DEWS2003 6-B-01

UMLによるDFDメタモデルの定義

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 本論文では、構造化手法で用いられるDFDをオブジェクト指向設計で用いられるUMLに変換することによって、既存システムから新規システムへの拡張、統合の為の設計支援をするのが目的である。本論文では、Object Constraint Language（OCL）とクラス図を用いてData Flow Diagram（DFD）のメタモデルを定義し、DFDメタモデルのインスタンスとUMLの観点から対応をとることによってDFD表現をUML表現に変換する規則を作成する。また、ケーススタディを用いて変換規則に基づきどのようにDFDからUMLが生成されるかを述べる。
キーワード ソフトウェア設計、UML、DFD

Defining DFD Meta Model using UML

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, our goals are to extend or to integrate from legacy system to modern system to support software design. Therefore, we convert Data Flow Diagram (DFD) used by structured design into Unified Modeling Language (UML) used by object-oriented design. In this paper, we define DFD meta model by using Object Constraint Language (OCL) and class diagram. Next, we compare DFD meta model with UML meta model. And, we make conversion rules which convert DFD expression into UML expression. Moreover, we describe how UML is generated from DFD by using case study based on the comparison.
Key words Software design,UML,DFD

1. 前 書 き

Data Flow Diagram(DFD) セマンティクスをUnified Modeling Language(UML) によるメタモデルで記述し、既存のDFDからUMLへの拡張支援の方法を述べる。ソフトウェアの長い保守サイクルでは新しい要求の追加や変化が生まれる。その際システムの全体を入れ替えるのではないく、新しい要求に対応するための差分を開発することの方が容易である。そこで本論文では、レガシーシステムから新規システムへの拡張・統合の方法について述べる。長い保守サイクルの中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。本論文では、この間違った認識の発生を排除するためにレガシーシステムで利用されたモデルのセマンティクスをメタモデルによって厳密に記述するアプローチを提案する。ここで我々はDFDのメタモデルをUMLのクラス図とObject Constraint Language(OCL) によって記述する。DFDはレガシーシステムを設計するために用いられてきた手法である。また、我々はUMLとDFDメタモデルを比較し、DFDメタモデルのインスタンスがUMLとどう対応するかを検討する。検討の結果、DFD表現をどのようにUML表現に変換するかを示す。

2章で定義すべきDFDセマンティクス、3章でDFDメタモデルの仕様を述べる、4章でDFDをUMLで表現するための変換規則を述べ、5章でケーススタディを用いてDFDのDFDメタモデルのインスタンス、変換規則による変換を示し、6章が結びとなる。

2. DFD

ここではDFDの概要を述べ、さらに詳細なセマンティクス

はOCLによる制約と共に述べる[3]。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つ。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を定義することが出来る。よって、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を定義できる。

（1） ソース、シンク

- システム外部を表すノード。

（2） データストア

- 一時的なデータの貯蔵庫を表すノード。

（3） プロセス

- システムの機能を表すノード。
- 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。
- 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。

（4） データフロー

- 各ノード間のデータの流れを表す有向エッジ。
- 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。
- 途中から分岐することも出来る。ただし、流れるデータは同一。
- 入出力のバランスは一致する (図1)。

例えば、図1では左側ダイヤグラムは右側ダイヤ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。DFDはデータの流れを表現するのであって、コントロールを

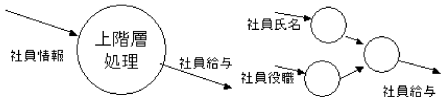

図1　入出力の一致

示さない。しかし、データの流れが図2のような場合、データフローA → B → Cに反復が存在する可能性を読み取れる。ただし、実際に反復が存在するかどうかは仕様を確認する必要がある。また、DFDは起動因子を示さない。よって、プロセス

