DEWS2004 6-C-04

システム仕様統合CASEツールの開発

DFD表現からUML表現への変換

白岩政太郎† 三浦 孝夫† 塩谷 勇††

† 法政大学 工学研究科 電気工学専攻 〒184–8584 東京都小金井市梶野町 3-7-2
†† 産能大学 経営情報学部 〒259–1197 神奈川県伊勢原市上粕屋 1573
E-mail: †{i02r3226,miurat}@k.hosei.ac.jp, ††shioya@mi.sanno.ac.jp

あらまし 本研究では、既存システムから新規システムへの拡張、統合の支援を目的としている. そこで、構造化分析とオブジェクト指向分析間の統合方法を提案する. これまで、筆者らはデータフロー図（DFD）で作成したシステム仕様を Unified Modeling Language（UML）のシステム仕様へ変換する規則を提案している. 本論文では、この変換規則を実現する CASE ツールの開発について述べる. CASE ツールは DFD と UML それぞれの描画ツール間で DFD 表現を UML 表現へ変換する. 本稿では、ケーススタディを用いて描画ツール間でどのように DFD 表現が変換されるかを示す.

キーワード ソフトウェア設計、CASE ツール、UML、DFD

A CASE Tool for Integration of System Specifications

Translation from DFD expression to UML expression

Masataro SHIROIWA†, Takao MIURA†, and Isamu SHIOYA††

† Dept.of Elect.& Elect. Engr., HOSEI University 3-7-2, KajinoCho, Koganei, Tokyo, 184–8584 Japan
†† Department of Management and Information Science, SANNO University 1573, Kamikasuya, Isehara city, Kanagawa 259–1197 Japan
E-mail: †{i02r3226,miurat}@k.hosei.ac.jp, ††shioya@mi.sanno.ac.jp

Abstract In this paper, we describe a CASE tool to extend and integrate from legacy system to modern system to support software design. We have already proposed conversion rules by which we convert DFD diagrams into UML. In this paper, we describe a CASE tool for this purpose, and we describe how UML is generated from DFD by this CASE tool.

Key words Software design,CASE TOOL,UML,DFD

1. 前 書 き

ソフトウェアシステム開発において、長い保守サイクルの中で、新しい要求の追加、変更が発生する. その際システムの全体を入れ替えるのではなく、新しい要求に対応するための差分を開発し、統合する必要がある. さらに、新しい概念、環境、ニーズの登場でソフトウェア開発で用いるモデルも変化し新しいものが開発されている. しかし、このモデル形式の相違は仕様書に関する間違った認識を生む原因となる. そこで本論文では、既存システムから新規システムへの拡張・統合を目的とする.

本研究では既存システムの分析が構造化手法でなされるとし、また、新規システムの分析手法としてオブジェクト指向手法が用いられると仮定する. 本稿では、構造化手法の分析ツールとしてデータフロー図（Data Flow Diagram,DFD）[4]、オブジェクト指向手法の分析のツールとして Unified Modeling Language（UML）[1] を取り上げ、DFD 表現から UML 表現への拡張支援の方法を述べる.

これまでに筆者らは DFD 表現で作成したシステム仕様を UML 表現で作成したシステム仕様へ変換する規則を提案している. その際、変換を行う DFD のセマンティクスを厳密に定義し解析するために、UML のクラス図と Object Constraint Language（OCL）を用いて DFD メタモデルを定義している. 以下ではこの DFD メタモデルとそこから導き出した変換規則について要約し、この変換を実現する CASE ツールの開発について述べる. ユーザはこの CASE ツールを利用して変換規則に従った、DFD モデルから UML モデルへの変換を半自動的に行うことが可能である.

2 章で変換対象である DFD セマンティクス、3 章で DFD メ

タモデルについて定義し、4 章で DFD を UML で表現するための変換規則を述べ、5 章で CASE ツールの概要を述べる. 6 章でケーススタディを用いて CASE ツールによる変換を示し、7 章が結びとなる.

