・<xs:enumeration value="Transition"/>
(3) conjugator.category.Type
・<xs:enumeration value="源"/>
・<xs:enumeration value="対"/>
・<xs:enumeration value="Source"/>
・<xs:enumeration value="Target"/>
(4) conjugator.kind.Type
・<xs:enumeration value="遷移.経過"/>
・<xs:enumeration value="遷移.合流"/>
・<xs:enumeration value="遷移.分岐"/>
・<xs:enumeration value="遷移.添加"/>
・<xs:enumeration value="遷移.脱落"/>
・<xs:enumeration value="遷移.消滅"/>
・<xs:enumeration value="Proceeded"/>
・<xs:enumeration value="Converged"/>
・<xs:enumeration value="Diverged"/>
・<xs:enumeration value="Promoted"/>
・<xs:enumeration value="Denoted"/>
・<xs:enumeration value="Terminated"/>

これらの図解などについては【資料５】【資料１７】を参照願いたい．ただし平成15年11月に学会発表した原稿ではアトリビュート値は旧版に即しているので注意されたい．なお生成したxmlは【資料１１】に掲げてある．

C．4 試作アプリケーションの全体構成

試作アプリケーションは次のportionから構成される：

・ログイン
・病名/プロブレム composer
・病名/プロブレム変遷 editor

この“コンテ”は【資料６】を参照願いたい．なお試作実装ゆえ“コンテ”に記載した構成や機能のうち最小限のみを実装した．

C．5 病名/プロブレム composer

C．5．1 機能仕様

試作アプリケーションにおいて用いる病名はMEDIS-DC病名集に典拠を置くこととし，以下のごとく実装設計した：

・病名コード体系はCSX XML Schemaに基づいたxmlにせず，DBMSであるcacheに格納する．
・前述「C．1」で云う根幹病名は『病名基本テーブル』に列挙されるエントリとする．
・病名構成要素としての「部位」は割愛し，前置修飾語にて代用する．
・病名および修飾語の並び順については『修飾語テーブル』の「修飾語位置区分」値のみで決定し，これはTopology/Dimensionに格納する．
・GUI上の病名選択または修飾語の選択は，System.Windows.Forms.TreeViewとSystem.Windows.Forms.ListViewとを併用する．
・検索機能については，『病名基本テーブル』の「病名表記」または「病名表記カナ（かな）」で文字列検索して「病名交換用コード」を取得た後に，『索引テーブル』にて対応する「対応用語コード」を持つエントリを全て抽出する，とした．

C．5．2 修飾語の扱い

さて修飾語は，現時点においてMEDIS-DC病名集分類では，その種類の区分分類が計画されているものの未完である．

しかし『病名/プロブレム composer』の操作性を向上させるためには修飾語の分類は必須なので，あくまでも本研究における試作アプリケーション用仮分類として試行した．ただ，その切り口は参考になろうと思われるので以下に列挙しておく：

・Anatomy（解剖）
・Artifact（人工物）
・Topology（形態的な方位方向など）
・Aging Stage（年齢的な時期）
・Clinical Stage（臨床的な経過時期）
・Event Stage（診療イベントに関わる時期）
・Cyclicality（周期性）
・Ordinal（基数）
・Type（型）
・Severity（重症度）
・Expression And Multiplicity（表現型など）
・Problem（プロブレム化させる修飾語）
・Etiology（病因論的な修飾語）
・Pathology（病理的な修飾語）
・Sex（性）
・Miscellaneous（その他）
・NotClassified（未分類事項）

なお試作アプリケーション用の仮分類ゆえ，（１）各修飾語は唯一の分類にのみ分類され，かつ（２）階層を持たない構造としていることなどの事由から，各修飾語は妥当に分類されていないことがあることを申し添えておく．

C．5．3 画面設計と試作実装

ここでは画面構成の概要のみを述べるに留めるので，具体の図譜については「C．4」の資料を参照願いたい．

画面は二つのportionから成る．一つは，構成された病名を格納する部分で，いま一つは病名および修飾語を選択する部分である．

前者はdragを可能としており，dragによって引き渡される情報塊は，交換用コード，病名表記，ＩＣＤコード，構築された文字列である．

後者は3枚のタグから成り，それぞれ前置修飾語，病名，後置修飾語を表示する．各々はTreeViewによって範疇選択し，ListViewから個別選択する構成とする．

試作実装したアプリケーションの画面は【資料7】を参照願いたい．

C．6　変遷editor：機能設計

変遷関係を表現するために，GUI上の関係線を具現するモジュールの機能を設計した．

当該モジュールは結果的に二つを用意することとなった：一つは扱いの容易さ，いま一つは拡張性の保持に主眼が置かれている．前者はTree pane，後者はGraph paneである．両者の差異は【資料8】の図説を参照願いたい．