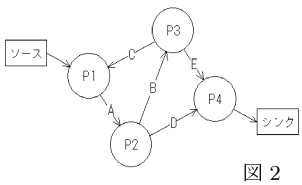

図2　ループ

からどのような条件で分岐するのか、どのような条件で反復が起こるのかは仕様を確認する必要がある。

3. DFDメタモデル

我々はDFDをUML[1]へ変換する規則を発見するために、DFDのメタモデル[2][4]をUMLで定義する。その為、我々はDFDメタモデルをUMLとOCLによって記述する。そして我々はDFDの表現をUMLを用いて解釈することを可能にする。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][5]がある。我々はここで同じアプローチを採用している。

3.1 メタモデル

メタモデルの役割はモデルの仕様を記述する言語を定義することである。メタモデルによってモデルのセマンティクスが規定され曖昧な解釈の発生を防ぐことができる。モデルはメタモデルのインスタンスとなる。

3.2 DFDメタモデル構成

DFDメタモデルは抽象構文、適格性規格によって記述される。

（1） 抽象構文

UMLのクラス図 (図3) 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。

ここで、OCL[1][5]はモデルに対して制約を記述する副作用を持たない言語である。UMLはモデルを表現するために、9つの図を用いる。しかし、図だけではモデルを正確に表現することはできない。OCLはオブジェクトが満足しなければならない不変条件、操作の前後で満足しなければならない条件などをモデルへの制約として表現する。モデルを詳細かつ正確に記述するために不可欠な言語である。制約は自然言語で記述することもできる。しかし、自然言語の曖昧な表現はモデル解釈の間違いを引き起こす。この曖昧さを排除するためにもOCLは不可欠である。また、OCLは副作用を持たない。つまり、OCLの式が評価されたとしても、式の値が返されるだけであり、モデル内に変化を与えることはない。OCLはUML仕様書の中でその仕様が定められ、UML仕様書の中でUMLメタモデルの正確な仕様を記述するために使用されている。OCLは以下の目的で使用可能である。

- クラス図のクラスや型の不変条件。
- 操作に関する事前事後条件。
- 操作の制約の記述。
- 現在のオブジェクトからほかのオブジェクトのプロパティを参照するための言語。

我々はDFDメタモデルの仕様を正確に記述するためにOCL

を使用する。

（3） DFD との対応

具象メタクラスのインスタンスが DFD セマンティクスのどこと対応するかここで述べる。

3.3 DFD メタモデル

3.3.1 クラス図

DFD メタモデルを示すクラス図は以下のようになる。図 3

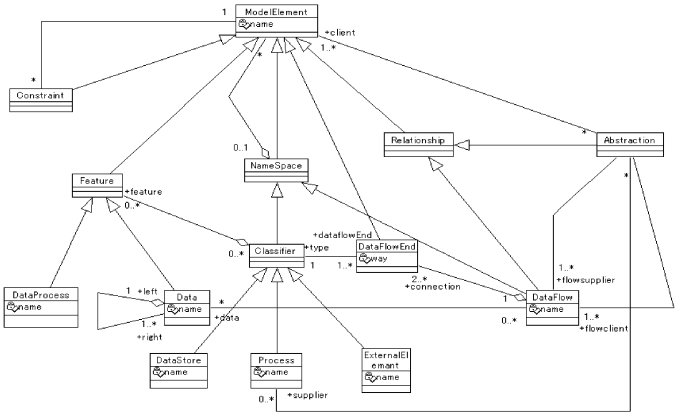

図 3 DFD メタモデル

の各要素は以下で定義される。

3.3.2 Abstraction

（1） 抽象構文

同一概念を抽象度の異なった段階で表現した要素集合を関連付ける関係を定義する具象メタクラス。メタモデルでは、Process を親 (supplier) とし、すべてのモデルの上位クラスである ModelElement を子 (client) としている。これにより、プロセスとそのプロセスの下階層の要素の間の抽象関係を記述することが可能である。また、DataFlow のインスタンス間の親子関係を定義し、その際の入出力バランスについて適格性規格で記述している。図 4 のインスタンスの例の場合、給与計算プロセスとその下階層の間に図のような Abstraction 関係を引くことが出来る。