2. Data Flow Diagrams

ＤＦＤは 70 年代に差管理利用されたシステム分析手法であり、システムをデータの流の観点から表現する. DFD はシステムをデータの流れの観点から表現する分析ツールである. すべての要素はラベルを持ち、各要素のラベルはある名前空間で唯一である. DFD で用いる要素にはソース、シンク、データストア、プロセス、データフローが存在する. DFD の要素で特にデータストアとデータフローに関しては、各要素が扱うデータの構成をデータ辞書を用いて記述する. DFD は分析の詳細さのレベルによって階層を構築することでシステムの全体像を表現する. その際、プロセスの下階層に新たな DFD を定義し、トップダウンに分析を行う. プロセスとプロセス、データストア、データフローの間には抽象関係が存在する. ソース、シンクはシステム外部を表すノードである. データストアは一時的なデータの貯蔵庫を表すノードである. プロセスはシステムの機能を表すノードである. プロセスへの入力データフローから出力データフローへのデータ変換を表す. また、プロセスは複数の入力、複数の出力への分岐を示すことも可能である. データフローは各ノード間のデータの流れを表す有向辺である. プロセスとプロセス、データストア、ソース、シンクの間に定義可能で、途中から分岐することもできる. ただし、流れるデータは同一である. また、入出力のバランスは一致する. 例えば、図 1 では左側ダイアグラムは右側ダイアグラムの上階層を表す. 上階層にはデータフロー「社員情報」が入力されるのに対して、下階層では「社員氏名」と「社員役職」が入力されている. 入出力のバランスが一致するということは、「社員情報」は「社員氏名」と「社員役職」に分解できることを示す. また、DFD は起動因

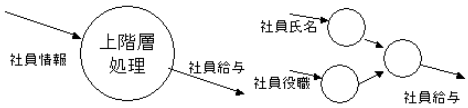

図 1　入出力の一致

子を示さない.

3. DFD メタモデル

筆者らは DFD 表現を UML 表現へ変換する規則を発見するために、DFD のメタモデル [2] [6] を提案している. 両者の方法論を形式的で厳密に比較するためにはそれぞれの仕様が厳密に定義されている必要がある. UML はその仕様書でメタモデルによりその仕様を厳密に定義している. そこで、DFD の仕様を同じようにメタモデルを作成し厳密に定義した. DFD メタモデルは UML のクラス図と OCL により表現する. この結果、どのような DFD 表現も UML 表現を用いて解釈することができるようになる. メタモデルによるアプローチとしては UML を UML により定義するアプローチ [1] [7] がある.

3.1　DFD メタモデル構成

DFD メタモデルは抽象構文、適格性規格により記述される.

（1） 抽象構文

UML のクラス図（図 2）によるモデルとそれを補足する自然言語記述からなる. 自然言語記述はクラス図の各クラスとクラスの属性、クラスが関係する関連の記述を含む. クラス図は具象メタクラスと抽象メタクラスに分類することができる. 具象メタクラスはインスタンスを作成できるメタクラス、抽象メタクラスはインスタンスを作成できないメタクラスである.

（2） 適格性規格

抽象構文のクラス図の不変条件を記述. OCL による記述と自然言語記述による補足からなる. また、付加的操作は OCL 式記述の為に必要な操作を定義している.

3.2　DFD メタモデル

DFD メタモデルを示すクラス図は以下のようになる.

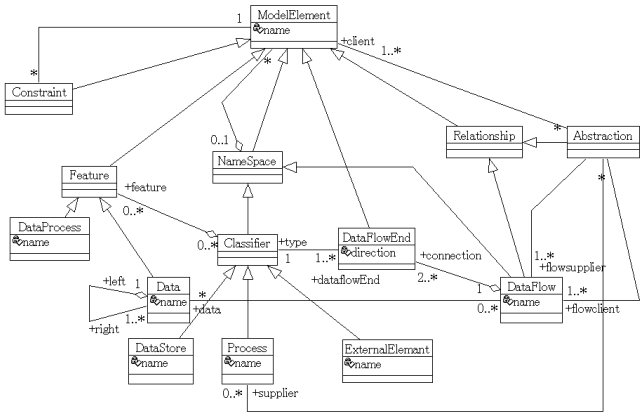

図 2　DFD メタモデル

以下では、図 2 の各要素の定義する. ここでは DFD の要素を表現する具象メタクラスに焦点を絞り定義する. また、適格性規格について説明を省略してある. 先ず、ソース/シンク、データストア、プロセスのセマンティクスはそれぞれ ExternalElement クラス、DataStore クラス、Process クラスで定義している. それぞれの抽象構文は以下のとおりである.

- ExternalElement

システムと相互作用するシステム外部の集合を定義する具象メタクラス.

- DataStore

データの貯蔵庫を示し、Data のインスタンスの集合を定義する具象メタクラス.

- Process

システムの機能を表し、DataProcess のインスタンスの集合を定義する具象メタクラス.

