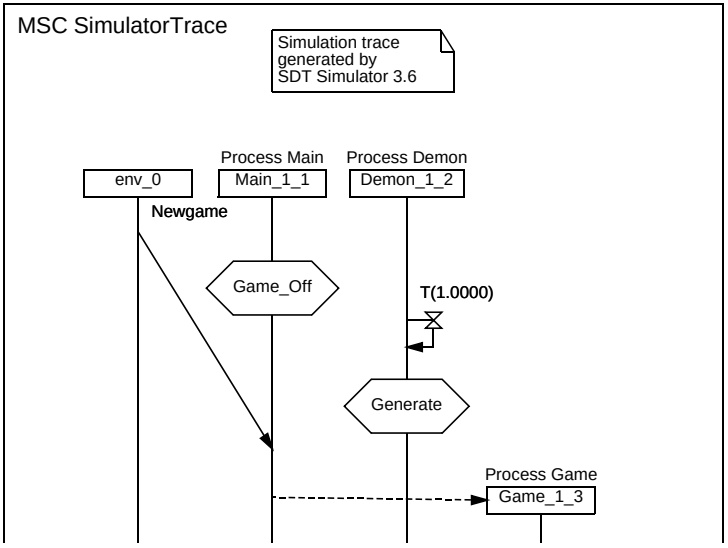


図123: Gameプロセスを生成した後のMSC

6. ［遷移］ボタンを2回クリックして、次の遷移を実行します。2回目の遷移の実行によって、GameプロセスはLosing状態になります。

7. 3回遷移を実行します。タイマ信号Tが取得され、Bump信号が送信および取得されます。Gameプロセスは現在Winning状態にあります。MSCで信号のやり取りがどのように表示されるかを確認します。

8. 次に、すぐに消費されるメッセージがどのように表示されるかを確認します。外部環境からProbe信号を送り、次の遷移を実行します。まず、Probeメッセージが表示されて(黒い点が付加されています)、次に、メッセージのラインが水平に引かれます。

9. 現在の遷移を完了させます。システムはWin信号で応答します。

10. Result信号を送信し、次の遷移を実行します。MSCで、Scoreメッセージのパラメータが1であることが確認できます。

11. Endgame信号を送信してゲームを終了します。次に、2回遷移を実行します。Gameプロセスが停止します。

MSCの現在の表示は次の図のようになっているはずです。MSCダイアグラムに表示されている水平の点線は、印刷した時にページが変わる境界線を示しています。

完成したダイアグラムと119ページの図90を比較します。手書きのダイアグラムと、生成されたダイアグラムが異なることに注目してください。SDLダイアグラムを見ただけでは、システムの動的な振る舞いを予測することは困難なので、このような相違が生じることがあります。

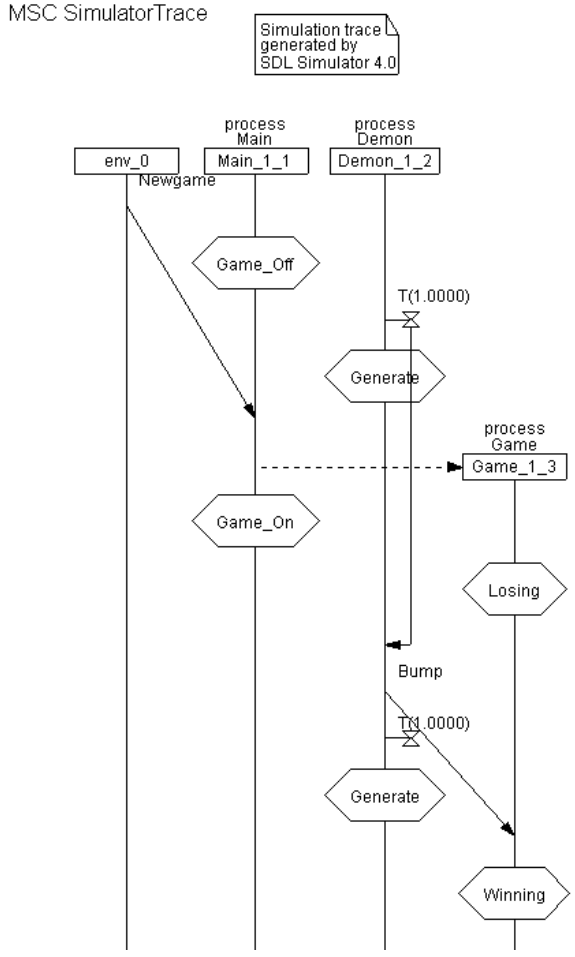


Figure 124: 完成した MSC 1 (2)

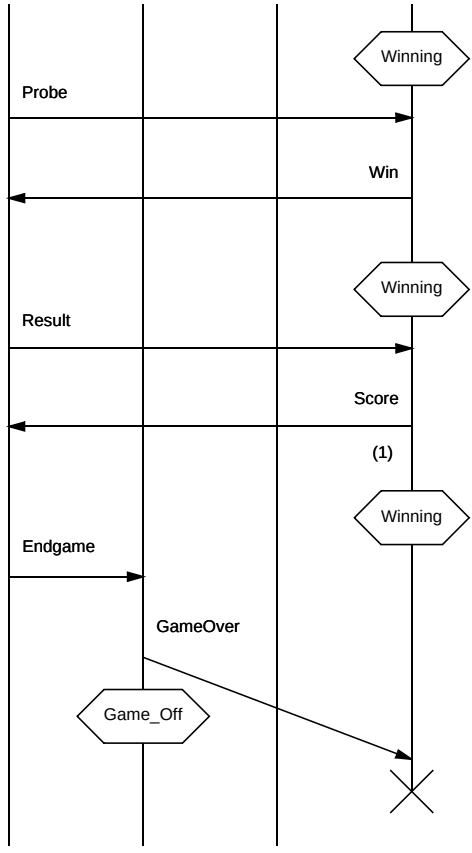


図125: 完成したMSC 2 (2)

SDLダイアグラムへのトレース バック

生成されたMSCダイアグラムから、SDLソース ダイアグラムへトレース バックすることができます。

1. MSCエディタの[ウィンドウ]メニューから、[情報ウィンドウ]を選択します。MSCエディタ上で現在選択しているオブジェクトに関する情報を表示したウィンドウが現れます（情報の数と種類は、選択しているオブジェクトによって異なります)。

図126: 情報ウィンドウ

ウィンドウは、Bumpメッセージに関連した情報を表示します。

2. MSCエディタでいくつかオブジェクトを選択し、[情報]ウィンドウの内容がどのように変わるかを確認します。

3. Bumpメッセージを選択し、[情報]ウィンドウで[SDLシンボルの表示]をクリックします。Bump信号を送信または受信するシンボルが選択された、SDLエディタが開きます。

   – Bumpメッセージの上で終点よりも始点に近い場所をクリックすると、Bumpメッセージの始点が選択され、対応するSDLの出力シンボルが選択されます。

   – 逆に終点を選択すると対応するSDLの入力シンボルが選択されます。

MSCトレースの終了

1. [トレース]メニューの[MSCトレース：停止]エントリを選択して、MSCイベントログの記録を停止します。

2. MSCエディタで、生成したMSCダイアグラムを「`SimulatorTrace.msc`」というファイル名で保存します。

3. [終了]を選択して、MSCエディタを終了します。

カバレジ ビューワー

最後に、カバレジ ビューワーの使用方法を学びます。カバレジ ビューワーは、シミュレーションで行った状態遷移やシンボル実行に関して、システム内のどのくらいの部分がカバーされたかをグラフィカルに表示するツールです。このカバレジをチェックすることによって、システムのどの部分がこれまでのシミュレーションで実行されていないか確認することもできます。

学習内容

- カバレジ ファイルの生成
- カバレジ ビューワーの起動
- カバレジ ツリー情報の解釈と変更
- より詳細なカバレジ チャートの起動
- シミュレータUIの終了

カバレジ ファイルの起動

1. シミュレータを再起動します。
2. Newgame信号を送信します。GameプロセスがWinning状態に遷移したことがトレースによって表示されるまで、遷移を7回実行します。
3. Probe信号を送信し、次の遷移を実行します。

   ここまでシステムのどのくらいの部分を実行したか確認します。シミュレータUIからカバレジ ビューワーを起動すれば、現在のカバレジ情報が表示されます。
4. [表示2]メニューから[カバレジ]を選びます。カバレジ ビューワーのメインウィンドウが開きます。

カバレジ ビューワーの使用

1. [カバレジ ツリーの表示]が表示されている場合は、[ツリー モード]クイックボタンをクリックして[Transition Coverage Tree]に表示を変えます。

2. [Transition Coverage Tree]をすべて表示するために、[すべてのノード]クイック ボタンをクリックします。

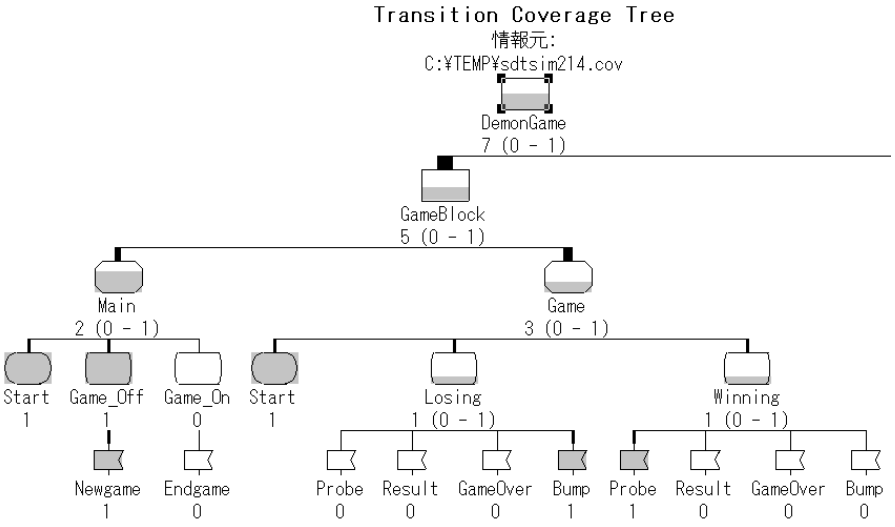


図127: 遷移カバレジ ツリー

ツリーの一部だけが表示されています。

– カバレジ ツリーはとても大きいので、カバレジ ビューワーを使用する際は、ウィンドウ マネージャーを使用してウィンドウのサイズを最大にすると良いでしょう。シミュレータUIに戻る時は、簡単にウィンドウのサイズを元に戻すことができます。

– または、ズーム アウトのクイック ボタンを何回かクリックして、ウィンドウの表示内容を縮小することもできます。

カバレジ ツリーはDemongameシステムのダイアグラム構造を反映しています。各プロセス ダイアグラム（Main、Gameおよび、Demon）の下には、そのプロセスで起こり得るすべての遷移が表示されます。表示される遷移は、状態や、状態から入力される可能性のある信号によって定義されたものです。スタート遷移は、SDLのスタート シンボルで表示されています。

個々のシンボルの下に示されている数は、これまでにシンボルが実行された回数を示しています。さらに、個々のシンボルは、灰色の表示によって実行の進行度が示されます。これまで一度も実行されなかった要素は、値が0でシンボルは白いままです。実際に実行された要素は、0以外の値が表示され、シンボルはグレー色で満たされます。

表示されたツリーから、以下の情報を読み取ることができます。

- Losing状態とWinning状態では、4つの遷移のうちで1つが実行されています。従って、これらの状態シンボルの1/4が灰色で満たされています。
- Mainプロセスでは、3つの遷移のうちで2つが実行されています。従って、プロセスシンボルは2/3がグレー色で満たされています。
- DemonプロセスおよびDemonBlockブロックでは、少なくともすべての遷移が最低1回実行されています。従って、これらのシンボルの全体が灰色で満たされています。
- システム全体では、起こり得る遷移の中の半数強が実行されています。

3. [最小]クイック ボタンをクリックすると、一度も実行されなかった遷移のみを表示することができます。システムの残りの部分を実行させるために、どの状態でどのような信号を送信するべきであるか確認することができます。
   - [最小]クイック ボタンの本来の機能は、シンボルの中で実行回数が最小のものを表示することです。点線で表示されたシンボルは構造全体を表示させるためだけの目的で表示されています。

4. 同様に、[最大]クイック ボタンをクリックすると、1回以上実行された遷移を表示することができます。システムでこれまでにどの信号が送信されているか参照できます。
   - [最大]クイック ボタンの本来の機能は、シンボルの中で実行回数が最大のものを表示することです。ただし、ここでは複数回実行された遷移は存在しません。

5. 再びツリーの全体構造を表示するために、[すべてのノード]クイック ボタンをクリックします。SDLエディタで遷移を参照する場合は、信号の入力シンボルかスタート状態シンボルの1つをダブルクリックします。この操作を実際に試してみてください。

カバレジ ビューワーでは、*シンボル カバレジ ツリー*を表示することもできます。つまり、プロセス ダイアグラムにおいて、各SDL シンボルが何回実行されたかを参照することができます。

6. [ツリー モード]クイック ボタンをクリックして、シンボル カバレジ ツリーへ表示を変更します。個々のプロセス ダイアグラムの下には、各SDLシンボルに対応する小さなアイコンが表示されます。小さなアイコンをダブルクリックすると、アイコンがどのSDLシンボルを示しているか確認できます。
7. 遷移カバレジ ツリーに表示を変更し、シミュレータUIに戻ります。

カバレジの増加

1. ゲーム プロセスを再びLosing状態にするために、3回遷移を実行します。
2. Result信号を送信します。Winning状態にするために、さらに4回遷移を実行します。
3. Endgame信号を送信します。ゲームプロセスを終了するために、さらに2回遷移を実行します。
4. [表示2]メニューから[カバレジ]を選択し、カバレジ ビューワーで現在のシステムのカバレジを確認します。
5. カバレジ ツリーに移り、[すべてのノード]コマンドですべてのノードを表示させます。Mainプロセスが完全に実行されていることが確認できます。LosingとWinning状態についての進行度を示すグレーの色が増えているはずです。
6. 実行されていない遷移を確認するために、[最小]クイック ボタンをクリックします。また、[最大]クイック ボタンをクリックした場合は、実行された遷移が表示されるのではなく、実行回数が最も多かった遷移やシンボルのみが表示されます。ここでは、Tタイマの入力になります。
7. より多くのツリーを表示させるために、[ツリー]メニューから[ツリーの増加]コマンドを選択します。Bumpの入力が追加されます。再度このコマンドを選択します。残ったすべての遷移が追加されるはずです。

カバレジの詳細を表示

1. カバレジ ビューワーに表示されているシステム ダイアグラムを選択します。[ツール]メニューから[詳細の表示]を選択するか、クイック ボタンをクリックします。[カバレジの詳細]ウィンドウが表示されます。

   表示されたカバレジ チャートには、特定の回数だけ実行された状態遷移の数が表示されます。チャートには、4つのバーが表示されます。

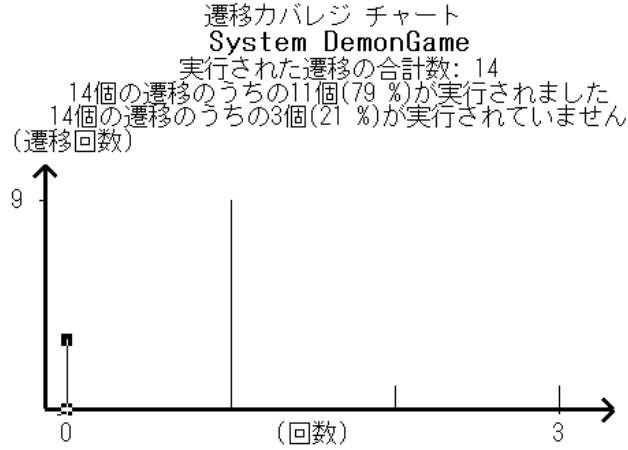


図128: [カバレジの詳細] ウィンドウ

2. バーを1つ選び、ウィンドウの下部のステータス バーを参照します。0回、1回、2回および3回の実行回数に対応する遷移の数を確認できます。以下の手順によって、実行されていない遷移を確認します。

3. 一番左の「0回を示すバー」を選択します。[エディタで表示]クイック ボタンをクリックします。SDLエディタが表示され、Gameプロセスにある、3つの実行されていない遷移が選択されます。

   – 1つ以上のダイアグラムに実行されていない遷移がある場合、ダイアグラムが開かれるたびに確認用ダイアログ ボックスが表示されます。確認用ダイアログ ボックスで[OK]をクリックすると、次のダイアグラムが表示され、実行されていない遷移を確認できます。

4. メイン ウィンドウのカバレジ ツリーから別のシンボルを選択します。[カバレジの詳細]ウィンドウは、選択したシンボルに対するカバレジ チャートを表示するために更新されます。

5. 任意にカバレジ ビューワーの機能を使用してみてください。ただし、カバレジ ビューワーの機能を試すには、Demongameシステムは簡単すぎるかもしれません。

シミュレータUIの終了

シミュレータUIを終了します。

1. [ファイル]メニューから[終了]を選択して、カバレジ ビューワーを終了します。
2. 次に、[ファイル]メニューから[終了]を選択して、シミュレータUIを終了します。ダイアログ ボックスが表示され、[ウォッチ]ウィンドウで変更した変数と、コマンド ウィンドウで実行したコマンド、および、追加したボタンを保存するかどうか質問されます。ここで、(自動的に入力された) ファイル名で保存すると、次に同じディレクトリからシミュレータUIを起動させた時に、これらがデフォルトで設定されます。

まとめ

この章では、シミュレータを生成し、実行およびトレースすることによって、SDLシステムの振る舞いを確認する方法について学習しました。

SDTにエクスプローラ ツールが使用できる場合は、引き続きエクスプローラの演習を学習してください。エクスプローラの演習は第5章「チュートリアル: SDLエクスプローラ」から始まります。

ここまでの演習では、DemonGameシステムという簡単なサンプルを取り上げてきました。SDL Suiteについてさらに知識を深めるために、SDL-92の利点を示すさまざまな演習を学習することができます。SDL-92は、オブジェクト指向のデザイン方法論を導入する際に有利な言語です。この演習は第6章「チュートリアル: SDL-92のDemonGameへの適用」に記述されています。

第 5 章

チュートリアル: SDLエクスプローラ

SDLエクスプローラは、状態空間を探索する技法が取り入れられたSDLシステムの振る舞いを検査するためのツールです。この章では、DemonGameシステムを実際に動作させます。

このチュートリアルを正しく理解するには、第3章「チュートリアル: SDLエディタとアナライザ」と、第4章「チュートリアル: SDLシミュレータ」の演習を終了しておくことが必要です。

エクスプローラの使用方法を理解するために、このチュートリアルの内容をすべて読んでください。また、記載されている手順にしたがって、お使いのコンピュータ システムで実際に演習してください。

このチュートリアルの目的

このチュートリアルでは、SDL Suiteにおけるバリデーションの基本的な機能を学習します。バリデーションとは、SDLシステムの状態空間を強力な手法やツールによって検査することをいいます。これらの手法やツールは、事実上発生し得るすべてのランタイム エラーを検出します。このようなエラーは、通常のシミュレーションやデバッグ技法では検出が困難であったものです。

このチュートリアルは、SDL Suiteの機能を理解するためのガイダンスとなるように配慮されています。チュートリアルを読む際は、説明されている各演習をコンピュータで実際に操作して確認してください。

このチュートリアルでは、内容を容易に理解できるように比較的簡単な題材を選んでいます。ただしSDL言語の学習を目的としていないため、説明はすべてSDL言語に関する基礎知識があることを前提に記述されています。

このチュートリアルでエクスプローラを使用するためには、第3章「チュートリアル: SDLエディタとアナライザ」と第4章「チュートリアル: SDLシミュレータ」の演習を終了しておくことが必要です。

メモ：Cコンパイラ

SDLシステムをバリデーションするには、使用するコンピュータ システムにCコンパイラがインストールされていなければなりません。このチュートリアルを始めるまえに、どのようなCコンパイラが使えるかを確認しておいてください。

メモ：プラットフォーム間の相違点

この章のチュートリアルやそのほかの章のチュートリアルに関する記述は、UNIXとWindowsの両方のプラットフォームで実行可能な項目に対して1種類の手順のみを記述します。プラットフォームによって操作が異なる場合は、「UNIX」または「Windows」などの指示を記述します。このように、プラットフォームを特定した記述がある場合は、ご使用のプラットフォームに関する記述のみを参照してください。

通常は、両方のプラットフォームで画面に表示される情報が等しい場合、どちらか一方の画面表示のみを示します。したがって、図のレイアウトや概観は、SDL Suiteを使用環境で実行したときに表示される画像と異なることがあります。重要な部分がプラットフォーム間で異なる場合に限って、それぞれの画面を示します。重要な部分がプラットフォーム間で異なる場合に限って、それぞれの画面表示を示します。

エクスプローラの生成と起動

SDLエクスプローラを使用すれば、システムをシミュレートするだけでなく、検査することが可能です。すなわち、システムのエラーや不整合を自動的に検出したり、要求条件と比較してシステムを検証することができます。

シミュレータと同様に、まずエクスプローラの実行形式ファイルを生成します。次に、適切なユーザー インターフェイスを使用してエクスプローラを起動します。

メモ：

この章の説明通りに動作するエクスプローラ実行形式ファイルを生成するには、作成したダイアグラムではなく製品に収められているSDLダイアグラムを使います。製品に収められているSDLダイアグラムは、以下の方法で使用できるようにします。

- UNIX の場合は、`$telelogic/sdt/examples/demongame`にあるすべてのファイルを作業用ディレクトリ`~/demongame`にコピーします。
- Windowsの場合は、`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥sdt¥examples¥de-mongame`にあるすべてのファイルを作業用ディレクトリ`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥work¥demongame`にコピーします。

これまでに作成したダイアグラムを利用する場合は、バリデーションの実行手順が多少異なります。

製品に収められたダイアグラムを使う場合は、オーガナイザで`demongame.sdt`システム ファイルを再度開くことが必要です。

学習内容

- エクスプローラ実行形式ファイルの迅速な生成と起動

エクスプローラの迅速な起動

エクスプローラの生成および起動には、以前に演習したシミュレータの生成、起動方法と、同じ方法で行うことができます。つまり、オーガナイザの[実装]ダイアログ ボックスと[ツール]メニューを使用してエクスプローラを生成し、起動することができます。ただし、ここではより簡単な方法を紹介します。

1. オーガナイザで、システム ダイアグラム アイコンが選択されていることを確認します。
2. 
[バリデート]クイック ボタンをクリックします。以下のような一連の処理が実行されます。
   - エクスプローラの実行可能ファイルが生成されます。シミュレータを生成したときと同様に、メッセージの最後に「分析が完了しました」とステータス バーに表示されます。この操作は、[実装]ダイアログ ボックスを手動で起動して、エクスプローラ カーネルを選択する操作と等価になります。オーガナイザ ログの末部を参照すれば、エクスプローラ カーネルが使用されたことを確認できます。
   - エクスプローラのグラフィカル ユーザー インターフェイスが起動されます。オーガナイザのステータス バーに「エクスプローラUIが起動しました」と表示されます。この操作は、[ツール]メニューから[エクスプローラUI]を手動で選択する操作と等価になります。
   - 生成されたエクスプローラが起動されます。エクスプローラUIに「Welcome to SDL EXPLORER」というメッセージが表示されます。この操作は、[開く]クイック ボタンで、エクスプローラの実行可能ファイルを選択する操作と等価になります。UNIXの場合のファイル名は`demongame_xxx.val`となり、Windowsの場合は`demongame_xxx.exe`となります。`_xxx`は、プラットフォーム、カーネルまたはコンパイラによって異なります。

メモ：

オーガナイザ ログ ウィンドウで[実装]を実行した際にエラーが発生した場合や、エクスプローラが起動しない場合は、以下の操作を行ってください。

- [実装]ダイアログ ボックスを開き、[gcc-バリデーション]や[Microsoft バリデーション]などの、コンピュータ システムで使っているCコンパイラに対応するバリデーション カーネルを選択します。
- [フル実装]ボタンをクリックして、エラーが発生していないことを確認します。
- [バリデート]クイック ボタンを再度クリックします。エクスプローラが上記のとおりに起動するはずです。

エクスプローラUIの図を以下に示します。

図129: エクスプローラUIのメイン ウィンドウ

エクスプローラのGUIは、前の演習で学んだシミュレータのGUIと非常に良く似ています。ただし、左側に表示されているボタン モジュールの内容が異なり、また、いくつかのメニューが追加されています。

エクスプローラには、シミュレータと同じタイプのモニタ システムがあり、利用可能なコマンド セットのみが異なります。

エクスプローラを起動すると、システムに静的なプロセス インスタンスが生成されますが（この場合MainとDemon）、インスタンスの初期遷移は実行されません。代わりにMainプロセスが実行されます。このMainプロセスは、以下の手順によってプロセスの待ち行列で確認することができます。

1. ウィンドウの左側にある[表示]ボタン モジュールの[レディキュー]ボタンをクリックします。待ち行列の最初のエントリにMainプロセスが表示され、スタート遷移の実行を待っていることがわかります。

   – [表示]モジュールにボタンが表示されていない場合は、モジュール名の左にあるトグル ボタンをクリックします。この操作によって[表示]モジュールが展開表示されます。各ボタン モジュールは、トグル ボタンによって展開または折りたたむことができます。

図130: ボタン モジュールの折りたたまれた状態

– [表示]モジュールのボタンは、シミュレータUIのボタンと同じタイプのコマンドを実行します。

2. 必要に応じて、ボタン モジュールをすべて展開できるようにエクスプローラUIウィンドウのサイズを変更します。また、次の演習では、複数のウィンドウを同時に使うので、テキスト エリアの幅を狭くすることも有効です。

エクスプローラの基本概念

エクスプローラの演習に入る前に、SDLエクスプローラの基本概念を理解する必要があります。

- エクスプローラを使ってSDLシステムを検査する場合、SDLシステムの動作は*動作ツリー*という構造でモデル化されます。このツリー構造内に表示されるノードは、SDLシステムの状態を表します。このように、システムの状態をまとめたものをシステムの*状態空間*と呼びます。
- 動作ツリー内を移動することで、SDLシステムの振る舞いを探索することができます。また、システムの各状態に移動するたびに、その状態を検査することができます。これを*状態空間の探索*と呼び、手動または自動で実行できます。
- 動作ツリーのサイズと構造は、エクスプローラのさまざまな*状態空間オプション*よって設定されます。これらの状態空間オプションは、SDLプロセスグラフ内の遷移に対して生成されるシステムの状態の数や、動作ツリー内の状態から分岐できる枝の数などに影響します。

動作ツリー内での移動

この演習では、動作ツリー内を手動で移動してDemongameシステムの状態空間を探索します。エクスプローラの動作は、シミュレータを実行した際の動作と似ています。しかし、以下で説明するような重要な相違点もあります。

エクスプローラのデフォルト設定は、できるだけ狭い状態空間を生成するように定義されています。この設定によって、動作ツリー内の2つの状態間の遷移は、常にSDLプロセス グラフの遷移と対応します。また、状態から分岐可能な枝の数は最小に抑えられています。

学習内容

- ナビゲータツールの使用
- テキスト トレースとGRトレースの表示

探索の設定

動作ツリーを手動で探索する場合、*ナビゲータ*というエクスプローラ ツールを使用します。

1. [検索]モジュールの[ナビゲータ]ボタンをクリックしてナビゲータを起動します。[Navigator]ウィンドウが開きます。

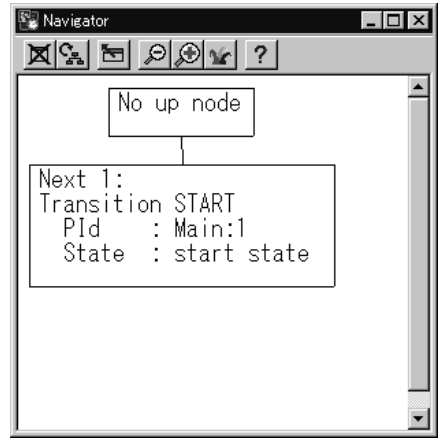


図131: ナビゲータツール

ナビゲータには、現在のシステムの状態やその近傍を表した動作ツリーの一部が表示されます。原則として、上側のボックスには、現在の状態に導いた動作ツリーの遷移が示されます。つまり、直前に実行された遷移が表示されます。下のボックスには、現在の状態から実行できるツリーの遷移が示されます。*Next 1*、*Next 2*.....などのラベルの付いた遷移は、まだ実行されていないことを表します。

システムは現在スタート状態にあるので、*上方向へのノード*はありません。唯一存在する下方向のノードはMainプロセスのスタート遷移です。

2. シミュレーションの場合と同様に、トレースを表示するには[表示]メニューから[コマンド]ウィンドウを開きます（トレースは、エクスプローラのメイン ウィンドウには表示されません）。

3. SDLシンボルのGRトレースを表示するには、エクスプローラ ウィンドウの[コマンド]メニューから[SDLトレースの開始/停止]を選択します。この時点でSDLトレースは有効となりますが、SDLエディタは最初の遷移が実行されるまで開かれません。

Navigatorの使用

1. [Navigator]で、*Next 1*ノードをダブルクリックすると次の遷移が実行されます。以下ような処理が随時発生します。

   – ナビゲータの*Up 1*ノードに、実行直後の遷移が表示され、*Next 1*ノードに次に実行可能な遷移が表示されます。ここでのNext 1ノードは、Demonプロセスのスタート遷移になります。

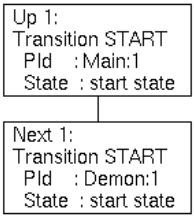


図132: 前の遷移と次の遷移

   – SDLエディタが開き、最後に実行されたシンボルが選択されます。シミュレータとの違いに注意してください。シミュレータの場合は、SDLエディタ内の次に実行されるシンボルが選択されていました。

– 実行された遷移（Mainプロセスのスタート遷移）のトレースが[コマンド]ウィンドウに表示されます（トレースを表示するために、ウィンドウをスクロールまたは縮小することが必要な場合もあります)。

図133: 実行された遷移のテキスト トレース

2. 必要に応じて、現在開いているすべてのウィンドウを画面上に整列表示させるために、ウィンドウの移動やサイズの変更を行います。

3. *Next 1*ノードをダブルクリックして次の遷移を実行します。Demonプロセスのスタート遷移がコマンドウィンドウとSDLエディタにトレースされます。

   現時点では、2つのアクティブ プロセスは信号を受信しない限り、遷移を続行できません。Mainプロセスは、外部環境から送信されるNewgame信号を待っており、DemonプロセスはTタイマからの信号送信を待っています。これらは、現在の状態から導き出される2つの遷移であり、ナビゲータに*Next 1*と*Next 2*として表示されています。遷移が表示されているボックスには、テキスト トレースと同じ情報が記述されています。

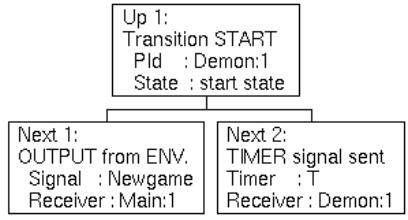


図134: ナビゲータに表示される遷移

これは、遷移を実際に実行しなくても、現在のシステム状態から実行できるすべての遷移が、エクスプローラによって表示されることを意味します（シミュレータでは、この情報を簡単に表示することができませんでした)。

4. *Next 2*ノードをダブルクリックして、タイマ信号を送信します。[コマンド]ウィンドウでタイマ信号が送信されたことを確認できます。また、ナビゲータで次の遷移がTタイマの取得であることを確認できます。

5. *Next 1*ノードをダブルクリックして次の遷移を実行します。シミュレータのチュートリアルで説明したように（第4章「チュートリアル: SDLシミュレータ」153ページの「動的エラー」を参照）、ここでDemongameシステムに動的エラーが発生します。ナビゲータは、次の遷移を表示する代りに、ボックスにエラー メッセージを表示します。

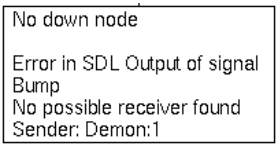


図135: 動的エラー

– [コマンド]ウィンドウの[Print-Trace]モジュールをスクロールすると、末尾にエラーメッセージを見つけることができます。

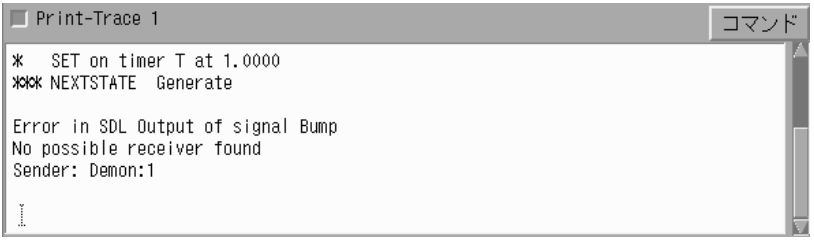


図136: [Print-Trace]モジュールの末尾

動作ツリーのこの分岐はこれ以上進めることができません。デフォルトでは、エラーが通知されると、その時点でツリーの探索が打ち切られてしまいます。代りに、Newgame信号の出力の遷移を選択できる位置まで状態を戻します。

6. *Up 1*ノードをダブルクリックして前の状態へ戻ります。この動作を繰り返して、上述の3項の状態へ戻ります。当然ながら、シミュレータの実行では遷移を後戻りすることはできませんでした。

   *Next 2*ノードには3つのアスタリスク「***」が付加されます。アスタリスクは、遷移を後戻りした回数を示します。

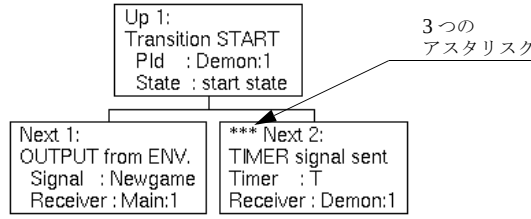


図137: 後戻りした遷移のマーク

7. 代わりに*Next 1*の次の遷移を実行します。Newgame信号が外部環境から送信されたことがテキスト トレースに表示されます。また、Mainプロセスは信号待ちの状態になっています。なお、信号は手動で送信する必要はありません。エクスプローラによって自動的に送信されます。

8. Next 1をダブルクリックして、次の遷移を実行します。テキスト トレースとSDLトレースは、MainプロセスがGame_On状態にあることを表示します。新しく生成されたGameプロセスのスタート遷移がナビゲータに表示されます。

9. Gameプロセスのスタート遷移を実行します。ナビゲータによって、遷移を継続するために必要な各入力信号が表示されます（Endgame、Probe、Result、およびTタイマ）。

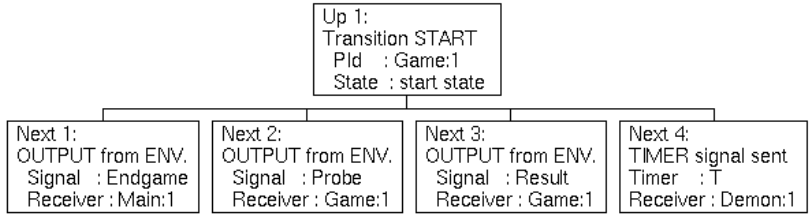


図138: 遷移の継続に必要な入力信号

ツリー構造を使用している際は、ある状態から多数の遷移が導き出されると、すべての遷移をナビゲータに表示できないことがあります。これを避けるために、リスト構造で遷移を表示することも可能です。

10. [ツリーの切り替え]クイック ボタンをクリックすると、以下のようなリスト構造に変わり、信号が見やすくなります。

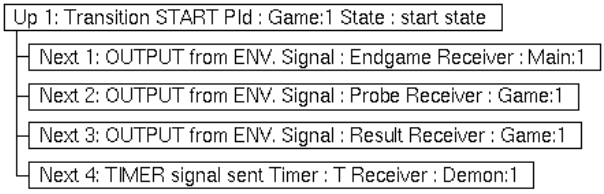


図139: リスト構造

11. ツリー構造に戻します。

    この演習では、これ以上動作ツリーを進めません。これまでに探索した動作ツリーの一部を199ページの図140に示します。図に記述されている各ノードは、SDLシステム全体の状態を示しています。各ノードには、前のシステムの状態から変化したアクティブ プロセスのインスタンスと、それらのインスタンスが存在するプロセスの状態、そして、それらの入力の待ち行列が記述されています。ノード間の矢印は実行可能なツリーの遷移を表しています。矢印には、番号と、遷移を可能にするSDLの動作が表示されています。矢印の番号は、ナビゲータの*Next*ノードに表示されていた番号と同じものです。

    図150の動作ツリーは、ナビゲータに表示されていたツリーと少し異なるはずです。ナビゲータでは、ノードはツリーの遷移を表し、プロセスの状態は表示されていませんでした。