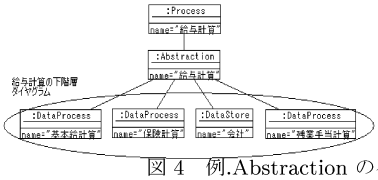

図 4 例.Abstraction のインスタンス

（a） 関連

client、flowclient:supplier 要素を低い抽象度で表現した要素。

supplier、flowsupplier:client 要素を高い抽象度で表現した要素。

（2） 適格性規格

入出力のバランスは一致する。

self.flowsupplier.data.allData=self.flowcliant.data.allData

（3） DFD との対応

プロセスとプロセス、データストア、データフローの間の抽象関係と対応する。また、データフローの入出力のバランスが親子ダイヤグラム間で一致することを定義している。

3.3.3 Classifier

（1） 抽象構文

Feature の集合を宣言する抽象メタクラス。Feature の集合を宣言する、DataStore、Process、ExternalElement の親クラス。

（2） 適格性規格

Data は一つの Classifier 内に同じ名前を持ってはならない。

self.feature->

select(a|a.oclIsKindOf(Data))->

forAll(p,q|p.name=q.name implies p=q)

（a） 付加的操作

操作 dataflows は、Classifier それ自体のすべての DataFlow を含む Set を返す。

dataflows:set(DataFlow);

dataflows=self.dataflowEnd.association->asSet

3.3.4 Constraint

（1） 抽象構文

文章による意味論的な条件や制限を定義する具象メタクラス。メタモデルでは関連した ModelElement の制約である。

（2） 適格性規格

Constraint はそれ自体には適用できない。

（3） DFD との対応

DFD への文章による付加的な説明と対応する。

3.3.5 Data

（1） 抽象構文

Classifier のインスタンスが持つ値の領域の名前を持った構成要素を定義する具象メタクラス。DataStore と DataFlow と関連を持つ。

（a） 属性

name:Data の識別子

（b） 関連

left:複数の Data から構成される Data。

right:ある Data に集約される Data。

（2） DFD との対応

Data のインスタンスはデータストアの蓄えるデータの項目、データフローを流れるデータの項目に対応する。また、データ辞書の式における各項と対応する。

3.3.6 DataFlow

（1） 抽象構文

Classifier 間で伝達される Data の集合を定義する具象メタクラス。メタモデルでは DataFlow 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。Data と関連を持つ。DataFlow のインスタンスは図 5 のようにそのインスタンスを構成する Data のインスタンスと関連する。

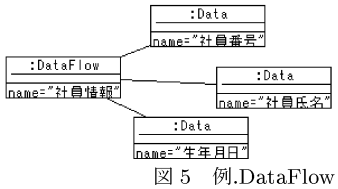

図 5 例.DataFlow のインスタンス

（2） 適格性規格

DataFlow は少なくとも 2 つ以上の DataFlowEnd を持つ。

self.allConnections->size>=2

（a） 付加的操作

操作 allConnections は、DataFlow のすべての DataFlowEnd の集合を返す。

allConnections:Set(DataFlowEnd);

allConnections=self.connection

（3） DFD との対応

データフローを通るデータの集合と対応する。

3.3.7 DataFlowEnd

（1） 抽象構文

DataFlow を Classifier に結合する端点を定義する具象メタクラス。Classifier に対しての入出力方向を示す属性を持つ。インスタンスは図 6 のようになる。way=in の時、各 Classifier のインスタンスへの入力、out の時、出力を表す。DataFlow 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。

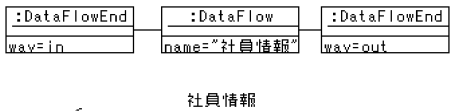

図 6 例.DataFlowEnd のインスタンス

（a） 属性

way:結合要素への入出力を表す属性。

（2） 適格性規格

同一 DataFlow の複数の DataFlowEnd が一つの Clasifier と結合することはない。

self.allConnections->size>=2 implies

forAll(r1,r2|r1.type.oclIsKindOf(Classifier)

=r2.type.oclIsKindOf(Classifier) implies r1<>r2)