それぞれのクラスはある同一名前空間で一意な値を持つ属性 name を持っている. また、それぞれのインスタンスは DFD のソース/シンク、データストア、プロセスの要素と対応する. DataStore クラスは DFD と共に用いるデータ辞書の項目を定義する Data クラスを集約している. この関係は図 3 のようにインスタンス化される. 同様に Process クラスは DFD には現れない

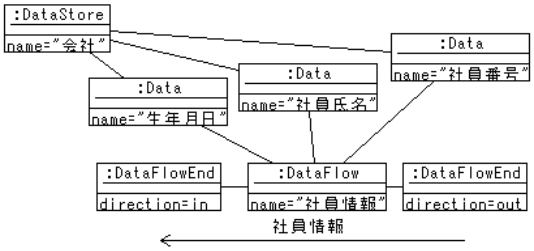

図 3　例.DataStore,DataFlow のインスタンス

最小限度まで分解されたプロセスを定義している DataProcess クラスを集約している. ExternalElement クラス、DataStore クラス、Process クラスは親クラスとして Classifier クラスを持ちこの性質を継承する.

DataFlow クラスは Data クラスのインスタンスの集合を定義する具象メタクラスである. この関係は図 3 のようにインスタンス化される.

また、メタモデルでは DataFlow クラスのインスタンスは DataFlowEnd のインスタンスを 2 つ以上集約している. DataFlowEnd クラスは DataFlow クラスのインスタンスを Classifier クラスのインスタンスに結合する端点を定義する具象メタクラスである. Classifier クラスのインスタンスに対しての入出力方向を示す属性（direction）を持つ. インスタンスは図 3 のようになる. direction=in の時、各 Classifier クラスのインスタンスへの入力、out の時、出力を表す. DataFlow クラスのインスタンスと合わせてデータフローの方向を表現する.

4.　UML による DFD 表現の記述

DFD の表現を UML の表現へ変換する規則を示す. DFD メタモデルのインスタンスはその抽象度によって対応する UML のダイアグラムが異なる. 変換に際しても抽象度に応じた変換が必要となる. 本稿では DFD の表現を UML のユースケース図、クラス図、シーケンス図へ変換する. 詳細は [8], [9] を参照.

4.1　ユースケース図の作成

ユースケース図への変換ルールは以下のとおりである.

U1:　ソース/シンクをアクターに変換する.

U2:　プロセスをユースケースに変換する.

U3:　データフローに従った関連を引く.

抽象度が高い DFD において、ExternalElement クラスのインスタンスは UML のアクターと、Process クラスのインスタンスは UML のユースケースと意味的な対応が取れる. ExternalElement クラスのインスタンスとアクターは共にシステムの外部を表し、Process クラスのインスタンスとユースケースは共にシステムの機能を表現しているからである.

4.2　クラス図の作成

クラス図への変換ルールは以下のとおりである.

C1:　データストアをクラスに変換する.

C2:　データストアへ直接入出力しているプロセスをメソッドに変換する.

C3:　複数のデータストアと直接的/間接的に関係しているプロセスはデータ辞書を検討して割り当てるクラスを決定する.

C4:　クラスへ変換したデータストア間に存在するデータフローをクラス間の関連に変換する.

UML のクラスは共通の属性とメソッドを持つオブジェクトの集合である. これより、DataStore クラスのインスタンスと Process クラスのインスタンスの集合は UML のクラスと対応が取れため C1 の変換ができる. DataStore クラスのインスタンスはクラスが持つ共通の属性と、Process クラスのインスタンスはクラスが持つ共通のメソッドと対応している. このため、C2、C3 の操作を行い、クラスの属性を操作するプロセスを発見しメソッドに変換する. データフローはデータの流れであり認識されたクラス間にデータフローがある場合にはそのクラス間に「利用する」などの関係があるためルール C4 により関連を与える.

4.3　シーケンス図の作成

シーケンス図への変換ルールは以下のとおりである.

S1:　ユースケース図とクラス図の作成によりアクター/クラスを認識する.

S2:　シーケンス図のアクター/クラス間にデータフローの矢線の向きと同じ向きでメッセージを引く.

Process クラスのインスタンスの方向はシーケンス図のメッセージで表現可能である. データフローに従い、順にメッセージを引くことで実行順も表現できる. ただし、このシーケンス図は DFD のデータフローに従った起こりうる実行順序を列挙したものにすぎず、必ずしも全体の実行順序を示すものではない.

5.　システム統合 CASE ツール

4 章で示した変換規則を半自動的に実行するための CASE ツールについて述べる. CASE ツール（Computer-Aided Software Engineering tool）とは構造化手法、オブジェクト指向手法などの方法論をシステム開発に効果的に適用するために不可欠である. CASE ツールは開発の一部を自動化したり、システム開発時に扱う情報を一元的にコンピュータで管理するために利用される. このツールは変換規則を半自動的に行うことにより、変換の際の人為的な見落としを防ぎ、高速化を図る.

