ソフトウェア工学 106－4
（1995．11．17）

構文知識を利用したソフトウェア設計支援ツール

臼井義美
日本電子計算株式会社

ビジネス・アプリケーションの設計においては、業務用語を用いて仕様を記述でき、しかも記述内容が厳密であることが望ましい。
そこで、我々は既に開発した業務用語を用いて仕様を記述できる設計支援ツールに、構文知識を組み込むことによってより厳密な記述を支援することとした。構文知識を組み込んだ設計支援ツールを利用した結果、利用者に対して記述内容を的確にガイドするとともに仕様の矛盾を早期に知らせることができた。
本稿では、ビジネス・アプリケーションの設計に構文知識を利用した仕様記述の支援方法と、それを適用した設計支援ツールについて報告する。

A CASE tool using specification description rules

Yoshimi Usui

Japan Information Processing Service Co.,Ltd.
E-mail: usui@ss.osaka.jip.co.jp

In designing business application, it is desirable to describe the specification without ambiguity by using its terminology. We developped a CASE tool with which users can describe specifications with thir terminology.
In this paper we give the CASE tool specification description rules to aid specification description without ambiguity. The CASE tool can help users describe specifications correctly as well as notify them of contradictions of the specifications in an early stage.

1 はじめに

ビジネス・アプリケーションの設計においては、できるだけ対象となる分野で利用されている業務用語を用いて仕様を記述でき、しかも記述内容が厳密であることが望ましい。そこで、我々はアプリケーションの設計に用いるデータや処理の仕様にできるだけ業務用語を用いるとともに、用語自体をオブジェクトとして扱い、そのメソッドを業務用語で記述した処理ユニットとして扱うことにより、記述しやすくかつ分かり易い設計支援ツールを開発した。また、仕様をより厳密に記述するために、仕様に用いる構文を中心とした記述のための知識を整理し、各設計フェーズごとにルールとしてツールに組み込むこととした。

本稿では、支援ツールの概要を述べた後、データに関する仕様記述に適用した構文知識とプロセスに関する仕様記述に適用した構文知識について説明し、最後にその適用結果をデータフローによる検証を例として述べる。

なお、ここでいう構文知識とは仕様記述に関わる各種のルールを指しており、そのルールを適用することで利用者が記述すべき仕様のガイダンスを行なったり、記述した仕様の矛盾の発見に役立つようなものを総称している。

2 目的

本支援ツールは ObjectFlow と呼び、その目的はビジネス・アプリケーションの開発にあたって、構文知識を利用することにより分かり易く厳密な仕様が効率良く記述できる環境を提供することにある。

このツールは、ビジネス活動におけるモノの名称を体系化した論理的クラスと、コンピュータの世界におけるデータの型による物理的クラスから、アプリケーションで用いるデータ項目オブジェクトを定義し、そこに関係づけられたメソッドを処理ユニットとして構造化チャート上に配置することによって設計するという方法を採用した。

このため、ビジネス活動におけるモノの体系化や仕様部品としての取り扱いはオブジェクト指向設計の考え方を取り入れ、集合データの定義やプロセスの設計には構造化設計の考え方を用いた。オブジェクト指向設計は対象世界に存在するモノ同士の関係を自然に表現するのに適しており、一方の構造化設計は振舞いの手順を的確に表現するのに優れているからである。