同一 DataFlow のすべての DataFlowEnd が同じ way であることはない。

self.allConnections->size>=2 implies

self.allConnections->exist(r1,r2|r1<>r2

implies r1.way<>r2.way)

（3） DFD との対応

データフローの有向性と対応する。

3.3.8 DataProcess

（1） 抽象構文

Classifier のインスタンスが持つ名前を持った操作を定義する具象メタクラス。

（2） DFD との対応

最小限度まで分解されたプロセスと対応する。

3.3.9 DataStore

（1） 抽象構文

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。インスタンスは図 7 のようになり、DataStore のインスタンスを構成する Data のインスタンスと関連する。

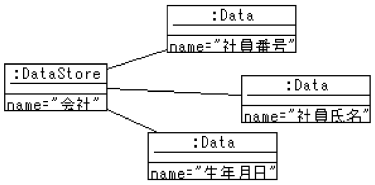

図 7 例.DataStore のインスタンス

（a） 属性

name:識別子

（2） 適格性規格

DataStore は Process に対する DataFlow だけを持つことができる。

self.dataflows->forAll(a|a.allConnections->

exists(r|r.type.oclIsKindOf(Process)))

（3） DFD との対応

データストアと対応する。

3.3.10 ExternalElement

（1） 抽象構文

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。

（a） 属性

name:識別子

（2） 適格性規格

ExternalElement は Process に対する DataFlow だけを持つことができる。

self.dataflows->forAll(a|

a.allConnections->exists(r|r.type.oclIsKindOf(Process)))

ExternalElement は client にならない

self.oclAsType(ExternalElement).client->isEmpty

（3） DFD との対応

ソース、シンクと対応する。

3.3.11 Feature

（1） 抽象構文

Classifier 内に集約されるプロパティを定義する抽象メタクラス。Classifier の Instance または Classifier 自体の操作や構造上の特徴を宣言する。Classifier に集約される Data、DataProcess の親クラスである。

3.3.12 ModelElement

（1） 抽象構文

メタモデル内の最上位メタクラスである抽象メタクラス。

（a） 属性

name:ModelElement の識別子。

3.3.13 NameSpace

（1） 抽象構文

NameSpace は ModelElement の集合を含むモデルの一部で定義される抽象メタクラス。NameSpace に集約される ModelElement の属性:name はその NameSpace 内で一意な要素を示す。

3.3.14 Process

（1） 抽象構文

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。インスタンスは図 8 のようになり、Process のインスタンスを構成する DataProcess のインスタンスの集合と関連する。

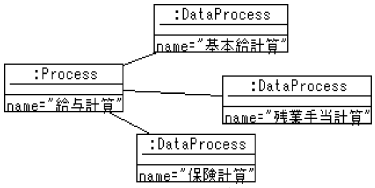

図 8 例.Process のインスタンス

（a） 属性

name:Process の識別子。

（b） Stereotypes

component:そのプロセスが上位となる Abstraction 関係が存在することを表す。

intermediate:そのプロセスが下位となる Abstraction 関係が存在することを表す。

（2） 適格性規格

Process は Process,DataStore,ExternalElement に対する DataFlow を持つことができる。

self.dataflows->forAll(a|a.allConnections->
exists(r|r.type.oclIsKindOf(Process)or
r.type.oclIsKindOf(DataStore)or
r.type.oclIsKindOf(ExternalElement)))

同一 DataProcess のすべての DataFlowEnd が同じ way であることはない。

self.allConnections->size>=2 implies
self.allConnections->exist(r1,r2|
r1<>r2 implies r1.way<>r2.way)

（3） DFD との対応

プロセスと対応する。

3.3.15 Relationship

（1） 抽象構文

Classifier のインスタンス間の関係を定義する抽象メタクラス。Abstraction と DataFlow の親クラスである。

4. DFD への変換

我々は DFD の表現を UML の表現へどう変換するかを示す。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 に変化がある。よって、変換に際しても抽象度に応じた変換が必要となる。我々はここでユースケース図、クラス図、シーケンス図へ DFD の表現を変換する。