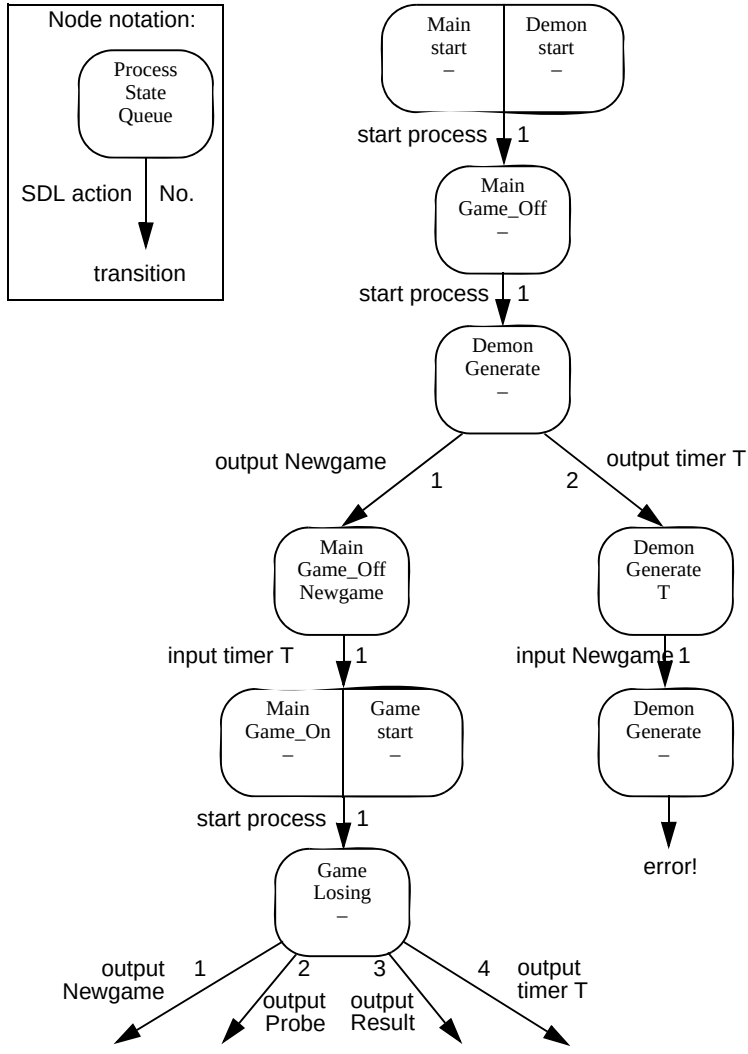


図140: Demongame動作ツリー

その他のトレースと表示機能

この演習では、エクスプローラのトレースと表示機能のいくつかを紹介します。

学習内容

- スタート状態からトレースをすべて表示
- ビュー コマンドの使用
- MSCトレース機能の使用
- パス コマンドの使用による特定の状態への移行

ビュー コマンドの使用

1. 前の演習の終了時と同じ状態にあることを確認します。

   スタート状態から現在までのすべてのテキスト トレースを参照するには、Print-Traceコマンドを使用します。Print-Traceコマンドを使うと、遷移を何回戻した状態からトレースを開始するか、パラメータで指定できます。

2. エクスプローラUIの入力行に、「`pr-tr 9`」と入力します（指定する数字には、これよりも大きな数字を入力しても構いません）。メイン ウィンドウのテキスト エリアにトレースが表示されます。トレースによって、これまでに実行されたSDLシステムでの動作を確認できます。

3. SDLエクスプローラでは、SDLシミュレータと同じ表示機能がサポートされています。Demonプロセスでセットされた、アクティブ タイマを参照するには、[表示]モジュールの[タイマー一覧]ボタンをクリックします。

4. MainプロセスのGameP変数を確認します。まずスコープをMainプロセスに設定します。[スコープ設定]ボタンをクリックしてMainプロセスを選択します。次に、[変数]ボタンをクリックしてGameP変数を選択します。

   – また、エクスプローラUIの[ウォッチ]ウィンドウを使用して、変数の値をモニタリングすることもできます。

MSCトレースの使用

エクスプローラでは、テキスト トレースやGRトレースの他に、MSCトレースを表示することもできます。

1. [コマンド]メニューから[SDLトレースの開始/停止]を選択して現在表示されているSDLトレースを非表示にします。次に、同じメニューから[MSCトレースの開始/停止]を選択してMSCトレースを有効にします。MSCエディタが開き、スタートから現在の状態までのトレースを表示したメッセージ シーケンス チャートが表示されます。
   – ここで、画面にたくさんのウィンドウが表示されて煩雑になるのを避けるために、SDLエディタを閉じても構いません。

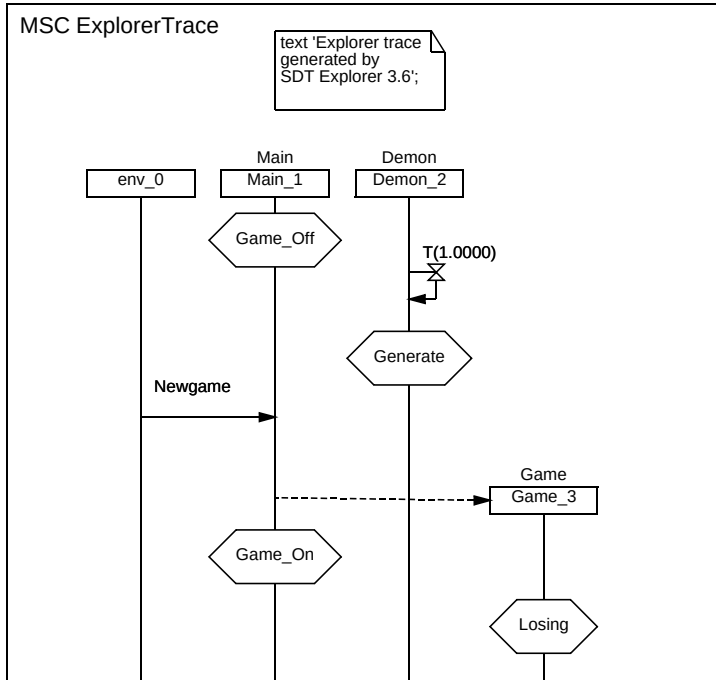


図141: 現在のMSC トレース

2. MSCが表示されたら、ナビゲータに表示されている、Probeなどの信号の遷移のいずれかをダブルクリックして遷移を実行します。まだ消費されていないメッセージがMSCに追加されます。
3. ナビゲータで、遷移を数回戻します。

MSCエディタで選択されるシンボルがどのように移動し、現在の状態に対応するMSCイベントを表現していくか確認してください。

4. 再び探索を進めます。ただし、今回は前回と異なるノードを選択します。つまり、前回とは異なる信号を送信します。

   新しいシステムの振る舞いを表している、MSCダイアグラムがどのように再記述されるかを確認します。

5. [コマンド]メニューで[MSCトレースの開始/停止]をオフにします。他のMSCダイアグラムを開いていなければ、MSCエディタも終了します。

Pathコマンドによる特定の状態への移動

Print-PathとGoto-Pathコマンドを使用すると、以前の状態へ戻ることができます。

1. 入力行に「print-path」と入力して、コマンドを実行します。動作ツリーに表示されていた、スタート状態から現在の状態までの遷移経路が表示されます。

   ```
   Command : print-path
   1 1 1 1 1 3 0
   ```

   – 遷移経路を示す番号は、ナビゲータで表示された遷移番号や、199ページの図140に記述されている矢印の番号と同じものです。

2. ナビゲータで、遷移を数回戻します。

3. テキスト エリアで、Print-Pathコマンドによって出力された遷移経路を見つけます（見えにくい場合はテキスト エリアをスクロールします）。UNIXの場合は、遷移経路の表示までマウスを移動して、遷移経路の数字を最後の「0」まで正しく選択します。

4. 入力行に「`goto-path`」と入力し、さらにprint-pathコマンドで出力された遷移経路の数字（ここでは「1111130」）を入力します。UNIXの場合は、マウスのポインタを入力したテキストの最後に移動し、マウスの中央のボタンをクリックして、遷移経路の番号を貼り付けます。

5. `Enter`キーを押して入力したコマンドを実行します。この操作によって、前の状態に戻ることができます。

   – 遷移経路の番号を入れ間違えた場合は、「↓」キーを押して入力行をクリアし、再度入力します。

SDLシステムの検証

前の演習では、動作ツリーを手動で移動しました。また、ナビゲータと[コマンド]ウィンドウのテキスト トレースを使って、エラーになる状況を検出しました。

この演習では、Demongameシステムの状態空間を自動的に探索させて、エラーや問題点を検出します。この操作は、SDLシステムの*バリデーション*といいます。

学習内容

- 自動状態空間探索の実行
- レポート ビューワーによるエラーの参照
- 状態空間と探索オプションの変更
- 振る舞いに影響しない状態空間への制限
- 探索におけるシステム カバレジの確認
- ユーザー定義ルールの使用
- ランダム ウォーク探索の実行

ビット状態探索の実行

自動状態空間探索は、様々なアルゴリズムによって実行することができます。*ビット状態探索*というアルゴリズムを使うと、規模の大きなSDLシステムでも効率良く検査することができます。探索時に生成されるシステムの状態を表示させるために、*ハッシュ テーブル*というデータ構造を使用します。

自動状態空間探索は、現在のシステム状態から必ず探索を開始します。したがってDemongameシステム全体を探索するのであれば、動作ツリーを最初の状態に戻さなければなりません。

1. [探索]モジュールの[初期状態へ]ボタンをクリックして、ツリーの先頭に戻します。

2. [探索]モジュールの[ビット探索]ボタンをクリックして、ビット状態探索を開始します。しばらくすると、[Report Viewer]というツールが開きます。このウィンドウについては、すぐ後で説明しますが、それまでの間、このウィンドウがメイン ウィンドウを隠さないように遠ざけておきます。

3. Demongameのような小さなシステムの場合、探索はすぐに終了し、統計データがテキスト エリアに表示されます。表示される統計データは以下のようになります。

```
** Starting bit state exploration **
Search depth    : 100
Hash table size : 1000000 bytes

** Bit state exploration statistics **
No of reports: 1.
Generated states: 2569.
Truncated paths: 156.
Unique system states: 1887.
Size of hash table: 8000000 (1000000 bytes)
No of bits set in hash table: 3642
Collision risk: 0 %
Max depth: 100
Current depth: -1
Min state size: 68
Max state size: 124
Symbol coverage : 100.00
```

表示された情報の中で、以下のデータに注目します。

- `Search depth    : 100`

  *Search depth*は探索の深さの制限を示しています。つまり動作ツリーの最大の深さの制限を示しています。探索中にこの深さに達すると現在の経路の探索が打ち切られ、動作ツリーの他の枝で探索が継続されます。エクスプローラUIのオプション設定を変えると探索の深さを変更できます。

- `No of reports: 1.`

  探索中にエラーが1つ検出されたことを示しています。このエラーについては、次の演習で調査します。

- `Truncated paths: 156.`

  最大の深さが、156回であることを示しています。つまり、動作ツリーにおいて探索されなかった箇所が存在します。無限の状態空間を持つSDLシステムでは、これは正常な結果です。Demongameシステムは、ゲームを永遠に続行できるので、無限の状態空間を持つシステムです。

- `Collision risk: 0 %`

  生成されたシステムの状態を表示するハッシュ テーブルにおいて、衝突のリスクが非常に低かったことを示しています。この値が0より大きい場合、オプションを設定してハッシュ テーブルのサイズを大きくしなければなりません。そうしないと、パスの一部が誤って打ち切られてしまう可能性があります。このチュートリアルでは、この状況は発生しません。

– `Symbol coverage : 100.00`
探索においてシステム内のSDLシンボルがすべて実行されたことを示しています。シンボル カバレジが100%ではない場合、バリデーションが完了したことにはなりません。この後の演習では、この状況が発生します。

レポートの検査

状態空間探索で報告されたエラーを[Report Viewer]で調べることができます。[Report Viewer]ウィンドウには、レポートがツリー構造内のボックスとして表示されます。

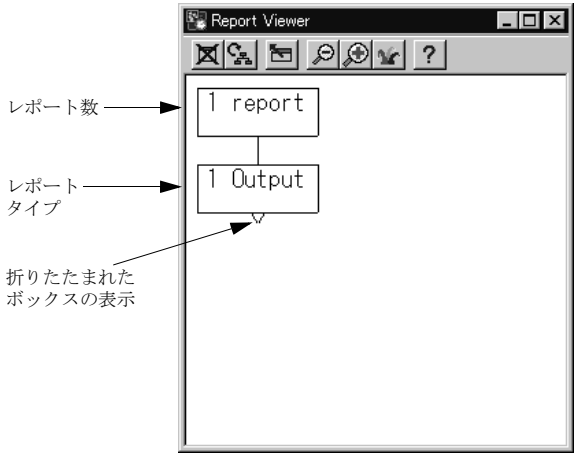


図142: [Report Viewer] ウィンドウ

- 一番上のボックスにはレポートの数が表示されます（この場合は1個）。
- レポート ツリーの次のレベルでは、レポート タイプごとにボックスが現れ、該当するタイプのレポート数が表示されます。
- 次のレベルでは、実際のレポートを参照できます。ただし、デフォルトでは、ツリーのこのレベルは折りたたまれ、レポート タイプ ボックスの下に小さい三角形のアイコンが表示されます。

1. レポートを展開するには、*Output*と表示されたレポート タイプ ボックスをダブルクリックします。以前に、手動で検出したエラーを報告するボックスが表示されます。さらに、エラーが発生した位置のツリーの深さも表示されます。

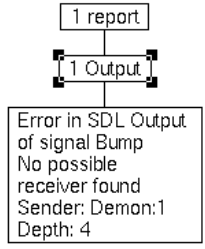


図143: 展開されたレポート

状態空間探索をすでに実行している場合でも、ナビゲータや[コマンド]ウィンドウを確認すると、エクスプローラがシステムのスタート状態にあることがわかります。ここで以下の方法によってエラーが発生した状態へ進みます。

2. [Report Viewer]のレポート ボックスをダブルクリックすると、以下の処理が実行されます。

   – エクスプローラUIのテキスト エリアと[コマンド] ウィンドウにエラーの内容を記述したテキスト トレースが表示されます。

   – ナビゲータがエラー状態へ移動し、エラーを表示します。

   – MSCエディタが開き、現在の状態までのMSCトレースが表示されます。Gameプロセスはまだ生成されていないので、どのプロセスもBump信号を受信していないことがわかります。ここで、MSCエディタ ウィンドウを移動して、他のウィンドウを隠さないようにします。

   [Report Viewer]を使ってエラーが報告された状況まで進むと、現在までの遷移経路を簡単に上下に移動することができます。ナビゲータのノードをダブルクリックしなくても、[探索]モジュールの[1つ上に移動]および[1つ下に移動]ボタンをクリックすれば移動が可能です。

3. [1つ上に移動]ボタンで、上に向かって遷移を2つ移動します。この状態から遷移可能な2つの状態のうち、エラーに通じる遷移には3つのアスタリスク「***」が表示されています（197ページの図137参照）。[1つ下に移動]ボタンを使用すると、3つのアスタリスク「***」が付いた遷移が選択されます。

4. [探索]モジュールの[初期状態へ]ボタンをクリックして、表示をツリーの先頭へ戻します。[1つ下に移動]ボタンを使って、エラーが報告された状態まで再び移動します。

   ツリーが分岐していても進む方向を選択する必要はありません。エクスプローラがエラーまでの遷移経路を記憶しているため、別の遷移をナビゲータで選択しない限り、[1つ下に移動]ボタンをクリックするだけで遷移を進めることができます。

規模の大きな状態空間の探索

ここでは、状態空間オプションの設定を変えて、高度なビット状態探索を実行します。この設定変更によって、状態空間が拡大し、より多くのエラー状況を検出することができます。

1. [初期状態へ]ボタンをクリックして、動作ツリーの先頭へ戻します。

2. [オプション1]メニューの[高度]ボタンをクリックします。以下のように、テキスト エリアには複数の実行された複数のコマンドが表示され、複数の状態空間オプションが一度に設定されます。

   ```
   Command : def-sched all

   Command : def-prio 1 1 1 1 1

   Command : def-max-input-port 2
   Max input port length is set to 2.

   Command : def-rep-log maxq off
   No log for MaxQueueLength reports
   ```

   ナビゲータには、スタート状態から移行できる2通りの遷移が示されます。状態空間の拡張は、すぐにナビゲータに反映され、このように表示が更新されます。

3. さらに、探索の深さをデフォルトの「100」から「300」に変更します。[オプション2]メニューから[ビット状態探索：探索の深さ]を選択します。ダイアログ ボックスに「`300`」と入力し、[OK]をクリックします。

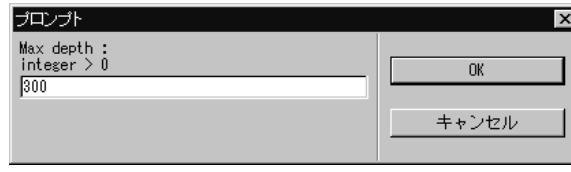


図144: 深さを「300」に指定

これらのオプション設定によって動作ツリーが拡大するため、探索の完了にはより多くの時間が必要になります。したがって、ここでは探索を手動で停止する方法を紹介します。

4. 新たにビット状態空間探索を開始します。20,000回の遷移が実行されるたびに、ステータス メッセージがテキスト エリアに表示されます。[探索]モジュールの[探索の中止]ボタンをクリックして、最初のステータス メッセージが表示された後で探索を止めます。テキスト エリアの表示は以下のようになります。

```
*** Break at user input ***

** Bit state exploration statistics **
No of reports: 2.
Generated states: 50000.
Truncated paths: 1250.
Unique system states: 21435.
Size of hash table: 8000000 (1000000 bytes)
No of bits set in hash table: 41557
Collision risk: 0 %
Max depth: 300
Current depth: 235
Min state size: 68
Max state size: 168
Symbol coverage : 100.00
```

メモ：

手動で探索を停止する前に、探索が停止してしまった場合は、207ページの1.から再度操作を行ってください。ただし、探索する深さを「400」または「500」に変更します。

前回の探索と比較すると、表示されている情報には以下のような違いがあります。

- `No of reports: 2.`
  探索によって、新しいエラー状態が検出されています。これは、動作ツリーの各状態から実行できる遷移数が増えたためです。

- `Max depth: 300`
  `Current depth: <number>`
  探索の停止時、探索は動作ツリーで表示されている深さにいます。しかし、探索では深さ優先アルゴリズムを使用しているため、より早い段階で最大の深さ300に到達しています。動作ツリーの残りの部分を探索したい場合は、現在の深さから探索を継続することができます。

5. [Report Viewer]で、2つのレポート タイプ ボックスを開き、ボックスを別々にダブルクリックして両方のレポートを見ます。[Report Viewer] ウィンドウの表示は次のようになります。

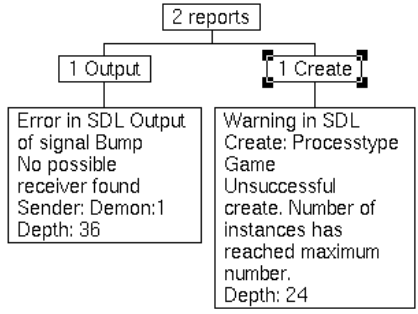


図145: ウィンドウに表示された2つのレポート

6. ここでは、各レポートがどれくらいの深さで発生したかを確認するだけで構いません。どちらのレポートも、ダブルクリックしないでください。

7. [探索]モジュールの[ビット探索]ボタンを再度クリックして探索を継続します。ダイアログ ボックスが開き、中断した探索を継続するか、最初から探索を実行するか質問されます。

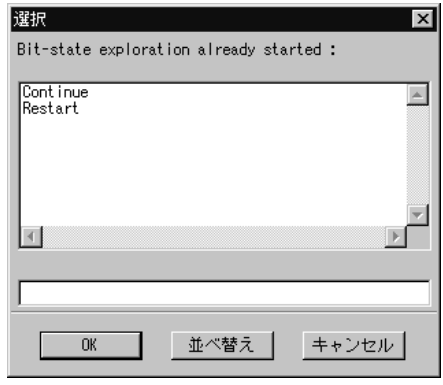


図146: 探索の続行

8. ダイアログ ボックスで[Continue]を選択し、[OK]をクリックします。探索が完了するのを待ちます。

9. [Report Viewer]で2つのレポートを再度開きます。深さの値が変化したことが確認できます。これは、各レポートが1つしか表示されていないためです。深さの値は、今までで最も小さな値を示しています。

10. [Report Viewer]の[Create]レポートをダブルクリックして、Gameプロセスの生成に失敗した状態に移動します。

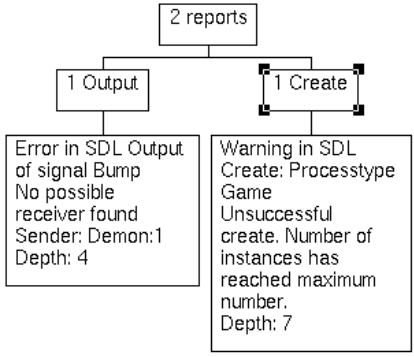


図147: 失敗したCreateプロセスのレポート

11. プロセスの生成に失敗した原因を確認するために、MSCトレースを参照します。

    最後のNewgame信号を受信すると、MainプロセスはGameプロセスの生成を試みます。しかし、すでにアクティブになっているGameプロセスは、前のGameOver信号を取得していないために終了していません。Demongameシステムではプロセスの複数のインスタンスを生成できないので、プロセス生成を実行することができずエラーになってしまいます。

状態空間の制限

エクスプローラでは、さまざまな方法で状態空間を制限することができます。ここでは、それらの方法の中から、多くの状況で役に立つ、Define-Variable-Modeコマンドの実行方法を紹介します。

このコマンドによって、状態空間探索において状態が一致した際に、エクスプローラに特定の変数を無視するように指定できます。各変数には「Skip」または、「Compare」のモードを設定できます。モードを「Skip」に設定することよって、新しい状態が検出された場合でもその状態に対する探索を実施しないように設定できます。新しい状態とすでに検出されている状態の相違点が、Skipを指定した変数の値だけであれば、Skipが適用されます。

ここで、DemonGameシステムに、この制限を設定します。*Game*プロセスの*Count*変数には、ゲームの現在得点が格納されます。そして、この変数の値はシステムの振る舞いに実質的には影響しません。以下の手順で、探査が実行された際に、エクスプローラがこの変数を無視するように設定します。

1. [初期状態へ]ボタンをクリックして、ツリーの表示を先頭に戻します。

2. エクスプローラUIの入力行にコマンド「`define-variable-mode`」を入力し、最初のダイアログ ボックスで*Game*プロセスを選択します。2番目のダイアログ ボックスで*Count*変数を選択し、最後のダイアログ ボックスで*Skip*を選択します。これでエクスプローラがCount変数を無視するように設定できました。

3. [ビット探索]ボタンをクリックして、ビット状態探索を開始します（ダイアログ ボックスが開いている場合は、[再開]をクリックして探索を再実行します）。

4. 探索が終了したら、前回の探索の結果と比較します。2つの探索の相違点は、Count変数が今回の探索で無視されていることだけです。しかし、前回の探索では処理が終了するまでに長い時間を要したのに対して、今回は数秒で終了しています。また、統計データに表示されている回数は、前回と比較すると非常に小さい値を示しているはずです。

この演習の結果からわかるように、システムのある変数の探索に多くの遷移回数がかかる場合は、これを省略することによって大幅に実行回数を削減できることがあります。したがって、振る舞いに影響しない変数、つまり、分岐式に影響しない変数や、「output to」が記述されている式に使用されている変数などを見つけることができれば効率の良い探索の実行が可能になります。また、探索の間、値が変わらない変数も同様です。たとえば、システムの起動時に初期化され、その後変更されないか、少なくとも探索の間変更されない配列などがこれに相当します。このような変数のモードは「Skip」に設定すべきです。

エクスプローラ カバレジのチェック

自動状態空間探索後のシンボル カバレジが100%未満の場合は、カバレジ ビューワーでシステムのどの部分が実行されていないかを確認することができます。Demongameシステムで100%未満のシンボル カバレジを得るためには、特別な方法で探索をセットアップしなければなりません。

1. ツリーの先頭へ表示を移動させます。
2. まず、初期の小さな状態空間に戻す必要があります。[オプション1]メニューの[デフォルト]ボタンをクリックします。ナビゲータに先頭のノードから引き出された遷移が1つだけ表示されます。
3. すべてのシステム内の状態へ到達しないように、探索の深さを「100」から「10」に変更します。[オプション2]メニューから[ビット状態探索：探索の深さ]メニューを選択し、最大の深さを「10」に設定します。
4. ビット状態探索を開始します。表示される統計データには、シンボル カバレジが約82%であることが記述されます。
   - シンボル カバレジがまだ100%の場合は、[オプション1]メニューの[リセット]をクリックして、上記3項と4項の手順を繰り返します。
5. Demongameシステム内の到達していない部分を調べるために、[コマンド]メニューから[カバレジビューワの表示]を開きます。

   シンボル カバレジ ツリーが現れ、実行されていないシンボルがすべて表示されます。

6. [ツリー モード]クイック ボタンをクリックして、遷移カバレジ ツリーに表示を変更します。

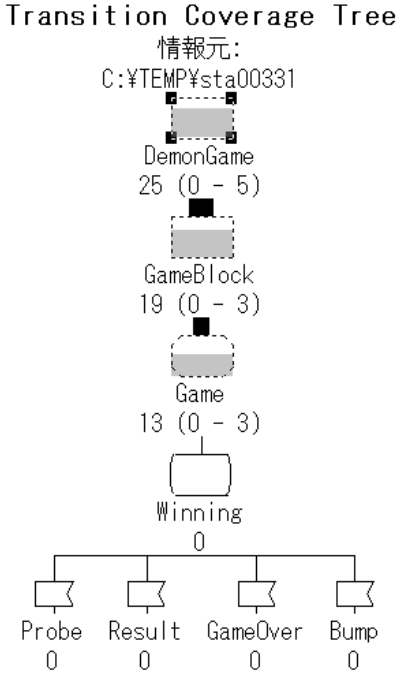


図148: 遷移カバレジツリー

GameプロセスのWinning状態からの遷移は、1回も実行されていないことがわかります。エクスプローラでは、システムの実行されていない部分を探索するために、Winning状態へ移動し、そこから新しい探索を始めることができます。次の演習では、この操作方法を説明します。

ユーザー定義ルールを使って状態を進める

あるシステム状態へ移動するには、[コマンド] ウィンドウの遷移記述とテキストトレースの内容を確認し、ナビゲータを使ってその状態を手動で見つける方法があります。この方法は、特にDemongameよりも大きいシステムでは、面倒で、難しい方法です。ここでは、手動による方法の代わりに、*ユーザー定義ルール*を使ったより簡単な移動方法を紹介します。

状態空間探索の実行時、エクスプローラは、到達するシステムの状態ごとに複数の定義済みルールをチェックします。レポートの生成も、そのようなルールが満たされたときに発生します。

この演習では、状態空間の探索時にチェックされる新しいルールの定義方法を学習します。このルールを使って、GameプロセスのWinning状態を検索します。

1. 現在の位置が動作ツリーの先頭にあることを確認します。

2. [コマンド]メニューの[規則の定義]を選択し、新しいルールを定義します。表示されたダイアログ ボックスにルールの定義を入力します。
   `state(Game:1)=Winning`

図149: 新しいルールの定義

   このルールの定義は、プロセス インスタンスの状態Game:1がWinning状態と同じでなければならないことを示しています。このルールを定義すると、ルールを満たす状態に状態空間探索が達したときにレポートが生成されます。

3. ビット状態探索を開始します。前回の探索からオプションを変更していないので、追加のレポートが生成されることを除くと、統計データは同じ内容になります。

4. [Report Viewer]を使って、ユーザー定義ルールに適合して、レポートが生成された状態へ進みます。これで、GameプロセスがWinning状態になっている、動作ツリーの最初の状態に到達できます。

5. エクスプローラで、現在状態が動作ツリーのルートになるように設定します。入力行で「`define-root`」コマンドを実行し、ダイアログ ボックスから[Current]を選択します。

ここで、オプションを変更し、新しいルールを定義するか、MSCをロードします。これらの新しい設定は、新しいルートから実行される探索に対して適用されます。また、あらゆるリスト関連のコマンドや、遷移経路を進めるコマンドも新しいルートを基準にして実行されます。さらに、MSCトレースも新しいルートから開始されます。

6. 作業を継続する前に、ユーザー定義ルールをクリアします。入力行で「`clear-rule`」を実行します。

ここで、先ほど定義したユーザー定義ルールをクリアし、この状態からランダムウォークと呼ばれる、異なる形式の状態空間探索を開始します。

ランダム ウォークの実行

ビット状態探索の他に、*ランダム ウォーク*という探索方法を使用できます。ランダム ウォークとは、ツリーの下方向への経路を繰り返しランダムに選択して動作ツリーを探索する方法です。この方法は、状態空間の規模が大きくなるSDLシステムにおいて有効です。ただし、Demongameのように小さなシステムの場合、ランダム ウォークの効果は他の探索方法とあまり変わりません。

1. [ランダム検索]ボタンをクリックして、現在の状態からランダム ウォーク探索を開始します。統計データのシンボル カバレジが100%になったことを確認します。

2. [コマンド]メニューから[カバレジビューワの表示]を選択し、新しいカバレジ情報に更新されたカバレジ ビューワーを表示させます。遷移カバレジ ツリーに表示を変更して、ツリー全体を表示させます。すべての遷移が数回実行されていることを確認できます。自動探索時のランダムなパスの選択によって、ツリーの下方向へパスが選択される際、探索済みパスの再探索を回避する機能はありません。したがって、同じ遷移が何度も実行されることがあります。

3. [ファイル]メニューの[終了]を選択して、カバレジ ウィンドウを終了します。

4. [オプション1]メニューの[リセット]ボタンをクリックしてシステムをリセットします。状態がツリーの先頭に戻り、ツリーのルートが元のシステムのスタート状態にリセットされます。

メッセージ シーケンス チャートの検証

エクスプローラのもう1つの主要な用途には、メッセージ シーケンス チャート（MSC）の検証があります。MSCの検証とは、SDLシステムにおいてMSCを満たす実行パスがあるかどうかをチェックすることです。MSCをロードし、MSCの検証用に最適化された状態空間の探索を実行することによって、検証を行います。

学習内容

- MSCの検証

システム レベルMSCの検証

この演習では、システム レベルで生成されるMSC、つまり、外部環境との間で送受信される信号のみを定義するMSCを検証します。MSCのファイル名は、`SystemLevel.msc`であり、DemonGameのサンプル ファイルがあるディレクトリに保存されています。MSCは下図のような表示になります。

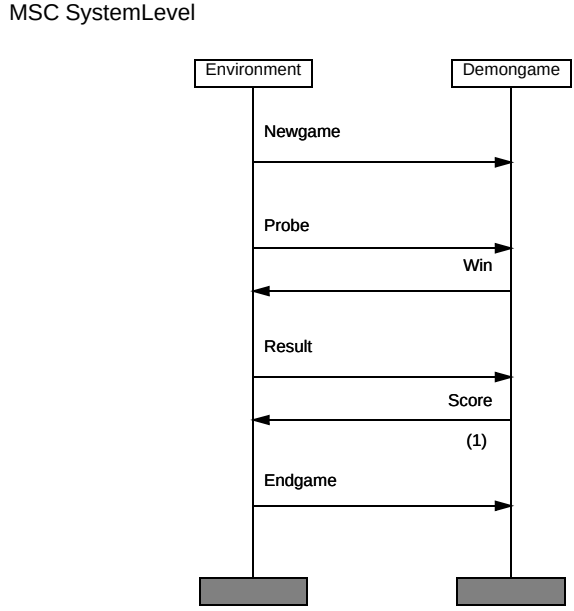


図150: システム レベルMSC

1. システムをリセットします。今回は[ファイル]メニューから[再開]を選択します。保存するかどうかを質問された場合は、[No]を選択します。[再開]コマンドによってエクスプローラが終了し、再起動します。
2. [MSC検証]ボタンをクリックして、MSCの検証を開始します。ファイル選択ダイアログ ボックスが開きます。このダイアログ ボックスで、検証するMSCを選択します。
3. `SystemLevel.msc`を選択し、[OK]をクリックします。ロードされたMSCによって状態空間探索がただちに開始されます。

   表示された統計データを参照すると、すべての遷移経路に対して実行が中断されずに探索が完了していることを確認できます。これは、ロードしたMSCの動作ツリーのサイズが制限されているためであり、MSC内のイベントと関連する部分だけが実行されます。この場合の、動作ツリーの最大の深さは20程度になります。

   MSCの検証が正常終了したこと、または、違反が検出されたことを示す以下の行を確認します。

   ```
   ** MSC SystemLevel verified **
   ```

今回、MSCの検証は正常終了しています。つまり、MSCに記述されている振る舞いが、実行可能であることが確認されています。ただし、[Report Viewer]では、レポートのうちの1つ（または2つ）にMSCのロード*違反*が示されており、もう一方のレポートにMSCが検証されたことが示されています。探索によって、MSCが違反している状態を簡単に検出できます。つまり、検証結果として出力されたMSCを確認することによって、違反している状態が存在していることを確認できます。

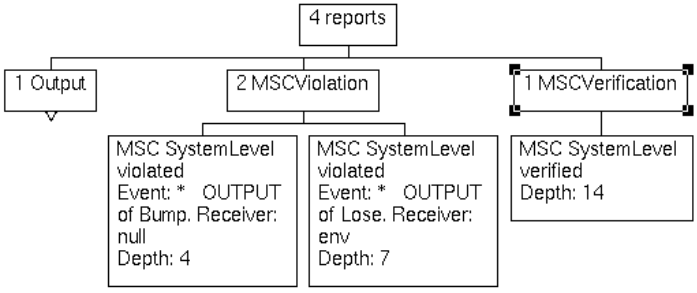


図151: MSCの違反と検証

4. MSCが検証された状態へ移動します。[コマンド] ウィンドウのテキスト トレースには、MainプロセスがEndgame信号を受信し、GameOver信号をGameプロセスへ送信したことが表示されます。

```
*   OUTPUT of GameOver. Receiver: Game:1
*     Signal GameOver received by Game:1
```

5. MSCトレースの内容と、217ページの図150のロードされたMSCとを比較します。ロードされたMSCは外部環境との間で送受信される信号のみを定義するので、MSCトレースの内容ほど詳細ではありません。エクスプローラのMSCトレースは常にプロセス レベルで生成されます。

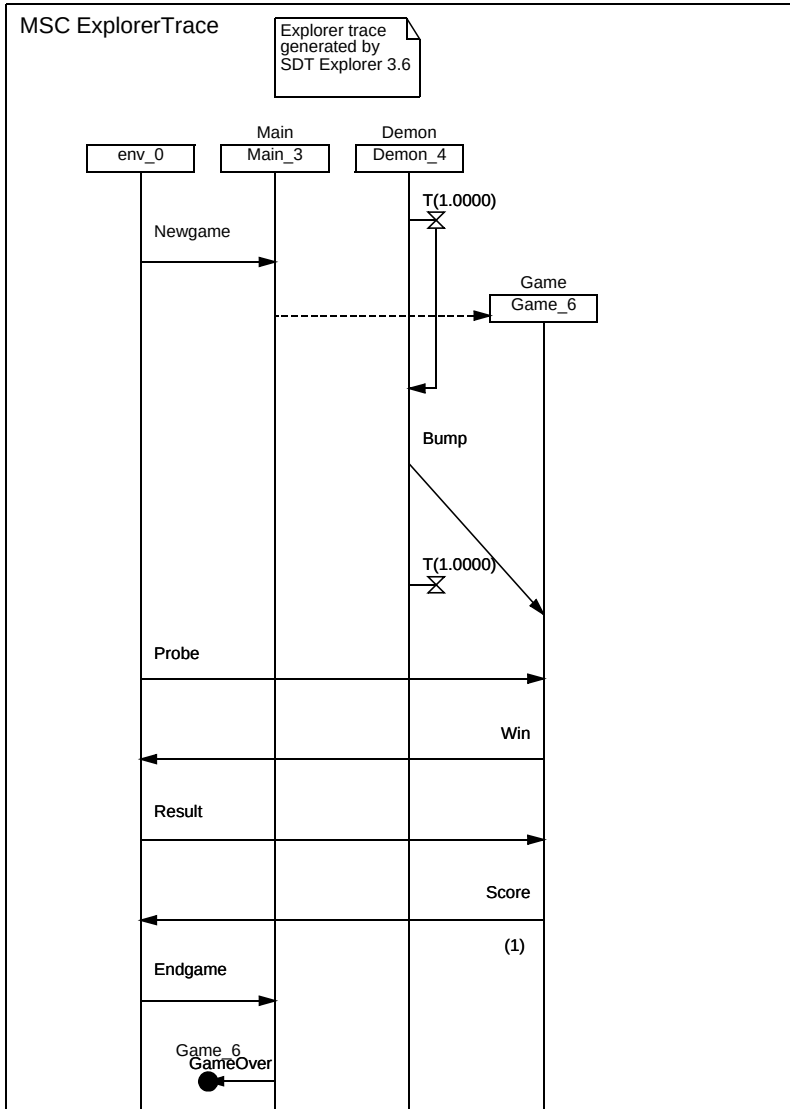


図152: MSC トレース

上図のトレースにはプロセスの状態を示す条件シンボルは表示されていません。

エクスプローラUIの終了

エクスプローラのチュートリアルの前半部分はこれで終了でです。以下の手順でエクスプローラ ウィンドウを閉じます。

1. ナビゲータと[Report Viewer]のそれぞれのウィンドウで[閉じる]クイック ボタンをクリックして、各ウィンドウを閉じます。
2. [コマンド]ウィンドウの[ファイル]メニューから[閉じる]を選択して、[コマンド]ウィンドウを閉じます。
3. [ファイル]メニューの[終了]を選択して、エクスプローラUIを終了します。変更したエクスプローラのオプションを保存するダイアログ ボックスが表示されます。