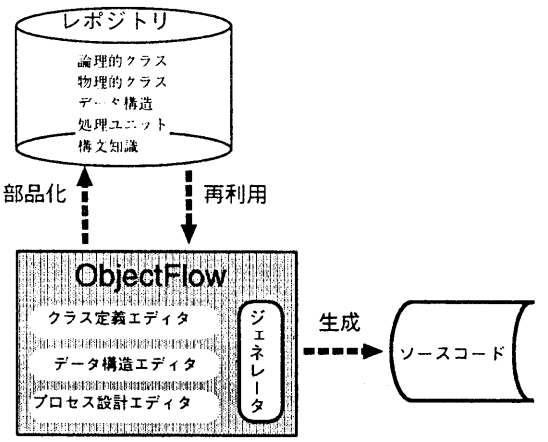

図 1: ObjectFlow の構成

3 ObjectFlow の概要

3.1 システムの構成

本支援ツールは、図 1 に示すように、仕様を記述するための 3 種類のエディタと、部品化された仕様や構文知識などを保管するリポジトリ、ソースコードを生成するジェネレータから構成されている。仕様記述のためのエディタは、データ名をオブジェクトとしてクラス階層に体系化するクラス定義エディタ、データの集合を構造的に定義するデータ構造エディタ、処理ユニットを記述するプロセス設計エディタから成っており、それぞれのエディタで記述した仕様を部品化し、再利用する機能を持っている。

3.2 モデル化の方法

本支援ツールにおいては、設計の対象となるビジネス世界や計算機環境から仕様の記述に用いる名詞と動詞を抽出する。こうして抽出した名詞のうち複合名詞はそれを構成する基本的な語彙に分割し、データ要素としてオブジェクトのクラス階層に体系化する。次にこれらのデータ要素を組み合わせて仕様で用いるデータ名や処理名を組み立て、これらの用語

で表される意味を仕様で表し、オブジェクトの形で仕様部品として蓄える。アプリケーションの設計に当たってはこれらの仕様部品を再利用しながら、データとプロセスに関してそれぞれ仕様を記述していく。

本支援ツールでは、図2に示すように従来ER図やオブジェクト図として表記されていたオブジェクト同士の関係を、次の3つの関係に注目して設計を行なうことにした。

3.2.1 意味的な関係

オブジェクトの意味的な関係を、汎化－特化の関係として表わす。そのために、共通の動作や属性を持つデータ要素をクラス階層として体系化する。これは、オブジェクト指向におけるクラスの定義を行う作業と同じで、結果をis-aのクラス階層図として実現する。

3.2.2 構造的な関係

クラス階層に整理されたオブジェクトを取り出して、全体－部分の関係を定義する。この関係は、オブジェクト同士の構造的な排他関係や包括関係をhas-aの関係として表わしたものであり、構造化設計におけるデータ構造図という形で記述する。

3.2.3 機能的な関係

データ要素間の機能的関係を表わす処理ユニットは、オブジェクトに記述されたメソッドとメッセージ交信をパッケージングしたものであるといえる。複数のオブジェクトが協調して1つの機能を表わす場合は、それらの間のメッセージ交換によって関係付けられるメソッド群を統合して1つの処理ユニットとして表わす。このように機能的な関係はオブジェクト間のメッセージ交換で実現することとなるため、そのタイミングを明確にする必要があるが、ここでは構造化チャート上に処理ユニットを配置することによって確定する。

4 データに関する記述支援

4.1 クラスの定義

本支援ツールでは、クラス定義エディタを用いてデータ要素を物理的クラスと論理的クラスの2つに

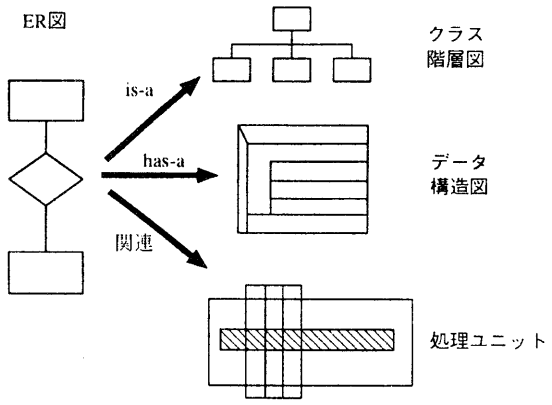

図 2: 関係の表記方法

分類し、それぞれクラス階層に体系化している。

論理的クラスは、対象となる業務に特有のデータ要素を体系化するためのもので、ビジネス活動を構成するモノを表す語彙のうち、できるだけアプリケーションで扱うデータ項目の名称の構成要素となる語彙をデータ要素として抽出する。そしてこれらのデータ要素ごとに、業務上で共通の認識が得られる属性やメソッドを記述していく。

