ソフトウェア工学 83－9
（1992. 2. 6）

大規模通信ソフトウェアの統合化開発支援環境
－ 上流設計の統合化支援システム －

金地 克之　　加賀谷 聡
株式会社 東芝 システム・ソフトウェア技術研究所

大規模な通信システムのプロトコルを、シーケンス・チャートを用いて開発する場合の問題点を明確にする。膨大な量の仕様書を作成・管理するために、設計するシステム全体の振る舞いをイベント・ツリーとして記述し、そのイベント・ツリーに基づいて、シーケンス・チャートによるプロトコルの設計・保守・管理までを一貫して支援する方法を提案する。提案する方法では、イベント・ツリーを用いることにより、仕様書間の関連が明確になると共に、イベント・ツリーに基づいたプロトコルの差分プログラミングを実現し、イベント・ツリーによる修正の支援機能を実現できる。

An Integrated Development Supporting Environment for Large Scale Communications Software
- An Integrated Upper Design Supporting System -

Katsuyuki Kanaji and Akira Kagaya
Systems & Software Engineering Laboratory, Toshiba Corporation
70, Yanagi-cho, Saiwai-ku, Kawasaki-shi, 210 Japan

For large scale communications systems, we clarify problems of protocol design methods based on sequence chart specifications. We propose a method to resolve a part of these ploblems that protocol designers have to design and manage a lot of specifications. In our proposed method, protocol designers design event-trees which specify the behavior of a entire communications system. Then our supporting system can support them utilizing event-trees for designing, managing and modifying protocol specifications. Our system prepares a differential programming technique in the designing phase, presents the relationship of specifications visually in the managing phase, and automatically modifies the specifications relating to designers modified in the modifying phase.

１．はじめに

近年、通信システムの大規模化、複雑化にともなって、通信ソフトウェアの効率的な開発・保守・管理を統合的に支援するための、統合化ＣＡＳＥ環境が強く望まれている。

通信ソフトウェアを開発する際に重要なことは、通信プロトコルを厳密に記述し、そのプロトコルの正しさを確認できることである。これまでの研究では、通信プロトコルを厳密に記述するために、ＳＤＬ（Specification and Description Language）やＬＯＴＯＳ（Language of Temporal Ordering Specification）などの形式的仕様記述言語を用いた研究が行われてきた。しかし、現在のところ、実現レベル（プロトコルを設計・検証し実行させることができるレベル）としてはＳＤＬが広く一般に普及し、製品開発においても使用されている。

更に、ＳＤＬによるプロトコル開発では、プロトコルの設計や保守を効率よくするために、シーケンス・チャートでプロトコルの設計・検証を行ってから、それらのプロトコル記述に対応したＳＤＬ記述を自動的に合成するシステムが研究・開発[1]されている。

本論文では、シーケンス・チャートによるプロトコル設計、保守、管理における問題点を明らかにして、大規模な通信システムを効率よく、高信頼性のもとに分散開発するための通信ソフトウェア開発支援環境について報告する。

第２章で、従来のシーケンス・チャートによる設計の問題点を明らかにし、第３章で、問題点に対して我々が提案する解決法の概要について説明する。そして、第４章で、我々が開発している大規模通信ソフトウェア開発支援システムの概要について説明した後で、我々が提案する解決方法の実現について述べる。最後に、第５章でまとめと今後の課題について述べる。

２．シーケンス・チャートによるプロトコル設計

２．１　従来の研究

シーケンス・チャートによるプロトコル設計では、通信システムを構成するプロセス群において、プロセス間での信号の処理手順（プロトコル）を、図形を用いてビジュアルに設計[2]する。シーケンス・チャートによるプロトコル設計の例を図１に示す。図１のシーケンス・チャートはプロセスＰ１，Ｐ２間での信号の処理手順を示しており、サービス１では、Ｐ１が外部から信号 call を受けた場合に、Ｐ２に信号 call を送信し、Ｐ２から信号 ack を受けることを記述している。

このようなシーケンス・チャートによるプロトコルの設計では、システムが提供するサービスの数だけシーケンス・チャートを記述する（サービス１、サービス２）。そして、シーケンス・チャートでプロトコルの設計を行った後に、プロトコル検証を行って、各プロセス（Ｐ１、Ｐ２）のＳＤＬ記述を合成する。

シーケンス・チャートによる設計では、システムが提供するサービスにおける信号の処理手順を、ビジュアルに設計できるため、個々のサービスに対してはプロトコル設計や理解が容易であり、ドキュメントとしても優れた特徴を持つ。

２．２　従来設計の問題点

シーケンス・チャートによるプロトコル設計では、上述の利点があるものの、大規模の通信システムを開発する際に、以下のような問題点が生じる。

（１）シーケンス・チャートの記述枚数

設計すべきシーケンス・チャートの枚数が、設計するシステムのプロセス数と、システムに対する外部イベント数、そしてシステム内部での条件判断数に比例して増加し、膨大な数になる。