4.1 ユースケース図の作成

DFD を UML に変換する際には抽象度の高いダイヤグラムから変換を始める。抽象度が高い DFD において、ExternalElement のインスタンスは UML のアクターと、Process のインスタンスは UML のユースケースと対応が取れる。ExternalElement 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 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。ExternalElement のインスタンスはソース/シンクと、Process のインスタンスはプロセスとそれぞれ対応している。よって、以下のルールによってユースケース図を作成する。

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。

4.2 クラス図の作成

DataStore のインスタンスと Process のインスタンスの集合は UML のクラスと対応が取れる。DataStore のインスタンスは共通のデータの集合を表し、Process のインスタンスは共通の操作の集合を表しており、それぞれデータストアとプロセスに対応している。また、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。データストアはクラスが持つ共通の属性と、プロセスはクラスが持つ共通のメソッドと対応している。そこで、我々はデータストアをクラス認識の起点として考えてクラスとする。次に、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。データストアへ直接入出力しているプロセスはクラスの属性を操作していると考えられる。あるプロセスが 1 つ以上のデータストアと直接的もしくは間接的に関係している場合、プロセスとデータストアが扱うデータ辞書をそれぞれ調べ、プロセスをどのクラスに割り当てるか決定する。クラス間の関連については、クラスとして認識したデータストアの間に存在するデータフローの経路をクラス間の関連として与える。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」などの関係があるからである。よって、以下のルールによってクラス図を作成する。

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: 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。

4.3 シーケンス図の作成

DataFlowEnd は属性:way を持ち、これにより DataFlow のインスタンスの方向を表現できる。また、DataFlow のインスタンスが集約する少なくとも 1 つ以上の DataFlowEnd のインスタンスは必ず Process のインスタンスと結合する。よって、それぞれ関連する DataFlowEnd、DataFlow、Process の一連のインスタンスにより Process のインスタンスの方向を表現できる。

この Process 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。先ず、ユースケース図とクラス図の

作成によってアクター/クラスを認識する。次に、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。我々は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過ぎず、必ずしも全体の実行順序を示すものではない。メッセージの起動条件がわかる場合には、メッセージのラベルの前に [条件] で示す。プロセスからデータフローが複数に分岐した場合、シーケンス図の同一の活性区間からメッセージを記述する。また、そのメッセージに [条件] を加えることで、分岐を表現する。あるクラスのメソッドとして認識されるプロセスから出たデータフローが他のクラスのメソッドとして認識されるプロセスを経由して戻ってきた場合、クラス間にメッセージのループが存在している。さらに、データ辞書を調べて戻ってきたデータフローの構成要素が出て行くデータフローの構成要素となっているならば、これらのクラス間には繰り返しが存在している。この繰り返しをシーケンス図では繰り返しの存在する部分に矢印を用いた反復の記号を付け、repetition と記述しておくことで表す。繰り返しの条件がわかる場合には [条件] の中に添えておく。よって、以下のルールによってシーケンス図を作成する。

S1: ユースケース図とクラス図の作成によってアクター/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。

S3: 条件の記述。

S4: 分岐の記述。

S5: 繰り返しの記述。

5. ケーススタディ

この章で我々はケーススタディを用いることによって DFD が DFD メタモデルのインスタンスとしてどう表現できるか、また、変換規則によって DFD がどう変換されるかを示す。我々はこれらの図の作図に Rational Rose を使用している。

5.1 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出来る。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。

5.2 TRMCS の DFD

図 9 から図 15 は TRMCS を表す DFD である。コンテキストダイヤグラムにより、システムとその外部の境界を示し、コンテキストダイヤグラムの単一プロセスの下階層をダイヤグラム 0 が表現している。またダイヤグラム 0 の各プロセスのラベルの前についた番号とプロセスの下階層のダイヤグラムの番号は一致する。