なおGraph paneについては，本項にて述べる機能設計も，また次項にて述べる構造設計も，留意考察すべき点は少なくなかった．

C．6．1　Tree pane

Tree paneは，病名/プロブレムの変遷のみをGUI上に表示するために考案し，深さ一階層の関係を表現することを役割とする．つまり深さ一層のツリー構造のみ表現を可能とするコントロールである．

表示項目はpaneの外側に別のコントロールを用意してこれに表示する仕様とした．このため「関係線」は，片側の“アンカー”領域と対側アンカー領域とを結ぶことになる．

アンカー領域と，これに相応する表示項目との相応保持は，アプリケーションの責任において為される．これを容易または感覚的に実施するためには，両側の“アンカー”領域のindexと表示項目コントロールのindexとを一致させるのみで良いように設計した．ただしその下準備などは，当然アプリケーション側の責任である．

アンカーや関係線の形状その他の属性などはprogrammableまたはend-user editableを可能とするようコントロールとアプリケーション間のインターフェイスを整えた．勿論，lock/unlock等も可能である．なお仕様詳細については【資料9】を参照願いたい．

このコントロールはレガシーコーディングに慣れたプログラマ等にとっても解り易く扱い易い印象を与えるものと思われる．また通常の臨床現場において病名/プロブレムを追加編集削除する場合，ほぼ充分な機能を与えるものとなっている．

とはいえ幾つかの制限があることも否めない：

- グラフ構造をサポートできない．
- 結合の可否はanchorの唯一つの属性によって決定されてしまう．

前者は項目間の同時性（共起性）表現の場合に難を生じる．また後者は多重結合における結合選択性（拒否可能性）の実現に難を生じることとなる．

C．6．2　共起性表現と結合選択性

CSX Ontological XML Schemaの応用可能性は，単に病名/プロブレム変遷表現のみに限らず，後述する「D．2」と「E．2」に挙げた如く，今後の重要な施策に関わる各種情報ソースを標準的な形式で表現する際には極めて有効な手法となりうる．その際，多重グラフ構造および共起性制約の表現は必須となるが，Tree paneでは，これらを実現できない．

また病名/プロブレム変遷表現のみに限っても，表示項目間の関係は時間に沿った有向グラフとなりうる．Tree paneは時刻t-1プロブレムリストと時刻tプロブレムリストとの間で，各元の変遷関係を表現可能としているが，深さ1であるがゆえに，合流（convergence）表現も擬似的に可能となっているのである．

さらに，表示項目が病名/プロブレムであれ，診療行為であれ，その項目が属するdomainやsubdomainを超えつつ関係が形成される場合がある．ということはConjugator数のみならず，「結合の意義」も多様とならざるをえない．

> *本研究主題でいえば，病名/プロブレムやプロブレムリストは同一のsubdomainに属すが，診療行為は別subdomainに属すると考えられる（「B．1」を参照のこと）．*
>
> *そしてsubdomain『問題空間』に属する元は同一subdomain内で結合しうると同時に，subdomain『診療行為』に属する元とも結合しうるのである．*
>
> *ただしsubdomain『治療薬剤』やsubdomain『検査機器』などに属する元は，subdomain『問題空間』に属する元とは，直接的には*

結合できず，subdomain『診療行為』に属する元の介在を要する．

よって唯一つの“アンカー”属性によって結合可能性が規定されてしまうコントロールとは，本質的には機能脆弱と云わざるをえない．ただ「C．6．1」に記した如く，コーディングや実用面での易可用性を否定するものではない．

いずれにせよ，これらの点を解決するために，次項のGraph paneを開発することとした．

C．6．3 Graph pane

Graph paneは病名/プロブレム変遷のみならず，種々の事物要素とそれらの関係をGUI上に表示するために考案し，グラフ構造を有する関係の表現を役割とするコントロールである．

表示項目はpaneの内側に用意してこれに表示する仕様とした．よってGraph paneでは関係線のみならず表示項目も管理することになる．そして表示項目とは事物要素のみならず，その属性をも包含しうることを想定している．また，前述した共起性表現やDomain間結合選択性のサポートをも実現することを目標としている．