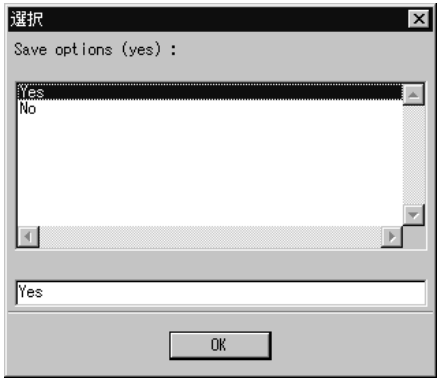


図153: 変更したオプションの保存

4. [Yes]を選択し[OK]をクリックした場合、UNIXでは`.valinit`というファイルに、またWindowsでは`valinit.com`というファイルに保存されます。エクスプローラUIを同じディレクトリから起動するときや、エクスプローラUIからエクスプローラをリスタートまたはリセットした際に、このファイルが読み込まれます。[No]を選択し、[OK]をクリックします。

テスト値の使用

最後の演習では、エクスプローラのテスト値機能について学習します。この機能は、状態空間探索を実行する際に、システムと外部環境との間の信号の送受信を制御するために使用します。演習では、外部環境からシステムに送信する信号を定義します。これらの定義には、パラメータへの具体的な値の定義も含まれます。

ここでは、Inresシステムという新しいSDLシステムを使用します。

1. Inres システムを、インストール ディレクトリから、作業用ディレクトリにコピーします。コピーするファイルは、UNIXの場合は `$telelogic/sdt/examples/inres`に、Windowsの場合は `C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥sdt¥examples¥inres`にすべて保存されています。

学習内容

- 自動的に生成されたテスト値の確認と使用
- テスト値の手動変更

テスト値の自動生成機能の使用

エクスプローラを開始すると、多数のSDLソートに対してテスト値が自動生成されます。たとえば、すべての整数パラメータには、テスト値として-55、0、または55のいずれかが与えられます。ここではInresシステムに自動的に与えられたテスト値を確認します。

1. オーガナイザの[ファイル]メニューで[開く]を選択し、Inresシステムのファイルを開きます。既存のDemonGameシステムに保存しなければならないファイルがある場合は、[開く]ダイアログ ボックスが表示される前に、それらの保存を促されます。作業用ディレクトリにコピーしたファイル `inres.sdt`を探して開きます。

2. [バリデート]クイック ボタンをクリックして、このシステムのエクスプローラを作成し、起動します。この操作の詳細については、190ページの「エクスプローラの迅速な起動」を参照してください。ダイアログ ボックスが表示され、新規と既存のエクスプローラUIのいずれを選択するか質問されたら、既存のエクスプローラUIを一覧から選択して[OK]をクリックします。

3. エクスプローラUIに表示されている[テスト値]ボタン モジュールを展開し、すべてのボタンが表示されるようにウィンドウを広げます。

   ボタン モジュールには、ボタンの行が4行あります。最初の行には、[値の一覧]、[値の定義]、および[値の消去]の3つのボタンがあり、これらのボタンに

よってSDLシステムの各ソート（データ型）へのテスト値の定義が可能になります。2番目の行には、[パラメーター一覧]、[パラメータ定義]、および[パラメータ消去]の3つのボタンがあり、これらのボタンによって信号パラメータへのテスト値の定義が可能になります。最後の行のボタンは、全信号に対してテスト値を設定する際に使用します。

4. [List Value]ボタンをクリックし、デフォルトのテスト値にどのような値が割り当てられているか確認します。以下のような値が表示されます。

```
Sort integer:
0
-55
55

Sort Sequencenumber:
zero
one

Sort IPDUType:
CR
CC
DR
DT
AK
```

テスト値は、定義済みソートの*integer*と、2種類のシステム専用の列挙ソートである*Sequencenumber*および*IPDUType*に対して定義されています。列挙ソートに対しては、値が10個以下の場合すべての値がデフォルトでテスト値に割り当てられます。外部環境との通信で使用する信号に割り当てられているソートのみが、パラメータの上に表示されます。

5. [信号一覧]ボタンをクリックして、ソートに与えられたテスト値を基にしてどのような信号がシステムに送信されるかを確認します。

以下のような信号の一覧が表示されます。なお、MDATreq信号に与えられているパラメータは、以下の表示内容と異なる可能性があります。これは、ランダム関数によってパラメータが生成されているためです。ランダム関数は使用しているコンパイラによって生成される値が異なることがあります。

```
ICONreq
IDATreq(0)
IDATreq(-55)
IDATreq(55)
IDISreq
MDATreq((. CR, zero, -55 .))
MDATreq((. CR, zero, -55 .))
MDATreq((. CC, one, 55 .))
MDATreq((. AK, one, 55 .))
MDATreq((. CR, one, 55 .))
MDATreq((. DT, one, -55 .))
MDATreq((. CC, one, 0 .))
MDATreq((. CC, one, -55 .))
MDATreq((. AK, one, 55 .))
```

```
MDATreq((. DR, one, 0 .))
```

*ICONreq*信号と*IDISreq*信号はパラメータを取りません。したがって、これらの信号の定義は1種類になります。IDATreq信号は整数のパラメータを1つ取ります。したがって、この信号には3つのテスト値が割り当てられ、各テスト値はintegerソートに定義された値に対応します。

MDATreq信号は、3つのフィールドで構成されたパラメータを取ります。そのフィールドは、IPDUType、Sequencenumber、および、integerの3つのソートに対応します。エクスプローラが、構造を検出すると、ソートに対応したテスト値を生成します。テスト値には、各フィールドが取り得るすべての組み合わせの値が割り当てられます。ただし、テスト値の数が許容できる最大の数を超えてしまった場合は、ランダムにテスト値が選択されます。テスト値を許容できる最大数は、デフォルトでは10ですが、この数はDefine-Max-Test-Valuesコマンドを使って変更できます。

結果的にMDATreq信号には、ランダムに生成された異なる10個のパラメータ値が割り当てられます。

ここで、状態空間探索を実行した際のシステムの振る舞いに、テスト値がどのように反映されるかを確認します。

6. [探索]ボタン モジュールの[ナビゲータ]ボタンをクリックして、ナビゲータを起動します。

7. 複数のノードが下方に現れるまで、下方のノードを4回ダブルクリックします。

   11個のノードが下方に表示されます。各ノードは外部環境からSDLシステムへ向かう入力を表しているはずです。これら11個の信号の内容を確認すると、ICONreqとMDATreqに割り当てられたテスト値に対応していることがわかります。

   以上が状態空間の探索において割り当てられるテスト値の生成方法です。外部環境からシステムへ信号の送信が可能な際、エクスプローラは信号に定義されたテスト値を使用して、信号送信に使用するパラメータを決定します。

テスト値の手動変更

ここでは、[テスト値]ボタン モジュールのほかのコマンドのボタンを使用して、手動でテスト値を変更します。

1. [探索]モジュールの[初期状態へ]ボタンをクリックして状態空間をスタート状態に戻します。

   以下の手順で整数型に1と99が割り当てられるようにテスト値を変更します。

2. [値の消去]ボタンをクリックして、ダイアログ ボックスから[integer]型を選択します。値ダイアログ ボックスに、値として「-」（ハイフン）を入力します。「-」（ハイフン）を入力すると、現在ソートに設定されているすべてのテスト値が削除されます。

   整数型に割り当てられているテスト値を変更すると、エクスプローラは各種ソートおよび信号に対してテスト値の再生成を試行します。ただし、整数型が、多くの他のソートや信号で使用されている場合は、変更対象のソートや信号に新しいテスト値を生成することはできません。

3. [信号一覧]ボタンをクリックして、現在使用されている信号定義を確認します。現在の信号定義は以下のようになります。

   ```
   ICONreq
   IDISreq
   ```

   整数型のテスト値は、現在割り当てられていないため、整数型のパラメータを含んでいない信号のみが表示されます。これは、現時点で探索を開始した場合、ICONreq信号とIDISreq信号だけが、外部環境からシステムに送信されることを意味します。

4. [値の定義]ボタンをクリックして、ソート ダイアログ ボックスで[integer]を選択します。そして、値ダイアログ ボックスに、値として「1」を入力します。

5. 再度、[値の定義]ボタンをクリックします。ソート ダイアログ ボックスで[integer]を選択して、値ダイアログ ボックスに、今度は「99」を入力します。

6. [信号一覧]ボタンをクリックして、信号の定義を表示させます。整数型のパラメータを持つ信号が、再度一覧表示されることを確認します。今回は、テスト値に1と99が使用されています。

```
ICONreq
IDATreq(1)
IDATreq(99)
IDISreq
MDATreq((. AK, zero, 1 .))
MDATreq((. DR, zero, 99 .))
MDATreq((. AK, one, 1 .))
MDATreq((. DR, one, 1 .))
MDATreq((. DR, zero, 99 .))
MDATreq((. AK, zero, 1 .))
MDATreq((. AK, one, 99 .))
MDATreq((. AK, zero, 99 .))
MDATreq((. AK, one, 1 .))
MDATreq((. CR, one, 1 .))
```

ここでは、エクスプローラにおいて、テスト値の生成の際に使用する主要な機能を学習しました。他にも、特定のパラメータにテスト値を設定する機能や、すべてのテスト値を手動で定義する機能などがあります。テスト値の設定に関するこの他の情報については、『User´s Manual』の第53章「Validating a System」の2421ページ、「Defining Signals from the Environment」を参照してください。

エクスプローラの終了

前回と同様の手順でエクスプローラを終了します。

1. [ファイル]メニューから[終了]を選択します。
2. 新しく設定したオプションを保存するかどうか質問されたら[Yes]を選択します。ダイアログで[Yes]を選択すると、UNIXの場合は`.valinit`の拡張子が付いたファイルに、また、Windowsの場合は`valinit.com`という名前のファイルに、新しいテスト値の定義を再生成するコマンドが保存されます。

まとめ

ここまでのチュートリアルでは、SDL Suiteの基本概念を学習しました。これらのチュートリアルでの学習が有意義に活用されることを期待します。ただし、演習に使用したDemongameやInresはきわめて簡単なシステムです。SDL Suite コンポーネントについてさらに知識を深めるために、SDL-92の利点を示すさまざまな演習を以降の章で学習することができます。SDL-92は、オブジェクト指向のデザイン方法論を導入する際に有利な言語です。これらの演習は第6章「チュートリアル: SDL-92のDemonGameへの適用」に記述されています。

第 6 章

チュートリアル: SDL-92のDemonGameへの適用

このチュートリアルでは、SDLで採用されている、オブジェクト指向の拡張表現の使用方法について説明します。このオブジェクト指向の拡張表現は一般的にはSDL-92として知られています。また、ここでは、第3章「チュートリアル: SDLエディタとアナライザ」と、第4章「チュートリアル: SDLシミュレータ」のチュートリアルですでに学習済みのDemonGameを題材として取り上げます。

SDTにおけるオブジェクト指向の拡張表現を理解するために、このチュートリアルの内容をすべて読んでください。また、記載されている手順に従って、お使いのコンピュータ システムで実習してください。

このチュートリアルの目的

このチュートリアルでは、SDL Suiteツールで利用できるオブジェクト指向SDLの基本的な機能を学習します。このチュートリアルは、SDL Suiteの機能を理解するためのガイダンスとなるように配慮されています。チュートリアルを読むときは、説明されている各演習をコンピュータで実際に操作して確認してください。

このチュートリアルでは、内容を容易に理解できるように比較的簡単な題材を選んでいます。ただしSDL言語の学習を目的としていないため、説明はすべてSDL言語に関する基礎知識があることを前提に記述されています。

ここでは、このマニュアルの前の章で使用したDemonGameを題材に取り上げています。この章を学習するには、第3章「チュートリアル: SDLエディタとアナライザ」と第4章「チュートリアル: SDLシミュレータ」の演習を終了しておくことが必要です。

> メモ：プラットフォーム間の相違点
>
> この章やその他の章のチュートリアルでは、UNIXとWindowsの両方のプラットフォームで実行可能な項目について1種類の手順を記述しています。プラットフォームによって操作が異なる場合は、「UNIXのみ」または「Windowsのみ」などの指示を記載してあります。このようにプラットフォームを特定した記述がある場合は、ご使用のプラットフォームに関する記述のみを参照してください。
>
> 画面表示についても、両方のプラットフォームで同じ情報が表示される場合、通常はどちらか一方の画面表示のみを示します。したがって、図のレイアウトや概観は、SDL Suiteを使用環境で実行したときに表示される画像と異なることがあります。重要な部分がプラットフォーム間で異なる場合に限って、それぞれの画面を示します。

DemonGameへのSDL-92の適用

これまでのチュートリアルでは、SDLの基本的な言語要素を使った演習を行いました。これらの言語要素は、SDL-88と呼ばれるオブジェクト指向の概念が導入されていないSDLで定義されているものです。

このチュートリアルにはいくつかの演習が用意されており、DemonGameへ機能を追加する作業をとおして、SDL-92についての理解を深めることができます。DemonGameへの機能の追加は、オブジェクト指向の手法を利用した、Gameプロセスへのプロパティの追加や再定義によって行います。

また、演習では、以下のようなSDL-92の言語構造を導入します。

- プロセス タイプ
  - プロセス タイプの継承とプロパティの追加
  - 仮想プロセス タイプと再定義プロセス タイプ
  - 仮想遷移と再定義遷移
- パッケージ
  - パッケージの使用
  - パッケージの再使用
- ブロック タイプ
  - ブロック タイプの継承とプロパティの追加

> メモ：
>
> この章で使用する「SDL-92」という用語は、1992年度版で導入されたオブジェクト指向のSDLを意味します。また、オブジェクト指向の仕様はSDL-96でも変更されていません。

事前準備

ここでは、元のDemonGameシステムをそのまま使用するのではなく、SDL-92の導入用に設計されたDemonGameシステムを使用して学習を進めます。SDL-92用のDemonGameシステムに加えられている変更は以下のとおりです。

- 外部環境から送信されるすべての信号（Newgame、Endgame、Probe、Result）は、管理プロセスであるMainへ送信するように変更されています。そして、Gameプロセスが存在する場合は、それらの信号がMainプロセスからGameプロセスに送信されます。
- Bump信号は、Mainプロセスにも送信され、MainプロセスがGameプロセスにBump信号を送信します。これにより、存在しない受信者に信号が送信されてしまう問題が解消されます。
- 信号ルートと信号リストは、信号の新しい経路を反映するように更新されています。
- 内部信号のGameOverは実際には必要ないため、EndGame信号に置き換えられています。

以上の変更によりユーザーから見たシステムの機能は変わらずに、より安定した動作を得ることができます。

新しいGameBlockブロックとMainプロセスを、図154と図155に示します。

新しいDemonGameを使用するには以下の準備が必要です。

1. 新しい作業用ディレクトリ`sdl92`を作成します。UNIXでは`~/demongame`の下に、Windowsでは`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥work`の下に作成します。
2. UNIXの場合は、`$telelogic/sdt/examples/demongame/sdl92/process_type`にあるすべてのファイルを新しいディレクトリにコピーします。
   Windowsの場合は、`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥sdt¥examples¥demongame¥sdl92¥process_type`にあるすべてのファイルを新しいディレクトリにコピーします。

メモ：インストール ディレクトリ

UNIXでは、インストール ディレクトリの環境変数は、`$telelogic`になります。UNIX環境内にこの変数が設定されていない場合は、システムの管理者やSDL Suite環境の管理担当者に正しい環境変数の設定方法を確認してください。

このチュートリアルでは、Windowsのインストール ディレクトリに対して`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J`を想定しています。PCにこのディレクトリが設定されていない場合は、システムの管理者やSDL Suite環境の管理担当者に正しいパスの設定方法を確認してください。

3. SDL Suiteを起動し、オーガナイザを使ってこの新しいディレクトリにあるシステム ファイル`demongame.sdt`を開きます。完全なシステムを構成したダイアグラムが表示されます。

ここで開いたDemonGameシステムには、前述の修正が加えられています。

以降の演習で作成するダイアグラムのすべてが、新しく作成したディレクトリ内に保存されています。演習では、SDL-92の使用方法を理解するためにSDLエディタでダイアグラムを作成しますが、ダイアグラムを作成しない場合は、保存されているダイアグラムをコピーしたり、保存されているダイアグラムへの直接接続することも可能です。

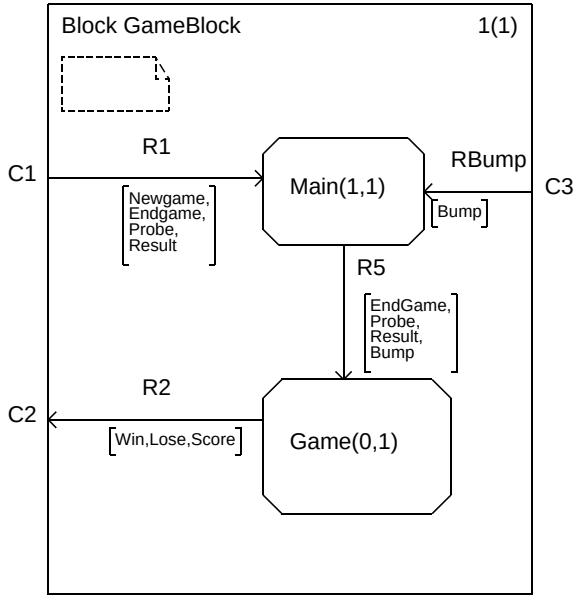


図154: 変更後のGameBlockブロック

実際に開いたダイアグラムのレイアウトは上図のレイアウトと多少異なることがあります。

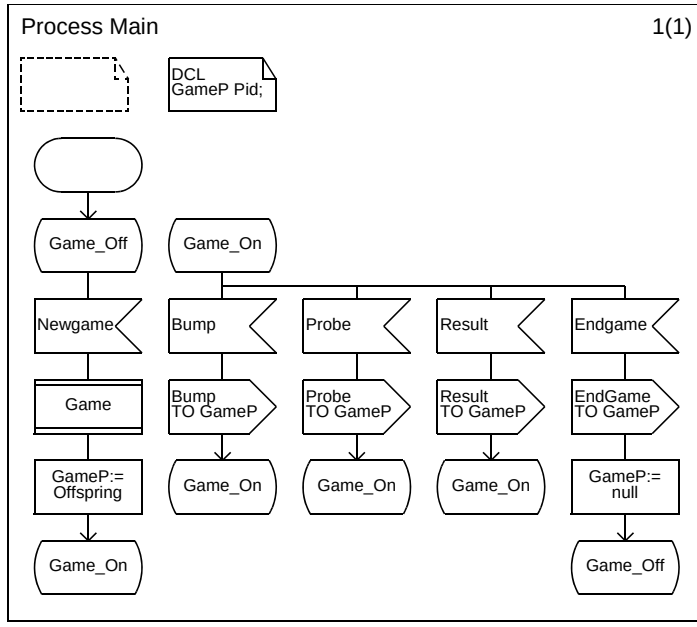


図155: 変更後のMainプロセス

実際に開いたダイアグラムのレイアウトは上図のレイアウトと多少異なることがあります。

プロセスからのプロセス タイプの作成

学習内容

- プロセス ダイアグラムのプロセス タイプへの変更
- プロセス タイプへの参照とプロセス タイプの実体化
- ゲートを使ったプロセス タイプとブロックまたは他のプロセス（タイプ）との接続
- 仮想遷移としての遷移の定義

プロセス タイプへの変更

導入した新しい機能を使用するために、Gameプロセスをプロセス タイプに変更し一般化します。この変更によって、後でプロセス タイプを特殊化したり再定義することができます。

1. Gameプロセスを開き、ダイアグラムのタイプをプロセスからプロセス タイプに変更します。タイプへの変更は、ダイアグラム ヘッダ シンボルを選択し、テキストを「Process Type Game」に修正するだけです。

図156: ダイアグラムの形式を変更

2. SDLエディタで[ファイル]メニューの[名前を付けて保存]をクリックし、Process Type Gameダイアグラムを`new_game.spt`などの新しいファイル名で保存します。ファイル名をファイル選択用のダイアログ ボックスで修正して、[OK]をクリックします。（ファイル選択用ダイアログ ボックスに表示される各`.spt`ファイルは、SDL Suiteのサンプル ファイルとして製品に収められている完成したシステムをコピーしたものです）。

3. オーガナイザ ウィンドウを表示させます。

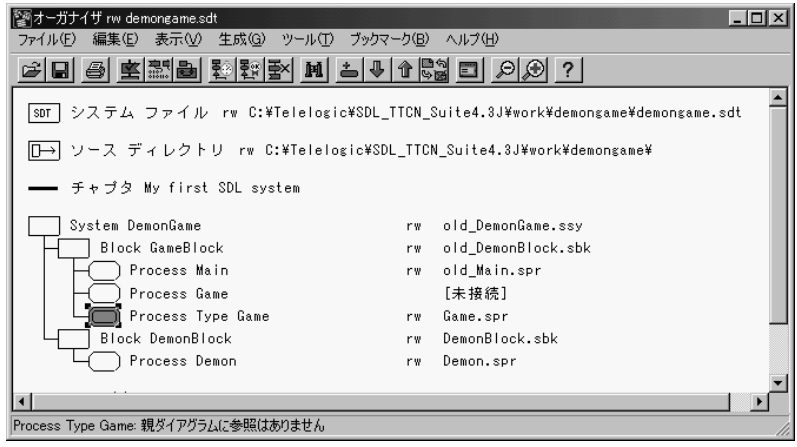


図157: オーガナイザに表示された有効でない参照

オーガナイザで参照シンボルがProcess Type Gameに変更されていることを確認できます。また、シンボルは、親ダイアグラムにこのプロセス タイプへの参照が存在しないことを示しています。さらに、古いGameプロセスには[未接続]と表示されます。このシンボルは、しばらくの間そのままにしておき、後からインスタンス シンボルで置き換えます。

4. GameBlockダイアグラム ブロックを開きます。GameプロセスがGameプロセス タイプのインスタンスを参照するようにGameBlockダイアグラム ブロックを変更します。インスタンスを参照するための構文は「Game(0,1):Game」です。テキストを入力する際は、テキストをシンボルに合わせるために改行することもできます。

   – テキストの編集を始める前の段階では、テキスト カーソルは点滅していません。この時点で`Delete`を押すと、選択されているシンボルがすべて削除されてしまいます。テキストの編集を開始すると、テキスト カーソルが点滅し、`Delete`を押すたびに1文字だけが削除されるようになります。

5. シンボルの選択を解除するとすぐに、信号ルートの各接続ポイントにテキスト入力用の四角いフィールドが表示されます。接続ポイントにG2やG5などの名前を入力します。

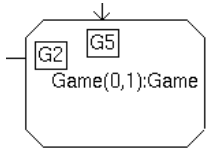


図158: 接続ポイントに名前を付ける

6. Gameという名前のプロセス タイプ参照シンボルを追加します。この時点でブロック ダイアグラムは次のようになります。

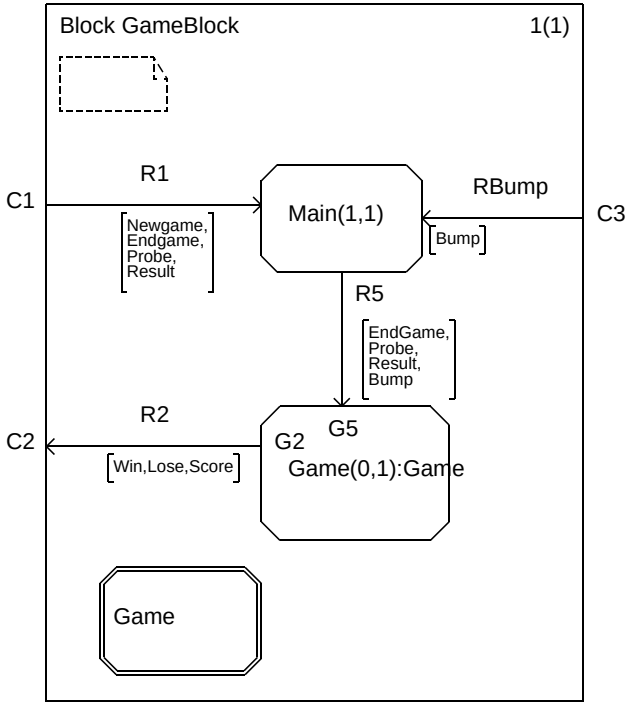


図159: 変更後のブロック

7. 手順2と同じように[名前を付けて保存]を使って、ブロック ダイアグラムを`new_gameblock.sbk`などの新しいファイル名で保存します。

ゲートと仮想遷移の挿入

ここでは、Gameプロセス タイプを完成させるために、以下の手順でGameプロセス タイプを編集します。編集せずに完成したダイアグラムに接続することもできますが（239ページの「完成したダイアグラムへの接続」を参照してください）、その場合も以下の説明を必ず読んでください。

プロセス タイプ ダイアグラムの編集

1. SDLエディタでGameプロセス タイプを再度開きます。信号ルートへの接続を定義する際は、ゲート シンボルを使います。ゲート シンボルには、先ほど定義した接続ポイント名に対応する名前を割り当てます。

2. ゲート シンボルをフレーム シンボルに接続します。ゲート シンボルをフレームの左側や上側に接続する場合は、フレームを選択して、下または右にドラッグします。

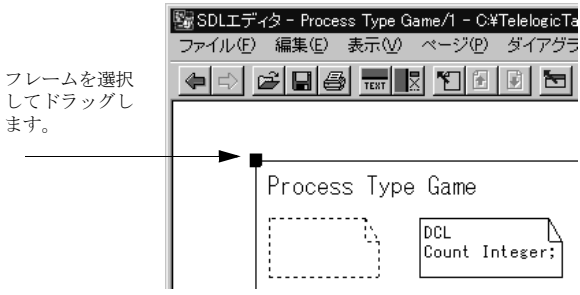


図160: フレーム シンボルの調整

3. G2とG5のゲートに対応する、ゲート シンボルを追加し名前と信号リストを入力します。
   - ゲート シンボルは矢印のような形をしています。シンボル メニューでシンボルをポイントするか、シンボルを選択すると、ステータス バーにそのシンボルの種類が表示されます。
   - フレームを指すようにゲートの向きを変えるには、まずゲートを追加し、次に、[リダイレクト]コマンドを使って方向を変更します。ゲートは双方向にすることもできます。
   - [新しいウィンドウを開く]コマンドを使うと、Gameプロセス タイプとGameBlockブロックの両方を同時に表示することができます。[コピー]と[貼り付け]コマンドを使ってダイアグラム間でテキストをコピーします。
   - 信号辞書ウィンドウを利用すれば、[挿入]コマンドで上位のGameBlockブロックの信号を挿入することができます。
4. Probe、Bump信号の入力とスタート遷移を仮想状態にします。信号名の前に「VIRTUAL」というテキストを追加します。
   - この変更によって、後からGameの機能を簡単に変更できるようになります。

変更後のダイアグラムは以下のようになります。

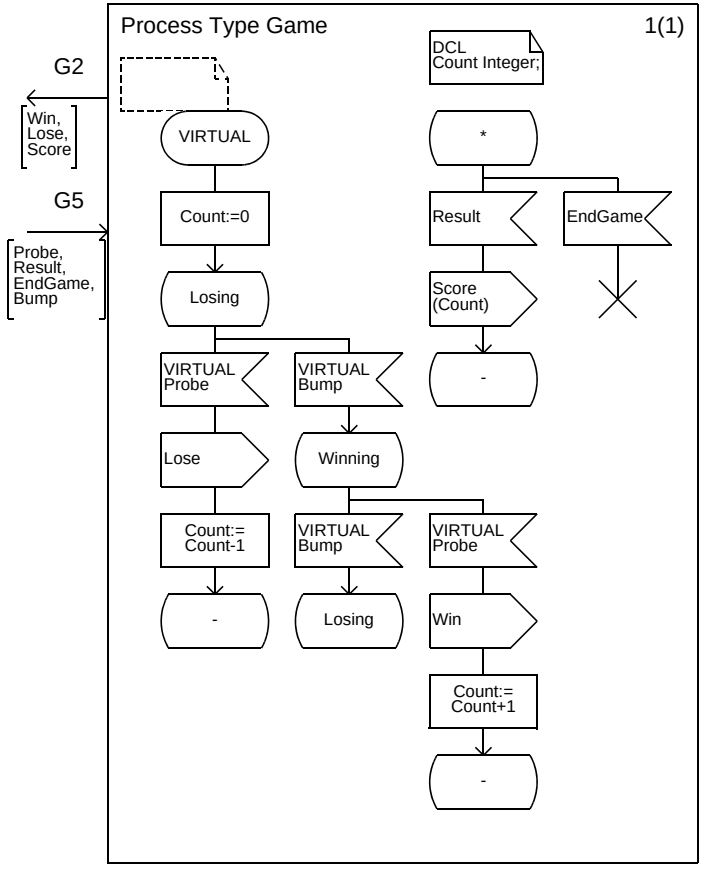


図161: 変更後のGameプロセス タイプ

完成したダイアグラムへの接続

完成したGameプロセス タイプ ダイアグラムも`game.spt`というファイル名で保存されています。ダイアグラムを作成する代わりにこのファイルを使用する場合は、以下の手順でこのファイルに接続します。

1. SDLエディタに表示されている、Gameプロセス タイプ ダイアグラムを閉じます。
2. オーガナイザで、Process Type Gameダイアグラムを選択し、[編集]メニューの[接続]をクリックします。

3. [既存のファイルを対象]オプションをオンにします。ファイル名を「`game.spt`」に変更するか、フォルダの絵のボタンをクリックしてファイルを選択します。
4. [接続]をクリックし、オーガナイザで新しいファイルへ接続されていることを確認します。

オーガナイザの構造

1. すべてを保存します。オーガナイザの表示は以下のようになります。

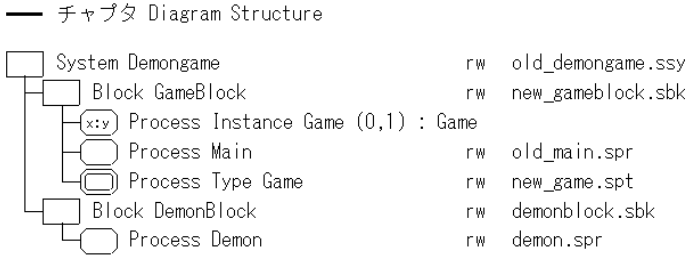


図162: 変更後のオーガナイザの表示

*インスタンス シンボル*は通常のシンボルと形状が似ていますが、インスタンス シンボル内には一般表記形式を表す「X:Y」（「インスタンス:タイプ」を意味します）が示されています。インスタンス シンボルは、ダイアグラム内で実際にタイプからインスタンスが生成されていることを示しています。

オーガナイザに表示されているインスタンス シンボルはダイアグラムを参照していないため、ダブルクリックしてもシステム階層を移動できません。*タイプ ビューワー*ツールを利用するとインスタンス シンボルをSDLエディタ内で使用できるようになります。インスタンス シンボルの使用については、このチュートリアルの以降の項で演習します。

2. システムを分析してから、終了します。構文エラーや意味エラーが報告された場合は、対応する箇所を修正します。

プロセス タイプのプロパティの再定義

学習内容

- プロセス タイプに他のプロセス タイプのプロパティを継承させる
- プロセス タイプの遷移を再定義する

JackpotGameプロセス タイプ

ここまでの演習では、DemonGameシステムの設計を修正することによって再設計を行いました。ここでは、10%の確率で「jackpot」が当たる機能を新たに追加します。jackpotはランダムにスコアを10点増やす機能です。この機能は、0から9を返し、Probe信号を受信したときに乱数をチェックする擬似乱数発生器をインプリメントすることで簡単に実現できます。

また起動するゲームの種類を実行時に指定できるようにします。この機能を実現するには、入力するNewJackpotGame信号を外部環境から加える必要があります。JackpotGameを起動するNewJackpotGame信号はMainプロセスとシステム ダイアグラムへ追加します。

これらの機能を実現するために、JackpotGameをプロセス タイプとしてインプリメントします。JackpotGame プロセス タイプには、Gameプロセス タイプのプロパティを継承させます。また、Bump信号を扱う遷移を再定義することによって、乱数発生の機能を追加します。擬似乱数発生器は、Bump信号を受信するたびに起動させます。以下にJackpotGameをインプリメントする手順について示します。

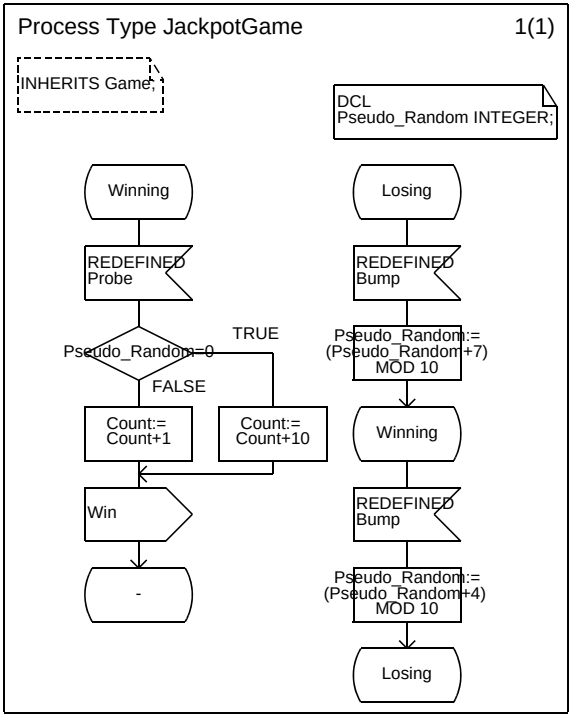


図163: JackpotGame プロセス タイプ

- 上記のダイアグラムを作成し、`new_jackpotgame.spt`などのファイル名で保存します。次に、SDLエディタで[ファイル]メニューの[ダイアグラムを閉じる]をクリックして、ダイアグラムを閉じます。

  – ここでは、`jackpotgame.spt`というファイル名の作成済みのダイアグラムが利用できます。ダイアグラムを作成しない場合は、このファイルをコピーするか、直接使用します。

GameBlockブロックの変更

JackpotGameプロセス タイプを親ブロックで使えるようにするために、前述のGameプロセス タイプと同じ方法でプロセス参照シンボルとプロセス インスタンス シンボルを追加します。また、NewJackpotGame信号をMainプロセスへの信号リストへ追加します。

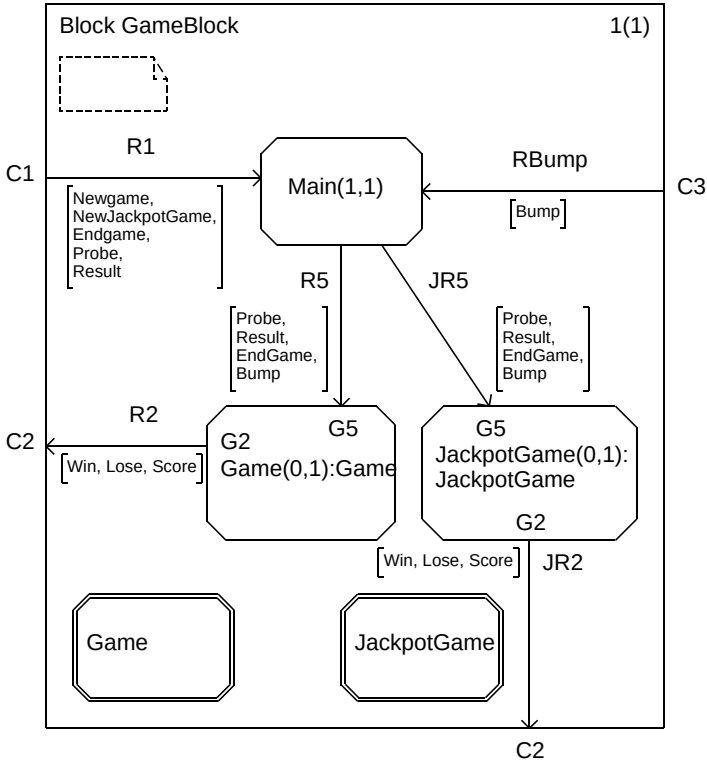


図164: GameBlockブロック

- 既存のGameBlockダイアグラムを上の図のように変更し、ファイルに保存します。

  – 上記のダイアグラムも`gameblock2.sbk`というファイル名で保存されています。ダイアグラムを作成せずにこのファイルを使用する場合は、SDLエディタ内のGameBlockを閉じ、オーガナイザで[編集]メニューの[接続]と[既存のファイルを対象]を使ってGameBlockダイアグラムを新しいファイルに接続します。

Mainプロセスと DemonGameシステムの変更

次に、NewJackpotGame信号の宣言と新しいコードの追加によって、Mainプロセスと DemonGameシステムを変更します。追加する新しいコードには、NewJackpotGame信号を受信し、JackpotGameのインスタンスを生成する処理を記述します。