なぜなら、それぞれのシーケンス・チャートは、グローバル状態遷移図の遷移可能な１遷移系列を表しており、規模の大きなシステムになると膨大な数のシーケンス・チャートを設計することになるからである。

（２）シーケンス・チャートの管理

設計するシーケンス・チャートの枚数が増えるに従って、設計するシーケンス・チャート間の関連が分かりにくくなる。このため、シーケンス・チャートの設計し忘れとか、シーケンス・チャートの修正のし忘れといった問題が生じる。

（３）シーケンス・チャートの設計と修正

プロトコルは、正常なシーケンスに対して割り込みがあった時点で異なるシーケンスを実行するのが特徴である。また、シーケンス・チャートによる設計では、システムが提供するサービスを全て記述する必要がある。このため、設計したサービスの数が増えるに従って、部分的に同じ記述を何度も記述することになる。

また、部分的に同じ記述が複数のシーケンス・チャートに分散しているため、ある修正が他のシーケンス・チャートに影響を与える場合が多く、シーケンス・チャートの修正のし忘れといった問題が生じる。

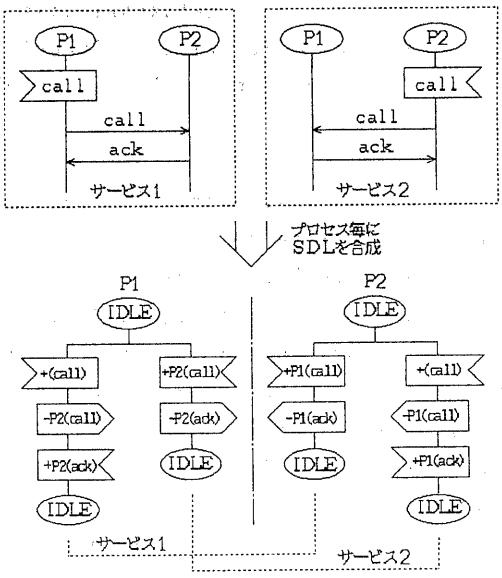


図１　シーケンス・チャートによるプロトコル設計

以降では、それぞれの問題点に対処した、統合的な開発支援システムについて説明する。

3．問題点に対する解決方法の概要

この章では、2章で述べたそれぞれの問題点に対する我々の提案する解決方法について述べる。

3．1　シーケンス・チャートの記述枚数の削減

シーケンス・チャートの記述枚数を削減するために、我々は、要求定義や機能分析後に得られるプロセス関連図を用いて、プロセス群を分割して、抽象的なプロトコル設計から、段階的に詳細化しながらプロトコルを設計する設計手法を提案した[3,4]。

提案した方法には、プロセスの抽象化による設計手法と、チャネルの抽象化による設計手法の2つがある。これらの設計手法は共に、プロセス群の分割処理を行って、シーケンス・チャートの分散開発を行い、目的とする全体システムのシーケンス・チャートを、自動的に合成する方法である。これらの設計手法により、プロトコル設計者が記述すべきシーケンス・チャートの枚数を、著しく削減することができる。

3．2　シーケンス・チャートの管理方法の提案

プロトコル開発において、設計されたシーケンス・チャート間の関連（正常処理、異常処理、割込処理などの関連）が不明確になるという問題点については、これまでのところ余り研究されていない。我々はこれらの問題に対して、従来からプロトコル開発で作成されているイベント・ツリー（プロセス群の振る舞いを規定したもの）を用いて視覚的にシーケンス・チャートを管理する方法を提案する。

イベント・ツリーは図2に示すように、設計を行うプロセス群に対する外部信号の受信と、外部信号の受信に対するシステム内部での条件判断に基づいて、プロセス群の外部仕様を記述したものである。

図2では、”+ 信号名”が、プロセス群に対する外部信号の受信を示している。また、図2には記述されていないが、”if 条件文”がシステム内部での条件判断の記述である。イベント・ツリーのそれぞれの遷移系列（ルートノードから、リーフノードまでのパス、例えば、+OffHook,+OnHook,+UnAuthorizedなどのイベントの遷移系列）が、それぞれ1枚のシーケンス・チャートに対応している。イベント・ツリーのリーフノードを用いてそれぞれのシーケンス・チャートを管理することで、シーケンス・チャート間の関係を視覚的に理解することができる。