このツールは DFD と UML の描画ツール間でその変換を行うものである. 今回は DFD の描画ツールとして Microsoft VISIO、UML の描画ツールとして ArgoUML [5] を利用する. ArgoUML は UML のダイアグラム情報を XML ファイルを利用してインポート/エクスポートすることが可能である. この XML ファイルは XML Metadata Interchange （XMI）[3] と呼ばれ、Object Management Group（OMG）により Meta Object Facility（MOF）と共に仕様が提供されている. XMI は MOF モデルを利用して定義したメタモデル、また、そのメタモデルに従って記述されるモデルを XML 形式で表現するための仕様である. UML メタモデルは MOF モデルに従っており、UML メタモデルに従ったモデルも XMI で表現が可能である. XMI による UML 情報のデータ交換は標準的な方法となっており、多くの CASE ツールが XMI をサポートしている. ArgoUML もこの XMI ファイルを利用してモデルを表現している.

5.1　システム構成

このCASEツールはDFDを記述可能なMicrosoft VISIO 2002とUML描画ツールのArgoUML version0.12の間で変換規則に従った変換を支援する（図4）. ユーザはVISIOでシステムをDFDにより記述し、XMLファイルとしてその情報をエクスポートする. DFDは各階層ごとに記述する必要がある. ユーザは以下のファイルをCASEツールへの入力ファイルとして用意する.

（1） リンク情報ファイル

VISIOのエクスポート機能を利用して出力したXMLファイルである. DFDの各要素間のデータフローに対応する情報を含んでいる.

（2） ノード情報ファイル

VISIOのレポート機能を利用して出力したXMLファイルである. リンク情報ファイルの要素がどの種類であるかの情報を含んでいる. ある階層の情報を表すノード情報ファイルのファイル名と上階層のプロセス名は一致させる. これにより階層構造を表現する.

（3） データ辞書ファイル

DFDのデータストアとデータフローについてのCSV形式の辞書ファイルでる.

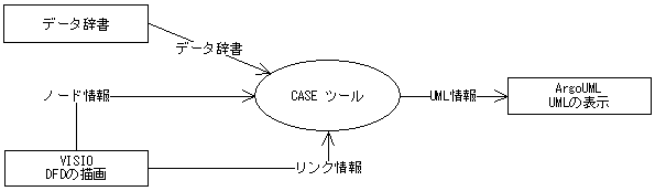

図4　CASEツールと描画ツールの構成

CASEツールはこれらのファイルを受け取りArgoUMLでインポートが可能なXMIファイルを出力する. システムはJAVAを用いて開発されている.

5.2　システム機能

CASEツールによる変換の流れを図5に示す. システムは以下のような機能を持つ.

（1） DFD情報の作成

CASEツールはリンク情報ファイルとノード情報ファイルよりDFD情報を作成する.

（2） 抽象度に応じた作成ダイアグラムの問い合わせ

UMLへの変換は変換対象のDFD表現の抽象度に応じてユースケース図もしくはクラス図を作成する. しかし、抽象度の判断はデータストアのダイアグラムへの出現のように判断基準がないわけではないが、ダイアグラムの意味を理解する必要があるためユーザに問い合わせる. ユーザはDFD表現の抽象度より適切なダイアグラムを選択する.

（3） ユースケース図の作成

抽象度が高いDFD表現の場合、変換規則に従ってユースケース図を作成し、XMIファイルを出力する.

（4） クラス図の作成

クラス図を作成する場合にはデータ辞書ファイルの情報とあ

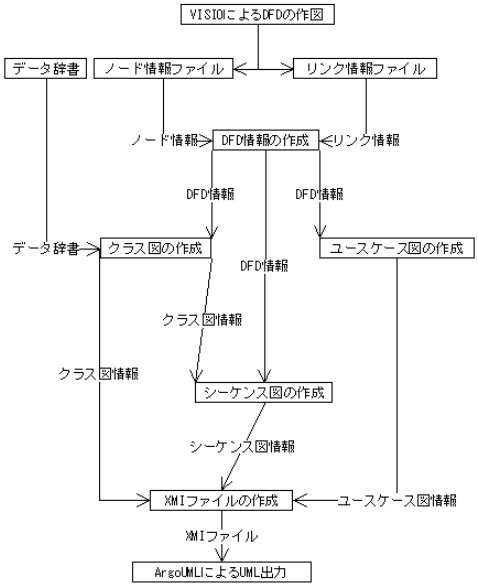

図5　変換の流れ

わせて変換規則に従って処理をし、XMIファイルを出力する.