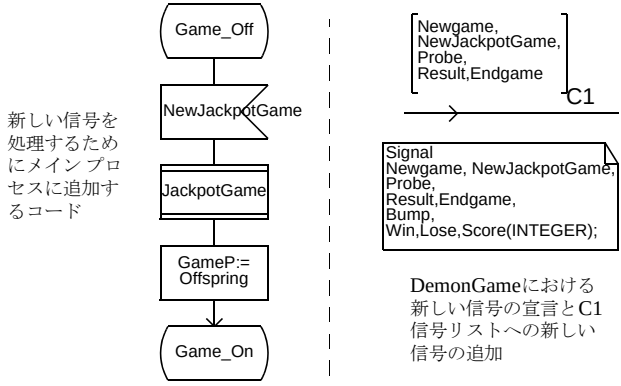


図165: Mainプロセスと DemonGame システムの変更

1. MainダイアグラムとDemonGameを上図のように変更し、ファイルに保存します。保存する際には、`new_demongame.ssy`や`new_main.spr`などの名前で新しいファイルに保存してください。

   – 上記のMainダイアグラムも`main2.spr`というファイル名で保存されているので、ダイアグラムを編集せずに保存されているファイルを使用することができます。オーガナイザの[編集]メニューを使ってダイアグラムを新しいファイルに接続します。

   – DemonGameダイアグラムは手動で編集する必要があります。既存のファイルには再接続しないでください。

2. オーガナイザで、JackpotGameプロセス タイプ ダイアグラムが先ほど作成したファイル`new_jackpotgame.spt`に接続されていることを確認します。接続されていない場合は、[編集]メニューの[接続]と[既存のファイルを対象]を使って接続します。

オーガナイザの表示は以下のようになります。

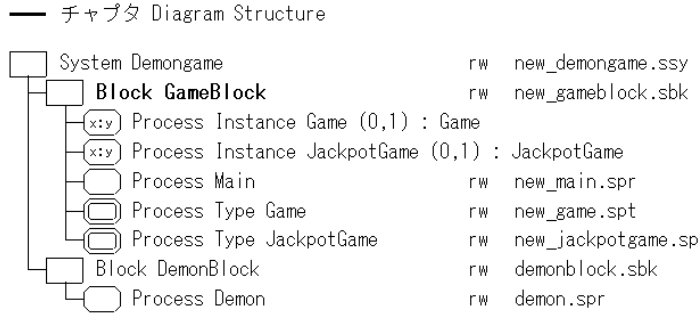


*図166: オーガナイザのリストに追加された**JackpotGame***

JackpotGameのシミュレーション

ここで、変更後のシステムを理解するために、数分程度の時間でシステムのシミュレーションを実行することができます。

1. シミュレータのチュートリアルで学習したように、最初にシステムを分析してシミュレータを生成します。次に、シミュレータUIで生成されたシミュレータを開きます。

2. 以下の機能を確認します。

   – 実行時にNewGame信号かNewJackpotGame信号によって、GameまたはJackpotGameのインスタンスを1つ起動できます。ただし、同時に2つのゲームを実行することはできません。`output-via`コマンドを使用して、C1経由でNewJackpotGame信号、Newgame信号、EndGame信号を送信し、ゲームの起動と停止を行います。

   – ゲームを開始していない場合でも、Bump信号を受信するMainが常に存在するため、Bump信号が動的なエラーを発生させることはありません。

   – 図式によるMSCトレースを有効にして、どのように信号が伝送されるかを表示します。また、SDLのGRトレースも有効にします。JackpotGameをすでに起動している場合でも、GameとJackpotGameの2つのプロセス タイプが実行されていることを図式で確認します。実行を確認するには、シンボル レベルでの実行が必要な場合もあります。

3. 実際の手順でゲームを試してみます。

   – まず、シミュレータUIに*Probe*という名前のボタンを作成します。このボタンはProbe信号を送信し、次にResult信号を送信します。そしてGoコマンドによって実行を再開します。ボタンをクリックするたびにScore信号が返されます。このボタンの定義には、「`output-to Probe Main; output-to Result Main; go`」と入力します。

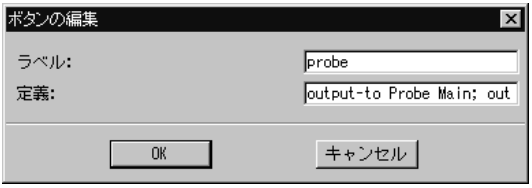


図167: [Probe]ボタンの定義

   – 次にシステムのトレースを1に設定します。1は、外部環境とやり取りする信号だけがトレースされることを意味します。実行速度を速くするには、以下のコマンドで図式によるトレースをすべてオフにします。
   `set-gr-trace 0; stop-msc-log`

   – 以下のコマンドでNewJackpotGameに信号を送信し、シミュレータを起動します。
   `output-to NewJackpotGame Main; go`

   – [Probe]ボタンを繰り返しクリックして、トレースを観察します。ときどき10点を獲得できます。

4. [Break]ボタンをクリックして実行を停止します。

プロセス タイプへのプロパティの追加

学習内容

- プロセス タイプの継承とプロパティの追加
- 破線のゲートの使用

DoubleGameプロセス タイプ

jackpotの機能が追加されても、DemonGameで多く得点するには時間がかかります。そこで、より多くの点数を獲得するために掛ける点数を任意に2倍にすることができる機能を追加します。

以下の手順でこの機能を追加します。

1. DoubleGameというプロセス タイプを作成します。Gameプロセス タイプのプロパティを継承し、さらに以下の変更を加えます。
   - 整数型のStake変数を宣言する。
   - スタート遷移の再定義によってStakeを1に初期化する。
   - Stakeの値を2倍にするDoubleStake信号を受信する。
   - 現在のStakeの値をCountに対して加算または減算するように、Winning遷移とLosing遷移を再定義する。
2. 作成後のプロセス タイプを以下の図に示します。ここまで学習した方法で、このダイアグラムを作成し、`new_doublegame.spt`というファイル名で保存します。次にSDLエディタで、[ファイル]メニューの[ダイアグラムを閉じる]をクリックし、ダイアグラムを閉じます。
   - 以下のダイアグラムは`double.spt`というファイル名で保存されており、ダイアグラムを作成しない場合は、このファイルをコピーまたはそのまま使用することができます。

メモ：

以下のダイアグラムには*破線*のG5ゲート シンボルが記述されており、このゲートによってDoubleStake信号が伝送されます。破線のゲートは継承元のタイプであるスーパータイプですでに定義されているゲートを参照するために使い、新しいゲートを追加する際と異なる手順になります。

ゲートを破線に変更する方法を以下に示します。

- ゲートを選択します。
- SDLエディタで[編集]メニューの[Dash]をクリックします。このコマンドで*破線/実線*を切り替えます。

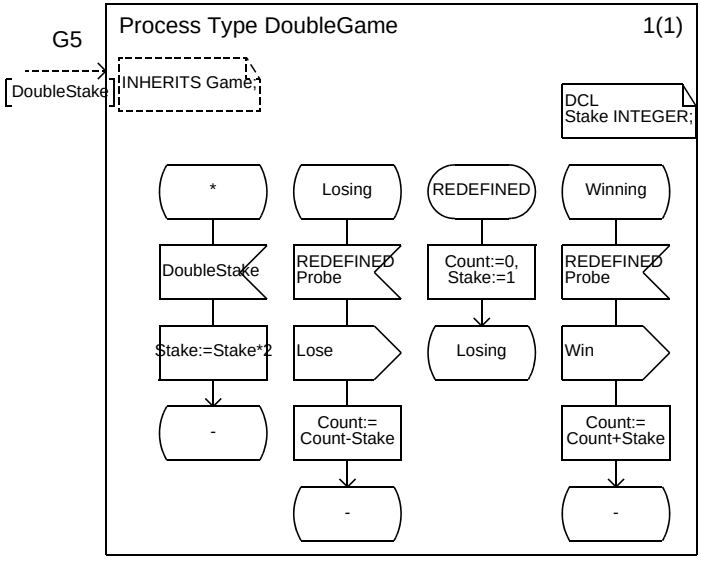


図168: DoubleGame プロセス タイプ

3. DoubleGameのプロセス タイプ参照シンボルと「DoubleGame (0,1):DoubleGame」と入力されたプロセス インスタンス シンボルを、GameBlockブロック ダイアグラムに追加します。以下の図を参照してください。
   – このダイアグラムは、`gameblock3.sbk`というファイル名で保存されています。ダイアグラムを作成せずに、このファイルを使用する場合は、SDLエディタに表示されているGameBlockダイアグラムを閉じて、オーガナイザでGameBlockダイアグラムを新しいファイルに接続します。

4. NewDoubleGame信号とDoubleStake信号をR1信号ルートの信号リストに追加します。また、DoubleGameプロセスへ送信する信号の信号リストにDoubleStake信号を追加します。以下の図を参照してください。

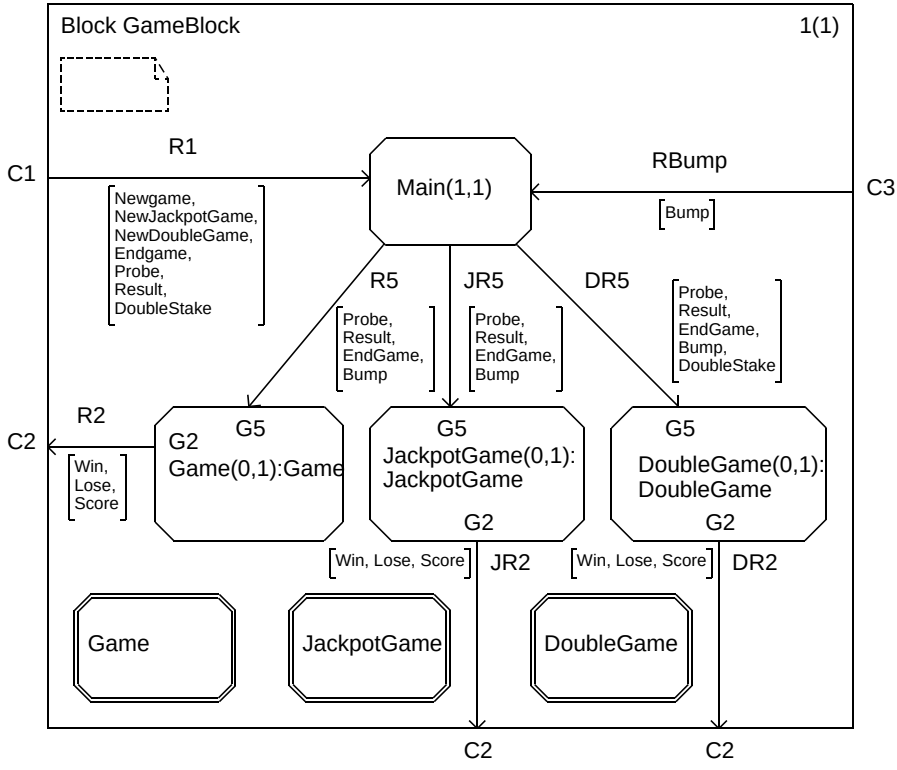


図169: 変更後のGameBlockブロック

5. 新しいNewDoubleGame信号とDoubleStake信号をシステム レベルすなわちDemonGameダイアグラム内に追加します。両方の信号を信号宣言とC1チャネルの信号リストに追加します。

   – これらの変更は手動で行う必要があります。

6. Mainプロセスを変更します。まず、DoubleStake信号とNewDoubleGame信号を受信するコードを記述します。次に、DoubleGameのインスタンスを作成します。以下の図を参照してください。

   – ここで使用するMainダイアグラムは、`main3.spr`のファイル名であらかじめ保存されており、ダイアグラムを編集する代わりに、このファイルを使うことができます。あらかじめ保存されているファイルを使う場合は、オーガナイザでダイアグラムをファイル`main3.spr`に接続してください。

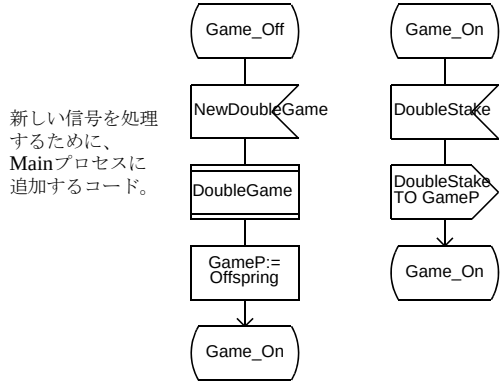


図170: Mainプロセスの新しいコード

7. 必要に応じて、オーガナイザからDoubleGameプロセス タイプ ダイアグラムを、前に作成したファイル`new_doublegame.spt`に接続します。

DoubleGameのシミュレーション

ここで、JackpotGameと同じようにDoubleGameをシミュレーションすることができます。DoubleGameはNewDoubleGame信号で起動します。

1. ゲームを実際に試してみるために、*Double*ボタンをシミュレータUIに追加し、「Double」というテキストと「`Output-to DoubleStake Main; go`」コマンドを割り当てます。

2. たとえば点数がマイナスになったときなどに、掛け点を2倍にして実行してみます。

2つのプロセス タイプのプロパティを組み合わせる

学習内容

- *タイプ ビューワー*（SDL Suiteのクラス ブラウザ）の使用
- 多段階でのプロセス タイプの継承

ここまでの演習では、ゲームの基本バージョンとしてスーパータイプのGameプロセス タイプを作成し、これをJackpotGameプロセス タイプとDoubleGameプロセス タイプの2つのサブタイプに拡張しました。タイプの継承や実体化の状況を把握できるように、SDL Suiteにはクラス ブラウザの一種であるタイプ ビューワーが備えられています。

タイプ ビューワーの使用

1. オーガナイザで[ツール]メニューの[SDL]サブメニューを選択し[タイプ ビューワー]をクリックして、タイプ ビューワーを開きます。

   タイプ ビューワーが起動し、2種類のウィンドウが表示されます。1つはすべてのタイプが一覧表示されるメイン ウィンドウで、もう1つはタイプの継承と実体化が表示される*タイプ ツリー* ウィンドウです。

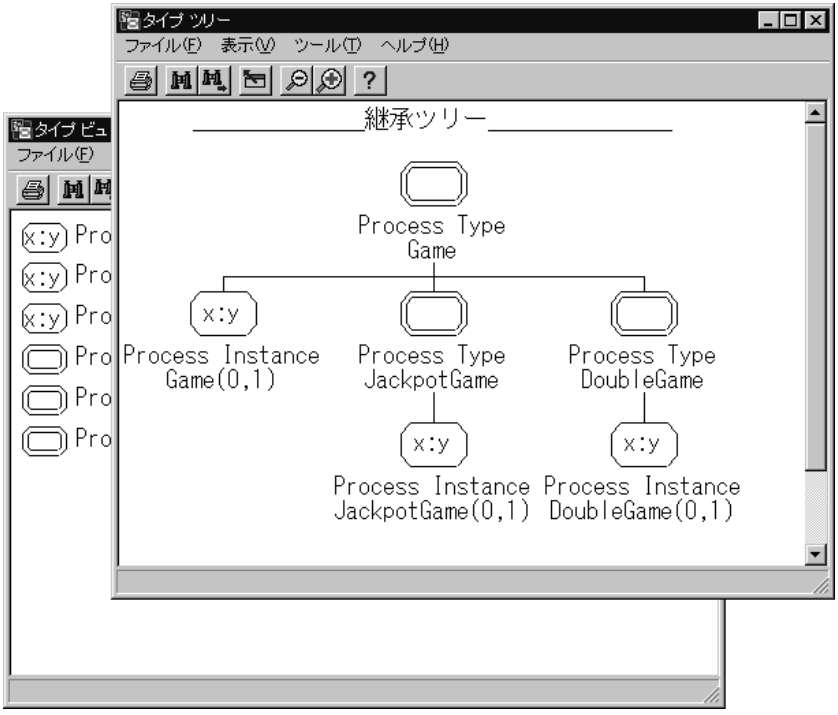


図171: タイプ ビューワーの2つのウィンドウ

メインのウィンドウには現在システム内に存在するすべてのタイプとインスタンスが表示されます。メインのウィンドウでオブジェクトを選択すると、タイプ ツリー ウィンドウの表示が変わり、選択したオブジェクトの継承ツリーが示されます。

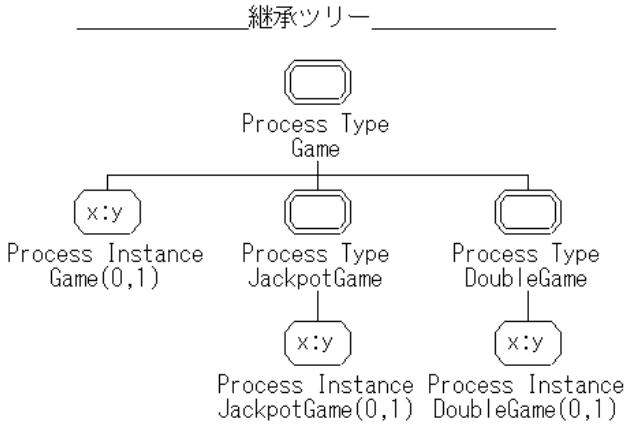


図172: Gameプロセス タイプの継承ツリー

図172は、GameJackpotGameプロセス タイプとDoubleGameプロセス タイプの継承ツリーを示しています。上の図のとおり、これらの継承は1レベルです。また、タイプ ビューワーには、SDLシステム内で生成されているインスタンスの元のタイプも表示されています。

– タイプ ビューワーのシンボルをダブルクリックすると、元のSDLダイアグラムが表示され、タイプの宣言やインスタンスを確認できます。

多重継承に代わる方法

ここでは、jackpotとdoubleの両方の機能を備えた新しいゲームの設計を検討します。SDL-92では多重継承がサポートされていないため、JackpotGameとDoubleGameの両方を継承するSuperGameという新しいプロセス タイプを簡単に作成することはできません。したがってJackpotGameかDoubleGameのいずれかを継承、すなわち再使用して、いくつかのプロパティを再定義または追加する必要があります。この定義には、再記述しなければならないコードを最小にする方法が望まれます。

JackpotGameとDoubleGameのどちらを再使用すべきでしょうか。DoubleGameのコードはSuperGameでも利用できるので、ここではDoubleGameプロセス タイプを継承し、JackpotGameの機能に合わせていくつかのプロパティを再定義します。

以下の手順でSuperGameを作成します

1. SDLエディタでJackpotGameプロセス タイプを開き、[名前を付けて保存]によって`new_supergame.spt`などの名前で新しいファイルとして保存します。
   - このダイアグラムは、`supergame.spt`というファイル名であらかじめ保存されており、ダイアグラムを作成しない場合は、このファイルをコピーするか、そのまま使用することができます。その場合は、手順6.から始めます。
2. ダイアグラムの名前をSuperGameプロセス タイプに変更します。
3. 継承を「INHERITS Game」から「INHERITS DoubleGame」へ変更します。
4. 以下の手順でダイアグラムの内容を変更します。
   - Winning/Probeのフローを変更し、勝った場合に1の代わりにStakeをCountに加えます。そしてjackpotに勝った場合は、Stakeの10倍の値を加えるようにします。
   - Losing/Probeのフローを再定義し、1の代わりにStakeをCountから引くようにします。

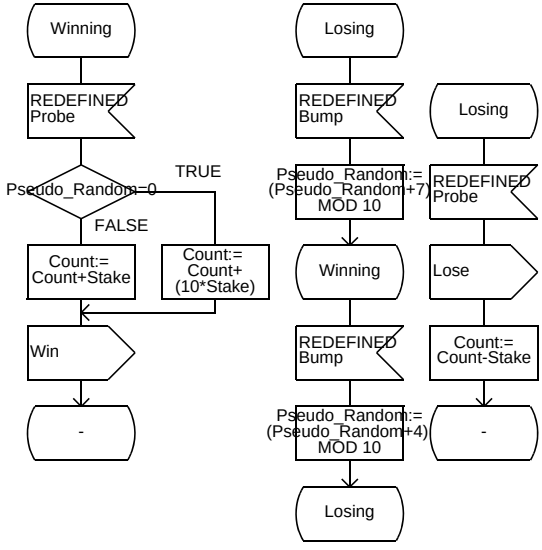


図173: SuperGameプロセス タイプに対する変更

5. オーガナイザに表示されているProcess Type JackpotGameが[未接続]になっている場合は、以前使用していたnew_jackpotgame.sptに再接続します。

6. GameBlockのダイアグラム内にSuperGameという名前のプロセス タイプ参照シンボルを追加します。次に、新しく追加したSuperGameプロセス タイプ ダイアグラムをオーガナイザで、先ほど作成したファイル new_supergame.sptに接続します。

7. ここまでに加えた変更は、タイプ ビューワーで確認できます。すべてを保存し、タイプ ビューワーで[ファイル]メニューの[更新]をクリックします。ダイアグラムを変更しても、タイプ ビューワーは自動的に内容を更新しないため、この操作が必要です。継承ツリーは以下のように表示されます。

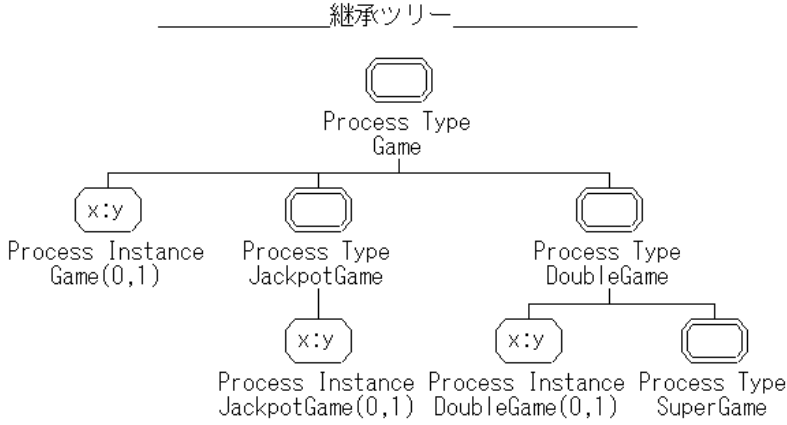


図174: 追加されたSuperGameプロセス タイプ

8. SuperGameを実行するには、SuperGame(0,1):SuperGameプロセス インスタンス シンボルをGameBlockに追加し、JackpotGameやDoubleGameで行った手順と同様に、ゲームを起動するNewSuperGame信号を追加します。ここで必ずシステム ダイアグラムを更新します。

   – これらのダイアグラムはgameblock.sbkとdemongame.ssyというファイル名で保存されており、ダイアグラムを編集せずに、これらのファイルを使うことができます。オーガナイザで、ダイアグラムをこれらのファイルに接続します。また、完成した各SDLダイアグラムに接続されている完全なシステム ファイルdemongame_sdl92.sdtも保存されており、同じように使うことができます。

パッケージとブロック タイプの使用

学習内容

- パッケージ ダイアグラムの作成
- システムでのパッケージの使用
- ブロック タイプへの参照と実体化
- 仮想状態としてのプロセス タイプの定義

パッケージ（再使用可能なコンポーネント）

パッケージを使用することによって、タイプ定義をさまざまなシステムで使うことができるようになります。また、コンポーネントを再使用できるような設定も可能になります。ここでは、パッケージの概念を使って、基本的なProbeの機能を持つDemonGameと、JackpotとDoubleStakeの両方の機能を持つDemonGameの2種類のシステムを作成します。

これらのシステムを構築するために、BasicFeaturesというパッケージを作成します。このパッケージは基本的なDemonGameで使用でき、さらに高機能なDemonGameでも100%再使用できるものにします。

この演習で、パッケージを最大限に使用するには、JackpotGame、DoubleGameおよびSuperGameを作成した場合と同じように、Gameプロセスをプロセス タイプに変換するだけでなく、Mainプロセスを再使用できるプロセス タイプに変換し、GameBlockブロックを再使用できるブロック タイプに変更する必要があります。

Mainプロセス タイプでは、追加するjackpotやdoubleの機能も拡張する必要があるので、再使用が可能なタイプに設定することをお勧めします。この演習では、再使用できるように設定されたタイプがあらかじめ用意されています。以下の手順で2つのパッケージを作成するために必要な結合部分を追加します。

1. まず、この演習で使う`gameblock.sbt`と`main.spt`という名前のファイルを、UNIXの場合は`$telelogic/sdt/examples/demongame/sdl92/packages`から、またWindowsの場合は`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥sdt¥examples¥demongame¥sdl92¥packages`から、このチュートリアルの始めに作成したものと同じディレクトリにコピーします。
   – 以降の演習で使用するダイアグラムはすべて上記ディレクトリ内にあります。以下の演習でダイアグラムを作成しない場合は、あらかじめ保存されているダイアグラムを作業用ディレクトリにコピーします。そして、オー

ガナイザでダイアグラムのシンボルを作成し、対応するファイルに接続します。

パッケージの作成

下の手順でパッケージを作成します。

1. オーガナイザで[編集]メニューの[新規追加]をクリックします。[新規追加]ダイアログ ボックスで、ドキュメント タイプにSDLパッケージを選択し、ドキュメント名を「`BasicFeatures`」に設定します。

図175: 新しいBasic Featuresパッケージの追加

[OK]をクリックすると、パッケージ ダイアグラムがルート ダイアグラムに設定された新しいダイアグラム構造がオーガナイザに作成されます。このように、オーガナイザでは同じシステム ファイル内で複数の構造を管理できます。

新しく作成されたパッケージに、以下のDemonGameの基本的な機能を定義します。

- 基本機能のDemonGameと外部環境との間の信号インターフェイスを宣言し、信号インターフェイスをサポートするMainプロセス タイプを定義します。
- 基本機能のGameプロセス タイプを定義します。
- プロセス タイプを内部に持つブロック タイプを定義します。

2. SDLエディタを使用して、信号の宣言をシステム ダイアグラムからパッケージ ダイアグラムに移動します。また、Gameプロセス参照シンボルとBasicGameBlockブロック タイプ参照シンボルをパッケージ ダイアグラムに追加します。以下の図を参照してください。

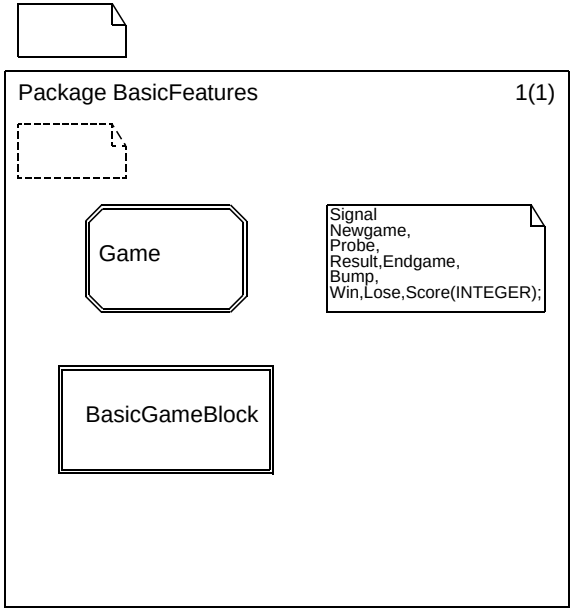


図176: Basic Features パッケージ

3. パッケージ ダイアグラムを`basicfeatures.sun`などのファイル名で保存します。

4. オーガナイザを使用して、BasicGameBlockブロック タイプを先ほどコピーした`gameblock.sbt`に接続します。`gameblock.sbt`のダイアグラムを図177に示します。

   – Mainプロセス タイプが、VIRTUALとして宣言されています。これは、Mainの名前をSuperMainなどに変更することなく、Mainに機能を追加し、外部環境からMainへの信号を送るための重要な定義です。この手順を、Gameプロセス タイプをJackpotGameなどに特殊化したときの手順と比較してください。プロセス タイプをVIRTUALに定義することで、後からREDEFINEDキーワードを使って名前を変更せずに機能を追加で

きます。この操作は264ページの「再定義されたMainプロセス タイプ」で説明します。

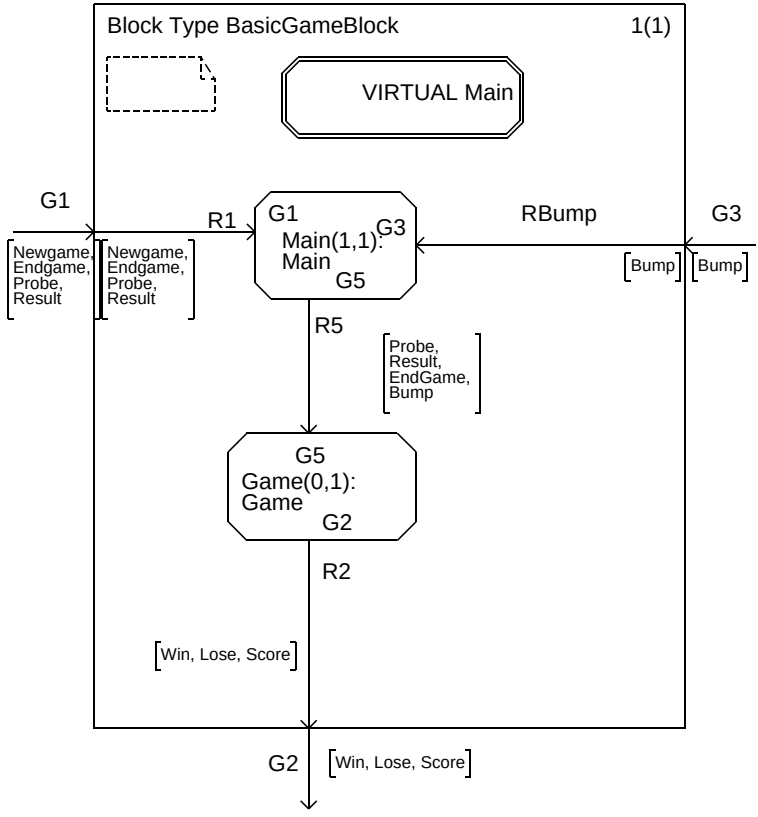


図177: BasicGameBlockブロック タイプ

5. Mainプロセス タイプを、コピーした`main.spt`に接続します。前の手順で[接続]ダイアログ ボックスの[サブストラクチャの展開]をオンにしている場合は、すでに接続されている場合もあります。

パッケージの使用

パッケージを使用するには、パッケージ参照シンボルにUSEステートメントを追加する必要があります。パッケージ参照シンボルは、以下の図のようにテキストシンボルに似たもので、フレーム シンボルのすぐ外に配置されています。

1. 基本機能を備えたDemonGameを作成するために、USEステートメントをシステム ダイアグラムに追加します。ここで、必ずパッケージ内のBasicGameBlockブロック タイプを実体化します。以下の図を参照してください。

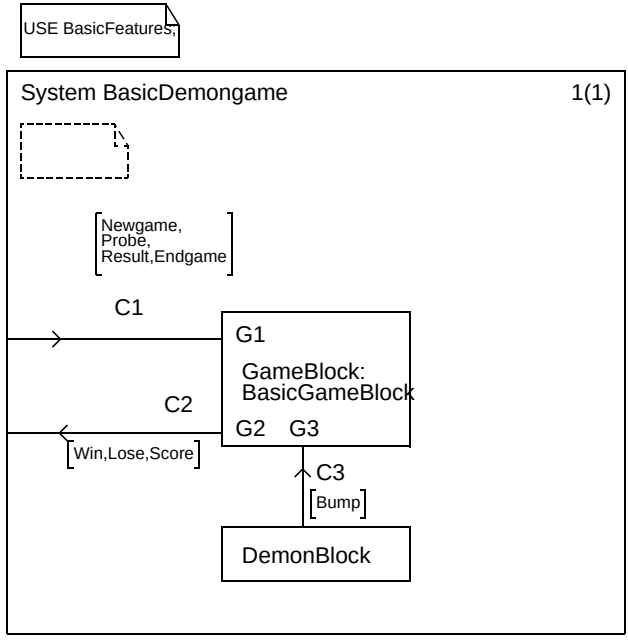


図178: パッケージのUSEステートメント

   – システム ダイアグラムは、`basicdemongame.ssy`などの名前で新しいファイルとして保存できます。

2. オーガナイザで、システム構造の既存のGameBlockを[未接続]に設定します。変更後のオーガナイザの表示は以下のようになります。なお、シンボルが表示される順序が図と異なる場合があります。

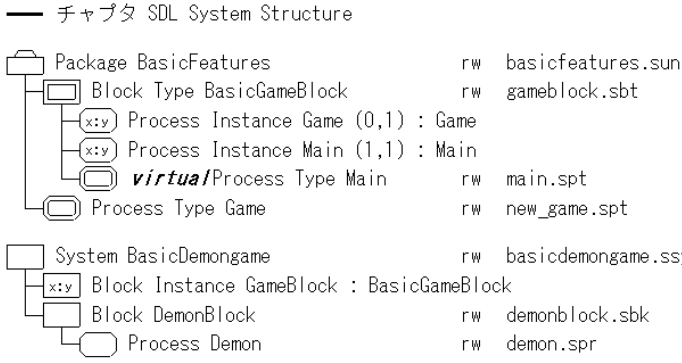


図179: BasicFeaturesパッケージを使用するBasicDemonGameシステム

3. 完成したシステムを分析して、演習を終了します。

メモ：

オーガナイザでパッケージ シンボルをアナライザの入力として選択してから[分析]コマンドを実行すると、パッケージを分析することができます。ただしこの場合、パッケージに対して意味分析の一部分のみが実行されます。アナライザは、パッケージとパッケージを使用するシステムとの間の一貫性をチェックしません。

完全な意味分析を実行するには、[分析]を実行する前にシステム ダイアグラムを選択しておく必要があります。

パッケージの再使用

すべての機能を持つDemonGameを作成するために、AdvancedFeaturesパッケージを作成して機能を追加します。AdvancedFeaturesの作成の際は、Basic Featuresパッケージを再使用します。

学習内容

- ほかのパッケージ内でのパッケージの再使用
- ブロック タイプの継承
- プロセス タイプの再定義

AdvancedFeaturesパッケージ

1. Basic Featuresパッケージと同じ手順でAdvancedFeaturesを作成します。257ページの図175を参照してください。

   – 現在オーガナイザ内にはBasicFeaturesとAdvancedFeaturesの2つのパッケージが表示されています。

以下の手順でAdvancedFeaturesパッケージを作成します。

2. Basic Featuresパッケージを使用するためにUSE句を定義します。
3. 新しい信号の宣言を追加します。
4. JackpotGameなどのプロセス タイプへ参照を追加します。
5. AdvancedGameBlockブロック タイプへの参照を追加します（AdvancedGameBlockでは、BasicGameBlockブロック タイプを継承し、再定義されたMainプロセス タイプを参照します）。
6. パッケージ ダイアグラムを`advancedfeatures.sun`というファイル名で保存します。

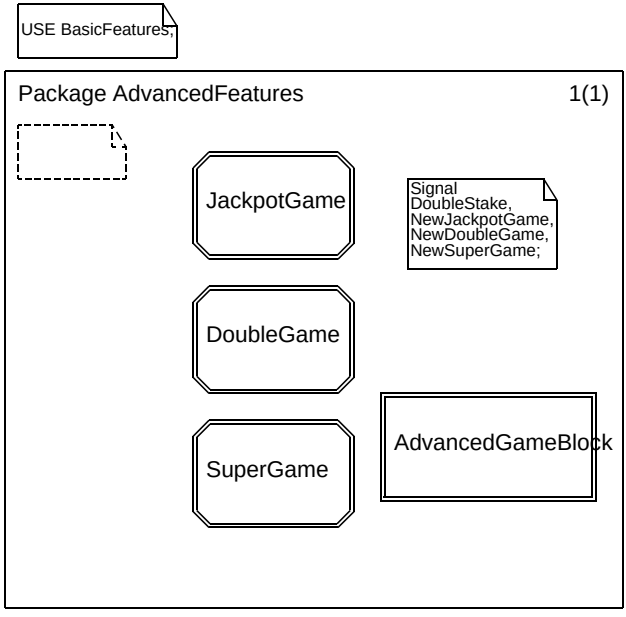


図180: AdvancedFeaturesパッケージ

AdvancedGameBlockブロック タイプ

AdvancedGameBlockブロック タイプのダイアグラムには、REDEFINEDが定義されたMainプロセス タイプへの参照があり、破線で表示されたMainインスタンス シンボルがあります。