したがってTree paneの如きアンカー領域は，もはや存在しえない．替わりに用意すべきは，表示項目における結合場所であるhotSpotと，関係線と結合とを表すNexusとLigand，そして結合選択性を示すReceptorである，とした．

なおLigandやNexusは形状やlock/unlock等の諸属性を有しており，programmableまたはend-user editableを可能としている．

以下にReceptorとLigandの振舞いとその付帯事項を述べる．なお，全体像の簡略な把握には【資料8】を，クラスの詳細については【資料10】を参照願いたい．

結合と表示

- 表示項目はViewLabelに格納され表示される．
- ViewLabelはSubstanceのcategory/kindほかdomainにおけるbusiness logicに応じて0..*の結合選択性（mood）を持つ．
- ViewLabelは当該項目のmoodに応じた0..*のReceptorを持つ．ただしReceptorはGUIには表示されない．
- ViewLabelは上下左右にhotSpotを有する．
- hotSpotに対するGUI operationによって，Receptorが感応する．同時に，Ligandが生成されるか/または既存の対側Ligandも感応する．
- ReceptorはLigandに対して選択性を有する．より正確にはLigandを介して対側Receptorのmoodに対する選択性を有する．
- hotSpotにおけるGUI操作によってLigandとReceptorが通信し，ReceptorはLigandへmood，connectivityそしてIDを渡す．
- Ligandは互いに同側Receptorのmoodを対側Ligandに伝え，対側Ligandから伝達されたmoodと同側Receptorのmoodとを比較して結合可否を判別する．
- 両端Receptor間にmood適合性がある場合のみLigand - Nexus - Ligandがsurviveし，mood適合性の無い場合にはsuicideする．なおこの三つ組みをLigamentと呼ぶこととする．
- 生き残ったLigandsはhotSpotに配置される．
- 結合と関係線の表示はLigamentによって実現される．

よってReceptorとLigandは結合されるもののReceptor同士は直接的には結合されず，対側Receptorへの結合選択性を示すのみである．

表示管理系

上述した機能を下支えするために以下を用意する：

- Ligamentを統括するLigamentManager
- ViewLabelとLigamentManagerとを統括するGraphPane
- ViewLabelでのReceptorの発生等を管理するMoodManager

そのほか

その他については重複記述を避けるため，次項「C．7」にて述べることとする．

C．7 変遷editor：構造設計

Graph paneはTree paneの弱点を解消し，かつ汎用性を獲得するためにデザインすることとしたが，アプリケーション構造は一挙に複雑化することとなるゆえ論理アーキテクチュアに分割して各クラスおよびそれらの配置そして必要となるイベントなどを設計した．

なお各論理層（tier）間の相互干渉を回避するため，可及的にイベントを介しての通信またはインターフェイスとして扱えるCollectionを介してのアクセスとするよう設計努力した．

全体像の簡略な把握には【資料8】を，クラスの詳細については【資料10】を参照願いたい．

C．7．1 論理層

論理層は以下の如く5層に分割した：

・View tier
・Control tier
・Application Logic tier
・Business Logic tier
・Entity tier

なお5層に分割してはいるものの概念的にはLayers パタンあるいは BCE に拠る3層分割 boundary (View tier, Control tier) - control (Application Logic tier, Business Logic tier) - entity (Entity tier) である.

このデザインパターンを採用したのは，変更や拡張を局所化するとともに，下層については再利用を狙ったからである．ただ，このデザインパターン自体が有する複雑度の増加，効率の低下，そして伝播性の低下という側面も併せ持つこととなる.

なお論理アーキテクチュアの設計は，本研究の本質に関わるクラスのみに留めている.

View tier

この層はいわゆる view であり Control tier に管理されつつ view 生成する．また GUI 操作の受け取り口でもある.

ViewLabel とその hotSpot ならびに Receptor, そして Ligament (Ligand - Nexus - Ligand) はここに配される．それらの振る舞い等は前述した通りである．したがって基礎的・原則的な結合可否については，この tier 内の messaging のみで完結することになる.

Control tier

この層は View tier とともに boundary もしくは presentation を担う.

LigamentManager, GraphPane, MoodManager が配される．それらの責務の概要は前述した通りである．なお，これより下層の tier との間で状態伝播に関わる役割も果たすこととなる.