一方、物理的クラスは、計算機環境に特有なデータ要素を体系化したもので、業務分野に影響されずターゲットマシンや言語などのシステム環境によってのみ規定される。このように、アプリケーションを設計する際の物理的な属性やメソッドを論理的クラスと完全に分離したことによって、クラス定義が複雑になるのを防ぐことができる。

以上に述べたクラス階層への体系化は、オブジェクトの型を明確にし、概念的な枠組を整理するのに役立っている。本ツールで提供するクラス定義エディタでは、データ要素を最適なクラスに配置するような支援を行なう予定である。

4.2 データ構造制約ルールの記述

データ構造エディタによってデータのhas-a関係をデータ構造図として記述する。仕様に用いるデータ構造はアプリケーションの稼働環境によって異なるが、ここではビジネス・アプリケーションのためのデータ構造の仕様記述に適した関係として、順次、反復、選択、包含、排他的な存在関係などを取り上げ

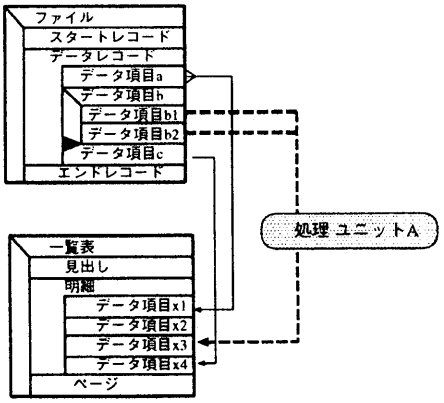

図 3: データ構造図と結合関係

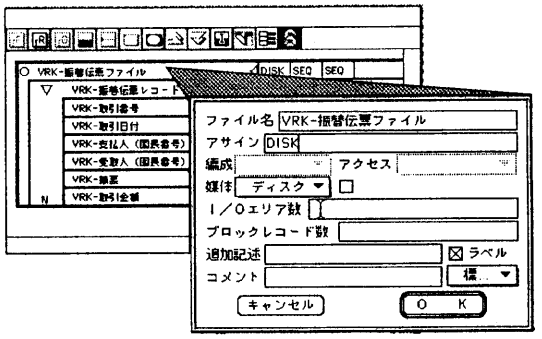

図 4: データの属性の記述

ている。利用者はこのエディタを用いてアプリケーションで実際に取り扱うファイルや帳票のような、複数のデータから成る集合オブジェクトとしてデータ構造図を定義する。

図3はデータ構造エディタ上で表示されるデータ構造図の表記例である。この図で示されるファイルは、1つのスタートレコード、複数のデータレコード、1つのエンドレコードから構成されており、データレコードは1つのデータ項目aと、データ項目bかデータ項目cのいずれか一方から成っていることを表している。

データ構造エディタは、データ構造の仕様記述に関する知識から得られるデータ構造制約ルールを用いて、データの包含関係に矛盾を起こさないような支援を行なっている。例えば、データ構造図に物理的クラスに含まれる「ファイル」というデータ要素を使った場合、ファイルはデータ構造の最上位レベルにしか記述できないこと、ファイルの構造的な下位レベルには複数のレコードが配置されなくてはならないこと、ファイルの下位レベルにはレコードというクラスのオブジェクト以外は記述できないこと、などのデータ構造制約ルールに反する操作はできないようにしている。

また、データ構造図に記述された各種のデータ項目は、マウスを用いて任意の位置に移動することができるが、データ構造制約ルールに適合しない位置には移動できないようにガードしている。

4.3 カーディナリティ制約ルール

データ構造図にオブジェクト間の対応関係を図式で表示しておくと相互の関係が分かり易いうえ、この対応付けは結合関係にあるオブジェクトの数を制限することが可能であるから、カーディナリティ制約ルールとして役立てることができる。データ構造エディタではデータ項目同士の結合関係をそれぞれのデータ項目間にMartin/Odellの提唱した表記法を用いて示すことにしている[4]。この結合関係には、1:1、1:nの結合関係の他、1:0または1の結合関係、1:0またはnの結合関係を記述できる。