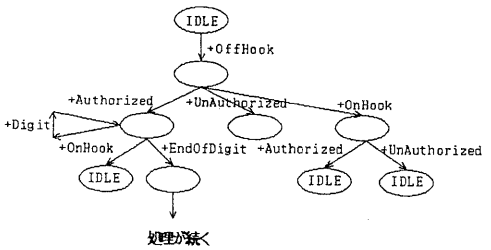


図2　イベント・ツリーの例

3．3　シーケンス・チャートの設計と修正の支援機能

設計したサービスの数が増えるに従って、部分的に同じ記述を何度も記述する必要があるといった問題や、修正する際に修正のし忘れが生じ易いという問題に対しては、イベント・ツリーによる仕様書管理機能を用いて仕様書の設計や修正を統合的に支援する方法を提案する。

イベント・ツリーによる設計・修正支援機能では、シーケンス・チャートでプロトコルを設計する際に、設計するプロセス群に対する外部信号の受信か、あるいは、システム内部での条件判断から、次の外部信号の受信か、あるいは、プロセス群内部での条件判断の前までをプロトコル単位（以下、モジュールと呼ぶ）とする。そして、各モジュールをイベント・ツリーの対応するエッジに割り当てて管理する。

これらのイベント・ツリーとモジュール管理機能を用いて、以下に示す差分プログラミング機能と、変更支援機能を実現することで、部分的な記述の繰り返しを削減し、修正のし忘れを削減することができる。

（1）シーケンス・チャートの設計支援

差分プログラミングによる設計では、最初のシーケンス・チャートを設計する際は、これから設計する遷移系列をイベント・ツリーで選択し、その遷移系列上の全てのモジュールを設計する。

以降のシーケンス・チャートの設計においては、設計する遷移系列を選択した場合に、既に設計されているモジュールを、これから設計するシーケンス・チャートに自動的に組み込み、設計されていないモジュールの設計のみを行う。そして、新しく設計したモジュールの登録・更新を行う。

例えば、図2で、右端のリーフノード以外のリーフノードに対応するシーケンス・チャートが既に設計済みであり、これから右端のリーフノードに対応するシーケンス・チャートを設計する際には、+OffHook と、+OnHook のモジュールがシーケンス・チャート内に自動的に組み込まれるので、残りの +UnAuthorized に相当するモジュールを設計するだけでよい。

このようにして、イベントツリーに従ってプロトコルを設計することにより、プロトコルの差分プログラミングを実現し、部分的に同じ内容の記述を繰り返すことを削減することができる。

（2）シーケンス・チャートの修正支援

部分的に同じ記述を、モジュールとしてイベント・ツリーのエッジで一元管理しているので、あるシーケンス・チャートにおいてモジュールを修正した場合には、そのモジュールを使用している他のシーケンス・チャートでも、その修正に合わせて表示を替えることができる。このため、シーケンス・チャートの部分的な修正による影響箇所の修正し忘れといった問題を解決することができる。

例えば、図2の右側の2つのリーフノードに対応するシーケンス・チャートでは、+OffHook と +OnHook のモジュールを共有しており、一方のシーケンス・チャートにおいて、+OnHook のモジュールを修正した場合には、他のシーケンス・チャートでも該当する部分を自動的に修正する必要がある。

更に、シーケンス・チャートにおいて、ある部分のモジュールを修正した場合に、そのシーケンス・チャートの別なモジュールにも影響を与える場合がある。このた

め、我々のシステムでは、イベント・ツリーにおいて、モジュールの修正が影響を及ぼす範囲を、イベント・ツリーのビュー機能として提供する。

以上のように、3.1で述べた方法でシステムを分割し、分割されたサブシステム毎に、3.2、3.3のようにイベント・ツリーを用いて仕様書の作成から管理、修正までを一貫して支援することで、2章で述べた問題点を効率的に解決することができる。

4．大規模通信ソフトウェア開発支援システム

この章では、4.1節で、現在、我々が開発中の大規模通信ソフトウェア開発支援システムと、その中の上流設計の統合化支援システムの概要について説明し、4.2節で、イベント・ツリーの実現、作成及び修正方法について説明する。そして、4.3節で、イベント・ツリーを用いた仕様書の作成と修正方法について説明し、4.4節では、イベント・ツリーや仕様書の作成におけるシステムの統合化支援について説明する。

4．1　上流設計の統合化支援システムの構成

（1）システム構成

大規模通信ソフトウェア開発支援システムのシステム構成を図3に示す。

大規模通信ソフトウェア開発支援システムは、上流設計の統合化支援システムと、ツール群を統合化する統合化ＧＵＩ（Graphical User Interface）、ＮＴＴが開発し市販しているプロトコル検証システムＳｏｆｔＳＤＥ（Software Systems Development Environment）、プロトコルのドキュメント自動生成システムであるＫＩＮＤＲＡ（British Telecommunications plc が開発）からなる。