Application Logic tier

この層は，実装アプリケーションが提供すべき「場」を構成する責務を担う：

・solution における限定的な特定の業務
・Data provider のクライアント
・Form と GUI control のクライアント

この層と Business Logic tier との分離により，Business Logic tier を他のアプリケーションで再利用できる可能性を高めることを目的としている．或いは，boundary が変更された場合でも，「場」の形成におけるロジックを再利用できることを目論んだものである.

この層には AppItem, ItemAllignMap, そして AppLigament, AppConjugator, AppRelation, RelationManager が配される.

AppItem は BizItem から生成されるが必ずしも Substance とは限らず，その子エレメントほかアトリビュートであることも可，としている．なお ItemAllignMap は，ViewLabel と AppItem とのマッピングを管理している.

AppLigament は Ligament と相応しているが，RelationManager によって boundary と entity との間の差異が緩衝になっている．GUI 操作で関係が生成変更された場合，AppConjugator や AppRelation の管理，そしてそれらのアトリビュート値の管理決定等は RelationManager が担う．詳細は「C．7．2」を参照のこと.

AppItem と AppLigament の編集状況は，必要な場合には，Business Logic tier に伝えられ，対象 domain における business logic の検証を受けることとなる.

Business Logic tier

この層は，対象領域の記述に要する entity を生成するとともに，諸関係について business logic や domain semantics への適合性の検証を実施することを責務とする．さらに将来的には RuleBase や Knowledgebase にアクセスして censing engine を動作させることを想定している.

したがって上位層へのインターフェイスとなる Collection を生成する際には，事物要素とその諸関係の形式的意味的な整合確認をしたうえで，これを実施することとなる．また前述したとおり，boundary 独立であることを目標としている.

この層には BizItem, BizRelation, BizIndex が配される．BizItem は Substance Collection である.

BizIndex とは，多次元空間における Substance 間の直接的または直近の上下前後左右などの Topology を纏めた要約 Collection である.

Entity tier

この層は，Storage に対するアクセスや対象 domain にフレームワーク（本研究の場合には CSX Ontology model）を適用するための課題を担う．今回の試作実装では CSX ontological XML

Schema に則ったファイルそのものをソースとしている．

Entity tier の構造は，CSX Ontology model が下支えしていることになる．

event と interface

各 tier で発生する event は，BCE での各層内に留まるものと層間を超えるものとが区別されている．

なお原則として event 名に .FIX とある event は，上位層クラスへの event 発行としている．また boundary では，event を受信するクラスは，その event によって状態遷移するクラスでなく，当該クラスを管理する Manager クラスが受信することがある．

インターフェイスの役割を果たすクラスは，AppItem（と ItemAllignMap）と AppLigament，また BizItem，BizRelation，BizIndex である．

結合強度の表現

現時点では結合強度を格納する属性は，何れのクラスにも用意していない．これがどのような微少構造形式によって具現化されるべきかは哲学的な論議を含みがちとなるが，Ligand や Nexus においても表現可能ではある．

C．7．2 辺と端点と頂点

生成編集された Ligament（Ligand - Nexus - Ligand）情報を Control tier から Entity tier へと伝達する際の留意について述べる．

一つに Relation[@category and @kind]および Conjugator[@category and @kind]を決定する必要がある．

加えて Conjugator 数と，Nexus 数・Ligand 数との間の差異を吸収管理する必要がある．この不一致性は，画面構成クラスの編成と XML Schema 設計の間の視点不整合から生じている．すなわち頂点管理か，あるいは辺または辺端点の管理か，という差異に拠っている．

これらの整合と翻訳には Relation の存在が必要条件となる．

Relation ならびに Relation[@category]と Relation[@kind]は，アプリケーションが提供する『場』によって生成・決定される．また，このとき Conjugator[@category]も同時に決定されることになる．

というのも GUI operator が「ナニを編集する」という『場』は，まさにアプリケーション自身，言い換えれば AppLogic tier に位置する処理群が提供するからである．

しかし十分条件を満たすには他の管理情報も必要となる．その一つは関係付けられる二つの Substance の・category と kind で決定される domain/subdomain の同異であり，いま一つは，画面製作者が意図した「GUI 配置における意味・意義」である．

domain/subdomain に関しては，「B．1」「C．6．2」を参照願いたい．

この十分条件によって，Receptor に結合された Ligament は，その意義に応じた Relation の管理下に置かれて Conjugator とされる．また，一つの Relation 内の Conjugators には唯一つの「親」Conjugator が存在しうるという整理，換言すれば全ての Conjugator[@kind]の決定が為されうるのである．