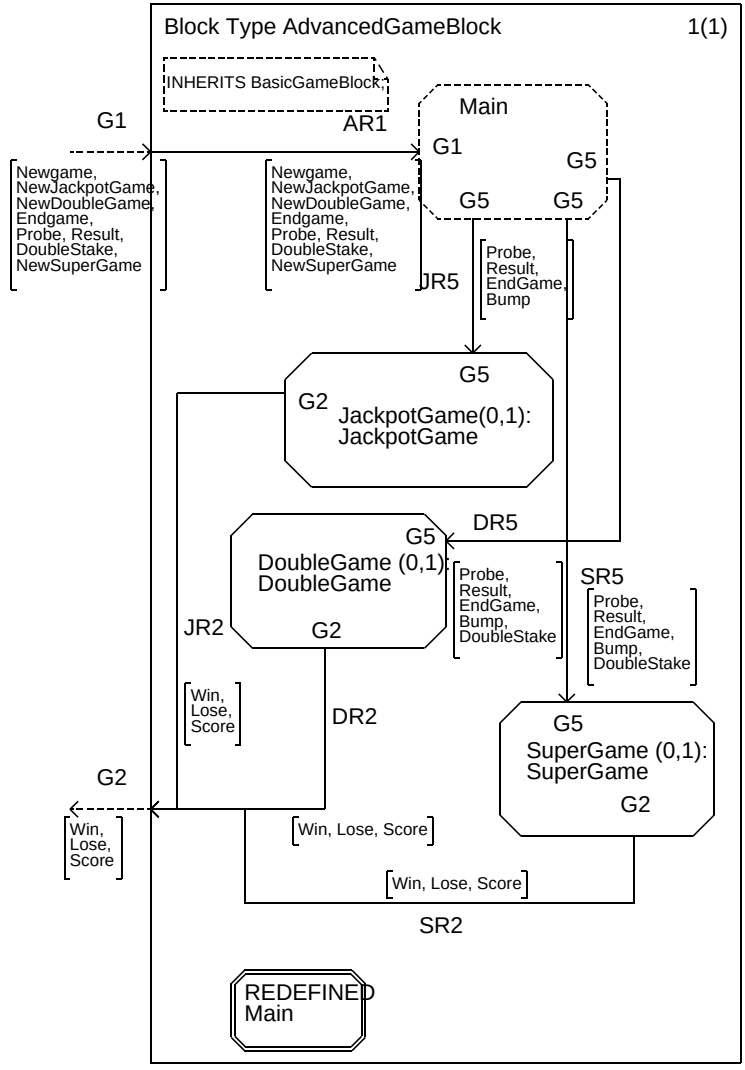


図181: AdvancedGameBlockブロック タイプ

- AdvancedGameBlockブロック タイプは、`advancedgameblock.sbt`というファイル名で保存されています。UNIXの場合は`$telelogic/sdt/examples/demongame/sdl92/packages`から、またWindowsの場合は`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥sdt¥examples¥demongame¥sdl92¥packages`から、このファイルをコピーし、オーガナイザを使用してダイアグラムをコピーしたファイルに接続します。

再定義されたMainプロセス タイプ

REDEFINEDが設定されたMainプロセス タイプには、BasicFeaturesパッケージ内のVIRTUALが設定されたMainプロセス タイプを暗黙的に継承しています。さらに、起動信号を受信するために必要な新しい機能のコードが追加されています。

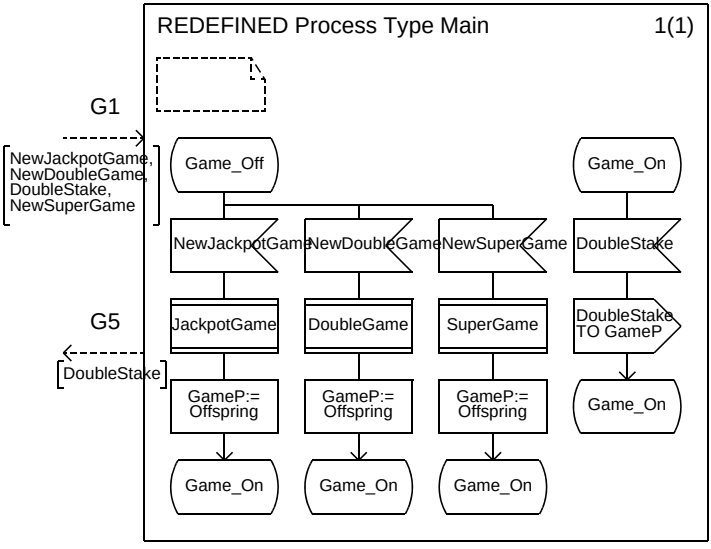


図182: 再定義されたMainプロセス タイプ

- REDEFINEDが定義されたMainプロセス タイプは、UNIXの場合はディレクトリ`$telelogic/sdt/examples/demongame/sdl92/packages`に、またWindowsの場合はディレクトリ`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥sdt¥examples¥demongame¥sdl92¥packages`に、`advancedmain.spt`というファイル名

で保存されています。このファイルをコピーし、オーガナイザでダイアグラムをファイルに接続してください。

AdvancedDemonGameシステムの作成

これまでの設定によって、以下のように比較的簡単にシステムを作成できます。

1. オーガナイザの[編集]メニューから[新規追加]を選択し、SDLシステムを追加します。AdvancedDemonGameなどのシステム名を設定して、`demongameadvanced.ssy`というファイル名で保存します。
2. SDLエディタを使用して、BasicDemonGameシステムの内容をコピーし、新しいシステムに貼り付けます。
3. USE BasicFeaturesに加えてUSE AdvancedFeaturesをシステムに追加します。
4. BasicGameBlockブロック タイプからAdvancedGameBlockへ参照を変更します。
5. C1の信号リストにJackpotGameなどの新しい信号を追加します。これでシステムは完成です。必要に応じて、システムの分析やシミュレーションを行います。

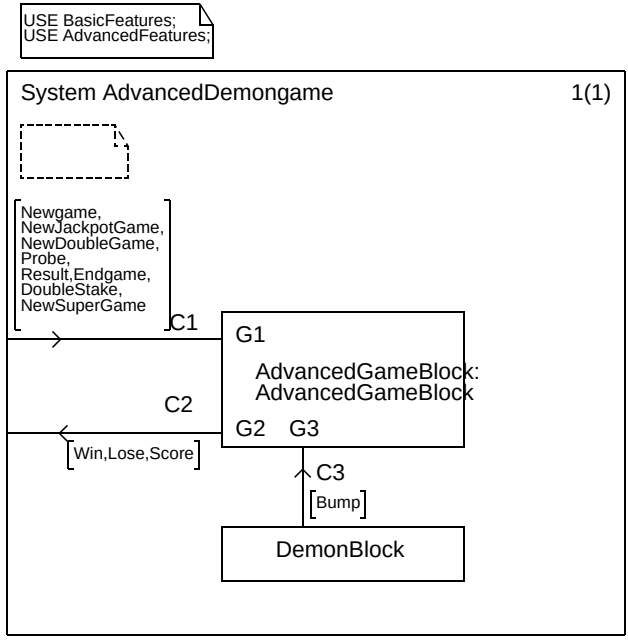


図183: AdvancedDemonGame システム

まとめ

このチュートリアルでは、SDL-92を使った小規模なSDLシステムを設計し、再使用可能にする方法について学習しました。そして、再使用可能なコンポーネントを作成することによって、機能の再使用や拡張に要する労力を削減できることがわかりました。

また、オブジェクト指向の設計体系をスムーズに導入するには、適切なシステムの設計が重要であることを学びました。

SDL-92をどれくらい理解できたかを確認するために、今度は完成したシステムを利用せずに自分自身で新しい機能を追加してみてください。

追加演習

以下に示す各項目は、新しい機能としてAdvancedDemonGameに追加することができます。これらの機能でAdvancedDemonGameを拡張してみてください。

1. システムが起動してから現在までの最高スコアを記録する機能の追加。なお、すべてのゲームで共通することですが、最高スコアは1つだけ存在するように設定します。
2. 現在までの最高スコアを達成したプレイヤーの名前を記録する機能の追加。名前は外部環境から入力されるものとします。
3. ゲームオーバー機能の追加。この機能は、現在のスコアが-100点などの所定の値より小さいかどうかをチェックし、所定の値より小さい場合ゲームを無効にし、プレイヤーが最初からゲームを始めなければならないようにします。

それでは、がんばってください。

> メモ：
>
> 上記の追加演習のヒントが、UNIXの場合は、`$telelogic/sdt/examples/demongame/sdl92/exercises`のディレクトリに、またWindowsの場合は`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥sdt¥examples¥demongame¥sdl92¥exercises`のディレクトリにそれぞれ保存されています。

付録: パッケージを使用したDemonGameのダイアグラム

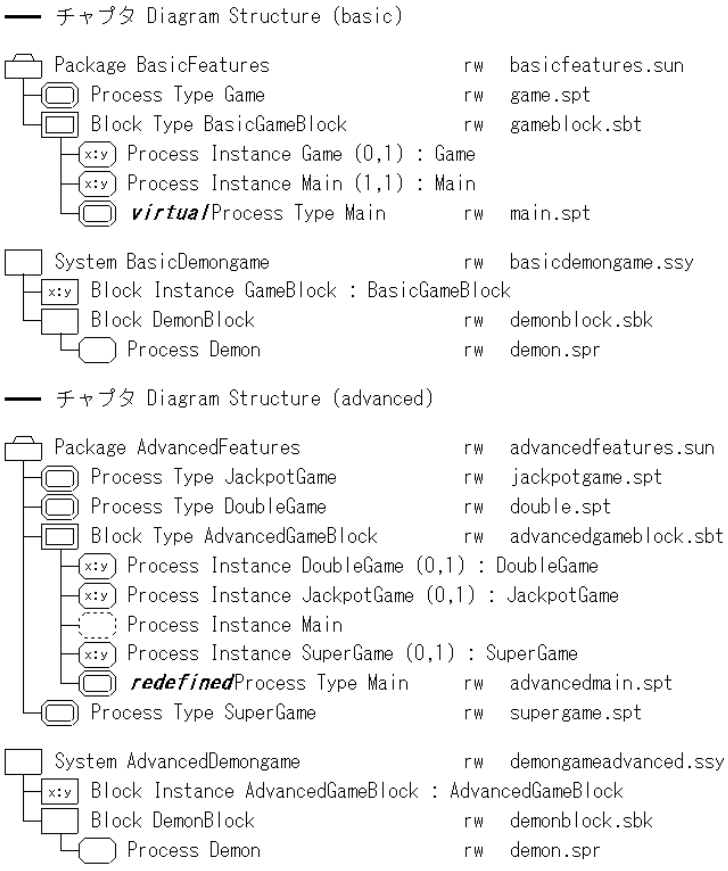


図184: 階層構造

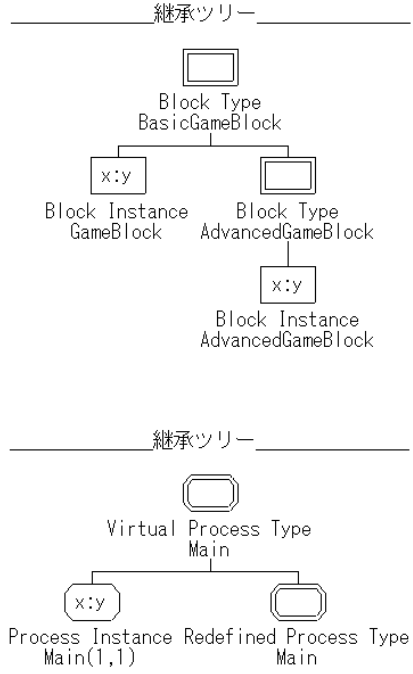


図185: ブロック タイプとMainプロセス タイプの継承ツリー

(Gameプロセス タイプの継承ツリーは253ページの図172に示されています)。

第 7 章

Cmicroターゲティング チュートリアル

このチュートリアルでは、ターゲティングの基本的な操作手順について説明します。使用するコンパイラとしては、Windowsの場合Microsoft Visual C、UNIXの場合gccまたはccを想定しています。

事前準備と表記規則

チュートリアルを学習する際は、SDL Suiteのツール、特にオーガナイザや、SDLアナライザ、SDLシミュレータの操作を理解しておく必要があります。これらのツールの学習を終了されていない方はこのマニュアルの以前の章に記載されているチュートリアルで学習されることをお勧めします。

このチュートリアルは、以下の表記法に従って記述されています。

- `<installation>`は、インストール ディレクトリを表します。これは、Windowsの場合は、`<システム ドライブ名>:¥IBM¥Rational¥SDL_TTCN_Suite6.3J`となります。また、UNIXの場合は、`$telelogic`になります。
- チュートリアルで説明するいくつかの手順は、WindowsとUNIXとの間の相違点がパスの区切り文字だけのものがあるため、区切り文字には「`/`」と「`¥`」の両方を混在して表記します。
- Windowsでは「ディレクトリ」のことを「フォルダ」と呼ぶことがありますが、このチュートリアルでは「ディレクトリ」に統一して表記します。

  このチュートリアルで使用するサンプル ファイルは、以下のディレクトリに格納されています。

  ```
  <installation>¥sdt¥examples¥cmicrotutorial¥wini386
  <installation>/sdt/examples/cmicrotutorial/sunos5
  ```

- Cmicroライブラリは以下のディレクトリに格納されています。

  ```
  <installation>¥sdt¥sdtdir¥wini386¥cmicro
  <installation>/sdt/sdtdir/sunos5sdtdir/cmicro
  ```

  Cmicroライブラリの説明については、『User´s Manual』の第66章、「The Cmicro Library」を参照してください。

はじめに

概要

このチュートリアルは3つのセクションに分かれています。最初のセクションでは、小規模のSDLシステムを作成し、そのシステムをSDLアナライザで生成します。ここでは、ターゲティング エキスパートを使用してできる構成内容と環境変数の作成方法について学びます。このセクションを読み終えると、ターゲット アプリケーションを構築できるようになります。

2番目のセクションでは、SDLを実際にテストし、SDLターゲット テスタの使用方法について学びます。

3番目のセクションでは、ターゲット アプリケーションからターゲット テスタのソースを削除する方法について学びます。

インテグレーション

ターゲティングは以下の方法で実装できます。

- ベア インテグレーション：

  SDLシステムは、ベア ターゲットで動作し、Cmicroカーネルによってスケジューリングされます。ほかのオペレーティング システム（OS）の介入はありません。

- ライト インテグレーション：

  SDLシステムは、Cmicroカーネルによってスケジューリングされますが、OSの1つのタスクとして実行し、OSの機能を使用することができます。

- タイト インテグレーション：

  すべてのSDLプロセス インスタンス集合とそのほかのタスクは、ターゲットのOSによってスケジューリングされます。

このチュートリアルでは、ライト インテグレーションの形式を使って実装します。これにより、ターゲットのアプリケーションを独立したOSタスクとして実行することができ、Cmicroカーネルを使ってSDLプロセスをスケジューリングできるようになります。

ターゲット テスタの通信

このチュートリアルでは、ターゲット テスタとターゲット アプリケーションとの通信はソケット（デフォルトではlocalhost、ポート 9000を使用）を使用して行います。

演習の準備

ページャ システム

このチュートリアルで使用するSDLシステムは、ページャ システムです。ページャは小さなハンドヘルド デバイスで、他の人と連絡をとるために使用されます。無線通信機能を備えているため、短いメッセージや電話番号などからなる特定の周波数の信号を受信できます。

ページャには、容量に制限はありますがメッセージを格納するためのデータバンク機能や、キーパッドおよびディスプレイなど、ユーザーとのインターフェイスとなる機器も備わっています。ページャの表示画面は小さいため、メッセージを読んだり削除したりするときは、画面をスクロールしながら操作することになります。

キーパッドは、右スクロール用、左スクロール用、削除用の3つのボタンから構成されています。メッセージを受信したときや、操作を間違えたとき、そして、すべきでないことをしようとしたときなどは、ページャは音を発して通知します。たとえば、データバンクが空なのにメッセージを削除しようとしたり、スクロールできない場所にスクロールしようとしたりしたときなどがこれにあたります。通常、ページャはメッセージを一定量しか格納できないため、いつかはいっぱいになります。

ページャの格納域が限界に達した場合は、受信メッセージを表示する前に警告メッセージが2秒間表示されます。

SDLオーバービューは、ブロックとプロセスに分割されたページャ システムを表しています。

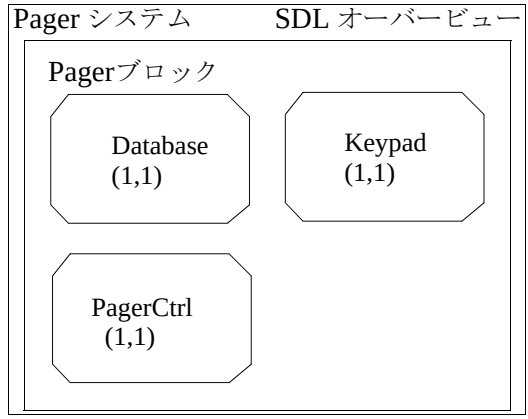


図186: Pager システムのオーバービュー

Process	Description
`Database`	ページャのメモリを構成するメッセージの配列を管理するプロセスです。データバンクのメッセージの順番を維持しながら、メッセージの格納、取得、および削除を行います。
`PagerCtrl`	このプロセスは、通常、システムのすべての入力と出力を処理します。このプロセスは、ユーザーが行ったキーパッドへの入力情報、無線通信で受信したメッセージ、およびデータベースに対する保存や削除などの情報を受け取ります。
`Keypad`	ユーザーの入力情報を信号に変換して`PagerCtrl`に送信するプロセスです。

ファイルの配信

このチュートリアルで使用するファイルは、次のディレクトリにあります。

```
<installation>/sdt/examples/cmicrotutorial/
<platform>/pager
```

`pager`プロジェクト ディレクトリ内には、`system`というサブディレクトリがあります。`system`ディレクトリにはページャ システムのSDL/GRファイルが保存されています。

さらに、システム ディレクトリに類似したディレクトリが用意されています。環境を自分でプログラミングしない場合は、ここにある環境設定ファイル`env.c`を使用することができます。

ターゲティング

準備 - ファイル構造

1. 作業用ディレクトリかローカル コンピュータのハードディスクに「`cmicrotutorial`」という名前の新しいディレクトリを作成します。以後、このディレクトリ名を`<MyTutorial>`と表記します。
2. `<installation>/sdt/examples/cmicrotutorial/<platform>/pager` ディレクトリを、ディレクトリ内のすべてのファイルやサブディレクトリと共に`<MyTutorial>`ディレクトリにコピーします。そしてコピーしたディレクトリ内のすべてファイルの書き込み制限を解除します。
3. オーガナイザで、`<MyTutorial>/pager/system`にあるページャ システム(`Pager.sdt`)を開きます。

ターゲティング エキスパートの使用

1. オーガナイザ ビューのシステム シンボルを選択します。
2. [生成]メニューからターゲティング エキスパートを起動します。ターゲティング エキスパートによって、デフォルトの分割ダイアグラム モデルが生成され、ディレクトリ構造がチェックされます。第59章「The Targeting Expert」2946ページの「Partitioning Diagram Model File」を参照してください。
3. オーガナイザで指定したターゲット ディレクトリが存在しない場合があるため、作成が必要な場合は作成を促されます。この場合は、[はい] をクリックしてください。

後で、ターゲティング エキスパートによってサブディレクトリ構造がターゲット ディレクトリに追加されます。詳細については、第59章「The Targeting Expert」2969ページの「Target Sub-Directory Structure」を参照してください。

> メモ：
> ターゲット エキスパートを初めて起動すると、初期ウィンドウが表示されます。[閉じる] をクリックすると、処理を進めることができます。ターゲティング エキスパートを起動するたびにこの初期ウィンドウが表示されることを避けたい場合は、[起動時に表示しない] チェック ボックスをオンにします。

ターゲティング エキスパートでの作業は、以下の4つステップに分類されます。

- ステップ1：必要なコンポーネントの選択
- ステップ2：インテグレーションの種類の選択
- ステップ3：生成プロセスの設定
- ステップ4：コンポーネントの実装

ターゲット エキスパートを初めて起動すると、操作方法を案内するアシスタントが自動的に起動されます。アシスタントを終了した場合、［ヘルプ］メニューの［アシスタント］を選択すれば再び起動するこたができます。

ヘルプ ビューワーが起動され、そこで、番号が表示されたボックスをクリックすると対応するマニュアルのページが表示されます。

ステップ1：必要なコンポーネントの選択

- 分割ダイアグラム モデルのコンポーネントをクリックします。
  デフォルトでは、componentという名前のコンポーネントに、完全なSDLシステムが生成されます。

> Hint:
>
> 配置エディタを使用してシステムを配置する場合、コンポーネントの名前は自由に付けることができます。システムの配置方法についての詳細は、『User´s Manual』の第40章、「The Deployment Editor」を参照してください。

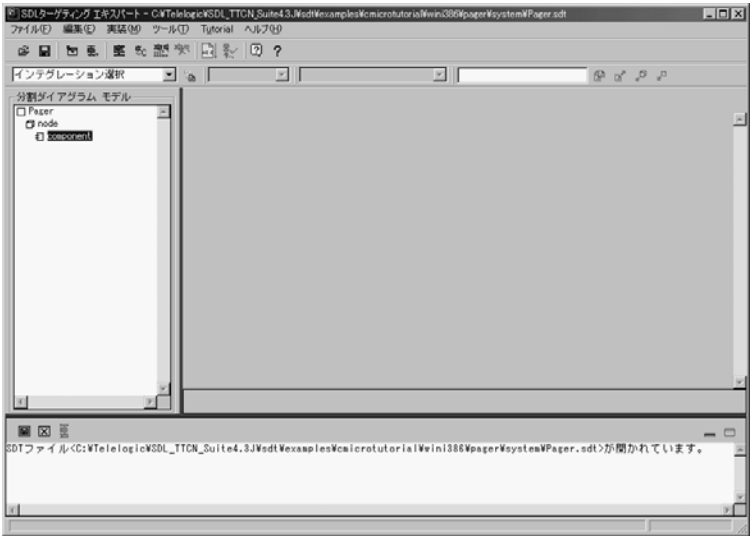


図187: ターゲティング エキスパートのメイン ウィンドウ

参考情報

- 分割ダイアグラム モデルに関する詳細については、第59章「The Targeting Expert」2946ページの「Partitioning Diagram Model File」を参照してください。
- 分割ダイアグラム モデルで選択可能な項目については、第59章「The Targeting Expert」2902ページの「Targeting Work Flow」を参照してください。

ステップ2：インテグレーションの種類の選択

1. メイン ウィンドウのインテグレーション ツール バーで、左にあるコンボ ボックスをクリックしてください。(または、分割ダイアログ モデルの要素のエントリを右クリックしてください). これは、図188に示されています。

図188: ターゲティング エキスパートのポップアップ メニュー

2. 表示されるポップアップ メニューに、定義済みインテグレーションをすべて含むツリー構造が表示されます。［ライト インテグレーション］の［アプリケーション テスト］を選択します。

SDL システムの妥当性がチェックされ、自動設定が行われます。

> Hint:
>
> 必要なファイルは`sdl_cfg.h`だけですが、SDL to Cコンパイラは、`componentname.c`、`componentname.ifc`、`componentname.sym`、および`sdl_cfg.h`を生成します。この処理が行われるのは、使用する機能と未使用の機能をSDL to Cコンパイラだけが把握しているためです（OO コンセプト）。

SDL to Cコンパイラが終了したら、ターゲティング エキスパートは次のことを行います。

- `env.c`ファイルを生成し、環境に送信するまたは環境から受信するSDL信号を一覧にします(イベント ログを参照してください)。生成されたファイルとライブラリ ファイルをコンパイルする前に、それぞれの信号を手動で適合させる必要があります。ターゲティング エキスパートは使用済みの信号を使用してスケルトンのみを生成するため、この作業が必要となります。

env.cのの編集方法は、280ページの「環境ファイルの編集」に記述されています。

- 既定のターゲット テスタ関連のオプションを生成します (sdtmt.opt)
- 既定の手動設定を生成します (ml_mcf.h)

> Hint:
>
> インテグレーションが選択されていると、ターゲティング エキスパートが自動的に既定のコンパイラを設定します。既定のコンパイラ名は、環境設定から付けられます。
>
> 環境設定で既定のコンパイラに設定したものとは別のコンパイラが必要な場合は、インテグレーション ツール バーのコンボ ボックスで変更することができます。

参考情報

のパッケージにはないインテグレーションを実行する場合は、インテグレーション ポップアップ メニューから[<ユーザー指定>]を選択してください。これにより、使用するハードウェアに必要なすべての設定が可能になります。インテグレーションがアクセス可能になるように、インテグレーション ポップアップ メニューで設定することもできます。

生成されるファイルの内容は以下のとおりです。

- component.c
  C言語の関数を使用した際のSDLシステムでの動作説明。
- component.ifc
  環境関数のヘッダ ファイルです。PIdや信号のデータ型定義などが記述されます。
- component.sym
  SDLシンボルに関する情報が記述されます。SDLターゲット テスタでSDLシステムをトレースするときに必要です。
- sdl_cfg.h
  Cmicroカーネルの自動設定用ファイルです。つまり、タイマを使う場合、タイマを有効にする定義がこのファイルに取り込まれます。

環境ファイルの編集

このセクションでは、env.cファイルに環境関数を記述する方法について学習します。

Hint:

予備のenv.cもあります。このファイルは次のディレクトリからコピーできます。

<MyTutorial>/pager/prepared into

<MyTutorial>/pager/target/pager._0/Application_TEST

このファイルを使用するときは、ライト プロテクションを解除してください。

予備のファイルと生成ファイルはまったく同じものではありません。

メモ：

ターゲティング エキスパートによってテキスト エディタが起動されます。既定では、あらかじめ組み込まれているエディタが使用されます。エディタは［ツール］メニューの［カスタマイズ］で変更できます。

1. ［編集］メニューの［環境の編集］をクリックして、env.cのファイルを開きます。

注意！

env.cファイルでは、以下の行の間でのみコードを編集できます。

```
/* BEGIN User Code ... */
/*   END User Code ... */
```

これは、以下の理由によるものです。

ターゲティング エキスパートが2回目にファイルを生成するとき、これらのセクションのコードが新しいファイルに読み取られ、コピーされます。上記の行の間に記述されたコードのみが変更されずに残ります。

以下の行内の文字を編集しないでください。

```
/* BEGIN User Code ... */
/*   END User Code ... */
```

2. globalセクション内の以下の行を見つけてください。

```
/* BEGIN User Code (global section)*/
/* It is possible to define some global variables here */
/* or to include other header files. */ */
```

このチュートリアルでは、コンソール アプリケーションを行う方法について説明しています。シミュレーションでは、画面とキーボードをユーザーとのインターフェイスに使います。したがって、使用する ヘッダ ファイルをインクルードする必要があります。

スタイルを向上させるために、いくつかの定義を使います。また、いくつかのグローバルな変数や関数を実装する必要があります。ディスプレイなど、環境内でシミュレーションしなければならないものがある場合には、この関数を呼び出します。

```
/* or to include other header files. */
```

の行の後に、以下のコードを挿入します。

```
#if defined(MICROSOFT_C)
  #include <conio.h>
#else
  #include <stdio.h>
#endif

#define key_was_pressed 1
#define key_not_pressed 0

int  KeySignalPresent  = 0;
char LastKeyPressed;
```

xInitEnv()

3. システムの起動時は、ほとんどの場合、初期メッセージが表示されます。`xInitEnv()`関数を使用することでこの操作を行えます。

```
printf("-------- Welcome to Pager system--------¥n¥n");
printf("get message :  0 to 4¥n");
printf("scroll right:  r¥n");
printf("scroll left :  l¥n");
printf("delete      :  d¥n¥n");

/* BEGIN User Code (init section) */
```

と

```
/*   END User Code (init section) */
```

の間にコードを挿入します。

xInEnv()

4. 次に、外部環境から送信されるデータの処理を記述する必要があります。このチュートリアルでは、キーボードからの入力の処理がこれに相当します。

> 注意！
>
> 環境ファイルにはカーネルの処理を中断する関数を使わないでください。
>
> 外部環境は、Cmicroカーネルのサイクルごとにポーリングされます。したがって、`getchar()`のようなカーネルの処理を中断する関数は使えません。このような関数は、カーネルを停止するため、SDLシステムの処理が継続できなくなります。

データは以下の方法によって処理できます。

キーを押すと、0から4までの数値はメッセージとして認識され、文字「r」、「l」そして「d」はスクロール用と削除用のコマンドとして認識されます。

`xInEnv()`関数コードの以下の行を含む場所に移動してください。

```
/* BEGIN User Code (variable section)*/
/* It is possible to define some variables here */
/* or to insert a functionality which must be polled */
```

これらの行の下に以下のコードを挿入します。

```
char my_inkey;

KeySignalPresent = key_not_pressed;

#if defined(XMK_UNIX)
  my_inkey = 0;
  if((my_inkey = getchar_unlocked()) != 0)
  {
    KeySignalPresent=key_was_pressed;
  }
#elif defined(MICROSOFT_C)
  if (kbhit())
  {
    my_inkey=_getch();
    KeySignalPresent=key_was_pressed;
  }
#endif

if (KeySignalPresent==key_was_pressed)
{
  if ((my_inkey == 'r') ||
      (my_inkey == 'l') ||
      (my_inkey == 'd') ||
      ((my_inkey>='0')&&(my_inkey<='4')))
    LastKeyPressed = my_inkey;
  else
    LastKeyPressed=0;
}
else
 LastKeyPressed = 0;
```

5. `xInEnv()`関数の以下の行を検索します。

```
/* BEGIN User Code <ScrollRight>_1 */
  if (i_have_to_send_signal_ScrollRight)
/*   END User Code <ScrollRight>_1 */
```

手順2では、`LastKeyPressed`変数を実装しました。手順4では、最後に押されたキーの値を持つ`my_inkey`を`LastKeyPressed`に割り当てました。`if()`ステートメントを次の式に修正します。

```
if (LastKeyPressed == 'r')
```

6. xInEnv()関数の以下の行を検索します。
   GLOBALPID(Who_should_receive_signal_ScrollRight,0));

   ScrollRight信号を受信するプロセス タイプIDを挿入する必要があります。プロセス タイプIDのオーバービューを取得するには、ターゲティング エキスパートにより表示されるダイアログ ウィンドウを確認してください。使用されているすべてのプロセス タイプIDは、このダイアログに含まれています。

Figure 189: プロセスタイプID ダイアログ

   このダイアログからXPTID_Keypadエントリを選択してクリップボードにコピーします。その後、以下のようにenv.c asに貼り付けます。

   GLOBALPID(XPTID_Keypad,0));
   これにより、xInEnv()関数から制御が返ると、信号が処理されます。

7. xInEnv()関数で以下の行を検索します。
   ```
   /* BEGIN User Code <ScrollLeft>_1 */
     if (i_have_to_send_signal_ScrollLeft)
   /*   END User Code <ScrollLeft>_1 */
   ```

   if() ステートメントを次のように修正します。
   ```
   if (LastKeyPressed == 'l')
   ```

8. xInEnv()関数の以下の行を検索します。
   GLOBALPID(Who_should_receive_signal_ScrollLeft,0));
   手順6の記述に従ってXPTID_keypadをコピーします。

```
GLOBALPID(XPTID_Keypad,0));
```

9. xInEnv()関数の以下の行を検索します。

```
/* BEGIN User Code <Delete>_1 */
  if (i_have_to_send_signal_Delete)
/*   END User Code <Delete>_1 */
```

if() ステートメントを次のように修正します。

```
if (LastKeyPressed == 'd')
```

10. xInEnv()関数の以下の行を検索します。

```
GLOBALPID(Who_should_receive_signal_Delete,0));
```

手順6の記述に従ってXPTID_keypadをコピーします。

```
GLOBALPID(XPTID_Keypad,0));
/* It is possible to insert any user code here. */ */
```

11. ページャ システムをシミュレートしますが、実物のターゲット ハードウェアは使用しないので、いくつかのメッセージを定義しておく必要があります。

xInEnv()関数の以下の行を検索します。

```
/* BEGIN User Code <ReceivedMsg>_1 */

if (i_have_to_send_signal_ReceivedMsg)

/*   END User Code <ReceivedMsg>_1 */
```

if() ステートメントを次のように修正します。

```
if ((LastKeyPressed>='0')&&(LastKeyPressed<='4'))
```

このIfステートメントは、キーボードで入力されたキーが定義済みのキーかどうかをチェックします。

12. 次の空のユーザー コードセクションに移動し、以下の行を挿入します。

```
char *p;
  xmk_var.Param1.MyText = (SDL_Charstring)NULL;

  switch(LastKeyPressed)
  {
    case '0':
         p = " Hello user";
        xmk_var.Param1.TelNumber = 12345;
        break;
    case '1':
        p = " How do you feel doing targeting?";
        xmk_var.Param1.TelNumber = 555555;
        break;
    case '2':
        p = " Targeting is all so easy!";
        xmk_var.Param1.TelNumber = 987654;
        break;
    case '3':
        p = " I only wanted to check if it works.";
        xmk_var.Param1.TelNumber = 45454;
        break;
```

```
    case '4':
         p = " ... and it works very fine!";
         xmk_var.Param1.TelNumber = 911911;
         break;
    default :
        break;
  xAss_SDL_Charstring(&(xmk_var.Param1.MyText), p,
XASS_AC_ASS_FR);
```

この部分に0から4までの番号に対応するメッセージが格納されます。どのメッセージを環境に割り振るかはswitchステートメントで決定します。

メモ：

ここでは実際のインターフェイスを使用しないので、このメッセージの受信方法はヘルパー機能にすぎません。

`xmk_var.Param1.MyText = (SDL_Charstring)NULL;`
は、ReceivedMsg信号のパラメータであるパラメータ メッセージの要素MyTextにNULLを設定することを意味します。

信号ReceivedMsgとパラメータ メッセージはSDLシステムで宣言されています。

`xAss_SDL_Charstring(&(xmk_var.Param1.MyText), p, XASS_AC_ASS_FR);`はメモリをポインタpに割り当てます。

メモ：

SDL文字列がCのchar型の配列にマップされている場合は、この配列（インデックス0）の最初の文字は内部でのみ使用可能となります。つまり、テキスト メッセージはインデックス1で開始する必要があります。この処理は、上で示した実装でテキストの前にスペースを入れることで行えます。

13. xInEnv()関数の以下の行を検索します。
```
GLOBALPID(Who_should_receive_signal_ReceivedMsg,0));
```
以下に示すように、ステートメントを変更します。
```
GLOBALPID(XPTID_PagerCtrl,0));
```

xOutEnv()

> 注意！
>
> xOutEnv()関数はxmk_TmpDataPtrという名前のポインタを提供します。このポインタで参照されるデータは、xOutEnv()関数が処理されていれば有効です。
>
> 関数を終了した後にデータを使用したい場合は、自分で定義した変数にコピーします。

14. 以下のコードセクションを検索します。

```
case CurrentMsg :
          {
            /* BEGIN User Code <CurrentMsg>_1 */
/* Use (yPDP_CurrentMsg)xmk_TmpDataPtr to access
the signal's parameters */
/* ATTENTION: the data needs to be copied..  Otherwise it
*/
/* will be lost when leaving xOutEnv */
            /*   END User Code <CurrentMsg>_1 */

            /* BEGIN User Code <CurrentMsg>_2 */
            /* Do your environment actions here. */
            xmk_result = XMK_TRUE; /* to tell the caller
that */
                                   /* signal is consumed
*/
            /*   END User Code <CurrentMsg>_2 */
          }
```

このコードは信号CurrentMsgを処理します。選択されたメッセージは画面に表示されます(電話番号、メッセージ、現在のメッセージの位置とメッセージの合計数)。/* Do your environment actions here. */の後に以下のコードを挿入します。

```
printf("¥r                                        ");
printf( "¥rCurrentMessage:  %6d  %s  (%d/%d)",
     ((yPDP_CurrentMsg)xmk_TmpDataPtr)->Param3.TelNumber,
     ((yPDP_CurrentMsg)xmk_TmpDataPtr)->Param3.MyText+1,
     ((yPDP_CurrentMsg)xmk_TmpDataPtr)->Param1,
     ((yPDP_CurrentMsg)xmk_TmpDataPtr)->Param2);
xFree(&(((yPDP_CurrentMsg)xmk_TmpDataPtr)->Param3.MyText));
```

行

```
xFree(&(((yPDP_CurrentMsg)xmk_TmpDataPtr)-
>Param3.MyText));
```

は、信号を送信するときにカーネルに割り当てられたメモリを解放します。

15. xInEnv()関数の以下の行を検索します。

```
case ServiceMsg :
```

```
  {
    /* BEGIN User Code <ServiceMsg>_1 */
    /* Use (yPDef_Close*)xmk_TmpDataPtr to access
       the signal's parameters */
    /* ATTENTION: the data needs to be copied.
                  Otherwise it */
     /*               will be lost when leaving xOutEnv
*/
     /*   END User Code <ServiceMsg>_1 */

     /* BEGIN User Code <ServiceMsg>_2 */
     /* Do your environment actions here. */

     xmk_result = TRUE; /* to tell the caller that */
     /* signal is consumed      */
     /*   END User Code <ServiceMsg>_2 */
  }
```

上記のコード部分で、ServiceMsg信号を処理します。このデータの処理は前の手順と同じです。`/* Do your environment actions here. */`に続けて、以下のコードを記述します。

```
printf("¥r
");
printf( "¥rServiveMessage: %s",
   ((yPDP_ServiceMsg)xmk_TmpDataPtr)->Param1+1);
xFree(&(((yPDP_ServiceMsg)xmk_TmpDataPtr)->Param1));
```

16. `xOutEnv()`関数にある以下のコードを検索します。

```
case ShortBeep :
     {
       /* BEGIN User Code <ShortBeep>_1 */
       /*   END User Code <ShortBeep>_1 */

       /* BEGIN User Code <ShortBeep>_2 */
       /* Do your environment actions here. */
          xmk_result = XMK_TRUE; /* to tell the caller
that */
                                  /* signal is consumed
*/
       /*   END User Code <ShortBeep>_2 */
     }
     break ;
```

このコードは信号`ShortBeep`を処理します。ページャがメッセージを受信したときやユーザーが不正な操作をしたときにビープ音で知らせます。これは`/* BEGIN User Code <ShortBeep>_1 */`コードの後に挿入します。

```
putchar(07);
```