図3の例では、データ項目aとデータ項目x1はn：1の結合関係があり、データ項目cとデータ項目x4は1：1の結合関係があることを示している。なお、処理ユニットAはデータ項目b1と同b2を入力として実行されるデータ項目x3のメソッドで、関数を介して複数のオブジェクトと関係している例である。この場合は、双方のオブジェクト間の関数をあらわす処理ユニットが存在する。

4.4 属性制約ルール

ソフトウェア設計で利用される抽象データ型には、スタック、キュー、テーブル、ファイル、集合、リスト、グラフ、ツリーなどがあるが、ここでは、ビジネス・アプリケーションでよく利用されるファイルを例に説明する。

ファイルのアクセス方法には、順次アクセス、索引アクセス、ランダムアクセスなどがあるが、ランダムアクセスを指定すれば対象となるデータ構造に

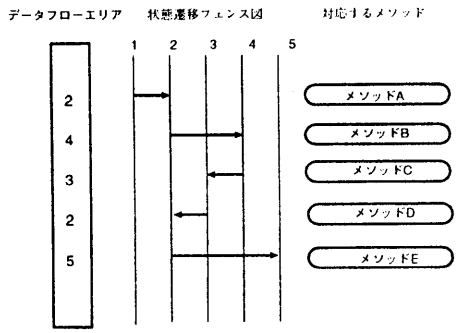

図 5: データフローエリアと状態遷移

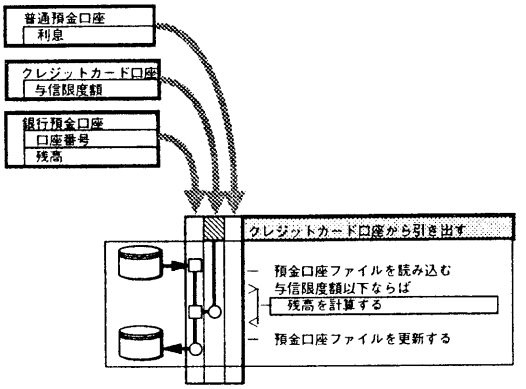

図 6: データフローエリア

プライマリーキーが指定されていなければならない。このようなルールをデータ構造エディタに持たせることによって、データの属性に関わる記述内容の矛盾や記述洩れを防止している。

図 4 はデータ構造エディタで表示されたデータ構造図と、データ項目の物理的属性を登録するダイアログの例である。データ項目に属性を登録するには、データ構造エディタに表示された当該データ項目をマウスでクリックすると、属性登録ダイアログが表示される。ダイアログでは属性制約ルールに基づいて必要な事項のみ入力可能となっている。また内容がいくつかの記述に限定されているものはメニュー形式にすることによって、記述ミスの防止を図っている。

さたに、上位クラスに定義した属性が継承された場合は、ダイアログが表示された時点で、エディタによって継承された内容が既に設定されている。

4.5 状態遷移ルール

オブジェクトは、そのオブジェクトが存在するか否かをはじめとして、時間とともにその状態が変化する。したがって、あるオブジェクトの状態を完全に定義するためには、オブジェクトの生成から消滅までのすべての状態を検討する必要がある。

オブジェクトの状態遷移は、生成、消滅、操作、更新、分類、結合などのメソッドによって引き起こされるので、各オブジェクトについて取り得る状態と遷移可能な条件は状態遷移ルールとして扱うことができる。従って、ある状態で実行可能なメソッドは、その時のオブジェクトの状態で許されている操作の種類により決定されるので、状態遷移ルールをエディタに組み込むことにより、特定のタイミングで実行可能なメソッドを限定することができる。

図 5 は、プロセス設計エディタのデータフローエリアに表示された状態と、それに対応する状態遷移フェンス図とメソッドの関係を表している。図ではあるオブジェクトについて 5 種類の状態が定義されており、それぞれのメソッドによって、ある状態から他の状態に変化することを表している。