なお Ligand - Nexus - Ligand と Conjugator とのマッピングのために，AppLogic tier に AppLigament と AppConjugator とを用意したが，AppConjugator には replica が用意され，相応と整合に貢献することとなる．この点は「C．7．1」を参照のこと．

AppLogic tier における RelationManager 等は，したがって，以下の二つの役割を担う：

・Relation と Conjugator の管理
・Ligament と BizRelation との間の整合と翻訳

なお，関係の意義付けについて一時的な混沌を許すような『場』を提供する場合には，その意義を GUI operator に尋ねるしかなかろう．このような事例は，変遷や連関をモデル化するツールや，archetype を作成するツールにおいて発生しうる．

通常業務システムでは，Relation [@category]，Relation [@kind]，Conjugator [@category]は画面製作者の管理下にあって自明であり，Conjugator [@kind]は Substance [@category and @kind]と画面製作者が意図する「GUI 配置における意味・意義」によって決定されうる．

C．8 変遷 editor：画面設計と試作実装

ここでは画面構成の概要のみを述べるに留めるので，具体の設計図譜については「C．4」の資料を参照願いたい．ただ「C．4」の資料では，Tree pane を用いた構成のみを例示している．

画面構成は使用するpaneによって異なるが，いずれにせよ最少一つ最多三つのプロブレムリストを表示できることとする．そして状況に応じて，表示プロブレムリスト数を変更可能とする．

一つのプロブレムリストを表示している状態では通常の1号様式画面と同等となる．二つの場合は，“動きの少ない”プロブレムリストを扱う場合を想定している．三つの場合は，逆に“動きの多い” プロブレムリストを扱う際に用い，医師の思考を支援することを想定した．

確定済みの過去の変遷関係は，医療というdomainにおけるsemanticsまたはruleにより編集不能とされるべきであるし，転記入力済み病名は後続する病名/プロブレムを持たないと考えられるが，これらの編集可否は「C．6」にて述べたようにprogrammableであり，またend-user editableとするか否かも制御可能である．

なお試作実装については，【資料12】を参照願いたい．用いたxmlは【資料11】である．

C．9 Character-Cast-Capacity model

これまでの殆ど全ての診療情報システムは，「人」の免許や所属部署という静的属性のみに応じて自動的に権限「付与」しているゆえに，権限を真に「管理」しているなどとは云い難いものであった．このような楽観的な機構には，セキュリティ強化用のハードウエアやinfrastructureを付加しても，本人確認が多少強化される程度に過ぎず，権限根拠の「管理」には至ることができないのである．

そして現実には「B．5」に記したとおり単一組織内においてさえ以下のような事象が発生しているのである：

・複数組織への所属に伴う役割の継承と集約
・診療場面に応じた様々な遷移と関与
・複数の立場から診療場面に応じた立場の選択
・権限の委譲と移譲

このような複雑な現実世界をシステム世界に適切に描写するには，権限の根拠を明確化する管理機構が必須となる．換言すれば権限根拠を直接的間接的に生じせしめる種情報塊を洗い出し，かつそれらの諸関係への言及が不可欠である．よって，権限管理，もしくは権限根拠の管理についての論理モデル構築は避けて通ることができないのである．

さて日常臨床では診療現場という『場の力』は無視しえない．そして場の状況は突発的であったり短寿命であったりすることが多いために，システム管理不能もしくはシステム管理不適ので，end userのシステム内“behaviour”は，自身の宣言と事後の監査により妥当性を検証することになる（「関係と状況」モデルから導かれる機構）．

また権限管理において権限根拠とその「固さ」を切り離して考えることはできない（「関係と状況」モデルでの権限管理技法の適用における尺度）．なお「固さ」の決定因子は以下の通りである：

・権限根拠を付与したauthorityの強さ
・権限根拠の寿命
・場における立場
・委譲や移譲の発生と期限

これらの既存成果を踏まえつつ，汎用性を付与するよう研究とリモデリングを進めるために，Character-Cast-Capacity model（役柄-配役-立場モデル）を構築することとした．