17. `xOutEnv()`関数にある以下のコードを検索します。

```
 case LongBeep :
     {
       /* BEGIN User Code <LongBeep>_1 */
       /*   END User Code <LongBeep>_1 */
```

```
        /* BEGIN User Code <LongBeep>_2*/
        /* Do your environment actions here. */
           xmk_result = XMK_TRUE; /* to tell the caller
that */
                                     /* signal is consumed
*/
        /*   END User Code <LongBeep>_2*/
      }
      break ;
```

これは/* BEGIN User Code <LongBeep>_1 */コードの後に挿入します。

```
putchar(07);
putchar(07);
```

外部環境の終了

このチュートリアルでは、外部環境の終了処理を行う必要はありません。マイクロプロセッサ ハードウェアなど、そのほかの場合には、終了処理が必要になることがあります。

`env.c`ファイル内の`xCloseEnv()`関数で以下のコードを探します。

```
/* BEGIN User Code (close section) */
/* Do the actions here to close your environment */
/*   END User Code (close section) */
```

必要なコードを挿入してください。

ステップ3：生成プロセスの設定

1. 分割ダイアグラムの［アプリケーション テスト］の下にある項目をクリックします。設定の追加または削除が必要な場合には、設定を編集できます
2. 保存］をクリックしてダイアログを閉じます。

このセクションのチュートリアルでは設定を修正する必要はありませんが、セクションの後半にある、297ページの「テスタを使用せずにTarget EXEを実行する」で変更を行います。

参考情報

各領域の簡単な説明を以下に示します。

- *Compiler / Linker / Make*

  この領域のすべての項目は、コンパイラ、リンカーおよびメイクツールによって設定できます。

  ここで、*Additional Compiler*についての補足説明を以下に示します。たとえば、生成したファイルのコンパイルにANSI Cコンパイラを使う必要があり、

オブジェクトのリンク先のオブジェクトが記述されているファイルをC++コンパイラでコンパイルしなければならない場合を仮定します。このとき、リンク先のファイルと使用するコンパイラに関する情報は*Additional Compiler*のセクションに記述できます。

詳細については、第59章「The Targeting Expert」2906ページの「Configure Compiler, Linker and Make」を参照してください。

- *ターゲット ライブラリ*

  この領域に定義や値を設定することにより、ターゲット ライブラリをスケーリングできます。詳細については、第59章「The Targeting Expert」2922ページの「Configure and Scale the Target Library」を参照してください。

  すべての設定は、`ml_mcf.h`という名前のファイルに保存されます。

- *ターゲット テスタ*

  この領域に定義や値を設定することにより、Cmicroテスタの機能をスケーリングできます。詳細については、第59章「The Targeting Expert」2923ページの「Configure the SDL Target Tester (Cmicro only)」を参照してください。

  すべての設定は、`ml_mcf.h`という名前のファイルに保存されます。

- *Host Connection*

  この領域には、ホストへの接続に使うパラメータを設定できます。たとえば、メッセージ コードの記述や実行形式名などを設定できます。Host Connectionの構成は、常にsdtmt.optファイルに格納されます。このファイルは、SDLターゲット テスタに必須です。詳細については、第59章「The Targeting Expert」2924ページの「Configure the Host (Cmicro only)」を参照してください。

ステップ4：コンポーネントの実装

1. ［アプリケーション テスト］が選択されている場合は、デフォルトで表示されるダイアログで、［コードの分析および生成］と ［環境関数］の2つのチェック ボックスを選択する必要があります。

2. コードの生成と実装を開始するには、［完全な実装］ボタンをクリックします。

SDL to Cコンパイラのコード生成が終了すると、ターゲティング エキスパートによって`env.c`が再生成されます（これによりこのファイルへの任意の修正が可能

です)。次に、メイクファイルが所定の設定にしたがって生成され、コードがコンパイルおよびリンクされます。

ターゲティング エキスパートがSDLターゲット テスタを開始します。

SDLターゲット テスタの使用

SDLシミュレータとSDLターゲット テスタとの違い

SDLターゲット テスタとSDLシミュレータとの相違点は、SDLシステムがターゲット ハードウェアで動作すること、および、実行されているSDLシステムが、SDLターゲット テスタのホスト パートが動作しているホスト システムへメッセージを送信することです。

このCmicroチュートリアルで使用するSDLターゲット テスタのホスト パートは、ターゲットとして生成したSDLシステムで動作します。

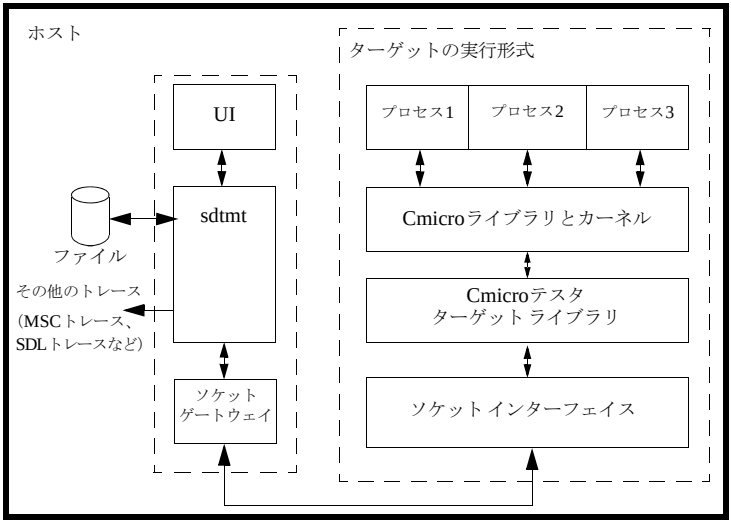


図190 ホスト シミュレーションのダイアグラム

このチュートリアルでの制限

ここでは、信号優先度、エラー チェック、テスタなど、Cmicroがサポートする機能の中で、プリエンプティブCmicroカーネル以外のすべての機能を使用できます。Cmicroカーネルを使用できない制限は、Cmicroが、小規模なスタンドアロン ハードウェア（ベア インテグレーション）をターゲットに設計されていることによるものです。

参考情報

- 現実のターゲット ハードウェアの動作についての詳細は、『User´s Manual』の第66章、「The Cmicro Library」を参照してください。
- makefileを編集する場合や参照する場合は、ターゲティング エキスパートの［編集］メニューの［Makefile］を使用してください。詳細については、『User´s Manual』の第59章、「The Targeting Expert」を参照してください。

ページャ システムのテスト

SDLターゲット テスタの実行

SDLターゲット テスタは、ターゲティング エキスポートから、または以下の方法で開始できます。

ターゲティング エキスパートの［ツール］メニューの［SDL］から［SDLターゲット テスタ］を選択し、クイック ボタン バーにあるクイックボタンを押します。

開始されたら、以下のステップを順番どおりに実行する必要があります。

1. ［Communication］グループの［StartGateway］または［Execute］メニューの［StartGateway］をクリックして、実行形式との通信を開始します。
2. ターゲティング エキスパートに戻ってメニュー エントリを選択します。
   ［チュートリアル］メニューの［ターゲットの起動（Windows）］またはそれに順ずるもの
   ［チュートリアル］メニューの［ターゲットの起動（UNIX）］

> Hint:
>
> ［チュートリアル］メニューは、他のSDLシステムでは使用できない設定可能なメニューです。メニュー エントリの作成についての詳細は、『User´s Manual』の第59章「The Targeting Expert」の2881ページ、「Configurable Menus」を参照してください。

> メモ：
>
> Windowsでは、独立したコマンド プロンプト ウィンドウからターゲット アプリケーションが開始されます。
>
> UNIXでは、独立したxtermウィンドウからターゲット アプリケーションが開始されます。

3. SDLターゲット テスタとターゲットとの通信が確立されます。SDLターゲット テスタは「Go Forever」で開始するメッセージを表示します。関連するボタンは実行グループにあります。ボタンをクリックしてください。
ターゲット アプリケーションのすべてのメッセージは、起動したUNIXシェルまたはDOSコマンド プロンプトに表示されます。ページャは、282ページの「xInitEnv()」で実装した初期メッセージを表示します。

4. 0、1、2、3、4のキーを押すことでメッセージを受信でき、rキーとlキーを押すとスクロールします。メッセージを削除するにはdキーを押してください。
外部環境の実装についての詳細は、282ページの「xInEnv()」を参照してください。

5. 最後に受信したメッセージ、現在のメッセージ数そして受信メッセージの合計数がページャに表示されます。スクロール、削除、および新しいメッセージの受信もできます。受信メッセージの数が上限である3通に達しているときに新しいメッセージを受信すると、一時的にメッセージは保存され、メモリがいっぱいのためメッセージを受信するにはメモリを解放する必要があるという内容の警告が2秒間表示されます。

6. キーボードのdキーを押すと、最後に受信したメッセージが削除され、新たに受信したメッセージが表示されます。

> メモ：
>
> ターゲット アプリケーションを終了するには、CTRLキーを押しながらCキーを押してください。

SDLターゲット テスタ コマンド

SDLターゲット テスタにはボタンのグループがあり、各ボタンはテスタ コマンドに対応しています。コマンドを入力するには、ボタンをクリックするか、テスタの下に表示されているコマンド ラインを使用します。コマンド ラインに「help」と入力すると、SDLターゲット テスタ コマンドの一覧が表示されます。

テスタのいくつかのコマンドに関する簡単な説明を以下に示します。詳細については、第67章「The SDL Target Tester」を参照してください。

MSCエディタによるSDLシステムのトレース

SDLターゲット テスタでシステムをテストしているときは、SDLシミュレータのGUIのように、MSCトレースを生成することができます。

- [Trace]グループで、たとえば、MSCトレースを開始するには、[Start MSC]ボタンをクリックします。
- 画面に表示するかファイルに出力するかを選択できます。

SDLエディタによるSDLシステムのトレース

ターゲット システムは、SDLエディタを使ってトレースすることができます。

- トレースを開始するには、SDLターゲット テスタのコマンド ラインに「`start-sdle`」と入力するか、[Start SDLE]ボタンを使用します。

MSCトレースとSDLトレースの機能は、システムを把握するのに役立ちます。

ターゲット情報

ターゲットの設定についての詳しい情報を得るには、SDLターゲット テスタのボタン エリアにある［Configuration Group］を開き、［Target］をクリックして現在のターゲットの設定を取得する必要があります。

カーネルについての情報を取得するには確認グループを開く必要があります。キューを押すとシステム内部にあるキューの現在の状況がわかります。現在のシステム上にあるホールドの最大値と信号の量を参照できます。確認グループのほかのボタンをクリックすることで実行しているシステムの詳しい情報を取得できます。

メモリ

`?memory`コマンドを使用すると、現在のメモリの状態を参照できます。

1. 293ページの「SDLターゲット テスタの実行」の記述に従ってページャ システムを開始します。
2. コマンド ラインに`?memory`または`?m`を入力します。メモリ プール サイズや現在のメモリの使用状況をチェックできます。

この時点で、メモリがページャ システムでどのように処理されるのかを確認することができます。プールにある現在のブロック数は4で最大数が5であることに注意してください。

3. ターゲット アプリケーションに切り替え、1から4のいずれかのキーを押してメッセージを取得します。

4. ターゲット テスタに戻り、?mコマンドを再び実行します。割り当てられているブロックが2つ増えているのが確認できるはずです。最後のメッセージを削除すると、メモリが再び開放され、ブロックが4つになっているのを確認できます。

ブレークポイントとキュー

システムのデバッグには、[Breakpoints]グループを使います。ブレーク ポイントは、信号入力やプロセスの状態に対して設定できます。ブレーク ポイントに到達したときは、[Continue]ボタンでシステムの動作を再開できます。

1. 293ページの「SDLターゲット テスタの実行」の記述に従ってターゲットを再び開始します。
2. ［Breakpoints］グループを展開しブレーク入力を選択します。
3. これで、プロセスIDを選択できます。この例では、キーパッドIDを使用します。
4. 信号IDリストが表示されたら、削除する信号を選択します。
5. 設定された入力のブレークポイントがテキスト エリアに表示されます。ターゲット アプリケーション ウィンドウに切り替え、最初にいくつかのメッセージを挿入した後にdキーを押します。この操作では何も起こりません。
6. テスタに再び切り替えます。この時点で、?memoryコマンドを使用したり、?queueコマンドを使用してキューの状態を見たりすることができます。
7. システムの状態を検証したら［Continue］をクリックします。

Keypadプロセスで信号の削除が発生するたびにシステムが停止するので、BAコマンドを入力してブレークポイントを削除してください。

テスタを使用せずにTarget EXEを実行する

ターゲット テスタを開始せずにターゲットの実行形式を実行することもできます。以下のことを最初に実行してください。

1. ターゲット アプリケーションとターゲット テスタを終了していなければ、終了します。
2. ターゲティング エキスパートに切り替えます。

以下では、ターゲット アプリケーションからターゲット テスタのソース コードを削除する方法について説明します。

1. 分割ダイアグラム モデルの［アプリケーション テスト］の下にあるターゲット テスタのエントリを押します。これで、選択されたターゲット テスタのフラグすべてが使用できなくなります。それらをオフにしようとしても、定義済みインテーグレーションが妨害するのでオフにすることはできません。

   それらのフラグにアクセスするために、ターゲティング エキスパートによって「アドバンス モード」が提供されています。(詳細は、『User´s Manual』の第59章「The Targeting Expert」の2900ページ、「Advanced Mode」を参照してください)。

2. ［ツール］メニューの［カスタマイズ］を選択するとダイアログがポップ アップします。［アドバンス モード］チェック ボックスをオンにし、ダイアログの［OK］ボタンをクリックします。

> 注意！
>
> ターゲティング エキスパートを入力するたびに、アドバンス モードがオンになります。ほかの定義済みインテーグレーション設定の制限を活用できるように、アドバンス モードがオフになっていることを確認してください。

3. ［アプリケーション テスト］が分割ダイアグラム モデルで選択されたときに表示されるダイアログで［実行］タブを選択する必要があります。［テスト アプリケーション］グループ ボックスでは、ラジオ ボタンは何も押さないでください。
   これで、ターゲット アプリケーションが正常に構築された後に、ターゲット テスタが実行されなくなります。
4. 分割ダイアグラム モデルで［ターゲット テスタ］を選択し、［ターゲット テスタを使用する］チェック ボックスをオフにします。以下のウィンドウで

［OK］をクリックします。(フラグがオフになっている場合)。最後のダイアログが実行された後は、［テスタ］タブにあるすべてのチェック ボックスがオフになっている必要があります。

5. ダイアログの下にある［保存］ボタンをクリックします。

> Hint:
>
> 新しい手動設定ファイル`ml_mcf.h`は、未定義のターゲット テスタ フラグと共に生成されます。

6. ［実装］ボタンをクリックします。(`ml_mcf.h`が修正されたことにより、すべてのファイルが再びコンパイルされます。)

コンパイルとリンクが完了すると、［チュートリアル］メニューの［ターゲットの起動］を使用してターゲットを起動でき、SDLターゲット テスタを使用せずにターゲットを使用できるようになります。

第 8 章

チュートリアル: ASN.1 データ型の使用

このチュートリアルでは、SDL SuiteのASN.1データ型と値の使い方について説明します。ここでは、ASN.1モジュールのSDLダイアグラムへのインポートおよびSDLダイアグラムでの使用の方法、コードの生成方法、BERまたはPERを使ってのASN.1データ型のエンコードおよびデコードの方法を学びます。

このチュートリアルには、ソースコードによるASN.1データ型の作成からASN.1データ型の実装に至るすべての手順が掲載されています。

機能とワークフローを解説するため、チュートリアル全体を通じて少しずつ例が示されています。SNMPプロトコルは、ASN.1を標準のSNMPスタックに適用する方法を表現するための基礎として使用されます。

このチュートリアルを十分に活用するためには、SDL SuiteとASN.1の基礎の知識が必要です。

ASN.1データ型に関するより詳しい情報、およびASN.1データ型をSDL Suiteと共に使った場合の使用法については以下を参照してください。

- 『Methodology Guidelines』の第2章、「データ型」
- 『User´s Manual』の第13章、「The ASN.1 Utilities」
- 『User´s Manual』の第57章、「アプリケーションの構築」
- 『User´s Manual』の第58章、「ASN.1 Encoding and De-coding in the SDL Suite」

はじめに

ASN.1（Abstract Syntax Notation One）は、ある種のインターフェイスや通信媒体を介して転送することを目的とした構造情報を記述するのに使用される表記言語です。特に、通信プロトコルを定義するためによく使用されます。

ASN.1が広く使用されているため、SDL SuiteではASN.1データ型をSDLに変換することができ、さらにはASN.1データ型のエンコードとデコードが可能になっています。

アプリケーションの実装時にASN.1データ型を使用することにより、開発プロセスを最適化できます。以下にASN.1の利点を挙げます。

- ASN.1は、ベンダー、プラットフォーム、言語から独立した、標準化された表記言語です。
- ほとんどのテレコミュニケーション プロトコルおよびサービスがASN.1によって定義されています。そのため、定義済みのASN.1パッケージおよびモジュールが存在しており、RFCなどの標準化団体から入手可能です。たとえば、SNMPを定義するASN.1データ型はRFC1157から入手できます。
- コンピュータ ネットワーク経由でASN.1データ型が転送される場合、その値はビットパターンで表す必要があります。ビットパターンを決定するエンコードとデコードの規則は、ASN.1の場合すでに定義されています。SDL Suiteでは、BERとPERによるエンコードがサポートされます。
- ASN.1 には拡張性があります。これは、ASN.1によって、長い時間を隔てて設計され、実装されたシステムの互換性が容易になることを意味します。
- SDL SuiteとTTCN Suiteは、共通のデータ型を別のASN.1モジュールに指定することにより、それらを共有できます。

ASN.1の実装

ASN.1 データ型をSDLシステムにインポートするときは、ASN.1の定義をSDLに変換する必要があります。SDL Suiteは、これをASN.1ユーティリティというツールを使用して行います。このツールは、SDLシステムの分析を行うと、自動的に起動します。

ただし、SDLに変換しても、それでASN.1データ型をアプリケーションに含むことができる訳ではありません。アプリケーションで生成された情報をコンピュータ ネットワーク経由で転送する場合は、データ型の値がエンコードされている必

要があります。また、SDLシステムへ、またはSDLシステムから信号を転送するときは、システムと環境の間のインターフェイスを作成する必要があります。

このようにしてASN.1データ型を実装する過程は、以下の3つの手順に分けることができます。

1. 抽象構文の作成
2. 転送構文の作成
3. アプリケーションのコンパイル

抽象構文と転送構文の定義は以下の通りです。

抽象構文

基本的な考え方は、ある型の情報を2つのノードの間で、プロトコル メッセージを使って転送するというものです。抽象構文は転送可能なあらゆるメッセージのセットとして定義されます。抽象構文を作成するには以下のことを行う必要があります。

- ASN.1などの高級プログラミング言語で定義された、ある形式のデータ構造を設計します。
- そのデータ構造が取り得るすべての値を定義します。

転送構文

転送構文は、抽象構文のメッセージを表すビット パターンのセットで、各ビット パターンがそれぞれ1つの値を表します。ビット パターンを決定する規則はエンコード規則と呼ばれます。

抽象構文の作成

抽象構文を作成するときは、以下のタスクを実行する必要があります。

- ASN.1モジュールのプロジェクトへの追加
- ASN.1モジュールのSDLダイアグラムへのインポート
- データ型への値の割り当て

ASN.1モジュールのプロジェクトへの追加

ASN.1 モジュールはASN.1データ型の定義を含むファイルです。標準の通信プロトコルを実装している場合は、定義済みのASN.1モジュールがすでに作成されている可能性が高いと考えられます。モジュールはRFCなどの標準化団体から取得できます。325ページの例10には、SNMPプロトコルを定義するASN.1モジュールが記載されています。このモジュールはRFC 1157で入手可能です。

ただし、使用するアプリケーションの型の定義済みモジュールが入手できない場合は、ユーザーがモジュールを作成する必要があります。ASN.1に関する資料に十分に目を通し、ASN.1モジュールの作成に関する手順や指針を調べてください。

ASN.1モジュール入手の方法にかかわらず、SDL Suiteにデータ型を含める前に、モジュールをプロジェクトに追加する必要があります。

以下の手順に従い、ASN.1モジュールをプロジェクトに追加してください。

1. サブディレクトリにあるASN.1モジュールをプロジェクトに保存してください。モジュールを保存するために、必ずファイル拡張子`.asn`を付けてください。
2. オーガナイザを開き、モジュールを含めるチャプタを選択します。これは、［その他のドキュメント］マーカなどのチャプタ マーカをクリックすることにより行うことができます。303ページの図191を参照してください。

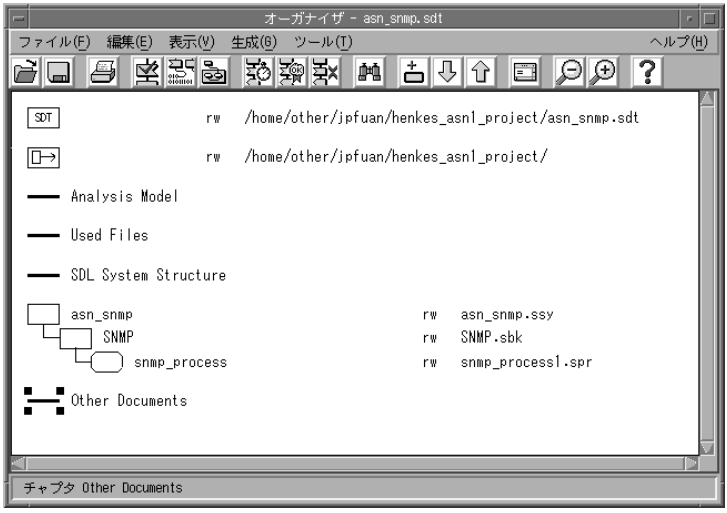


図191: チャプタ マーカの選択

3. ［編集］メニューから、［既存の追加］コマンドを選択します。［既存ファイルの追加］ウィンドウが開きます。

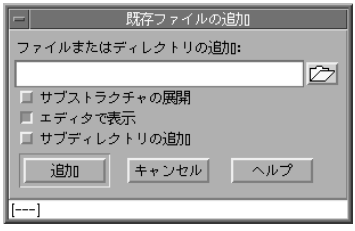


図192: ［既存ファイルの追加］ウィンドウ

4. フォルダが描かれたボタンをクリックし、ASN.1モジュールを見つけます。［追加するファイルの選択］ウィンドウが開きます。

5. 検索するディレクトリを選択し、［フィルタ］フィールドに「*.asn」と入力して、検索フィルタを変更します。［フィルタ］ボタンをクリックします。使用できるASN.1モジュールが［ファイル］ウィンドウに表示されます。モ

ジュールを選択し、[OK] ボタンをクリックします。[追加するファイルの選択] ウィンドウが閉じます。

6. 選択したモジュールが [既存ファイルの追加] ウィンドウに表示されます。[OK] ボタンをクリックしてモジュールをシステムに追加します。このモジュールは、選択したチャプタのオーガナイザで見ることができるはずです。

ASN.1モジュールがプロジェクトに追加されました。

例1:ASN.1モジュールのSDLプロジェクトへの追加 ——————————

SNMPの例では、RFC1155_SMI、RFC1157_SNMP、USE_SNMPの3つのモジュールがプロジェクトに追加されています。

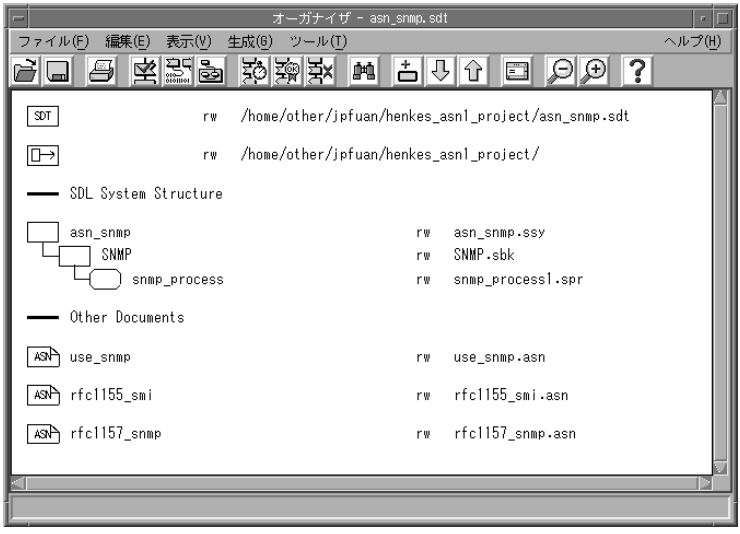


図193: 追加されたASN.1モジュール

ASN.1モジュールのインポート

プロジェクトにモジュールを追加した後、そのモジュールをSDLで使えるようにする必要があります。これは、モジュールをSDLシステム ファイルにインポート

することにより行うことができます。モジュールのインポートが済めば、ASN.1データ型は、通常のSDL型として使えます。

以下の手順に従い、モジュールをSDLダイアグラムにインポートしてください。

1. オーガナイザからシステム ファイル<system_name>.ssyを開きます。
2. 追加モジュールの名前をパッケージ リファレンス フレームに入力します。パッケージ リファレンス フレームはシステムフレームの外にあります。（図194を参照）
3. ダイアグラムを保存します。

例2:ASN.1モジュールのインポート

SNMPの例では、RFC1155_SMI、RFC1157_SNMP、USE_SNMPの3つのモジュールがインポートされています。

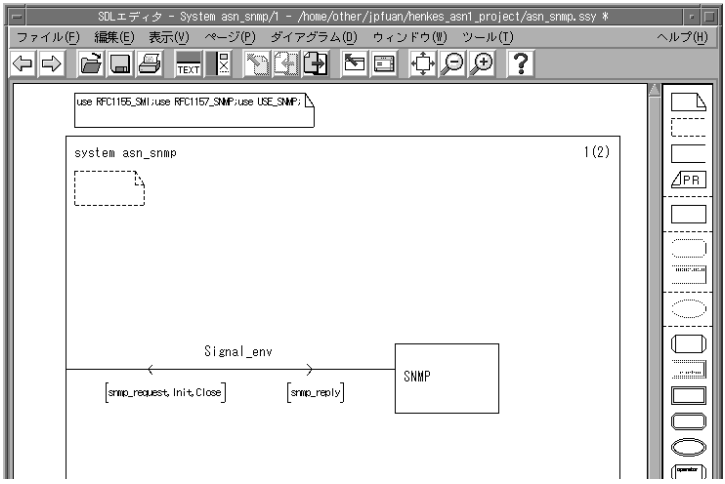


図194: インポートされたASN.1モジュール

データ型への値の割り当て

モジュールがSDLシステムにインポートされると、信号パラメータとASN.1データ型の変数の宣言を自由に行えます。パラメータと変数は通常のSDLのパラメータや変数として扱われ、通常行う方法で値を割り当てることができます。

ASN.1を使って定義された情報を転送する信号の宣言をするときは、ASN.1モジュールのトップレベルの型を、信号パラメータとして定義することをお勧めします。

変数に値を割り当てれば、抽象構文を作成したことになります。

例3: 変数への値の割り当て

この例では変数Replyは、Message型として宣言されています。この型は、ASN.1モジュールSNMP1157.asn で定義されたトップレベルの型です。325ページの例10を参照してください。

図195: 変数reply の宣言

変数replyが宣言されている場合、これをSDLダイアグラムで、通常のSDL変数として使用することができます。307ページの図196はreplyが、SDLシステムによって受信された信号の引数として使用される様子を示しています。

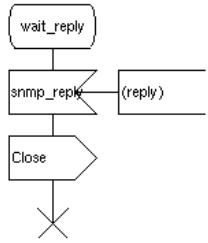


図196: `reply`変数の使用法

例4:信号の宣言

前の例の信号`snmp_reply`を使用する前に宣言を行う必要があります。前に述べたように、信号の引数は、ASN.1モジュールのトップレベルの型です。

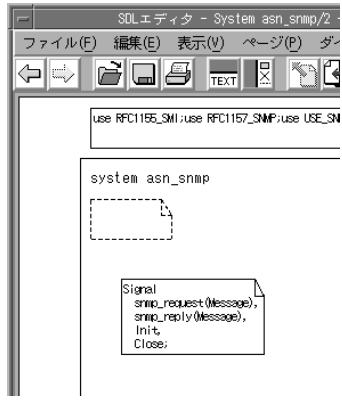


図197: 信号の宣言

転送構文の作成

SDL Suiteには、転送構文を作成する方法として何通りかあります。使用可能なコーディング インターフェイスは次のとおりです。

- 基本SDLインターフェイス
- 拡張SDLインターフェイス
- Cコード インターフェイス

このチュートリアルでは、Cコード インターフェイスのみを扱っています。ASN.1のエンコードとデコードの詳細については、『User´s Manual』の第58章、「ASN.1 Encoding and De-coding in the SDL Suite」を参照してください。

Cコード インターフェイスを使用すると、オーガナイザの［実装］ダイアログ ボックスまたはターゲティング エキスパートを使用して転送構文を作成できます。このチュートリアルには両方の方法が記載されています。ターゲット エキスパートを使用する場合、転送構文を作成するときに、Cadvanced SDL to C コンパイラかCmicro SDL to C コンパイラのどちらかを選んで使うことができます。両方の方法が利用できます。

このセクションは短い概説で始まります。実際の手順は以下を参照してください。

- 312ページの「テンプレート ファイルの生成 - オーガナイザ」
- 317ページの「テンプレート ファイルの生成 - ターゲティング エキスパート」

はじめに

ネットワーク上の2つのノード間で抽象構文を転送できるようにするためには、最初に転送構文を作成する必要があります。その後転送構文の表現はプロトコル バッファに転送されます。

転送構文作成時は以下のタスクを実行する必要があります。

- テンプレート ファイルの作成
- 生成されたテンプレート ファイルの編集

テンプレート ファイルは、ASN.1データ型をコンパイルやコード生成過程に含めるため、生成する必要があります。テンプレート ファイルによりSDLシステムから情報を抽出し、スケルトンを作成します。しばしばテンプレート ファイルには、アプリケーションの要求に応えるのに十分な情報が含まれていないことがあり、

そうした場合テンプレートの編集を行う必要があります。生成されるテンプレート ファイルには以下が含まれます。

- 環境関数
- 実装処理

ただし、環境ファイルを作成するには、SDL Suiteに追加情報が必要です。テンプレート ファイルの生成を行う前に、どのエンコードおよびデコード方法を使うかを決定し、データ型ノード ファイルを作成する必要があります。

> メモ： 環境ファイル
>
> 環境ファイルを作成する方法はいくつかあります。このチュートリアルではファイルを自動生成する方法について説明します。ただし、ファイルを最初から全部自分で作成することもできます。この手順はより高度なもので、このチュートリアルではその一部のみ説明します。

> メモ： データ型ノード
>
> データ型ノードはASN.1ユーティリティによって自動作成されます。編集はできません。

> メモ： 実装処理
>
> テンプレートの実装ファイルは［実装］ダイアログ ボックスを使用している場合のみ作成されます。デフォルトのターゲティング エキスパートの実装ファイルは、すべての必要な実装機能を処理します。

環境関数

環境は、アプリケーションにとって必要であり、SDLシステムで指定されていないすべての装置または関数として定義されます。環境に信号を送ることにより、SDLシステムは一定のタスクを実行しようとします。以下はその例です。

- ファイルからの情報の読み取り、またはファイルへの情報の書きこみ
- ネットワーク経由でのメッセージの送受信
- ハードウェア ポートまたはソケットからの情報の読み取り、またはこれらへの情報の書きこみ

ただし、SDLシステムは、システムで発生したイベントのみを制御します。SDLシステムは、システムから送信される信号が環境によってどのように処理されるかを指定しません。そのため、SDLシステムと環境の間のインターフェイスを提供する必要があります。このインターフェイスは環境関数により作成されます。

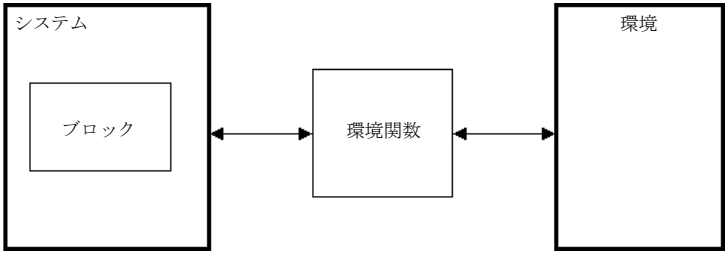


図198: 環境関数

SDL Suiteはむしろ役立つものであり、環境関数のスケルトンを含むテンプレートの環境ファイルを生成できます。環境ファイルはCコードで書かれており、このファイルを編集することにより、SDLシステムからの信号やSDLシステムに入ってくる信号の振る舞いを指定できます。

環境ヘッダー ファイルまたはシステム インターフェイス ヘッダー ファイルを作成することもできます。こうしたファイルには環境関数を実装するのに必要な、すべての型の定義およびその他外部の定義が含まれます。

> メモ： 環境ファイル
>
> 環境ファイルを作成する方法は以下の通りです。
>
> - ファイルを自動作成する方法。この手順はこのチュートリアルで説明しています。
> - ファイルを最初から全部自分で作成する方法。この方法はより高度な方法であり、このチュートリアルではその一部のみを扱っています。

エンコード/デコード

転送構文を作成するとき、転送するメッセージはエンコードされている必要があり、受信するメッセージはデコードする必要があります。適用するエンコード規則のタイプは環境ファイルで指定されます。つまり、エンコード/デコード関数の呼び出しが環境ファイルに含まれている必要があるということです。

SDL Suiteでは標準のBERとPERのエンコード/デコード方法がサポートされていますが、ユーザーが定義したエンコード方法を使用することもできます。SDL SuiteではASCIIエンコードも利用可能ですが、ASN.1データ型のエンコードはサポートされません。

データ型ノード

ASN.1データ型をアプリケーションに含めるためには、データ型をSDL Suiteが解読できる形式に変換する必要があります。SDL Suiteでは、この変換はASN.1ユーティリティが処理します。

ASN.1ユーティリティ ツールは、SDLシステムの分析が行われるときに自動的に起動します。ASN.1ユーティリティで以下の処理を行うことができます。

- ASN.1モジュールの構文および意味分析
- ASN.1モジュールからのSDLコードの生成
- BERまたはPERを使用してのエンコードとデコードに関するデータ型情報の生成

つまり、ASN.1ユーティリティを使用しているときはデータ型ノードを作成するということを意味します。データ型ノードは、ASN.1データ型のプロパティと特性を表す静的変数です。これには、BER/PERのエンコーダとデコーダで必要なタグ情報も含まれます。変数は次のように名前が付けられます。
`yASN1_<type_name>`

すべてのノードは`<asn1module_name>_asn1coder.c`という名前のついたファイルに生成され、ノードにアクセスするための宣言は`<asn1module_name>_asn1coder.h`という名前のついたファイルに生成されます。

> メモ：
>
> データ型ノードはASN.1ユーティリティによって自動作成されます。編集はできません。

実装処理

> メモ：［実装］ダイアログ ボックスのみ
>
> このセクションの内容が有効なのは、オーガナイザの［実装］ダイアログ ボックスを使用してプロジェクトの構築および分析を行う場合のみです。

SDL Suiteにあるデフォルトの実装ファイルは、プロジェクト内のソース ファイル、ヘッダー ファイル、オブジェクト ファイルとライブラリの関係を決定します。

ただし、デフォルトの実装ファイルには、実装処理で生成されたファイルは含まれません。環境ファイルやデータ型ノードファイルを実装処理に含むには、デフォルトの実装ファイルに付加されるテンプレート実装ファイルを生成する必要があります。図199を参照してください。テンプレート実装ファイルはSDL Suiteによって生成されます。

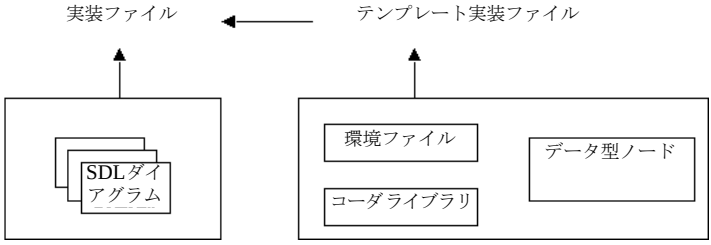


図199: 実装処理

テンプレート ファイルの生成 - オーガナイザ

以下の手順に従い、オーガナイザの［実装］ダイアログ ボックスを使用して、環境ファイル、データ型ノードファイル、テンプレート実装ファイルを生成してください。

1. オーガナイザでSDLシステム シンボルをクリックします。
2. ［生成］メニューから［実装］コマンドを選択します。［SDL実装］ウィンドウが開きます。
3. ［実装］ダイアログ ボックスで以下からオプションを指定します。
   - ［コードの分析と生成］を選択します。
   - ［コード ジェネレータ］ドロップダウン リストから［Cadvanced］を選択します。
   - ［環境ヘッダー ファイルの生成］を選択します。
   - ［環境関数を生成します］を選択します。
   - ［ASN.1コーダの生成］を選択してASN.1ユーティリティを起動します。
   - ［標準カーネルの使用］ドロップダウン リストから［Application］を選択します。