本支援ツールでは、あるオブジェクトについての状態とその状態で実行可能なメソッドを状態遷移ルールとして整理し、データ構造エディタとプロセス設計エディタを連係させることで実現している。

5 プロセスに関する記述支援

5.1 データフローエリア

データフローは、データのソースとそのデータを使うプロセスの間のつながりを明確にする一方、プロセス間の先行関係を示すこともできる。

プロセス設計エディタは、対象となるアプリケーションで用いられるすべてのオブジェクトに関して、対応するデータフローエリアを用意し、その状態遷移を表示する。データフローエリアは、利用者が処理ユニットを記述する毎に自動的に関係するデータフローを描画する機能を有している。図 6 は、3 つのデータ構造図で示された集合オブジェクトが、プ

ロセス設計エディタのデータフローエリアにそれぞれ対応して表示されることを示している。

5.2 操作制約ルール

オブジェクト指向による設計では、各オブジェクトをカプセル化して扱い、クラスにはそれらのオブジェクトの型ごとに必要な操作をメソッドとして記述する。従って、入力データの型や構造を定義することにより、各オブジェクトに属するメソッドは入力データの型によって組織化でき、実行可能なメソッドを一定のルールとして与えることができる。

例えば、物理的クラスに含まれるファイルというオブジェクトのメソッドには、ファイルのオープン、クローズ、レコードの読み込み、書き出し、更新、検索、ソートなどがある。プロセス設計エディタのデータフローエリアはそれぞれデータ構造オブジェクトの時間軸に対応したビューを提供しているので、ファイルに対応するエリアをマウスでクリックすると、そのデータ構造図が表示される。利用者はデータ構造図の中のファイルを示すデータ項目をクリックすることにより、ファイルに関するメソッドがメニュー形式で表示される。このとき、プロセス設計エディタで利用者が指定したタイミングにおけるデータの状態によって、状態遷移ルールに基づき実行不可能なメソッドは選択できないようにしている。

5.3 基本ユニットの構文ルール

処理ユニットにはオブジェクトに定義されたメソッドに対応する基本ユニットと、複数のオブジェクトに属するメソッド群を組み合わせて1つの機能を表す抽象ユニットの2種類を用意している。

基本ユニットは、オブジェクトに定義されたメソッドに対応するもので、物理的クラスのクラスメソッドとして登録したものである。本支援ツールは、実行環境に応じたプログラムソースを生成するための構文をリポジトリに蓄えており、プロセス設計エディタで関係するデータの型に応じた適切な構文を表示する。

図7は、プロセス設計エディタを用いて基本ユニットの記述を行なっている例である。利用者は、基本ユニットを組み入れたいタイミングをマウスで指定すると、基本ユニットの挿入される位置が確定する(図7の斜線部分)。次に、対象となるオブジェクト

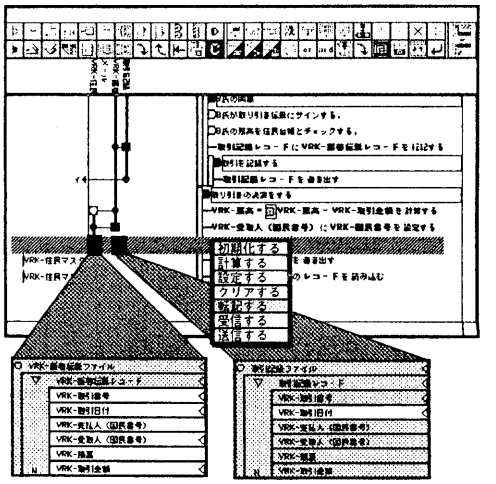

図7: 基本ユニットの記述

が含まれるデータフローエリアをマウスで指定すると、そのオブジェクトのデータ構造図が表示される(図7の左のデータ構造図)。このとき、データ構造図に含まれるオブジェクトのうち、指定したタイミングにおける状態遷移ルールによって、操作不可能なオブジェクトは網掛けで表示され、選択できないようになっている。利用者は選択可能なオブジェクトを指定すると、そのオブジェクトに属するメソッドの中から操作制約ルールに基づいて実行可能なメソッドがメニュー形式で表示されるので、希望のものを選択する。