本モデルを3C modelまたは3Cモデルと呼ぶことがある．

なお，3Cモデルの概念図は【資料13】に，またpreliminaryかつprimitiveなXML Schema設計は成果発表【資料18】に掲げてある．

C．9．1 行為点の形成

まず全構成の根元的な単位を構築する．それは点または行為点（actPoint）である．

actPointは，時刻，場所，関与者から成る．関与者は通常複数となる．場合によっては場所も複数となることがある．ただしこの場合でも，システム管理時刻およびシステムが管理するべきactPoint連番は，一つのactPointごとに唯一である．

actPointはPKI（Public Key Infrastructure）における点管理機構に沿うよう設計した．

C．9．2 行為場の形成

次に行為場（actField）を構築する．actFieldはactPointの連である．

すなわち，一人の患者の一連の診療に関わるactPointの連なりである．よってactFieldは，少なくとも暗黙的には目的や解決すべき問題をもって時間推移のなかで綴じられることになる．この表象は「B．3」との強い相応性が

ある．

なお，個々のactPointにおける関与者（登場人物）は，必ずしも一致しない．また場所も，actPointごとに異なることがある．

C．9．3 行為者

行為者（Actor）は，具体のヒト（Person）または組織単位（Party）である．

Personは，後述する「役柄」に「配役」される．よって後述する「実施者」とは相異なる．

Partyは，後述する「組織単位縦列」構成要素となりうる．

なお現時点では，Partyは Characterに castされない，と考えることとする．というのも3C modelの主眼とは，Personが執行すべき行為を支える権限に関する根拠の管理にある，からである．

C．9．4 役柄

役柄（Character）は三つしか用意していない：

・実施者（Participant）
・消費者（Consumer）
・親類縁者知人など（kithKin）

Participantは health service提供者である．Consumerは health serviceの消費者である．kithKinとは consumerの親類縁者知人などである．

なおCharacter自体は三種のみであるが各々が執りうる「立場」は様々なのである．

C．9．5 配役

Personは一つ以上のCharacterに配役（cast）されうる．

Personは Characterに castされなければ，actPoint（場面：scene）には登場することができない，とする．

ただし actFieldには登録されている可能性がある．

C．9．6 立場の適用

Characterは，actPointで立場（Capacity）が適用される・または選択する．Characterは，あるactPointにおいてあるCapacityのもとに行為（Action）を実施（act）する，とする．

言い換えればCharacterは，Characterであるのみでは何らActionを actできないし，そもそも当該のactPointにおけるCapacityが前提されないCharacterが actPointに登場しても無意味ゆえ，actPointには登場「できない」と云うよりもむしろ「しない」のである．

C．9．7 組織単位と役割

組織単位（Party）とは特定の役割（Role）を担った同定可能かつ機能的なヒトの塊である．Partyは，それが存する界において一つ以上のRoleを持っている．

例： 医療機関，診療科部，診療グループ，Taskforce，委員会など．

Partyには，Partyの Roleに即した Characterが求められる．必要とされている Characterを確保するために Personが castされ，同時に，そのPartyに配属（assign）される．

C．9．8 組織単位縦列

組織単位縦列（Fleet）は，多段の Partyにて構成された Partyである．あるいは，組織単位縦列におけるある境界 Partyまでの Party列である．

通常，祖先 Partyの Roleは子孫 Partyへ継承される．継承結果としての Roleは，Fleetの Roleに集約されることになる．

Fleetは，実装システムにおけるデータハンドリングを容易とするために設けられた．

C．9．9 根拠の継承と集約

Personは，Personが持つ資格や技能証明などの属性に応じた Characterに castされ，その Characterは特定の Party/Fleetに assignされて所属する Party/Fleetの Roleを引き継ぎ，Character（Participant）の Capacityは特定の actPointにおける・その場の situationに応じた Roleをも集約したうえで決定される．

このような事情から，Capacityはけっして静的ではありえないのである．

C．9．10 権限獲得と付与対象

Characterは特定の場において特定の Capacityを『執る』ことで初めて特定の Actionを actするための権限（privilege）を『獲る』ことができるのである．

言い換えれば権限とは Capacityに基づきつつ Capacityに対して与えられるのである．

権限付与の対象が Person または Participant であると考えるのは完全な誤解もしくは思慮不足であり，「である」故の権限付与ではなく「立場に基づく責務を遂行する」故の権限付与なのである．