（5） メソッド割り当てに関する問い合わせ

クラス図の作成ではプロセスを複数のクラスに割り当てできる場合がある. この場合、CASEツールはユーザにプロセスをどのクラスにメソッドとして割り当てるか問い合わせる. その際、CASEツールは認識したクラスが扱うデータ辞書の項目とプロセスが扱うデータ辞書の項目をユーザに提示する. ユーザはDFD表現とCASEツールが提示するデータ辞書の項目を見てどのクラスに割り当てるかを判断する.

（6） シーケンス図の作成

クラス図に対応したシーケンス図を作成し、XMIファイルを出力する. DFD表現にソース/シンクが含まれる場合にはクラスとアクター間にメッセージを作成する.

次章ではケーススタディを用いて変換の実験を行い、CASEツールの変換の流れを示す.

6.　ケーススタディ

この章ではケースステディを用いてDFD表現がCASEツールにより、どう変換されるかを示す.

6.1　遠隔医療介護システム（TRMCS）

このシステムは入院の必要がなく自宅で病状の経過を見守っている患者を支援するシステムである. 緊急時、またはモニタ監視による異常をシステムが感知し、主治医や医療機関、救急隊の要請を自動的に行い、また医療機関同士の連絡を円滑に行うことがシステムの主な役割である. このシステムは患者の病状を診断することはなく、診断はすべて医師に委ねられる. ユーザ（患者）はヘルプコールにより、認証を受け主治医への電話接続サポートの要求、救急隊の要請が可能である. また、モニタリングにより、心電図、脳電図等の監視を受けることができモニタデータに異常があればシステムが主治医への連絡、救急隊の要請を自動で行う. 主治医は、システムより患者のヘルプコールまたはモニタデータを受け取り、サポートまたは医師の

判断によりシステムを介して救急隊要請をすることができる. サポートの際、医師は接続要求を利用して患者への電話接続を自動で行うことができる. 救急要請は救急隊と医療機関の双方に患者情報を送信し、一般公開入札を発動させる. また、主治医はシステムと提携している医療機関に対し、急患の要請や医療機関の対応科を検索することやモニタデータにアクセスすることが可能である. 提携医療機関は他の医療機関の検索が可能である. また、提携医療機関は患者を受け入れる際、一般公開入札に参加できる. 一般公開入札に参加可能な医療機関についてはシステムが患者の症状と比較し自動で医療機関を選定する. システムはヘルプコールとモニタリング異常に際して、日時、時間、患者 ID をログとして保存する.

6.2 TRMCS の DFD

図 6 から図 12 は VISIO で描画した TRMCS を表す DFD である. コンテキストダイアグラムにより、システムとその外部の境界を示し、コンテキストダイアグラムの単一プロセスの下階層をダイアグラム 0 が表現している.