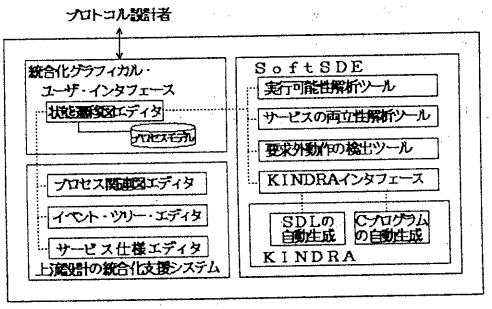


図3　上流設計の統合化支援環境

上流設計の統合化支援環境は、ＰＲＤ（Process Relation Diagram）エディタ、ＥＴ（Event Tree）エディタ、ＳＣ（Sequence Chart）エディタと、モジュール・データベースで構成される。これらの上流設計の統合化支援ツール群を用いて、シーケンス・チャートによるプロトコルの設計・管理・修正までの一貫した支援を行うことができる。

上流設計の統合化支援ツール群を使用してシーケンス・チャートを生成した後、生成されたシーケンス・チャート記述を、ＳｏｆｔＳＤＥへの入力データとしてのＳＡＬ（Service Addition Language）記述に変換する。

更に、ＳＡＬ記述に変換後は、ＳｏｆｔＳＤＥのプロトコル検証機能と、ＫＩＮＤＲＡのドキュメント生成機能を用いて、ターゲットのＳＤＬ記述を生成することができる。

（2）ツール間を統合する統合化ＧＵＩ

プロトコルの設計者は、上記のようにプロトコルを開発して保守するまでに、様々なツールをいろいろな状況で使い分ける必要がある。しかし、プロトコル設計者が、これらのツールの使用方法（コマンドや、オプション）や、これらのツールを用いた設計手法をマスターすることは非常に負担が大きい。これらの問題点が、実際にＣＡＳＥツールが普及しない原因の1つになっている。

このため、我々は、プロトコルの開発作業を分析して、その標準的な作業手順を状態遷移図で記述したプロセスモデルを作成し、このプロセスモデルに基づいて、ツール間を統合化するグラフィカル・ユーザ・インタフェースを開発した[5]。

我々が開発した統合化ＧＵＩ（図4）では、プロトコル設計者は、標準的なプロトコルの開発作業を、メニュー選択の形で行うことができる。我々のシステムでは、メニューには作業状態に合わせて、次に行うべき処理あるいは、次に行っても良い処理のみを表示する。

また、我々の開発した統合化ＧＵＩでは、プロトコル検証中にエラーがあった場合に、プロトコルエラーの種類を特定する為の情報と、プロトコルエラーを修正するための情報を収集し、プロトコル設計者に自動的に提示する機能を備えている。この結果、プロトコル設計者は、システムが提供するエラー情報を基にして、メニューに従って作業を行うことにより効率的に正しくエラーを修正することができる。

この結果、プロトコル設計者は、ＣＡＳＥツールの使用方法を意識することなくプロトコルの開発・保守作業に専念することができる。

4．2　イベント・ツリーの作成・修正支援

4.2.1で、イベント・ツリーの実現形式について説明し、4.2.2で、イベント・ツリー作成後の修正支援について説明する。また、4.2.3で、イベント・ツリーを作成および修正する際に有効な支援機能について説明する。

4．2．1　シーケンス・チャートとモジュールの管理

イベント・ツリーによりシーケンス・チャートとモジュールを管理する機能について説明する。イベント・ツリーに対応するテキストファイルを図5に示す。

イベント・ツリーＥＴ：

イベント・ツリーは、状態遷移図で表現されており、状態Ｓと、状態遷移Ｌの集合として記述されている。

状態Ｓ：

状態Ｓは、イベント・ツリーの各ノードに対応しており、初期状態S_1（ルートノード）、中間状態S_m（中間ノード）、終了状態S_e（リーフノード）からなる。

そして、各状態Ｓは、状態情報（遷移元の状態S_{org}と、状態遷移可能な数の情報）を持っており、更に、初期状態S_1と、中間状態S_mは、状態遷移Ｌを持っている。