もし「転記する」というメソッドを選択すると、そのメソッドに適した構文テンプレートが表示される。テンプレートで要求されている入力データに対して、その対象となるデータフローエリアをクリックするとデータ構造図が表示される(図7の右のデータ構造図)。このとき、属性制約ルールに基づいて入力データとして利用できないオブジェクトは選択できないようになっている。

このように、各種の制約を満たすようなオブジェクトやメソッドのみを選択対象とすることによって、利用者は容易に基本ユニットを記述することができる。

なお、図7のエディタの表示例には、既に記述された処理ユニット群が、実行手順に沿って配置された様子が示されている。

5.4 抽象ユニットの詳細化ルール

複数の基本ユニットが集団で特定の機能を表すようなものを、抽象ユニットとして部品化することができる。抽象ユニットは論理的クラスに登録されたオブジェクトのメソッド名を用いて記述した処理ユニットであり、ある機能を業務用語で表現した機能モジュールである。

図8は、抽象ユニットの例を示しており、「預金を引き出す」という抽象ユニットの内部に「残高を計算する」という抽象ユニットが含まれている。図に示すように、プロセス設計エディタ上で「残高を計算する」という抽象ユニットを展開表示すると、複数の基本ユニットから構成されていることがわかる。利用者は、複数の基本ユニットまたは抽象ユニットを組み合わせて、新しい抽象ユニットを作ることができ、これにその機能を表す動詞を付加して、抽象ユニットとして登録することができる。こうして登録された抽象ユニットは、利用者が随時オブジェクトを介して再利用することができる。

設計者にとっては、従来のオブジェクト指向による設計法ではこのような抽象的な機能を部品として取り扱うことが難しかったが、本設計支援ツールを用いると基本ユニットを組み合わせることによって容易に抽象ユニットを部品化することが可能となった。

また、基本設計フェーズにおいて、処理内容がまだ確定していない抽象ユニットを記述しておいて、詳細設計フェーズでその内容を基本ユニットを用いて詳細化するようなトップダウン設計にも適用可能である。このとき、詳細化にあたっては、抽象ユニットで定義された入出力データの型や構造を満たすように記述しなければならない。プロセス設計エディタでは、このような知識を詳細化ルールとして利用している。

5.5 制御条件制約ルール

メソッドの実行順序を確定する段階で、条件分岐やループの終了条件などの制御に関して多くのルールが存在する。例えば、条件分岐ではそれぞれの分岐の総和が必ず排他的論理和になっているべきであるし、ループの終了条件を満たすためには、ループ内の処理ユニットに終了条件を満たすための記述がなければならない、などである。プロセス設計エディタでは、条件文を記述すると必ずELSEにあたる条件を自動的に付加するなどの支援を行なっている。

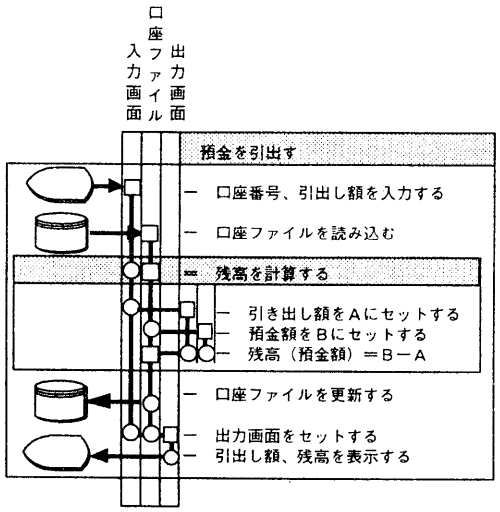

図 8: 抽象ユニットの記述

6 データフローによる検証

6.1 垂直フローと水平フローの意味

データフローには、図9に示すように垂直フローと水平フローが存在する。データフローエリアに縦のフローで描かれる垂直フローは、各々オブジェクトの状態遷移を示しており、データフローが描かれていない部分はオブジェクトの未生成区間もしくは