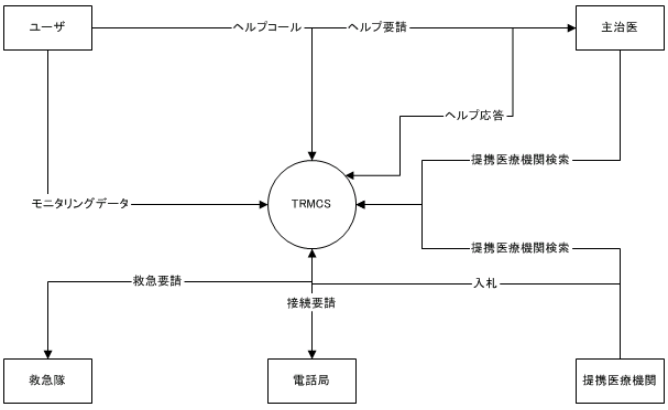

図 6　コンテキストダイアグラム

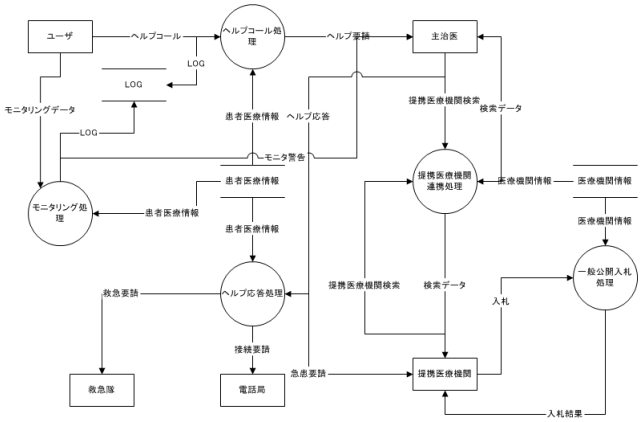

図 7　ダイアグラム 0

6.3 UML への変換

このケーススタディでは抽象度の高いコンテキストダイアグラム図 6 のみをユースケース図に変換する. 変換規則では次のような変換を行う. ルール U1 に従いソース、シンクを表す「ユーザ」、「主治医」、「救急隊」、「電話局」、「提携医療機関」をアクター に変換し、ルール U2 に従ってプロセス「TRMCS」をユースケースに変換する. ルール U3 により、データフローに従った関連を引く. 以上の操作により図 13 が得られる. 図 14 は図 6 を CASE ツールを利用して変換し、出力した XMI を ArgoUML へインポートした画面である. 変換規則どおりにユースケース図が変換できている. コンテキストダイアグラムのような抽象度の高いダイアグラムの場合はシステムと外部との境界を明確にすることが目的である. このような場合にはユースケース図に変換し、同一の意味のダイアグラムを作成する. ここでは、シ