また、終了状態S_eは、シーケンス・チャート名と、そのシーケンス・チャートが編集済みであるかどうかの情報（フラグがＹなら編集済み、Ｎなら未編集）を持っている。

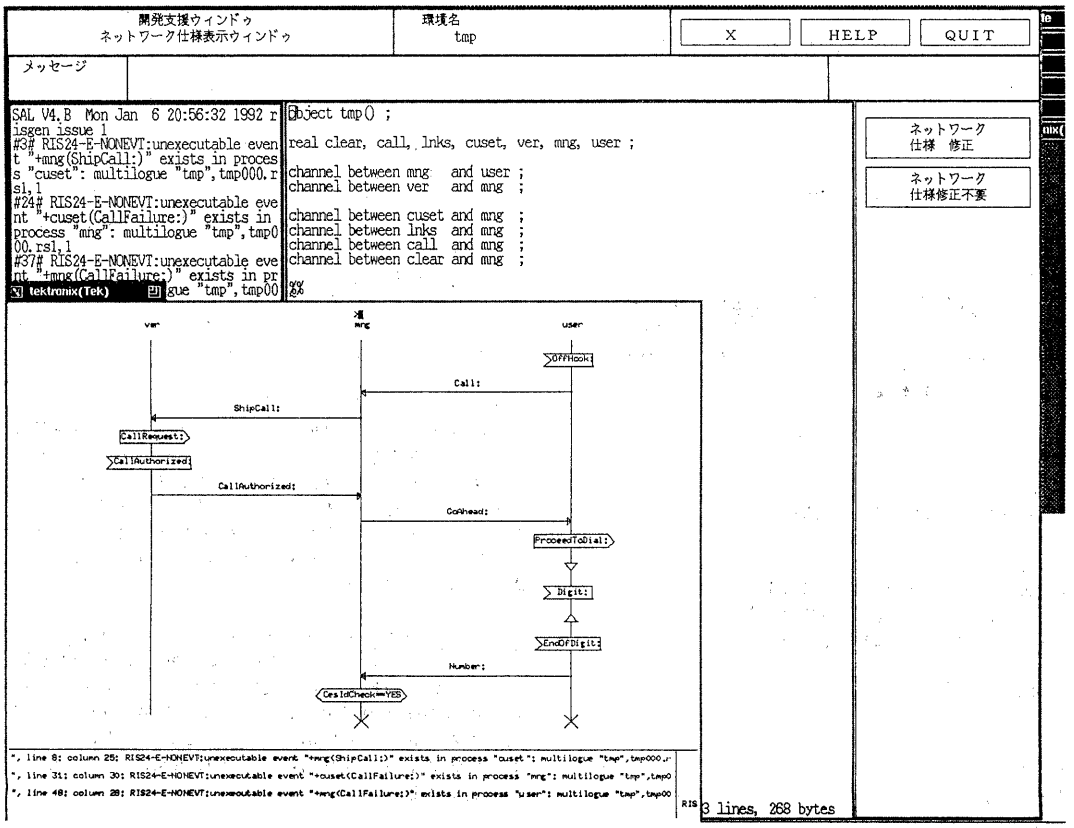


図４　統合化ＧＵＩの画面例

状態遷移Ｌ：

状態遷移Ｌは、イベント・ツリーのエッジに対応しており、イベント名（外部信号の受信は、+信号名で表されており、条件判断は、if条件文で表されている）、モジュール番号と、そのモジュールが編集済みであるかどうかの情報（フラグがＴなら編集済み、Ｆなら未編集である）を持っている。

そして、モジュール番号を用いて、モジュール・データベースに蓄えられているモジュールを管理する。

４．２．２　イベント・ツリーの修正支援

イベント・ツリーは、イベント・ツリー・エディタを用いて作成する。作成されたイベント・ツリーはテキスト形式に変換されて（上述（１））、シーケンス・チャートとモジュールを管理する為の情報がリンクされる。このため、一度作成したイベント・ツリーを修正する場合には、シーケンス・チャートとモジュールの管理情報（リンク）の張り替え作業が必要になる。

我々のシステムでは、次のようなイベント・ツリーの修正機能を提供している。

（ａ）リーフノードにイベントを付加する場合

この場合、例えば、図５のリーフノードＳ３に、イベント（+OnHook）を付加する場合。次に示すように、イベント・ツリー・ファイルのシーケンス・チャートを管理するためのリンクを操作する。

```
Ｓ３：
  {S1,1}
  {(+OnHook,(module058,F),S59)}
Ｓ５９：
  {S3,0}
  {(chart034,N)}
```

（ｂ）リーフノードへ遷移するイベントを削除する場合

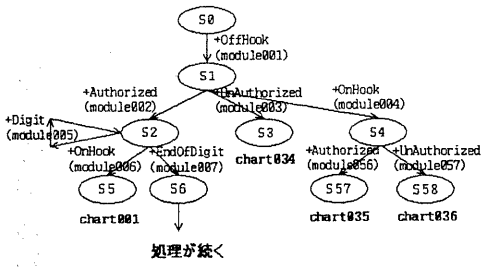
リーフノードの親ノードでの分岐数が１の場合（（ａ）で述べた図５のＳ５９へ遷移するイベント（+OnHook）を削除する場合）には、イベント・ツリー・ファイルのシーケンス・チャートを管理する為のリンクを操作する。

```
Ｓ３：
  {S1,0}
  {(chart034,N)}
```

リーフノードの親ノードでの分岐数が２以上の場合（図５のＳ５８へ遷移するイベント　(+UnAuthorized）を削除する場合）には、イベントとリーフノードを削除するだけでよい。

```
Ｓ４：
  {S1,1}
  {(+Authorized,(module056,F),S57)}
```