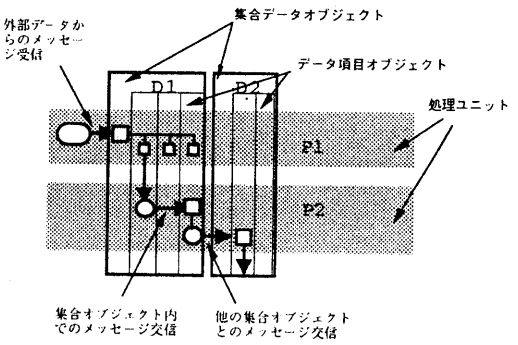

図 9: データフローの意味

その参照が不可能である区間を示し、逆にデータフローの描かれている部分は、状態制約ルールに合致したメソッドの実行が可能な区間を示している。

一方、横方向のデータフローすなわち水平フローは、オブジェクト間のメッセージ交信のタイミングを時間の経過に沿って表示したものであり、端末の操作員や外部ファイルからのデータ入出力を表すデータフローも、水平フローで表される。

6.2 矛盾の表示

本支援ツールでは、設計者が仕様を記述している間、データフローを解析することにより次のような整合性の検証を行っている。まず、垂直フローの検証は、新しく処理ユニットを作成したり、リポジトリに登録されている処理ユニットを再利用したときに、データを授受しているデータ項目のデータ構造制約ルールに基づいて、その矛盾を検証できる。また、それぞれのデータフローエリア上におけるデータの状態と、各処理ユニットにおけるデータ項目のアクセスの種類をもとに、状態遷移ルールによって処理の妥当性をチェックし、矛盾があればデータフロー上にエラー表示をする。

一方、水平フローは、データ項目オブジェクト間のメッセージ交信に用いる引数の検証に利用している。すなわち、データ項目オブジェクトをデータ構造図から選択すると、交信相手とのオブジェクトの引数の属性制約ルールに基づいて、矛盾を検証している。設計者は、処理ユニットを用いたプロセスの設計中に、絶えずデータフローの検証が行われており、その矛盾点が赤色で表示されるので、容易にかつ厳密に仕様を記述することができる。

7 おわりに

構文知識をもとにした仕様記述ルールを利用したソフトウェア設計支援ツールについて説明した。本支援ツールは、ここで述べたすべてのルールによる支援を実現している訳ではないが、利用者にとっては記述のガイダンスと矛盾の検証によって非常に効果的な支援ができることが分かった。

今後は、未装備のルールを実装するとともに構文知識をさらに整備し、開発現場おける仕様記述の支援に役立てたい。

参考文献

[1] J. Martin, J. J. Odell:Object-Oriented Methods:A Foundation,PTR Prentice Hall(1995)

[2] Ian Sommerville:"Software Engineering",Addison-Wesley(1992)

[3] Martin, James:Recommended Diagramming Standards for Analysts and Programmers,Prentice Hall(1987)

[4] 臼井義美：データフローによる部品探索機能を持つ仕様エディタの試作，情報処理学会ソフトウェア工学研究会報告，Vol. 94,No.55. pp.89-96(1994).

[5] Ian Hayes :"Specification Case Studies",Prentice Hall(1993)

[6] Kurt J. Schmucker:"Object-Oriented Programming for the Macintosh"Hyden Book.Co.(1986).

[7] Grady Booch: Object Oriented Disign,The Benjamin/Cummings Pub.Co.(1991).

[8] 臼井義美：Show-CASEにおける仕様部品の再利用支援環境，情報処理学会ソフトウェア再利用技術シンポジウム論文集，(1992).

[9] J. ランボー, M. ブラハ, W. プレメラニ, F. エディ, W. ローレンセン: 羽生田栄一 監訳: オブジェクト指向方法論 OMT モデル化と設計，トッパン (1992).

[10] 片岡雅憲：ソフトウェア再利用技術、日科技連 (1992)