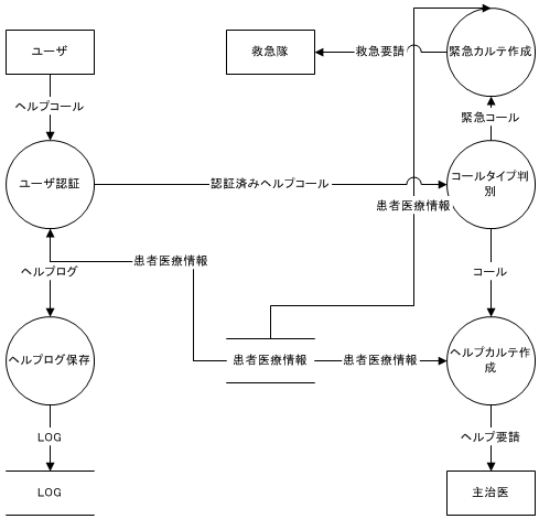

図 8　1. ヘルプコール

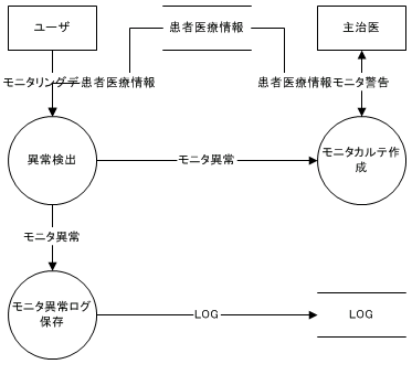

図 9　2. モニタリング

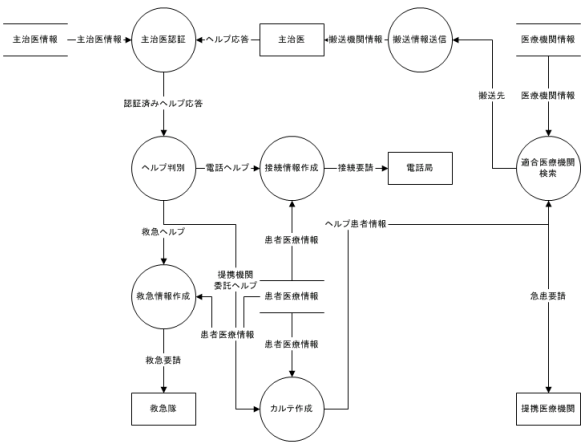

図 10　3. ヘルプ応答

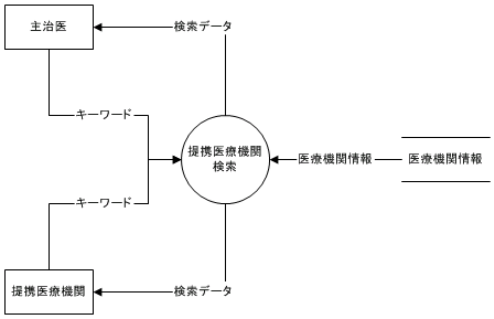

図 11　4. 提携医療機関連携

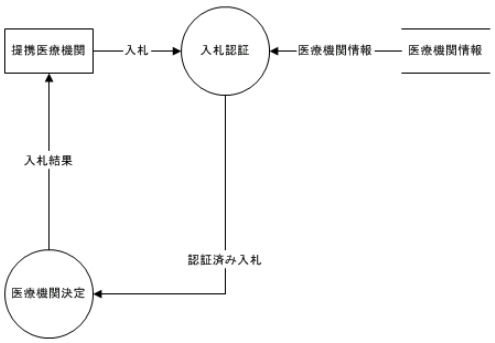

図 12　5. 一般公開入札

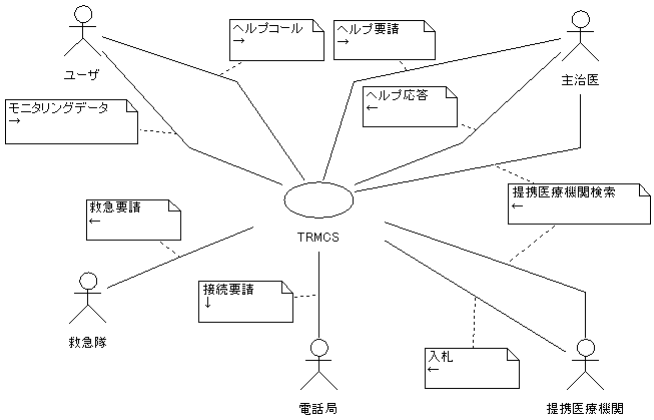

図 13　コンテキストダイアグラムをユースケース図に変換した図

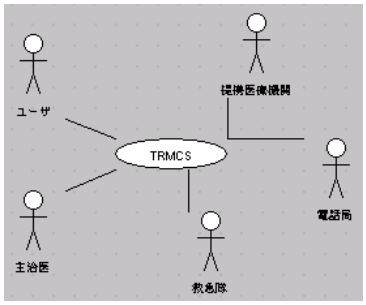

図 14　CASE ツールを利用してユースケース図に変換した図

ステム全体を現す図 6 の TRMCS をユースケースへ変換し、システムの外部をアクタで記述することで同一の意味のダイアグラムが生成された.

次にダイアグラム 0 より下階層の DFD をクラス図へ変換する. ここで、例として図 8 を変換規則に沿ってクラス図に変換してみる. 先ずルール C1 より、データストア「LOG」と「患者医療情報」をクラスとして認識する. 次にルール C2 より、データストアに直接入出力するプロセスをクラスのメソッドとする. ここでは、「ヘルプログ保存」が LOG クラスのメソッドとして、「ユーザ認証」、「緊急カルテ作成」、「ヘルプカルテ作成」が患者医療情報クラスのメソッドとして割り当てられる. ルール C3 に従い、「コールタイプ判別」はデータ辞書を見てどのクラスに割り当てるかを判別する. このプロセスが扱うデータ辞書の項目は、患者 ID、患者氏名、コールタイプ、症状であり患者医療情報に加えると判断する. ルール C4 により、データストア間のデータフローに対応した関連を引く. 以上の操作により図 15 が得られる. 図 16 は図 8 を CASE ツールを利用して変換し、

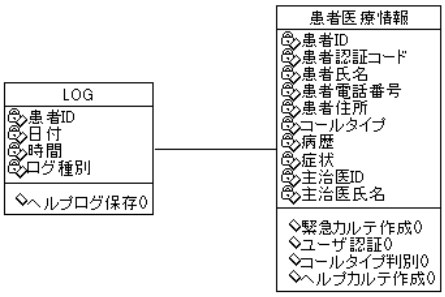

図 15　ヘルプコールをクラス図へ変換した図

出力した XMI を ArgoUML へインポートした画面である. こちらも変換規則どおりにクラス図が変換できた. データフロー図ではデータとそれに対する操作が表現されており、データと操作の関係をまとめてクラス図を作成する. 図 8 では、データストア「LOG」をクラスに変換し、そのデータを操作するプロセス「ヘルプログ保存」をそのメソッドに変換している. データストア「患者医療情報」についても同様にクラスに変換している. また、データフロー「ヘルプログ保存」の関係が両クラスの関連に対応している. 今回の実験では、プロセス「コールタイプ判別」を割り当てるクラスの判別をユーザに問い合わせるのではなく、このプロセスが扱うデータ辞書の項目を多く含むクラスを自動で選択するようにして実験を行った. 以下、図 17 から図 20 は同様の変換を行って、ArgoUML へインポートし、出力した画面である. これらの変換では変換規則に従って手動で変換したものと同一の意味を表すダイアグラムを出力することが出来た.