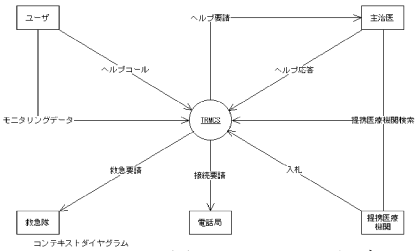

図 9 コンテキストダイヤグラム

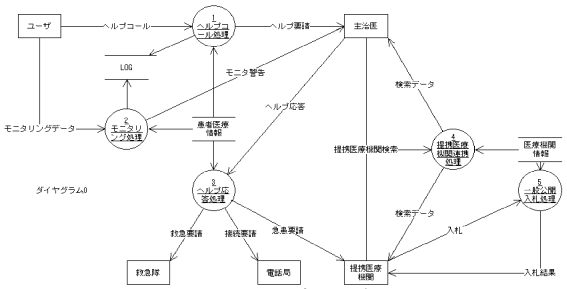

図 10 ダイヤグラム 0

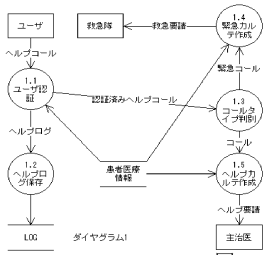

図 11 1. ヘルプコール

5.3 TRMCS の DFD メタモデルのインスタンス

次に、DFD を DFD メタモデルにしたがってインスタンス化したものを一部示す。図 16 のコンテキストダイヤグラムのインスタンスを見ると、中心にある属性:name が「TRMCS」の Process のインスタンスと外側にある各 ExternalElement のインスタンスが DataFlow のインスタンスによって、関係付けら

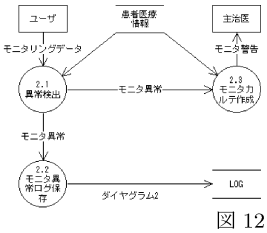

図 12　2. モニタリング

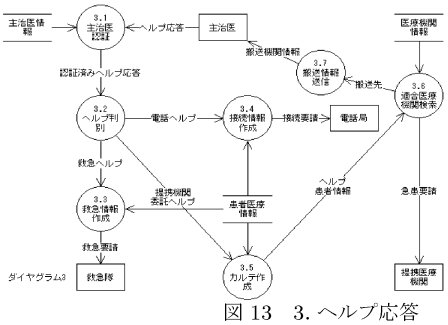

図 13　3. ヘルプ応答

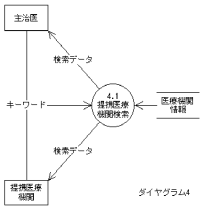

図 14　4. 提携医療機関連携

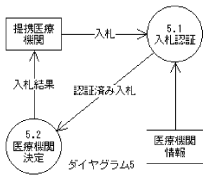

図 15　5. 一般公開入札

れているのがわかる。また DataFlow のインスタンスは 2 つ以上の DataFlowEnd のインスタンスを持ち、DataFlowEnd のインスタンスは各要素のインスタンスに関連付けられている。DataFlowEnd は way 属性を持ち DataFlow のインスタンスが各要素のインスタンスに対して入力であるか出力であるかを表現する。このように DFD の意味を DFD メタモデルのインスタンスとして UML で表現することが可能である。ただし、図が煩雑にならないようにするため、DFD メタモデル Feature に関するインスタンスを省略した。また、Abstraction に関するインスタンスも省略されている。Abstraction は同一概念を抽象度の異なった段階で表現した要素集合を関連付ける関係である。例えば、属性:name が「TRMCS」の Process のインスタンスとその下階層を表すダイヤグラム 0 のインスタンスを考える。この場合、属性:name が「TRMCS」の Process のインスタンスから下階層のすべてのインスタンスへ Abstraction を表す関係が存在する。

5.4　UML への変換

抽象度の高いコンテキストダイヤグラム図 9（図 16）をユースケース図に変換する。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」