   メモ：

   単にテンプレート ファイルを生成するだけなので、必ず［コンパイルとリンク］オプションをオフにするようにしてください。
4. 生成されたファイルを格納するターゲット ディレクトリを指定します。

図200は［実装］ダイアログ ボックスでオプションを選択したときの様子を示しています。

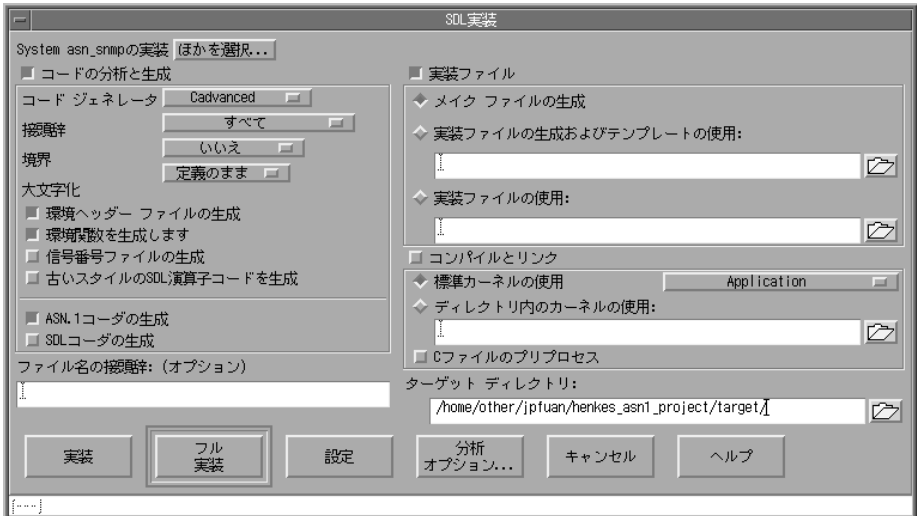


図200: ［実装］ダイアログ ボックス- テンプレート ファイルの生成

5. ［フル実装］ボタンをクリックします。

> メモ：
>
> エンコードやデコードの呼び出しは、［実装］ダイアログ ボックスで［ASN.1コーダの生成］オプションが有効になっている場合のみ生成されます。

ターゲット ディレクトリには以下のファイルが生成されています。

- `<system_name>_env.c`
  これは環境スケルトン ファイルです。
- `<system_name>.ifc`
  これは環境ヘッダー ファイルです。
- `<system_name>_env.tpm`
  これはテンプレート実装ファイルです。
- `<asn1module_name>_asn1coder.c`
  `<asn1module_name>_asn1coder.h`

これらのファイルはASN.1ユーティリティによって作成されるデータ型ノードです。

生成されたファイルの編集 - オーガナイザ

生成されたファイルはスケルトン関数のみで構成されているので、アプリケーションの機能に合わせてファイルを編集する必要があります。

> メモ：
>
> 環境ファイルやテンプレート実装ファイルを編集した後は、そのコピーを作成するようにしてください。［実装］ダイアログ ボックスで誤ってファイルを再生成した場合、編集した内容は上書きされてしまうからです。

1. エディタを使用して環境ファイル<system_name>_env.cを編集してください。スケルトン ファイルにはマクロが含まれていますが、定義はされていません。マクロを定義するには、<system_name>_env.hファイルを作成し、コードを手動で入力します。314ページの例5 は編集されていないSNMP環境ファイルを示しています。
2. 環境ファイルを保存します。
3. 必要であればテンプレート実装ファイル`<system_name>_env.tpm`を編集します。
4. テンプレート実装ファイルを保存します。
5. 編集済みファイルのコピーを作成し、別のフォルダに保存します。

> Notes:
>
> - 情報をネットワーク経由で転送するには、適切なヘッダー ファイルにソケット コマンドを追加する必要があります。
> - 複数のエンコードの方法を使用する場合、たとえばBERとPERを使う場合は、環境ファイルに適切なエンコード関数呼び出しを入力する必要があります。

例5: 環境関数 ——————————————————————————

以下のコードは環境ファイル スケルトンの一部であり、送信信号を扱う関数を示すものです。

```
XENV_OUT_START
```

```
  /* Signal_env チャネル経由で env に送信される信号 */

    /* 信号 : snmp_request */
    IF_OUT_SIGNAL(snmp_request,"snmp_request")
      OUT_SIGNAL1(snmp_request,"snmp_request")
      XENV_BUF(BufInitWriteMode(Buf));
      XENV_ENC(BER_ENCODE(Buf, (tASN1TypeInfo *)&yASN1_Message,
        (void *)&((yPDef_snmp_request *)(*SignalOut))->Param1));
      OUT_SIGNAL2(snmp_request,"snmp_request")
      XENV_BUF(BufCloseWriteMode(Buf));
      RELEASE_SIGNAL
    END_IF_OUT_SIGNAL(snmp_request,"snmp_request")

    /* 信号の初期化 */
    IF_OUT_SIGNAL(Init,"Init")
      OUT_SIGNAL1(Init,"Init")
      XENV_BUF(BufInitWriteMode(Buf));
      OUT_SIGNAL2(Init,"Init")
      XENV_BUF(BufCloseWriteMode(Buf));
      RELEASE_SIGNAL
    END_IF_OUT_SIGNAL(Init,"Init")

    /* 信号の終了 */
    IF_OUT_SIGNAL(Close,"Close")
      OUT_SIGNAL1(Close,"Close")
      XENV_BUF(BufInitWriteMode(Buf));
      OUT_SIGNAL2(Close,"Close")
      XENV_BUF(BufCloseWriteMode(Buf));
      RELEASE_SIGNAL
    END_IF_OUT_SIGNAL(Close,"Close")
}
```

エンコーダおよびデコーダ関数呼び出し

前に述べたように、環境ファイルを自動生成する必要はありません。別の環境ファイルをコピーするか、最初から自分で作成することにより、ニーズに合った環境ファイルをカスタマイズできます。カスタマイズする場合は、エンコードおよびデコード関数の構文を正確に記述する必要があります。

BER関数呼び出しの構文は以下の通りです。

```
BER_ENCODE (
   Buffer,
   &Typenode,
   &Signalparameter)

BER_DECODE (
   Buffer,
   &Typenode,
   &Signalparameter)
```

PER関数呼び出しの構文は以下の通りです。

```
PER_ENCODE (
   Buffer,
   &Typenode,
   &Signalparameter)

PER_DECODE (
   Buffer,
   &Typenode,
   &Signalparameter)
```

例6: エンコードおよびデコード関数呼び出し

以下の関数呼び出しはSNMPの例で使用されています。

```
BER_ENCODE(Buf, (tASN1TypeInfo *)&yASN1_Message,
   (void *)&((yPDef_snmp_request *)(*SignalOut))-
>Param1));

BER_DECODE(Buf, (tASN1TypeInfo *)&yASN1_Message,
   (void *)&((yPDef_snmp_reply *)SignalIn)->Param1))
```

受信信号のデコード

受信信号に含まれる情報をSDLシステムで利用するには、その前に多くのタスクを実行する必要があります。ほとんどのタスクはSDL Suiteによって自動的に実行されますが、手動で実行する必要のあるタスクもあります。

以下はデコード処理に必要な手順です。手順1と2は手動で実行する必要があります。手順3から5はSDL Suiteにより生成されます。

1. 正しい環境ファイルを自動生成するためには、SDLシステムで受信信号と送信信号を宣言します。信号のパラメータがASN.1データ型として宣言された場合、ASN.1モジュールのトップレベルのノードを使用します。307ページの例4を参照してください。デコードの際、エンコードで使うものと同じトップレベルの型を使用することをお勧めします。

2. エンコードされた情報を、プロトコルを特定したパケットから抽出し、これをデータ バッファに転送します。これは環境ファイルにおいて、Cコードで実装される必要があります。手順およびより詳しい情報は『User´s Manual』の第57章、「アプリケーションの構築」を参照してください。

以下の機能の関数と関数呼び出しは、環境ファイルで自動生成されます。

3. 信号構造用のメモリの割り当て。

4. BER_DECODE関数の呼び出し。この関数はデコード ライブラリで定義され、実際のデコード処理を行います。
5. SDLシステムへのデコードされた信号の送信。これはSDL_Output関数により実行されます。

テンプレート ファイルの生成 - ターゲティング エキスパート

以下の手順に従い、ターゲティング エキスパートを使用して環境ファイル、データ型ノード ファイル、テンプレート実装ファイルを生成してください。

1. ［生成］メニューから［ターゲティング エキスパート］コマンドを選択します。SDLターゲティング エキスパート ウィンドウが開きます。
2. 分割ダイアグラム モデル フレームの上にあるドロップダウン メニューから［ライト インテグレーション］を選択し、使用するSDL to Cコンパイラを選びます。CadvancedとCmicroのどちらも使用することができます。この定義済みのオプションにより、生成に必要なコンパイラの種類を指定できます。
3. ［SDL to Cコンパイラ］タブを選択します。
4. ［全般］ボックスで［分析とコード生成］ボタンを選択します。
5. ［環境］ボックスで以下を選択します。
   - ［環境関数］
   - ［環境ヘッダー ファイル］
6. ［通信］タブを選択します。［コーダ］ボックスで［ASN.1コーダ関数を生成する］チェック ボックスをオンにします。
7. ［完全な実装］ボタンをクリックします。この操作により環境ファイルが生成されます。

> メモ：
>
> エンコードとデコードの呼び出しは［コーダ関数］オプションが有効なときのみ生成されます。

ターゲット ディレクトリには以下のファイルが生成されます。

- `<system_name>_env.c` (Cadvanced)
  `env.c` (Cmicro)
  これは環境スケルトン ファイルです。
- `<system_name>.ifc`
  これは環境ヘッダー ファイルです。
- `<asn1module_name>_asn1coder.c`
  `<asn1module_name>_asn1coder.h`
  これらのファイルはASN.1ユーティリティによって作成されるデータ型ノードです。

生成されたファイルの編集 - ターゲティング エキスパート

生成されたファイルはスケルトン関数のみで構成されているので、アプリケーションの機能に合わせてファイルを編集する必要があります。

> メモ：
>
> 編集後、環境ファイルのコピーを作成するようにしてください。誤ってファイルを再生成した場合、編集した内容は上書きされてしまいます。

1. 環境ファイルの名前を変更します。
2. 必要に応じて環境ファイル`<system_name>_env.c`を編集してください。スケルトン ファイルにマクロは含まれていますが、定義はされていません。マクロを定義するには`<system_name>_env.h`ファイルを作成し、手動でコードを入力します。319ページの例7 は、編集されていないSNMP環境ファイルを示します。
3. 環境ファイルを保存します。

メモ：

- ネットワーク経由で情報を転送するには、適切なヘッダー ファイルにソケット コマンドを追加する必要があります。
- BERとPERなどの複数のエンコード方法を使用したい場合は、環境ファイルに適切なエンコード関数呼び出しを入力する必要があります。

例7: 環境関数 - Cadvanced

以下のコードは環境ファイル スケルトンの一部であり、送信信号を処理する関数を示すものです。

```
XENV_OUT_START


  /* Signal_env チャネル経由で env に送信される信号 */

    /* 信号： snmp_request */
    IF_OUT_SIGNAL(snmp_request,"snmp_request")
      OUT_SIGNAL1(snmp_request,"snmp_request")
      XENV_BUF(BufInitWriteMode(Buf));
      XENV_ENC(BER_ENCODE(Buf, (tASN1TypeInfo *)&yASN1_Message,
        (void *)&((yPDef_snmp_request *)(*SignalOut))->Param1));
      OUT_SIGNAL2(snmp_request,"snmp_request")
      XENV_BUF(BufCloseWriteMode(Buf));
      RELEASE_SIGNAL
    END_IF_OUT_SIGNAL(snmp_request,"snmp_request")

    /* 信号の初期化 */
    IF_OUT_SIGNAL(Init,"Init")
      OUT_SIGNAL1(Init,"Init")
      XENV_BUF(BufInitWriteMode(Buf));
      OUT_SIGNAL2(Init,"Init")
      XENV_BUF(BufCloseWriteMode(Buf));
      RELEASE_SIGNAL
    END_IF_OUT_SIGNAL(Init,"Init")

    /* 信号の終了 */
    IF_OUT_SIGNAL(Close,"Close")
      OUT_SIGNAL1(Close,"Close")
      XENV_BUF(BufInitWriteMode(Buf));
      OUT_SIGNAL2(Close,"Close")
      XENV_BUF(BufCloseWriteMode(Buf));
      RELEASE_SIGNAL
    END_IF_OUT_SIGNAL(Close,"Close")
}
```

例8: 環境関数 - Cmicro

以下のコードは環境ファイル スケルトンの一部であり、送信信号を処理する関数を示すものです。

```
  switch (xmk_TmpSignalID)
  {
     case snmp_request :
          {
            /* ユーザー コードの開始 */
            /* 信号パラメータへのアクセスに (yPDP_snmp_request)xmk_TmpDataPtr を使
用 */
            /* 注意： データをコピーしてください。コピーしない */
            /* と、xOutEnv から復帰するときに失われます。 */
            /* このセクションは、選択したコーダ関数を使用して */
            /* 送信データをエンコードするために使用できます。 */
            ** 通信インターフェイスでデータを送るときはコメントを削除してください。
            ** (<SendViaCommunicationsInterface( data, datalen )> は、
               置き換える必要があります。)
            char* data;
            int datalen;

            BufInitWriteMode( Buf );
            XENV_ENC( PER_ENCODE( Buf, (tASN1TypeInfo *)
&yASN1_z_RFC1157_SNMP_0_Message,
            (void *) &((yPDef_snmp_request *)xmk_TmpDataPtr)->Param1));
            BufCloseWriteMode( Buf );
            BufInitReadMode( Buf );
            datalen = BufGetDataLen(Buf);
            data = BufGetSeg( Buf, datalen );
            <SendViaCommunicationsInterface( data, datalen )>;
            BufCloseReadMode( Buf );
            */
            /* 環境のアクションをここに記述します。*/
            xmk_result = XMK_TRUE; /* 信号が届いたことを呼び出し元 */
                                   /* に通知します。       */
            /*   ユーザー コードの終了 */
          }
          break ;

     case Init :
          {
            /* ユーザー コードの開始 */
            /* Do your environment actions here. */
            xmk_result = XMK_TRUE; /* 信号が届いたことを呼び出し元 */
                                   /* に通知します。       */
            /*   ユーザー コードの終了 */
          }
          break ;

     case Close :
          {
            /* ユーザー コードの開始 */
            /* 環境のアクションをここに記述します。*/
            xmk_result = XMK_TRUE; /* 信号が届いたことを呼び出し元 */
                                   /* に通知します。       */
```

```
          /*   ユーザー コードの終了 */
        }
        break ;

    default :
        xmk_result = XMK_FALSE;/* 信号が届かなかったことを */
                               /* 呼び出し元に通知し   */
                               /* Cmicro Kernelに処理    */
                               /* を委ねます。...   */
        break ;
}
```

エンコーダおよびデコーダ関数呼び出し

前に述べたように、環境ファイルを自動生成する必要はありません。別の環境ファイルをコピーするか、最初から全部自分で作成することにより、ニーズに合った環境ファイルをカスタマイズすることができます。カスタマイズする場合は、エンコードおよびデコード関数の構文を正確に記述する必要があります。

BER関数呼び出しの構文は以下の通りです。

```
BER_ENCODE (
   Buffer,
   &Typenode,
   &Signalparameter)

BER_DECODE (
   Buffer,
   &Typenode,
   &Signalparameter)
```

PER関数呼び出しの構文は以下の通りです。

```
PER_ENCODE (
   Buffer,
   &Typenode,
   &Signalparameter)

PER_DECODE (
   Buffer,
   &Typenode,
   &Signalparameter)
```

例9: エンコードおよびデコード関数呼び出し

SNMPの例では、以下の関数呼び出しが使用されています。

```
BER_ENCODE(Buf, (tASN1TypeInfo *)&yASN1_Message,
   (void *)&((yPDef_snmp_request *)(*SignalOut))-
>Param1));

BER_DECODE(Buf, (tASN1TypeInfo *)&yASN1_Message,
```

```
(void *)&((yPDef_snmp_reply *)SignalIn)->Param1))
```

受信信号のデコード

受信信号に含まれる情報をSDLシステムで利用するには、その前に多くのタスクを実行する必要があります。ほとんどのタスクはSDL Suiteにより自動的に実行されますが、手動で実行する必要のあるタスクもあります。

以下はデコード処理に必要な手順です。手順1と2は手動で実行する必要があります。手順3から5はSDL Suiteにより生成されます。

1. 正しい環境ファイルを自動生成するためには、SDLシステムで受信信号と送信信号を宣言します。信号のパラメータがASN.1データ型として宣言される場合、ASN.1モジュールのトップレベルの型を使用する必要があります。307ページの例4を参照してください。デコードの際、エンコードで使うものと同じトップレベルの型を使用することをお勧めします。

2. エンコードされた情報を、プロトコルを特定したパケットから抽出し、データ バッファに転送します。このタスクは環境ファイルにおいてCコードで実装される必要があります。手順と詳しい情報については『User´s Manual』の第57章、「アプリケーションの構築」を参照してください。

以下の機能の関数および関数呼び出しは、環境ファイルで自動生成されます。

3. 信号構造用のメモリの割り当て。

4. BER_DECODE関数の呼び出し。この関数はデコード ライブラリで定義され、実際のデコード処理を行います。

5. SDLシステムへのデコードされた信号の送信。これはSDL_Output関数により実行されます。

アプリケーションのコンパイル

転送構文の作成が済めば、アプリケーションのコンパイルと構築を行う準備が整ったことになります。アプリケーションは編集済みの環境ファイルとテンプレート実装ファイルを含みます。以下の手順に従ってください。

- 323ページの「編集済みファイルの使用 - オーガナイザ」
- 324ページの「編集済みファイルを使用する - ターゲティング エキスパート」

編集済みファイルの使用 - オーガナイザ

以下の手順に従ってください。

1. オーガナイザでSDLシステム シンボルをクリックします。
2. ［生成］メニューから［実装］コマンドを選択します。［SDL実装］ウィンドウが開きます。
3. 以下の手順に従い、［実装］ダイアログ ボックスでオプションを変更してください。
   - ［コードの分析と生成］を選択します。
   - ［コード ジェネレータ］ドロップダウン リストで［Cadvanced］をクリックします。
   - ［環境ヘッダー ファイルの生成］をオフにします。
   - ［環境関数を生成します］をオフにします。
   - ［ASN.1コーダの生成］をオフにします。
   - ［実装ファイル］をオンにします。
   - ［実装ファイルの生成およびテンプレートの使用］をオンにし、テキスト フィールドに生成されたテンプレート実装ファイル「`.... /<new_name>_env.tmp`」を入力します。
   - ［コンパイルとリンク］を選択します。
   - ［標準カーネルの使用］ドロップダウン リストで［Application］を選択します。

   324ページの図201は［実装］ダイアログ ボックスでオプションを選択した様子です。

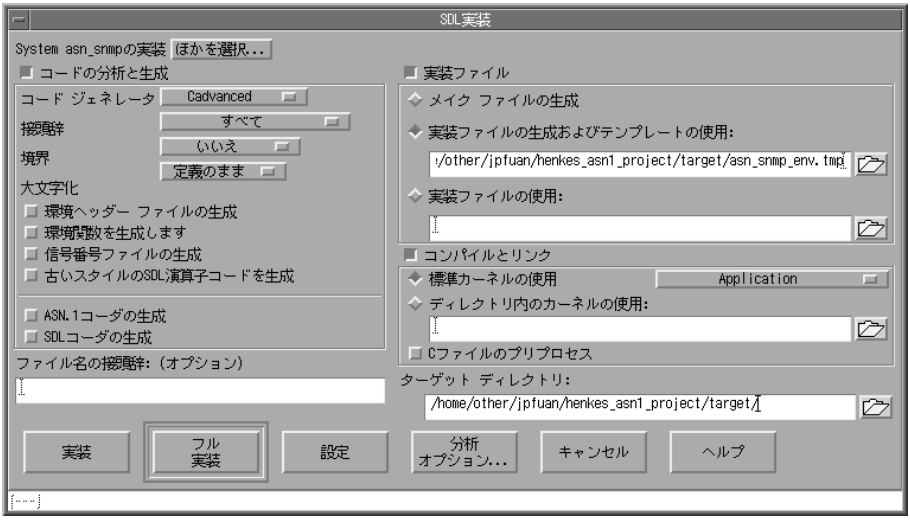


図201: ［実装］ダイアログ ボックス - コンパイル

4. 生成されたファイルを格納するターゲット ディレクトリを指定します。
5. ［フル実装］をクリックします。

編集済みファイルを使用する - ターゲティング エキスパート

以下の手順に従ってください。

1. ［SDL to Cコンパイラ］タブを選択します。
2. ［環境］ボックスで［環境関数］オプションと［環境ヘッダー ファイル］オプションをオフにします。
3. ［完全な実装］をクリックします。

付録A

例10: RFC1157 ASN.1 モジュール

```
RFC1157-SNMP DEFINITIONS ::= BEGIN

IMPORTS
  ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
             FROM RFC1155-SMI;

-- top-level message

        Message ::=
               SEQUENCE {
                     version        -- version-1 for this RFC
                        INTEGER {
                            version-1(0)
                        },

                    community      -- community name
                        OCTET STRING,

                    data           -- e.g., PDUs if trivial
                        PDUs --ANY--        -- authentication is being
used
                }

-- protocol data units

        PDUs ::=
                CHOICE {
                    get-request
                        GetRequest-PDU,

                    get-next-request
                        GetNextRequest-PDU,

                    get-response
                        GetResponse-PDU,

                    set-request
                        SetRequest-PDU,

                    trap
                        Trap-PDU
                     }

                  GetRequest-PDU ::=
                      [0]
                          IMPLICIT SEQUENCE {
                              request-id
                                  RequestID,

                              error-status        -- always 0
                                  ErrorStatus,

                              error-index         -- always 0
```

```
                        ErrorIndex,

                    variable-bindings
                        VarBindList
                }
            GetNextRequest-PDU ::=
                [1]
                    IMPLICIT SEQUENCE {
                        request-id
                            RequestID,
                        error-status        -- always 0
                            ErrorStatus,

                        error-index         -- always 0
                            ErrorIndex,

                        variable-bindings
                            VarBindList
                    }
            GetResponse-PDU ::=
                [2]
                    IMPLICIT SEQUENCE {
                        request-id
                            RequestID,
                        error-status
                            ErrorStatus,

                        error-index
                            ErrorIndex,

                        variable-bindings
                            VarBindList
                    }
            SetRequest-PDU ::=
                [3]
                    IMPLICIT SEQUENCE {
                        request-id
                            RequestID,

                        error-status        -- always 0
                            ErrorStatus,

                        error-index         -- always 0
                            ErrorIndex,

                        variable-bindings
                            VarBindList
                    }

    Trap-PDU ::=
        [4]

            IMPLICIT SEQUENCE {
               enterprise           -- type of object generating
                                    -- trap, see sysObjectID in [5]
                   OBJECT IDENTIFIER,

               agent-addr           -- address of object generating
                   NetworkAddress, -- trap

               generic-trap         -- generic trap type
```

```
                    INTEGER {
                        coldStart(0),
                        warmStart(1),
                        linkDown(2),
                        linkUp(3),
                        authenticationFailure(4),
                        egpNeighborLoss(5),
                        enterpriseSpecific(6)
                    },

                specific-trap     -- specific code, present even
                    INTEGER,      -- if generic-trap is not
                                  -- enterpriseSpecific

                time-stamp        -- time elapsed between the last
                  TimeTicks,      -- (re)initialization of the network
                                  -- entity and the generation of the
trap

                variable-bindings   -- "interesting" information
                     VarBindList
            }

                -- request/response information
                 RequestID ::=
                         INTEGER

                 ErrorStatus ::=
                         INTEGER {
                             noError(0),
                             tooBig(1),
                             noSuchName(2),
                             badValue(3),
                             readOnly(4),
                             genErr(5)
                         }

                 ErrorIndex ::=
                         INTEGER

                 -- variable bindings

                 VarBind ::=
                         SEQUENCE {
                             name
                                 ObjectName,

                             value
                                 ObjectSyntax
                         }

                 VarBindList ::=
                         SEQUENCE OF
                             VarBind


END
```

第 9 章

チュートリアル: SDL-2000の機能の使用法

このチュートリアルでは、SDL SuiteにおけるSDL-2000の各機能の使い方について説明します。

このチュートリアルを有効に活用するためには、SDL SuiteとSDLエディタの機能を理解している必要があります。

エディタとアナライザのチュートリアルにあるDemon Gameの例とこのチュートリアルは独立しています。

SDL-2000の機能の使い方についての補足情報は次の場所にあります。
『User´s Manual』の第43章「Using the SDL Editor」の1921ページ、「Working with Classes」

このチュートリアルの目的

このチュートリアルでは、SDL Suiteでサポートされている SDL-2000に慣れていただくことを目的としています。

はじめに

SDL Suiteでサポートされている機能

SDL Suiteでは、SDL-2000に関連する次の機能をサポートしています。

- データ型のグラフィカル デザイン(クラス シンボルの使用)
- テキストによるアルゴリズム
- 大文字と小文字の区別
- パラメータのない演算子
- 結果を返さない演算子

SDL-2000の利点

UMLからSDLへの転送を容易にするUMLの便利な機能がSDLに含まれています。

データ型のグラフィカル デザイン

SDL Suiteには、次のようなグラフィカルなデータ型のデザイン機能があります。

- *クラス シンボル*
- *関連ライン*
- *アグリゲーション ライン*
- ［*クラスの編集および参照*］ダイアログについては、*ダイアグラムの編集*を参照してください。
- ［*クラス情報*］ダイアログについては、*クラスの参照*を参照してください。

クラス シンボル

クラス シンボルでデータ型をグラフィカルにデザインすることができます。SDL エディタはほかのSDLシンボルと同じようにクラス シンボルを扱います。

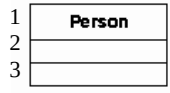


図202 クラス シンボル

クラス シンボルには次のように3つの独立したテキスト フィールドがあります。

- 名前フィールド(1)
- 属性フィールド(2)
- 演算子フィールド(3)

クラス シンボルのプロパティは、大文字と小文字の区別の面を除けば、ほかのSDLシンボルと同じように環境設定マネージャで設定できます。詳細については、338ページの「大文字と小文字の区別」を参照してください。クラス シンボルのサイズはテキストのサイズに応じて調節されますが、このサイズを手動で変更することはできません。クラス シンボルは、折りたたんだり展開したりできます。

関連ラインとアグリゲーション ライン

ラインには次の2種類があります。

- 関連ライン
- アグリゲーション ライン

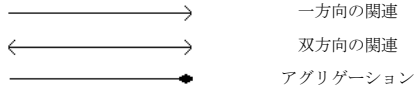


図203

関連ライン

関連はUML表記規則を使って2種類のタイプにリンクします。タイプの種類:

- ブロック タイプ
- プロセス タイプ
- データ型
- インターフェイス

関連ラインは、リダイレクト型または双方向型のどちらにもなり得ます。一方向の関連ラインは、1つの名前と1つのロール名を持ちます。双方向の関連ラインは、1つの名前と2つのロール名を持ちます。

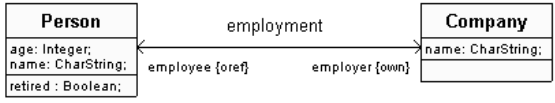


図204: 関連ラインの例

アグリゲーション ライン

アグリゲーション ラインは関連ラインと同じ形式です。

アグリゲーション ラインは、クラスがほかのクラスのサブセットであるか一部であるかを示すために使用されます。例えば、車のハンドルは車のサブセットまたは一部と言えます。アグリゲーション ラインが取り得る方向は一方向のみです。

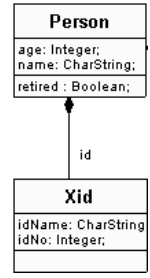


図205: アグリゲーション ラインの例。

SDL 構造の作成

ここでは、小さな例を作成していくつかの操作を実行することによって、SDL Suiteでサポートする SDL-2000における可能性と制限事項を学びます。

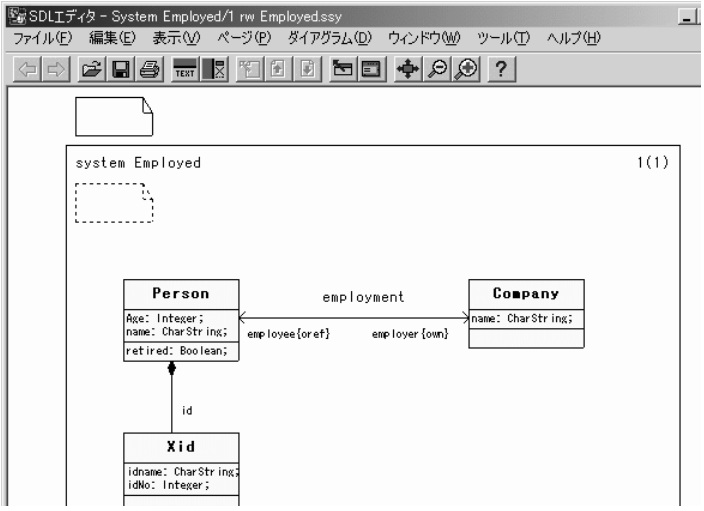


図206: クラス シンボルを使用したSDL ダイアグラム

クラス シンボルを使用しての作業

1. SDLエディタを起動します。
2. ダイアグラムにクラス シンボルを配置します。

   ほかのSDLシンボルを配置するのと同じようにシンボルを配置します。詳細については、第3章「チュートリアル: SDLエディタとアナライザ」59ページの「ブロック参照シンボルの設定」を参照してください。シンボルは移動できますが、シンボルのサイズは変更できません。
3. クラス シンボルに`Person`と名前を付けます。

   同じクラス名を持つすべてのクラス シンボルは、単一のマージ済みクラスとして扱われます。名前の付け方による問題については、338ページの「制限事項：」を参照してください。
4. 属性と演算子フィールドに値を入力します。

5. ダイアグラムにクラス シンボルを2つ以上配置し、`Company`と`Xid`と名前を付けます。

6. 属性と演算子フィールドに値を入力します。

ラインを使用しての作業

1. クラス シンボル`Person`をクリックします。ハンドルが2つ現れます。

2. 四角の関連ハンドルを選択してクラス シンボル`Company`にドラッグします。ドラッグしている間、ラインが描かれます。ほかのSDLダイアグラムと同じようにラインをドラッグします。詳細については、第3章「チュートリアル: SDLエディタとアナライザ」61ページの「ブロック間のチャネル作成」を参照してください。

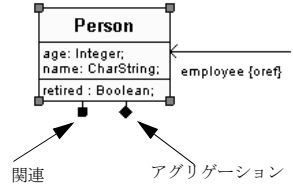


図207

3. クラス シンボル`Company`に到達したら、マウスのボタンをクリックします。ラインで両端が結ばれます。

4. 関連ラインとロールに名前を付けます。

関連の名前には動詞、ロールの名前には名詞が多くの場合使用されます。

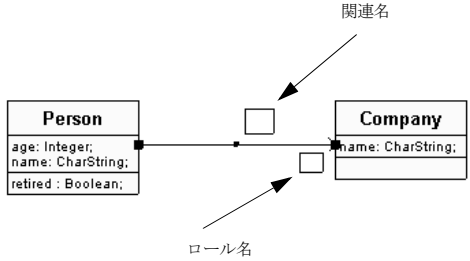


図208

5. ［*編集*］メニューの［*双方向*］を選択し、ラインを双方向にします。
6. 2つ目のロール名を入力します。
7. 菱型のアグリゲーション ハンドルを選択してクラス シンボル`Xid`にドラッグします。
8. アグリゲーション ラインとロールに名前を付けます。

シンボルとラインの移動

ほかのSDLダイアグラムと同じ方法でシンボルを移動することができます。詳細については、第3章「チュートリアル: SDLエディタとアナライザ」60ページの「シンボルの移動とサイズ変更」を参照してください。

ダイアグラムの保存

1. メニューの［*保存*］を選択してダイアログを保存できます。これには、［*保存*］ボタンをクリックする方法と、`<Ctrl> s`を使用する方法の2通りあります。

   保存したファイルには、拡張子として`.ssy`が付けられます。

ダイアグラムの編集

［*クラスの参照および編集*］ダイアログを使用すると、属性と演算子の間で整合性が取れるように、クラスの完全な定義の参照と編集を行うことができます。

ダイアログは2つの部分で構成されています。上の部分は、クラスと各クラスのすべての発生内容を参照するためのものです。下の部分は、クラスの名前、属性、および演算子を編集するためのものです。

［*クラスの参照および編集*］ダイアログで変更を行った場合、このダイアログで選択しているクラスに属するすべてのシンボルに影響します。1つのクラス シンボルのみの内容を編集するときは、シンボルを1つ選択してシンボルの内容を編集する必要があります。クラス情報の編集については、第43章「Using the SDL Editor」1922ページの「Browse & Edit Class Dialog」を参照してください。

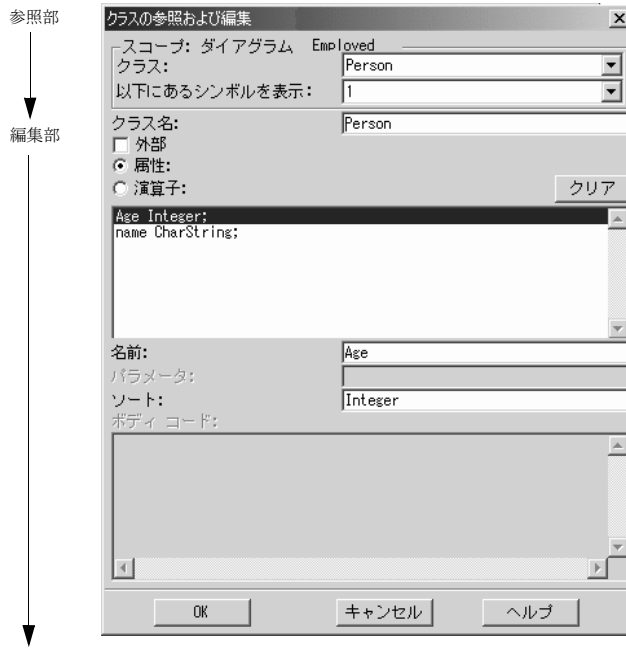


図209: ［クラスの参照および編集］ダイアログ

大文字と小文字の区別

大文字と小文字の区別はクラス シンボルに適用され、環境設定マネージャで解除することはできません。ただし、SDLエディタ内のほかのシンボルについては解除できます。

クラスの区別は、演算子フィールドと同様、名前フィールドと属性フィールドにも適用されます。

2つのクラスが同じ名前でも大文字と小文字の面で差があるときは(例えば`Person`と`person`)、［*クラスの参照および編集*］ダイアログでは別なクラスとして扱われ、コードを生成するときに1つのクラスにマージされることは**ありません**。

詳細については、第54章「SDLアナライザ」2465ページの「Set-Case-Sensitive」を参照してください。

制限事項：

1つのクラス シンボルを選択したときは、このダイアログでクラスを編集できますが、複数のクラス シンボルを選択した場合は、このダイアログは使用できません。

クラス名に誤った構文が含まれている場合、ダイアログは表示されません。

値の入力時は構文がチェックされるため、クラスに構文エラーが追加されることはありません。

シンボルに誤った構文が含まれている場合、ダイアログは表示されません。

ダイアグラムの編集

1. `Xid`クラスをダブルクリックするか、または`Xid`クラスを選択してから［*編集*］メニューの［*クラス..*］を選択します。

   ［*クラスの参照および編集*］ダイアログが、クラス名とその属性または演算子が入力された状態で表示されます。

   同じ名前のクラスが複数ある場合、それら同じ名前を持つすべてのクラスの属性または演算子がダイアログに表示されます。

2. ［*クラス*］ドロップダウン メニューからPersonクラスを選択します。属性および演算子フィールドの内容がPersonクラスの内容に変更されます。

3. *クラス名*フィールドのクラス名をPersonからEmployedに変更し、［*OK*］をクリックします。

   ダイアグラムのクラス名がEmployedに変更されていることを確認します。

4. ［*クラスの参照または編集*］ダイアログをもう一度開き、［*属性*］または［*演算子*］ボタンをクリックしてEmployedクラスの属性または演算子を参照または編集します。

   属性または演算子の一覧はラジオボタンの下にあるフィールドに表示されます。

5. この時点で、*名前*、*パラメータ*、*ソート*、*ボディ コード*フィールドの情報を編集することができます。

   *パラメータとボディ コード*フィールドは演算子用のフィールドです。

6. ［*OK*］をクリックします。

   ダイアログが閉じられ、現在のSDLダイアグラム内の適切なクラス シンボルすべてが更新されます。

クラスの参照

［クラス情報］ダイアログでは、クラスの完全なテキストの(PR)定義を読み取り専用形式で参照することができます。ダイアグラムを編集する場合は、337ページの「ダイアグラムの編集」を参照してください。