（ｃ）イベント系列中のイベントを挿入／削除する場合
　この場合には、リスト処理における挿入／削除のポインタ操作と同様に、状態の挿入／削除と、状態の遷移元、遷移先をリンクし直せばよい。



（ａ）イベント・ツリー

```
Ｓ０：
  {S0,1}
  {(+OffHook,(module001,T),S1)}
Ｓ１：
  {S0,3}
  {(+Authorized,(module002,T),S2),
   (+UnAuthorized,(module003,T),S3),
   (+OnHook,(module004,T),S4)}
Ｓ２：
途中省略

Ｓ３：
  {S1,0}
  {(chart034,Y)}
Ｓ４：
  {S1,2}
  {(+Authorized,(module056,T),S57),
   (+UnAuthorized,(module057,F),S58)}
Ｓ５７：
  {S4,0}
  {(chart035,N)}
Ｓ５８：
  {S4,0}
  {(chart036,N)}
```

（ｂ）テキストファイルの例（(a)に対応）

図５　イベント・ツリーのデータ構造

４．２．３　イベント・ツリーによるその他の支援機能

　イベント・ツリーにおけるその他の支援機能には、イベント・チェック機能と、イベント・ビュー機能の２つがある。

（１）イベント・チェック機能

　ユーザがシーケンス・チャートを設計する前に、イベント・ツリー・エディタでイベント・ツリーを作成する。イベント・ツリーの作成において、イベント・ツリー上でイベントのチェック機能をシステムが提供しており、シーケンス・チャートの作成し忘れなどを削減することができる。

　イベント・ツリー上でのイベント・チェック機能にはイベント・シーケンスに対する確認と、イベント待ち状態の確認の２種類がある。

　イベント・シーケンスに対するものは、イベントの起動順序をチェックするもので、

（ａ）イベントＡが起きる前に、必ずイベントＢが無ければならない。
（ｂ）イベントＣが起きた場合には、以降のイベント・シーケンス上でイベントＤが無ければならない。

などのチェックを行う。

　例えば、図５の場合、状態Ｓ０から状態Ｓ３までの遷移系列において、イベント（+OffHook）があった場合、イベント（+OnHook）が無ければならない制約条件に違反しており、プロトコル設計者の単純ミスを削減することができる。

　イベント待ち状態の確認は、外部環境からの信号を待っている場合には、必ず、タイムアウトというイベントを記述しなければならないなど、ある状態で起こり得るイベントの確認手段として用いることができる。

（２）イベント・ビュー機能

　イベント・ビュー機能は、プロトコル設計者が注目したいイベントを選択した時に、そのイベントを含むシーケンス・チャートを選択して表示する機能である。

　イベント・ビュー機能により、プロトコル設計者は、システムが提供する全サービスの中から、必要なサービスだけを選び出し、それらのサービスのみに専念して作業を行うことができる。

４．３　シーケンス・チャートの作成・修正支援

　4.3.1で、イベント・ツリーによるモジュールの管理方法について説明し、4.3.2で、イベント・ツリーを用いた仕様書作成の差分プログラミング実現方法について説明する。また、4.3.3において、モジュールの修正により影響を受ける仕様書の修正支援機能の実現方法について説明する。

４．３．１　モジュール記述

　イベント・ツリーで管理されるモジュールのデータ構造を図６に示す。モジュールは、モジュール番号と、モジュール本体（シーケンス・チャート記述）、表示中のシーケンス・チャート管理、編集中のシーケンス・チャート管理のデータで構成される。図６のモジュールの例はそれぞれ、図７（ｂ）のモジュール００１とモジュール０５６を記述したものである。

モジュール番号：

　モジュール番号は、システムによって名前が一意になるように自動的に割り当てられる。

モジュール本体：

　モジュール本体は、モジュールが管理するシーケンス・チャートの記述を管理している。シーケンス・チャート記述は、（信号名、信号の送信元プロセス、信号の送信先プロセス）からなる。モジュールが設計されていない初期状態では、ダミー・プロセスに対する外部環境からの信号の受信、あるいは、ダミープロセスにおける条件判断のみが記述されている。このシーケンス・チャート記述はＳＡＬ（Service Addition Language）言語記述に自動変換することができる。

表示中のシーケンス・チャート管理：

　表示中のシーケンス・チャート管理では、編集中のシーケンス・チャートのモジュールが修正された場合に、表示中のシーケンス・チャートのモジュール記述を自動的に修正するために用いる。すなわち、モジュールが修

正された場合に、このモジュールを表示しているシーケンス・チャートに対して再表示を行うことで、モジュール記述に一貫性を持たせることができる。

編集中のシーケンス・チャート管理：
編集中のシーケンス・チャート管理は、複数のシーケンス・チャートで、同時に同じモジュールを修正することを禁止するための排他制御として用いる。

```
module001 {
  body {(OffHook,env,dummy),
        (OffHook,dummy,user),
        (Call,user,mng),
        (ShipCall,mng,ver),
        (Request,ver,control),
        (Request,control,dummy),
        (Request,dummy,env)}
  display_file {2,(chart034,chart035)}
  edit_file {chart036}}

module056 {
  body {(Authorized,env,dummy),
        (Authorized,dummy,control),
        (Authorized,control,ver),
        (Authorized,ver,mng),
  display_file {1,(chart035)}
  edit_file {}}
```