C．9．11 試作実装

分担研究報告書（北野）資料を参照願いたい．

C．9．12 HL7 v3 との親和性

HL7 v3 RIM では，Role の scope と play により関係と役割を表現するよう規定されているが，これは輻輳状態を成していると云えよう．そのうえ，Entity-Role の直列化規定も曖昧な感を否めない現状である．したがって，少なくとも現況では，関係と役割の表現は整理しくいように思われる．

３Ｃモデルにおける Capacity は，単純例では，HL7 v3 RIM の Participation に相応可能である．しかし Capacity の本質は，権限根拠の継承と集約にある．したがって，これに真に対応するクラスは，HL7 v3 RIM には存在しない．

これは無理からぬことであって，３Ｃモデルが到達目標としている事項は，HL7 v3 RIM の開発開始の際の目標項目に挙げられていなかったためであろうと思われる．

さらには，Entity-Role と Participation-Act との結合は，双方が完全な整合を保持するよう管理されていることが前提とされている．ただ臨床現場においては，現実的な管理業務の量を超える処理要求・処理期限・組み合せが発生しうるのである．

対応許容量を超えれば，結果として整合齟齬や結合不能状態となる危険を否定できないかと思われる．そしてこれに引き続く事態とは security breach であろう．

C．10 Web アプリと視覚化

神田は以前より独自に種々のシステムを構築して地区または県の歯科医師会活動等の地域医療に貢献しているが，特に Linux ほか open system での DBMS や各種 script の扱い，そして統計や画像の処理を得意としている．

神田は二つの面，すわなち CSX ontological XML Schema を（1）Web アプリケーションを構築する際の諸情報の記述形式としての応用可能性とともに，早くも（2）診療データを二次利用する際の CSX XML Schema の応用可能性について研究した．

なお使用した開発環境は次の通りである：
Linux (Vine-Linux2.6) libxml2 ver 2.6.0, PHP ver 4.3.1, PostgreSQL ver 7.3.2, PXBASE ver 0.6.0, Apache ver 1.3.27, Panorama ver 0.18.01.

C．10．1 可搬性の確認

CSX ontological XML Schema の可搬性の確認は単純で，libxml2 による parsing と xml node structure のツリー構造認識の正当性とを確認するのみである．当然乍ら，これらはいずれも成功した．この基本的なサンプルコードは PHP にて記述されている．【資料１４】

C．10．2 RDBMS への格納

次に CSX XML Schema に基づいた xml を，そのツリー構造を保持したままテーブル変換してデータベースとするために，PostgreSQL のライブラリとして動作する PXBASE を利用した．

PXBASE の変換仕様と生成される schema は比較的単純な構造で，以下の通りである：

- ノード番号 (id int KEY)
- ノードタイプ (type int)
- 名前 (name text)
- 値 (value text)
- 親ノードの番号 (parent int)
- 同じ階層での位置 (pos int)

なお field ノードタイプには，xml での node type に相応しつつ PXBASE が規定する値が格納される．【資料１４】

この単純な mapping でも，xml エレメント同士の親子関係や位置関係などを含めた立体的な構造を RDBMS 内に一意に写像できた．

C．10．3 Web アプリ構築

このような DBMS 環境と Apache および PHP とを組み合わせて Web アプリケーションを試作した．

Usecase サンプルとしては，次項で三次元画像処理を扱うため，歯牙６点法歯肉溝深さの入力フォームとした．なお６点とは頬側近心，頬側中央，頬側遠心，舌側近心，舌側中央，舌側遠心の６箇所のことである．

PHP と Apache で生成されたフォームで入力された診療データは PXBASE により PostgreSQL の中に CSX ontological XML Schema 準拠の xml 形式にて格納されることになる．

ここまでで，診療データの二次利用における CSX XML Schema の応用可能性の検証環境が全て整ったことになる．

C．10．4 三次元画像処理

まず前述システムに蓄積された診療データを CSX XML Schema に準拠した xml ファイルとして抽出する．【資料14】

このデータ xml ファイルは臨床現場にて取得しやすい点情報しか保持していないため，三次元画像処理には適さない．そこで前処理として，Catmull-rom 曲線（Spline 曲線の一種）を算出することとした．【資料14】

また 3D graphics framework を利用して視覚化するには，光源位置（1または2個），カメラ位置，そして三次元方向の rotation に関するパラメータが必要となる．そこで，この諸元も CSX ontological XML Schema にて記述することとした．【資料14】