クラス情報の参照方法については、第43章「Using the SDL Editor」1921ページの「Class Information」を参照してください。

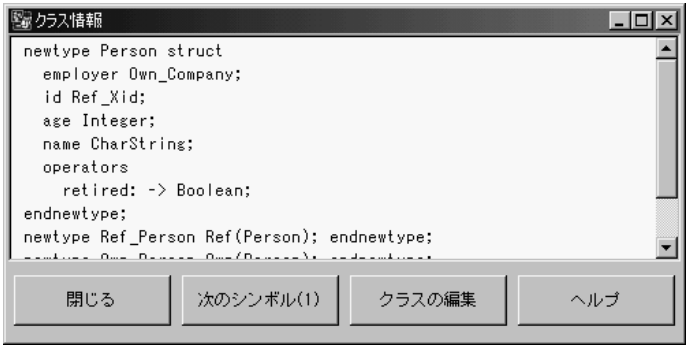


図210: ［クラス情報］ダイアログ

制限事項

複数のクラス シンボルを選択した場合はダイアログを使用できません。

定義の参照

1. 同じ名前が付いた複数のクラスの名前を変更します。この処理は、［クラス情報］ダイアログのテストを目的としています。
2. クラスを1つ選択し、［*ウィンドウ*］メニューの［*クラス情報*］を選択します。

   ［*クラス情報*］ダイアログが表示され、そこには、同じ名前を持つすべてのクラス用のPRコードが含まれています。
3. ［*次のシンボルを表示します*］ボタンをクリックします。

   カーソルが置かれたラインの近くに、次のシンボルが表示されます。

括弧内の数字は、現在のダイアグラムのこのラインの近くに作成されたシンボルの数を示します。［*次のシンボルを表示します*］ボタンをクリックすると、すべてのシンボルを1つずつ参照できます。

4. クラスを編集する場合は、［*クラスの編集*］ボタンをクリックします。すると、対象のクラスに対する［*クラスの参照および編集*］ダイアログが表示されます。

テキストによるアルゴリズム

テキストによるアルゴリズム

SDL Suiteはアルゴリズムのテキストによる記述をサポートしています。アルゴリズムはタスク シンボル内に表示できます。例えば、if-then-else、ループ、分岐などです。この機能の詳細については、第3章「SDLの拡張表現の使用」137ページの「SDLのアルゴリズム」を参照してください。

パラメータのない演算子と結果を返さない演算子

SDL Suiteは、パラメータのない演算子または戻り値のない演算子をサポートしています。たとえば、代入ステートメントの代替として、演算子アプリケーション ステートメントは、値を返さない演算子を起動する場合があります。

演算子とパラメータの詳細については、第2章「データ型」81ページの「演算子」を参照してください。

制限事項

次の制限事項は重要です。

- ラインの継承はサポートされていません。
- クラス シンボルに追加されたコメントは、生成されたPRコードには表示されません。
- クラス シンボルのCIFコードの生成はサポートされていません。
- 同じ関連が複数の場所で発生している場合は、すべてのインスタンスを編集して関連を変更する必要があります。
- クラス シンボルからの移動は、*ツール> 移動を使用した時のみ可能です。クラス シンボルをダブルクリックしても、［クラスの参照および編集］ダイアログが開かれるだけで、この処理は実行できません。*

クラス シンボルの使用例

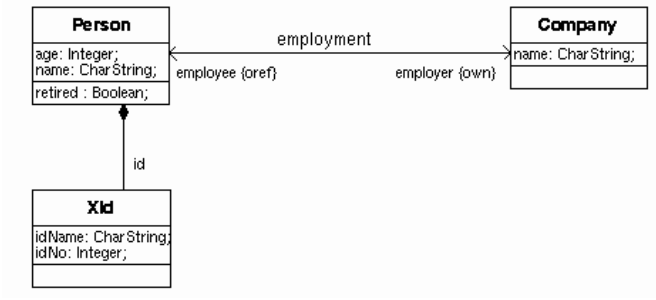


図211 データ構成のグラフィカルな表現

演算子定義を［*クラスの参照および編集*］ダイアログの*ボディ コード*部に追加する必要があります。内容は次のようになります。

```
operator retired returns Boolean { /* ユーザーが定義したボディ コードの開始 */
return age > 64;
/* ボディ コードで定義されたユーザーの終了 */}
```

上のグラフによって次のPRコードが生成されます。

```
newtype Person struct
  employer Own_Company;
  a_Company Ref_Company;
  id Ref_XId;
  age Integer;
  name CharString;
  operators
    retired: -> Boolean;
    operator retired returns Boolean {/* ユーザーが定義したボディ コードの開始 */
return age > 64
/* ユーザーが定義したボディ コードの終了 */}
endnewtype;
newtype Ref_Person Ref(Person); endnewtype;
newtype Own_Person Own(Person); endnewtype;
newtype Oref_Person Oref(Person); endnewtype;
newtype XId struct
  idName CharString;
  idNo Integer;
endnewtype;
newtype Ref_XId ref(XId); endnewtype;
newtype Own_XId own(XId); endnewtype;
newtype Oref_XId oref(XId); endnewtype;
newtype Company struct
  employee Oref_Person;
  a_Person Ref_Person;
  name CharString;
```

```
endnewtype;
newtype Ref_Company Ref(Company); endnewtype;
newtype Own_Company Own(Company); endnewtype;
newtype Oref_Company Oref(Company); endnewtype;
```

第10章 チュートリアル: スレッド インテグレーション

「スレッド インテグレーション」モデルは、Cのコードを生成し、それをオペレーティング システムに統合する際に使う、強力なテンプレートを提供するものです。チュートリアルでは、このモデルを使って実際にアプリケーションを構築する実習を行います。小規模なGSMシステムの実装例としてSDLシステム「Mobile」を構築し、このモデルに基づく開発がどのようなものか実体験してください。

実習には、SDL Suiteの エディタ、オーガナイザ、ターゲティング エキスパートに関する基本的な知識が必要です。SDLや、オペレーティング システムとの統合についても、知っていれば理解の助けになるでしょう。このチュートリアルの前に、第64章「Integration with Operating Systems」をお読みになるようお勧めします。

理解を深めるため、実習を始める前にチュートリアルを通読しておいてください。実習中は、指示に従って実機でいろいろお試しください。

はじめに

このチュートリアルの目的は、「スレッド インテグレーション」モデルに慣れていただくことです。「スレッド」実行形式ファイルの構築方法、外部コードとの統合方法について、基本的な手順を理解してください。

このチュートリアルは、SDL Suiteのビルド ツールを使い、指示に従って実習する形式になっています。配置エディタやターゲティング エキスパートの使い方も実習します。

> メモ：プラットフォームによる違い
>
> このチュートリアルでも、Solaris、Linux、Windowsのいずれにも通用する説明は、両方に共通の書き方をしてあります。プラットフォームによって違いがある事項は「Solarisの場合」、「Linuxの場合」、「Windowsの場合のみ」などというように明記しますので、お使いのプラットフォームに該当しない部分は無視してください。
>
> 画面表示例は、プラットフォームによって表示内容に違いがなければ、通常は片方の画面だけを示します。お使いの環境で動作しているSDL Suiteの画面と、**画面配置や表示形状が多少異なる**場合があるので注意してください。プラットフォームにより表示内容の重要部分に違いがある場合は、両方の画面を示します。

必要条件

Windows

- Cコンパイラ
- リソース コンパイラ

Solaris/Linux

- Cコンパイラ
- Motif (GUIウィジェットのライブラリ) 第1.2版以降

例題システムの説明

SDLシステム

機能の説明

例題はGSM(Global System for Mobile Communications)方式による電話システムです。ここではGSM規格の上位階層だけに限定し、次の機能を実装します。

- ユーザーのPINコード(暗証番号; Personal Identification Number)の確認機能。ただしPINコードは一桁の数字のみから成るものとします。
- VLR(Visitor Location Register)/HLR(Home Location Register)データベースによる、携帯電話の追跡。
- 携帯電話が移動した際の、基地局の切り替え。
- 携帯電話のIMEI(国際移動体機器識別; International Mobile Equipment Identity)コードの制御。
- 通話時間および費用を追跡する、課金サービス。

実装についての説明

システムはモジュール分割して実装します。各機能エンティティはブロック タイプで実装し、これを「GSM」というパッケージにまとめます。

SDLシステムはGSMパッケージを使い、そのブロック タイプを実体化します。「Mobile」システムには、4つのMobileStationブロック インスタンス、4つのBaseStationTransceiverブロック インスタンス、2つのMobileSwitchingCenterブロック インスタンスがあります。MobileStationインスタンスには、携帯電話の所有者に対応して、Marie、John、ParisPizza、LyonPizzaという名前が付いています。BaseStationTranscieverStationのインスタンスには、その所在地に対応して、Lyon11、Lyon12、Paris11、Paris12という名前が付いています。

ターゲット アプリケーション

配置

「Mobile」システムは5つの実行形式ファイルに分かれます。MobileStationの各ブロック インスタンスに対応して、4つの実行形式ファイルを作ります。それ以外のブロック インスタンス(Proxy、BaseTranscieverStation、BaseStationController、MobileSwitchingCenter、Database)は、まとめて1つの実行形式ファイルになります。

グラフィカル ユーザー インターフェイス

MobileStationブロック用に、グラフィカル ユーザー インターフェイス(GUI)を用意します。Cのソース コードの形で提供されるものをコンパイルし、MobileStationに対応する4つの実行形式ファイルを生成する際にリンクすることになります。画面インターフェイスは典型的な携帯電話を模擬したものです。簡単なメニュー システムが付いており、電話番号簿や課金レポートを呼び出せるようになっています。また、ウィンドウ下部のオプション メニューから、基地局を切り替えることができます。実際のGUIを図212に示します。

図212 携帯電話のGUI

TCP/IP通信

ターゲット アプリケーションの実行形式ファイルは、TCP/IPを介して、互いに信号を送り合って通信します。実際に通信処理を行うのはSDL SuiteのTCP/IP通信モジュールで、Cのソース コードの形で提供されています。これをコンパイルし、実行形式ファイルにリンクすることになります。

信号の受け取り手を特定できるよう、送信先実行形式ファイルのホスト名とTCPポート番号を、TCP/IPモジュールに知らせる必要があります。そのために、経路制御機能を主導でCに実装しなければなりません。

モバイル システムにはデフォルトの経路制御機能が付属しているので、経路制御情報をわざわざ与えなくても、コンピュータ上で実行形式ファイルを動作させることは可能です。

図213に、実際に配置されるモバイル システムを構成する実行形式ファイルの、UMLコンポーネント ダイアグラムを示します。各実行形式ファイルはTCP/IPサーバーを設定し、ある番号のTCPポートを監視します。リモートの実行形式

ファイルがSDL信号を送る際にも、この番号を使います。TCP/IPサーバーは、各コンポーネントのインターフェイスとして表示されています。破線の矢印は、あるコンポーネントから別のコンポーネントへの信号の流れを表します。

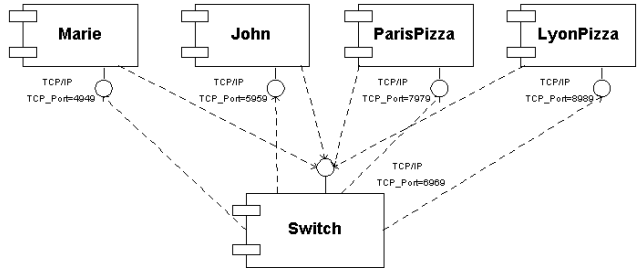


*図213:*配置されるモバイル システムを表すコンポーネント ダイアグラム

実習の準備

例題システムのコピー

実習に必要な例題システムを、SDL Suiteの組み込みディレクトリから、適宜作成した作業ディレクトリにコピーしておきます。

UNIX

`$telelogic/sdt/examples/mobile`
以下のファイルをすべて、作業ディレクトリ(例:`~/mobile`)にコピーしてください。

Windows

`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥sdt¥examples¥mobile`
以下のファイルをすべて、たとえば次のような作業ディレクトリにコピーしてください。
`C:¥IBM¥Rational¥SDL_TTCN_Suite6.3J¥work¥mobile`

システムの開き方

オーガナイザからシステム ファイル`Mobile.sdt`を開いてください。

配置ダイアグラムの作成

スレッド実行形式ファイルを生成する際、アプリケーションのスレッドに関する情報を設定する必要があります。すなわち、各スレッドでどのSDLインスタンス セットを実行するか、を指定するのです。その後、SDLシステムをいくつかの実行形式ファイルに分割することになります。

このような情報は配置ダイアグラムの形でモデル化します。編集には配置エディタを使います。配置ダイアグラムを見れば、ターゲット プラットフォームに関わらず、SDLシステムの配置をどのようにモデル化するかがわかります。

ここで学習する事項

- 配置エディタの起動
- SDLシステムを実行形式ファイルやスレッドに配置する方法

配置エディタの起動

オーガナイザで配置シンボル「network_depl」をダブルクリックすると、配置エディタが起動され、対応するダイアグラムが開きます。

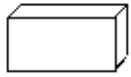
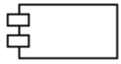
SDLシステムの配置

配置ダイアグラムには次のような機能があります:

- SDLシステムの分割
- SDLインスタンス セットを実行形式ファイルにオーガナイズ
- SDLインスタンス セットをスレッドにオーガナイズ(スレッド インテグレーションの場合のみ)

配置ダイアグラムは、ノード、コンポーネント、スレッド、オブジェクト、コメントの、5種類のシンボルを使って作成します。各シンボルについて表1に説明します。

表1 :配置ダイアグラムに使われるシンボル

シンボル	名称	説明
	ノード	計算機資源、すなわちコンピュータ
	コンポーネント	SDL インスタンス セットを含む実行形式ファイル
	スレッド	実行ポイント。このシンボルはスレッド インテグレーションの場合にしか使いません
	オブジェクト	SDLインスタンス セット。システム、ブロック、プロセスの3種類があります

各シンボルはコンポジションを使って接続します。ダイアグラム上でシンボルを選択すると、その下部にハンドルが現れます。ハンドルをクリックするとコンポジション リンクが作られます。次に接続先のシンボルをクリックすると接続されます。

各シンボルの名称はダイアグラム領域内で編集できます。シンボルによってはほかにも、[**シンボルの詳細**]ダイアログ ボックスで情報を編集できます。

配置ダイアグラム「*network_depl*」を図214に示します。

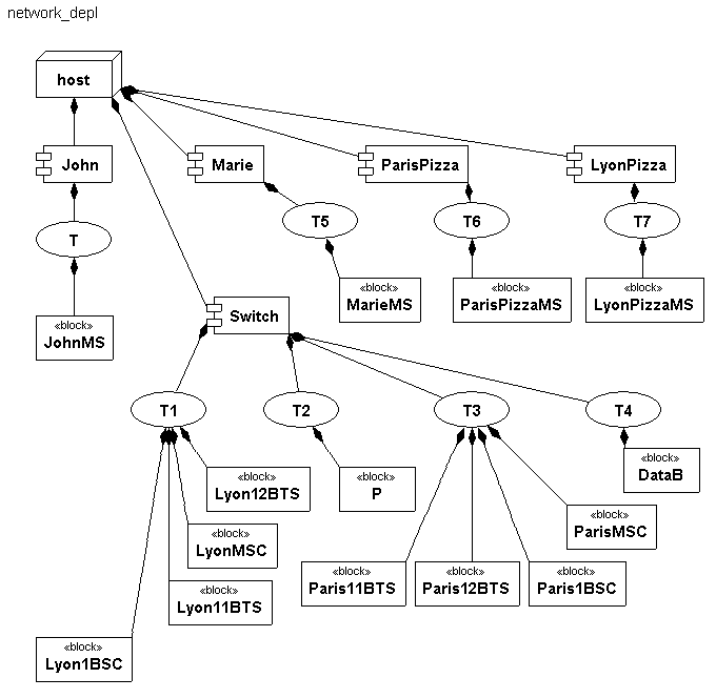


図214：配置ダイアグラム「network_depl」

ダイアグラムにはコンポーネントが5つあり、それぞれ実行形式ファイルを表しています。すべてのコンポーネントは1つのノードに属しています。また、各コンポーネントには1つ又はいくつかのスレッドがつながっています。さらに各スレッドにはオブジェクトが1つ以上属しており、スレッドの内部処理に使われるSDLインスタンス セットを表しています。

たとえば「John」コンポーネントには「T」というスレッドがつながっています。さらにこのスレッドには「JohnMS」というオブジェクトが属しており、ネットワークSDLシステムのSDLインスタンス セットを表しています。[オーガナイザ]ウィンドウを開き、システムに現れるブロック インスタンス セットを、配置ダイアグラム中のオブジェクトと比較してください。

[シンボルの詳細]ダイアログ ボックス

このダイアログ ボックスを開く方法は、次の2とおりがあります。

- ダイアグラム シンボルのダブルクリック。
- ダイアグラム シンボルを右クリックし、ポップアップ メニューから[*シンボルの詳細...*]を起動。

コンポーネント

「John」というコンポーネントをダブルクリックすると、[シンボルの詳細]ダイアログ ボックスが現れます。そのようすを図215に示します。

図215: コンポーネント「John」の、[シンボルの詳細] ダイアログ ボックス

ダイアログ ボックス中のドロップダウン リストで、インテグレーション モデルを選択してください。これにより、コンポーネントから生成される実行形式ファイルの、コード生成パラメータが決まります。[ライト]、[スレッド]、[タイト]の3つの選択肢があります。

コンポーネント「John」については、[スレッド]を選択してください。その他のテキスト ボックスの内容は、コード生成には使われません。

スレッド

スレッド シンボル「T」をクリックすると、ダイアログ ボックスの表示が、このスレッドの内容を反映するものに変わります。そのようすを図216に示します。

図216: スレッド「T」の、[シンボルの詳細] ダイアログ ボックス

スレッドについては次の4つのパラメータを指定して、実行形式ファイルの処理性能を最適化できます。

- スタック長
- スレッドの優先度
- キュー長
- 最大信号数

パラメータの値は、ターゲット オペレーティング システムごとに別々に設定してください。指定しなければデフォルト値になります。「*network_depl*」配置ダイアグラムでは、各スレッドともデフォルト値をそのまま採用します。

オブジェクト

オブジェクト シンボル「JohnMS」をクリックすると、ダイアログ ボックスの表示が、このオブジェクトの内容を反映するものに変わります。そのようすを図217に示します。

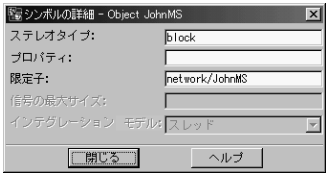


図217: オブジェクト「JohnMS」の、[シンボルの詳細] ダイアログ ボックス

オブジェクトについては、次の2つのパラメータの指定が必須です。

- ステレオタイプ
- 限定子

ステレオタイプとは、SDLインスタンス セットのタイプのことです。[システム]、[ブロック]、[プロセス]の中から選んでください。「JohnMS」のステレオタイプはブロックなので、[ステレオタイプ]テキストボックスにもそのように入力します。

限定子はSDLインスタンス セットの識別に使われます。[オーガナイザ]ウィンドウで「JohnMS」ブロックをさがすと、ブロック インスタンス セットは「network」システムのすぐ下にあります。したがって限定子は「network/johnMS」となります。

これで「*network_depl*」ダイアグラムが完成したので、次に実行形式ファイルを生成します。

配置エディタについて詳しくは『User´s Manual』の第40章、「The Deployment Editor」を参照してください。

ターゲティング エキスパートの使い方

ここで学習する事項

- ターゲティング エキスパートの起動方法
- 配置記述をもとに実行形式ファイルを生成する手順
- ターゲティング エキスパートGUIを使って、Cのコード生成に関する設定を行う方法
- ターゲティング エキスパートGUIを使って、コンパイラやリンカの設定を行う方法

ターゲティング エキスパートの起動

配置ダイアグラムを、ターゲティング エキスパートへの入力として使います。

オーガナイザ上の「*network_depl*」ダイアグラム シンボルを右クリックし、ポップアップ メニューから[*ターゲティング エキスパート*]を起動してください。

配置ダイアグラムの分析が始まります。エラーが見つかると、[オーガナイザ ログ]ウィンドウにエラー メッセージが表示されます。[オーガナイザ ログ]ツールバーの[エラーの表示]をクリックすると、配置ダイアグラム中にエラー箇所が表示されます。

これでターゲティング エキスパートが起動されました。

ターゲット プラットフォームの選択

起動したばかりの状態では、[ターゲティング エキスパート]ウィンドウは図218のようになっています。

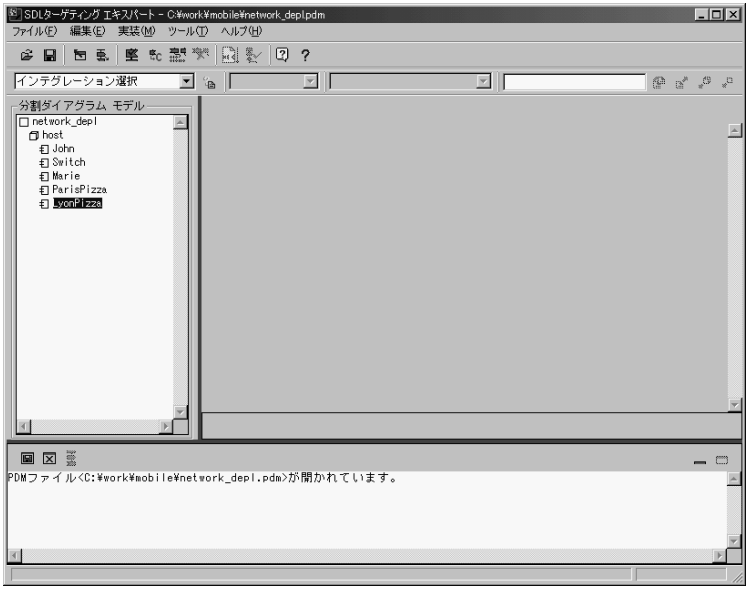


図218: 起動直後の[ターゲティング エキスパート]ウィンドウのようす

ウィンドウの左側は「*分割ダイアグラム モデル*」といい、配置ダイアグラムのうち特定の種類のシンボルを表示する部分です。ノードとコンポーネントが表示されています。

各コンポーネントに対し、次の操作を行います。

1. [*分割ダイアグラム モデル*]ウィンドウで、コンポーネントを選択してください。
2. 「インテグレーション選択」と表示されているドロップダウン リストをクリックすると、適用可能なインテグレーションを列挙したメニューが開きます。ただしコンポーネントの場合、配置ダイアグラムで[スレッド]が指定されているので、ほかの選択肢はありません。
3. サブ メニューからプラットフォームを選択してください。
   - **Windows:** [*Win32スレッド(CAdvanced)*]を選択
   - **Solaris:**[*Solarisスレッド(CAdvanced)*]を選択
   - **Linux:**[*Linuxスレッド(CAdvanced)*]を選択

SDLシステムを自動生成してよいか問い合わせるダイアログ ボックスが開きます。[*はい*]をクリックしてください。

[ターゲティング エキスパート]ウィンドウは図219のようになります。

図219: インテグレーション モデルが選択されたコンポーネント

Cのコード生成に関する設定

ターゲティング エキスパートのうち、Cのコード生成に関するGUIには、[*SDL to C コンパイラ*]、[*通信*]、[*環境ヘッダー ファイル*]、[*実行*]の4つのタブがあります。それぞれ、実行形式ファイルのコード生成方法を設定する画面です。

生成される各実行形式ファイルは、外部コードを介して通信し合うことになります。実行形式ファイルはすべてTCP/IP通信モジュールを使います。また、携帯電話に対応する実行形式ファイル(John、Marie、ParisPizza、LyonPizza)にはGUIが組み込まれています。外部コードは環境関数を介してSDLシステムと接続されます。環境関数の生成に関する設定は手動で行わなければなりません。

TCP/IPモジュールを簡単に設定できるよう、ターゲティング エキスパートにはウィザードが付属しています。これを起動すると、必要な環境関数がデフォルトですべて自動生成されます。

TCP/IPウィザードのダイアログ ボックスを図220に示します。

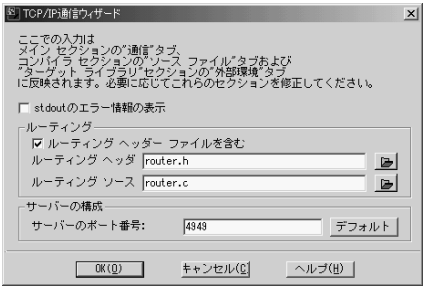


図220: TCP/IP ウィザード

信号を正しい送信先に送るため、TCP/IPモジュールに経路制御情報を与える必要があります。Cのヘッダー ファイルに経路制御関数を宣言し、これをインクルードしなければなりません。また、経路制御関数の定義は、Cのソース ファイルに記述することになります。入力信号のTCPポート番号も指定しなければなりません。ポート番号はリモートの実行形式ファイルから信号を送る際に必要です。

各コンポーネントに対し、次の操作を行います。

1. 左側のサブ ウィンドウで、インテグレーション名をクリックしてください。
   - **Windows:** [*Win32 Threaded*]を選択
   - **Solaris:** [*Solaris Threaded*]を選択
   - **Linux:**[*Linux Threaded*]を選択
2. [*通信*]タブを選択してください。
3. [*信号の送信*]グループ内の[*TCP/IP*]チェック ボックスをオンにすると、TCP/IPウィザードが開きます。
4. [*ルーティング ヘッダー ファイルをインクルードする*]チェック ボックスをオンにしてください。

5. ヘッダー ファイル名を指定します。ヘッダー ファイルテキスト ボックスの右側にあるファイル ボタンをクリックすると、ファイル選択ダイアログ ボックスが開きます。

6. 作業ディレクトリにある、ルーティングのヘッダー ファイルを選択してください。

   – John、Marie、ParisPizza、LyonPizzaのいずれかの実行形式ファイルを生成する場合は*router.h*を選択。

   – 実行形式ファイルSwitchを生成する場合は*switchrouter.h*を選択。

7. [*開く*]をクリックしてください。

8. ソース ファイル名を指定します。テキスト ボックスの右側にあるファイル ボタンをクリックすると、ファイル選択ダイアログ ボックスが開きます。

9. 作業ディレクトリにある、ルーティングのソース ファイルを選択してください。

   – John、Marie、ParisPizza、LyonPizzaのいずれかの実行形式ファイルを生成する場合は*router.c*を選択。

   – 実行形式ファイルSwitchを生成する場合は*switchrouter.c*を選択。

10. [*開く*]をクリックしてください。

11. 生成する実行形式ファイルに対応したサーバー ポート番号を、表2に従って設定してください。

表2:実行形式ファイルのTCPサーバー ポート番号

実行形式ファイル	TCPポート番号
Marie	4949
John	5959
ParisPizza	7979
LyonPizza	8989
Switch	6969

12. TCP/IPウィザードのダイアログ ボックスにある、[*OK*]をクリックしてください。

これでコード生成に関する設定は終わりです。

コンパイルやリンクに関する設定

*ターゲティング エキスパートの[コンパイラ/リンカ/実装]セクションには、[コンパイラ]、[ソース ファイル]、[追加コンパイラ]、[リンカ]、[ライブラリ マネージャ]、[実装]*の6つのタブがあります。画面のようすを図221に示します。

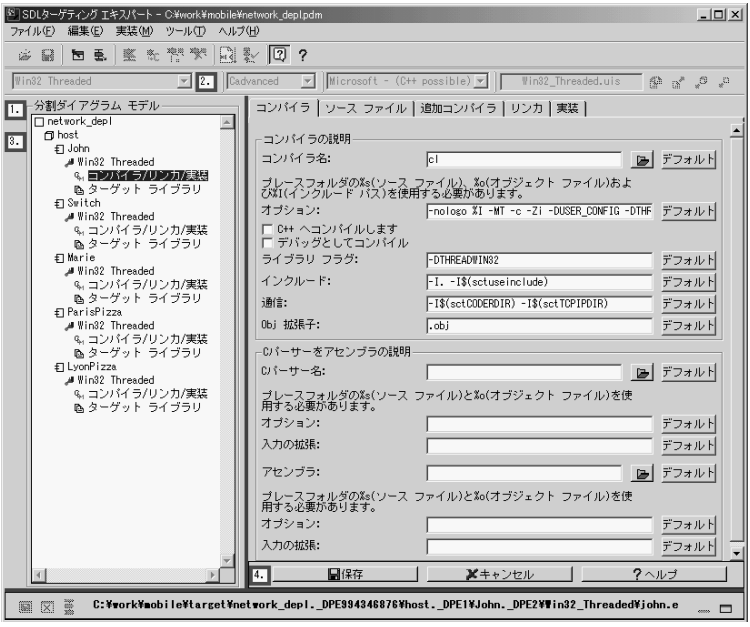


図221: ターゲティング エキスパートの[コンパイラ/リンカ/実装]セクション

*実行形式ファイルSwitch*を生成するためのコンパイル、リンク手順

実行形式ファイル*Switch*を生成するためのコンパイル、リンク手順を以下に示します。

1. [*分割ダイアグラム モデル*]ウィンドウで[コンパイラ/リンカ/実装]をクリックしてください。
2. [*インクルード*]ボックスに「-I../../../../../」と追記してください。テキスト ボックスは図222のようになります。

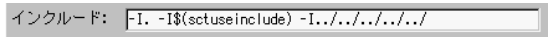


図222 : -Iフラグを追加した[インクルード]ボックスのようす

3. [*保存*]をクリックしてください。
4. [分割ダイアグラム モデル]ウィンドウで、*Switch*に対する[*ターゲット ライブラ*]をクリックしてください。
5. 必要に応じ、[*カーネル*]タブで、次のチェック ボックスを設定してください。
   - [標*準出力にエラーを表示する*]。オンにすると、実行時にエラーが発生した場合、ログ メッセージが表示されるようになります。
   - [*テキスト トレース*]。オンにすると、実行時、標準出力にテキスト トレースが出力されるようになります。
6. [*保存*]をクリックしてください。
7. [*完全な実装*]をクリックすると、SDLシステムの分析後、コードおよび実装ファイルが生成され、ついで実行が始まります。

これでSwitchという実行形式ファイルが生成されました。ファイル拡張子は、Windowsの場合は「.exe」、Unixの場合は「.sct」になります。

MobileStationの各実行形式ファイルを作成するためのコンパイル、リンク手順

*MobileStation*の各実行形式ファイルを生成するためのコンパイル、リンク手順を以下に示します。

1. [分割ダイアグラム モデル]ウィンドウで、*MobileStation*の各インスタンスの、[*コンパイラ/リンカ/実装*]をクリックしてください。
2. [*インクルード*]ボックスに「`-I<your working directory>, e.g. -I/home/mobile.`」と追記してください。テキスト ボックスは図222のようになります。

3. [*ライブラリ フラグ*]ボックスに、次のフラグを追加してください。
   - `-DXMAIN_NAME=SDL_Main`
   - `-DXEXTENV_INC="`<gui.h>"
   - 作成しようとしている実行形式ファイルに応じて、`-DMARIE`、`-DJOHN`、`-DPARISPIZZA`、`-DLYONPIZZA`のいずれか

「`XMAIN_NAME`」フラグを指定すると、SDLシステムの関数名mainが置換されます。関数自体はGUIソース コード内にあります。SDLのmain関数が「`SDL_Main`」という名前に変わり、スレッド内の新しいmain関数から呼び出されるようになります。

「`XEXTENV_INC`」フラグは、TCP/IPモジュールから環境コードを使う旨の指定です。ファイル*gui.h*には、生成される環境ファイルで使われるマクロの定義があります。

4. [*ソース ファイル*]タブをクリックすると、コンパイルするべき外部ファイルが列挙されます。
5. [追加]をクリックするとファイル検索ダイアログ ボックスが現れます。
6. 作業ディレクトリにある*gui.c*を選択し、[*開く*]をクリックしてください。ファイル一覧に追加されます。
7. **Windowsの場合のみ**: リソース コンパイラでGUIをコンパイルする必要があります。
   - [*コンパイラ名*]ボックスに「*rc*」と入力してください。
   - [オプション]ボックスに「*-l 0x41d %I -fo %o %s*」と入力してください。
   - コンパイルするべきファイルとして「*gui.rc*」を追加します。[*追加*]をクリックし、作業ディレクトリにある*gui.rc*を選択してください。
   - [*オブジェクトの拡張子*]ボックスに「*.res*」と入力してください。
   - [実装]タブをクリックしてください。
   - [実装ツール]ドロップダウンリストで、「Microsoft nmake (一時応答ファイルを使用)」を「Microsoft nmake」に変更してください。
8. [*リンカ*]タブをクリックしてください。

9. [*オプション*]ボックスに、次のように入力してください。
   - **Windows:**「*-subsystem:console*」という記述を「*-subsystem:windows*」に変更。
   - **Unix:**「*-lXm -lXt -lX11*」という記述を、「*-lpthread*」と「*-L/usr/lib*」の間に追加。
10. [*保存*]をクリックしてください。
11. [*完全な実装*]をクリックすると、SDLシステムの分析後、コードおよびメイクファイルが生成され、ついで実行が始まります。

これで実行形式ファイルが生成されました。拡張子は、Windowsの場合は「.exe」、Unixの場合は「.sct」になります。

ターゲット システム

実行形式ファイルは、ターゲティング エキスパートが作成するディレクトリ構造以下に生成されます。ターゲティング エキスパートは、オーガナイザで指定されたターゲット ディレクトリをルートとして扱います。

モバイル システムでは、ターゲット ディレクトリとして「*target*」が使われます。生成されるターゲット ディレクトリ構造を図223に示します。

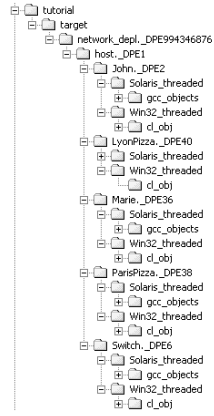


図223:：生成されるターゲット ディレクトリ構造

実行形式ファイルは、プラットフォームによって別々の場所(*Win32_threaded*または*Solaris/Linux_threaded*)に生成されます。オブジェクト ファイルが生成されるのはそれぞれのサブ ディレクトリです。

システムの実行

ここで学習する事項

- ターゲティング エキスパートで生成された実行形式ファイルを、実際に動かす方法
- モバイル システムの使い方

システムの概要

実行形式ファイルMobileStationの実行時アーキテクチャを図224に示します。

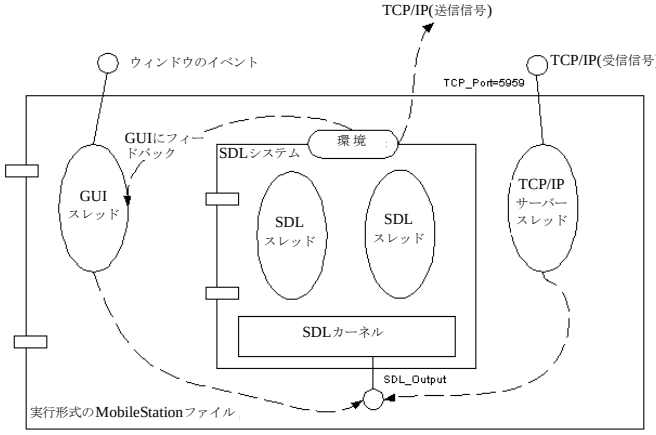


図224: 実行形式ファイルMobileStationの実行時アーキテクチャ

SDLシステムは環境を介して外部とやり取りします。GUIメッセージ ループは、それ自身の実行スレッド内で動作します。TCP/IPサーバー スレッドも同様です。環境スレッドはSDLシステムと、信号を挿入することによってやり取りします。これは、SDLカーネル関数`SDL_Output`の呼び出しという形で実現されています。

SDLシステムから環境に送信された信号は、GUIを経てフィードバックされるか、またはTCP/IPを使って外部の受け取り手に送られます。

システムの使い方

モバイル システムは次の順序で起動してください。

1. 実行形式ファイルMarie、John、ParisPizza、LyonPizzaを起動します。それぞれのGUIが画面に現れます。
2. 次に実行形式ファイルSwitchを起動します。SwitchはMobileStationの各インスタンスの実行形式ファイルに信号を送って初期化します。携帯電話ウィンドウの用意が整うと、[*オン*]ボタンが現れます。

通話は次のような手順で行います。

1. (発信側)MobileStationの[*オン*]をクリックしてください。PINコードを入力するよう求められます。PINコードは表3に示すとおりです。

表3:MobileStationのPINコード

実行形式ファイル	PIN コード
Marie	1
John	2
ParisPizza	3
LyonPizza	4

2. [*OK*]をクリックすると「PIN OK」と表示されます。これで通話の準備が整いました。
3. [*OK*]をクリックするとGUIのメニュー システムが使えるようになります。[<]ボタン、[>]ボタンで電話番号簿を選び、[*OK*]をクリックしてください。
4. 電話番号簿から通話相手の名前を選んで[OK]をクリックしてください。受信側のMobileStationがオンになっていれば、「Incoming Call」と表示されます。
5. 受信側のMobileStationの[*OK*]をクリックすると呼が確立し、通話状態になります。
6. 通話を終えるときは、発信側および受信側の[OK]をクリックしてください。

ここでは詳しく述べませんが、システムにはほかにもさまざまな機能があります。いろいろ実験してみてください。

これでチュートリアルを終わります。スレッドインテグレーションやTCP/IPモジュールについてさらに詳しく知りたい方は、『User´s Manual』の第64章、「Integration with Operating Systems」をお読みください。