図６　モジュールのデータファイルの例

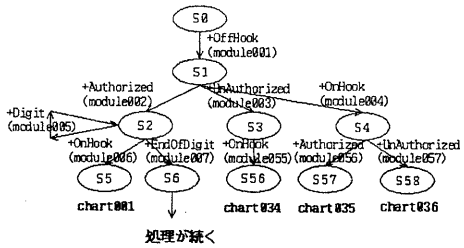
４．３．２　差分プログラミング支援

ユーザがプロトコルを設計する際のシステムの動作手順を以下に示す。

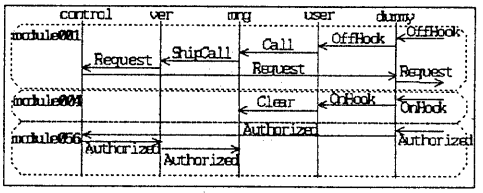
step1: ユーザが設計するシーケンス・チャートを、イベント・ツリーのリーフノードをクリックして選択する。
step2: システムは、選択されたリーフノードに該当する状態S_nから、親状態S_pに遷移する。親状態S_pがルートノードの場合には step4 へ、ルートノードでない場合には step3 へ行く。
step3: S_pからS_nへの状態遷移Ｌをスタックにプッシュする。そして、S_pをS_nに代入し、step2 へ行く。
step4: スタックが空であれば step7 へ行く。空で無ければ、状態遷移Ｌをポップして、これに該当するモジュールＭをシーケンス・チャート・エディタにロードする。
step5: モジュールＭをロードする場合に、モジュールＭの編集中のシーケンス・チャート管理にS_eが登録されていれば、そのシーケンス・チャートS_eを構成するモジュールM_aを全て表示中に変更する。
step6: モジュールＭの編集中のシーケンス・チャート管理に、現在のシーケンス・チャートS_nを登録する。そして、step4 に戻る。
step7: ユーザがシーケンス・チャート・エディタで、モジュールの中にプロトコルの設計を行う。
step8: モジュールの設計終了後は、4.3.3で述べる手順に従って処理を行う。

注）ユーザがシーケンス・チャートを表示するだけの場合には、step5 の処理を行わない。

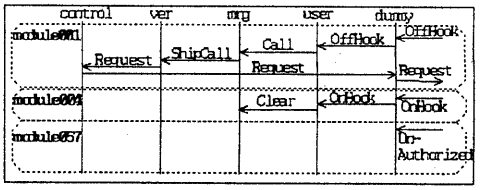
図７に、差分プログラムの例を示す。図７は、シーケンス・チャート０３５が既に設計されて表示されており、ユーザがシーケンス・チャート０３６を設計しようとしている場合である。この場合、ユーザはモジュール０５７を設計すればよい。



（ａ）シーケンス・チャート設計中のイベント・ツリー



（ｂ）シーケンス・チャート０３５



（ｃ）シーケンス・チャート０３６

図７　差分プログラムの例

４．３．３　モジュールの修正支援

ユーザが、プロトコルの設計を行い、モジュールに対して編集が行われた場合のシステムの動作手順を以下に示す。

step1: 編集されたモジュールＭに対するイベント・ツリー・ファイルのモジュールＭの編集フラグをＴにする。
step2: 編集されたモジュールＭの表示中のシーケンス・チャート管理に、シーケンス・チャートS_eが登録されている場合には step3 へ、登録されていない場合には step4 へ行く。
step3: シーケンス・チャートS_eを、上記（２）で述べた手順で表示し直す。
step4: シーケンス・チャートS_eを構成する全てのモジュールＭの編集フラグがＴの場合、イベント・ツリー・ファイルのシーケンス・チャートS_eの編集フラグをＹにする。
step5: モジュールＭの編集フラグ、シーケンス・チャートS_eの編集フラグの値によって、イベント・ツリーのノードとエッジの色を変更する。

以上の処理で、モジュールＭを共用して表示中であるシーケンス・チャートＳｏも自動的に修正することができる。そして、モジュールの変更後、イベント・ツリーのビュー機能を用いてモジュールＭの修正によって影響を受ける箇所を調べて、上記の手順で繰り返し修正を行うことにより、必要な箇所を正しく修正することができる。