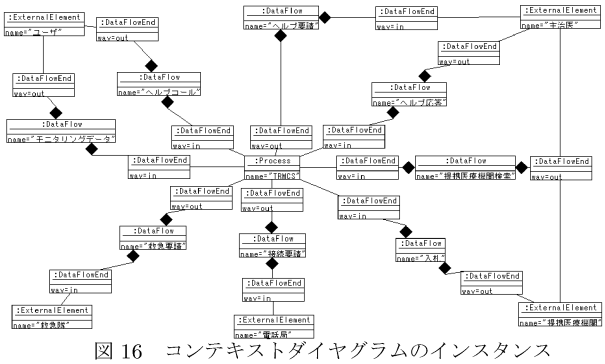

図 16　コンテキストダイヤグラムのインスタンス

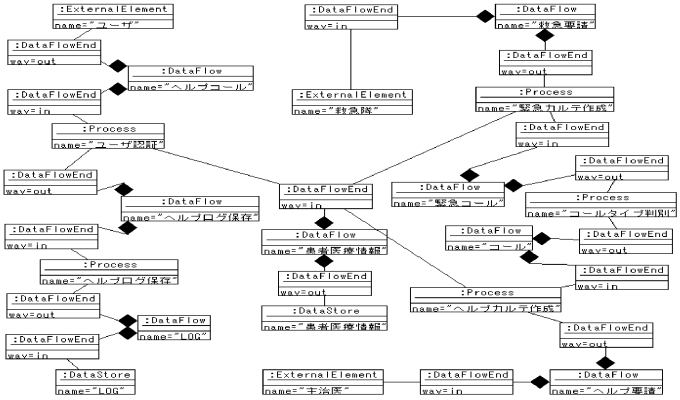

図 17　ヘルプコールのインスタンス

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き、図 18 のユースケース図を得る。次に変換

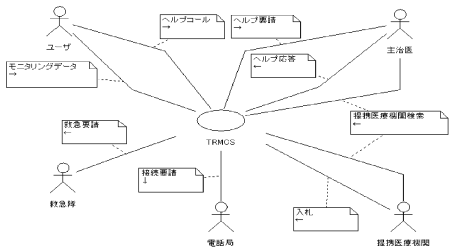

図 18　コンテキストダイヤグラムをユースケース図に変換した図

規則に基づいて DFD をクラス図に変換したものを示す。例として図 11 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き、図 19 のクラス図となる。以下、図 20 から図 23 は同様に変換できる。図 24 はすべてのクラスを結合したものである。　次に変換規則に基づいて