7. 結　　び

DFD の定義を DFD メタモデルにて厳密に行い、その後作成した UML への変換規則を実現する CASE ツールを作成した. そして、CASE ツールは変換規則に従って変換を行うことを示した. クラス図への変換でのプロセス割り当ての選択では、今回の実験ではこの CASE ツールは変換規則に沿って、ユーザに問い合わせるのではなく、プロセスが扱うデータ辞書の項目を多く含むクラスを選択するように変換を行った. しかし、ケーススタディで現在の機能でも十分クラス図への変換が可能なことが分かった. ユーザ判断が必要な項目は、

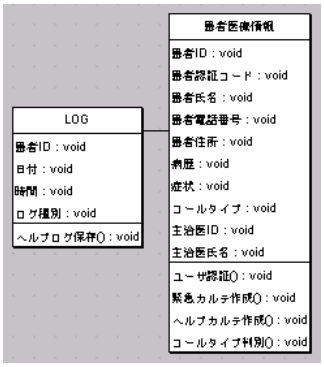

図 16　ヘルプコールをクラス図へ変換した図（CASE ツール）

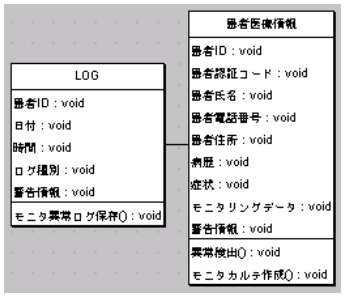

図 17　モニタリングをクラス図へ変換した図（CASE ツール）

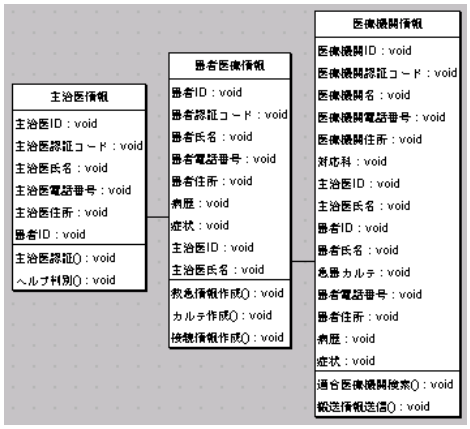

図 18　ヘルプ応答をクラス図へ変換した図（CASE ツール）

図 19　提携医療機関連携をクラス図へ変換した図（CASE ツール）

- メソッドの割り当てで複数のクラスへ割り当てが可能な場合どのクラスに割り当てるか
- DFD の抽象度に応じてユースケース図とクラス図のどちらを作成するのか

である. これらの機能を実現すればさらにシステムにふさわしい UML を出力することが可能になる.

また、変換規則に従った変換はデータ辞書を解析する作業が伴い、大規模なシステムでは手動でこれを行うのは困難である.

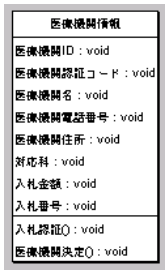

図 20　一般公開入札をクラス図に変換した図（CASE ツール）

CASE ツールによる変換規則の半自動化によって、変換に際しての人為的な間違いを防ぎ、また、変換の高速化が可能となった. これにより、高品質なソフトウェアの開発、開発期間の短縮、コストダウンの面でシステム開発への貢献が期待できる.

今後はユーザに問い合わせを行うシステムの実現を行う. また今回、CASE ツール作成の環境の問題でシーケンス図を出力することが出来なかったため、この解決も今後の課題である. シーケンス図と同じ相互作用図であるコラボレーション図を利用して DFD のプロセスの方向性を表現することも可能なため、これに関しての変換規則の変更と CASE ツールでの出力させることでこれが可能となる.

なお、DFD の作図に Microsoft VISIO を使用している.

文　　献

[1] OMG: Unified Modeling Language Specification, March 2000
[2] OMG: Meta Object Facility Specification, April 2002
[3] OMG: XML Metadata Interchange Specification, January 2002
[4] Tom DeMarco: 構造化分析とシステム仕様, May 2001
[5] Alejandro Ramirez, Philippe Vanpeperstraete, Andreas Rueckert, Kunle Odutola, Jeremy Bennett, Linus Tolke: ArgoUML User Manual, 2003
[6] 鯵坂恒夫, 佐伯元司: 方法論工学と開発環境, November 2001
[7] James S. Willans,Paul Sammut,Girish Maskeri,Andy Evans,Tony Clark: Defining OCL expressions using templates, The precise UML group
[8] Masataro Shiroiwa, Hidenobu Tokuda, Takao Miura: Documentation Maintenance DFD by Means of UML, *ISE* 2002
[9] Masataro Shiroiwa, Takao Miura, Isamu Shioya: Meta Model Approach for Mediation, *COMPSAC* 2003