例えば、図７においてシーケンス・チャート０３６を設計中に、モジュール００４を修正した場合には、表示中であるシーケンス・チャート０３５のモジュールを自動的に修正することができる。このとき、イベント・ビュー機能を用いて図８に示すように、修正したモジュール００４に依存する部分をイベント・ツリーで表示することができ、修正の影響箇所を視覚的に認識することができる。

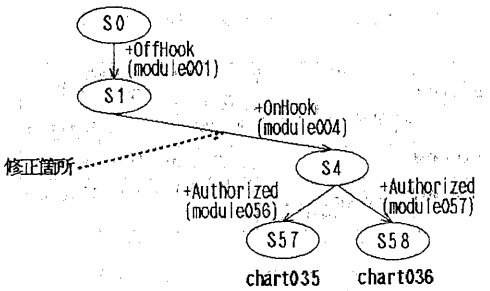


図８　モジュールの修正の影響箇所

４．４　統合化支援

（１）概略設計からのイベント・ツリーの自動生成

我々のシステムでは、大規模な通信システムを分割し、分割されたサブシステム毎にシーケンス・チャートによるプロトコル設計を行うことを前提としている。このとき、概略レベル（分割されたサブシステム間）でのプロトコルをシーケンス・チャートで作成した場合に、ＳｏｆｔＳＤＥで生成されるＳＤＬから、サブシステムに対するイベント・ツリーを自動的に生成することができる。

ただし、現在の段階では、一度詳細レベルまでプロトコル設計を行った場合に、概略レベルでのプロトコルの修正をイベント・ツリーにどのように反映するかという問題が残っている。

（２）イベント・ツリーからシーケンス・チャート作成のナビゲート

イベント・ツリーを用いて、シーケンス・チャートでプロトコルを設計する際に、既に作成されているモジュール記述を自動的にロードするだけでなく、まだ作成されていないモジュールも、モジュールの枠組みと、そのモジュールを起動する外部信号の受信か、条件判断が自動的に表示されるので、イベント・ツリーで設計した情報を基にして、プロトコル設計者をナビゲートすることができる。

５．おわりに

５．１　提案手法に対する考察

シーケンス・チャートによるプロトコル設計では、設計の容易性と、設計した仕様に対する理解の容易性という利点がある。本稿では、更に、シーケンス・チャートのこれらの利点に加えて、イベント・ツリーを用いることによる仕様の管理の容易性と、仕様の修正の容易性という特徴をもったプロトコル開発方法を提案した。

提案したイベント・ツリーを用いた差分プログラミング方法をある１事例に対してハンドシミュレートを行った。その事例では、全体のサービス数が３６枚で、プロトコル設計者の記述すべきシーケンス・チャートのモジュール数を２９８から５７（イベント・ツリーのエッジ数の総和）に削減できた。なお、ＳｏｆｔＳＤＥが提供するＧＳＡＬ[2]では、さらに、３３（イベント・ツリーの共通エッジをネットワークにした場合のエッジの総和）まで削減できるが、１枚１枚の仕様書が多階層にわたって記述されるため理解の容易性や修正の容易性が非常に損なわれてしまう。しかし、我々の提案した方法では、理解の容易性や修正の容易性を保ったまま、記述作業量を著しく削減することができる点で優れていると考える。

５．２　今後の課題

（１）プロトタイプシステムの作成・評価

これまでに、図３のシステム構成図において統合化ＧＵＩのプロトタイプは開発済みで実際に運用・評価を行っている。現在、我々の提案したイベント・ツリーによる仕様の作成・保守・修正を一貫して支援するプロトタイプシステムを作成中であり、早期に完成させ適用評価を行っていく必要がある。

（２）設計手法のナビゲート機能

大規模なシステムを分散開発する際に、システムをサブシステムに分割後、本手法であるイベント・ツリーを用いた設計を行うことができる。しかし、現在の段階では、分散開発中のサブシステム間の依存関係による、作成及び修正支援と、分散開発後の統合化処理の支援が実現されていない。これらの支援機能として、プロトコル設計者をナビゲートする機能が必要である。

［参考文献］

[1] H. Ichikawa, M. Itoh and M. Shibasaki, "Protocol-Oriented Service Specifications and Their Transformation into CCITT Specification and Description Language," The Trans. of the IECE of Japan, Vol.E 69, No.4, pp.524-535, Apr. 1986.

[2] 伊藤、加藤、市川：”通信サービス記述を指向した図形言語”，情報処理学会，SE-63，1988.11.

[3] 加賀谷聡、加地浩一：”プロセス抽象化によるプロトコル検証”，1990年　信学秋季全大，D-46，1990.10.

[4] 金地克之、加賀谷聡：”チャネルの抽象化によるプロトコルの設計・検証”，1991年　情報処理学会第４３回全国大会予稿集（１），pp.201-202.

[5] 金地、高橋、加賀谷：”プロセスモデルに基づいた統合化ＧＵＩの構築”，第４回ソフトウェアプロセスワークショップ投稿中