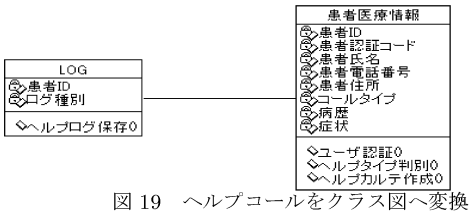

図 19　ヘルプコールをクラス図へ変換した図

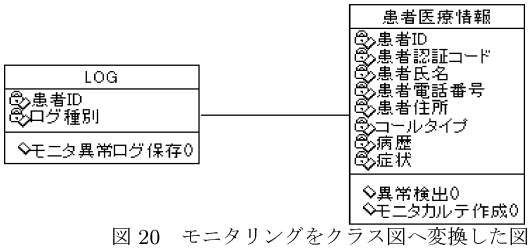

図 20　モニタリングをクラス図へ変換した図

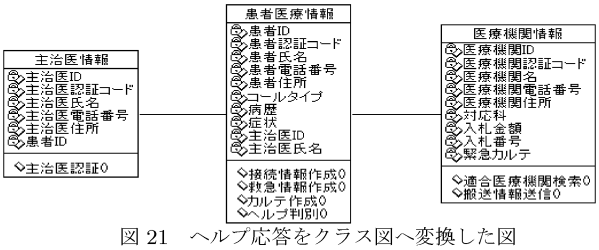

図 21　ヘルプ応答をクラス図へ変換した図

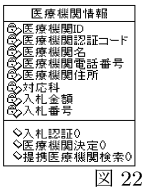

図 22　提携医療機関連携をクラス図へ変換した図

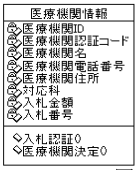

図 23　一般公開入札をクラス図に変換した図

シーケンス図へ変換を行う。例としてクラス図 19 へ変換した図 11 をシーケンス図へ変換する。ルール S1 より、アクターとして「ユーザ」、「救急隊」、「主治医」を認識し、クラスとして「LOG」、「患者医療情報」を認識する。ルール S2 より、図 11 を参照して各プロセスをメッセージとして与える。ここでは、「ユーザ認証」をユーザから患者医療情報へのメッセージとして与える。以下、同様の操作でメッセージを与えて図 25 が得られる。ここで一つの活性区間から複数のメッセージが発生しているのは、その活性区間の前のプロセスからのデータフローが分岐しているためである。例えば、「緊急カルテ作成」と「ヘルプカルテ作成」が同一活性区間から発生しているのはその前の「コールタイプ判別」からのデータフローが分岐しているためである。以下、図 26 から図 27 は同様に変換できる。

6. 結　　び

我々は UML のクラス図と OCL による制約によって、DFD メタモデルを作成し、DFD を DFD メタモデルのインスタンスとして表現することにより DFD メタモデルの正当性を示した。また、UML メタモデルとの比較をし、対応を考えることによって、DFD の表現を UML のクラス図とシーケンス図により記述できることを示した。

本研究の一部は文部科学省科学研究費補助金
(課題番号 14580392) の支援による。

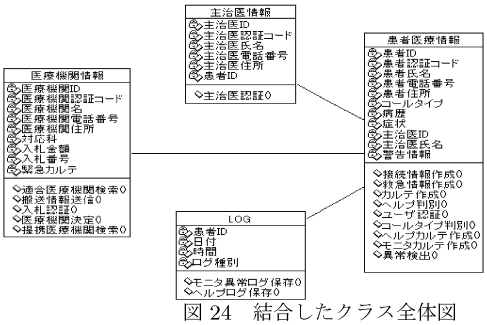

図 24　結合したクラス全体図

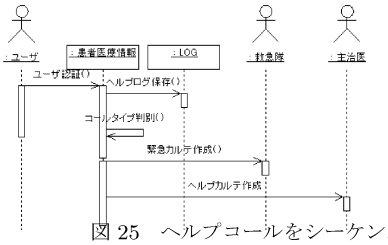

図 25　ヘルプコールをシーケンス図へ変換した図

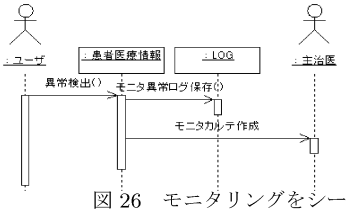

図 26　モニタリングをシーケンス図へ変換した図

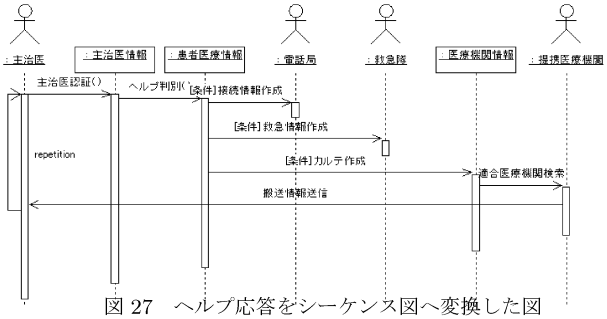

図 27　ヘルプ応答をシーケンス図へ変換した図

文　　献

[1] OMG: Unified Modeling Language Specification, March 2000
[2] OMG: MetaObjectFacility Specification, April 2002
[3] Tom DeMarco: 構造化分析とシステム仕様, May 2001
[4] 鯵坂恒夫, 佐伯元司: 方法論工学と開発環境, November 2001
[5] James S. Willans,Paul Sammut,Girish Maskeri,Andy Evans,Tony Clark: Defining OCL expressions using templates, The precise UML group
[6] Masataro Shiroiwa, Hidenobu Tokuda, Takao Miura: Documentation Maintenance DFD by Means of UML, *ISE* 2002