この様なパラメータ情報の記述も可能であるのは，CSX ontological XML Schema が ontology に即しており，かつ事物の諸属性を極めて抽象化して扱うことのできるよう設計理念を持ち，そしてこれを実現しているからである．

これらのデータファイルとパラメータファイルから，Panorama の処理形式ファイル .rt を生成した．なお歯槽骨の骨吸収を表現するためには円錐状の“ドリル”を用意しておき，Catmull-rom 曲線にて表現された歯肉溝の底の深さまで歯槽骨を“掘る”という手法で .rt を生成した．最終の処理は Panorama からの .png 画像ファイルの出力を得るのみである．【資料14】

この試作システムは，本研究主題である病名/プロブレム変遷や病名-診療行為連関とは趣を異にするものの，幾つかの点を実証したことの成果はそれ自体ですでに大きく，また本研究の発展性を立証するものとして極めて有意義と云えよう：

- POSIX 環境における実装可能性の立証
- 廉価なシステム環境における実装可能性の立証
- 診療情報の二次利用における CSX XML Schema 形式の有用性の立証（視覚化を含む）

C．11 診療行為との連関（preliminary）

研究協力者である矢嶋は以前より独自に歯科医院用のレセコンおよび電子カルテ『カルテメーカー®』を開発販売してきたが，その改訂に際し CSX ontological XML Schema を診療情報などの記述書式として用い，病名/プロブレム変遷や病名-診療行為連関に関わる情報交換を実現する開発を行っている．なお現時点では，α test 版に至っている．【資料15，19】

カルテメーカー® version3 は，DBMS ならびに画面構築に 4th Dimension®を用いている．なお version3 では，Windows 版と Macintosh 版とが用意されることとなった．

画面の構成展開についてはカルテメーカー®の諸元ならびに既存ユーザ等への配慮等の事由から本報告書に記した諸設計に完全に準拠しているわけではないものの，その思想を継承している部分も多い．

いずれにせよ，そのシステム環境において，現時点の CSX ontological XML Schema にて病名-診療行為連関の記述が可能であることを立証した意義は大きい．

ただし，この XML Schema は制約表現については開発中ゆえ，事実のみの記述となっている．また診療行為連関を表現する際に，CSX ontological XML Schema で用いるべき属性値も検討中の段階である．

よってカルテメーカー® version3 α test 版による病名-診療行為連関の記述・表現・実装は，本研究班としては preliminary report として位置づけ，第二年度における研究活動の参考とすることとなる．

C．12 制約表現（preliminary）

本研究を実施開始した時点における CSX ontological XML Schema は，本質的には，事実表現ならびにパタン表現を為すのみであった．これらの記述とくに後者は制約や規則を含んでいる・あるいは・そのものであることがある．

よって事実やパタンの記述と制約の記述とを分離し・かつ・明示的に扱うようにしたなら，加えて，他の記述表現枠組ではなく CSX XML Schema の内部表現つまり自己完結的に記法を整えれば，表現力と明晰性は一挙に向上すると共に，真に，知識表現の能力を獲得することになる．よってこれへの挑戦を開始した．

さて制約表現の範囲は，構造制約（連結制約）もしくは値範囲制約を表現するのみであって，処理規則の記述は含まないこととする．

ただ，前者が後者を示唆することを否定せず，

またそのような示唆が実装処理で活用されることを期待するものである．

C．12．1 制約類型と記述原型

制約の類型

制約に関する表現は多種だが，その類型としては以下の如く要約可能であろう：

・ドメイン親和性
・基数（多重度）
・数値範囲（または属性値限定）
・比較と同値
・取捨自由度
・存否
・順序と共起
・選択
・排他と依存

これらの類型に分類される実際の制約事項は，domain semantics や business rule に即しつつ記述されることになる．なお全ての類型が互いに独立というわけではないがこれについては後述する．

記述形式の原型

次に，これらの制約類型を如何に記述しうるかについて，候補としての原型を挙げる：

・定義体 Substance またはその子エレメントにおけるアトリビュートとその値
・先ず定義体 Substance と制約体 Substance を準備し，次に制約 Relation により両者を連結
・制約 Relation か制約 Relation/Conjugator におけるアトリビュートとその値

Substance は，Relation による構成により構成体を定義できる．したがって Substance は事実やパタンを記述しつつ定義体を構成できるとともに，制約体をも構成できる．

しかし Substance は，自身のみでは自身に関する制約を表現できない．したがって定義体は，制約 Relation による構成によって制約体からの「説明」を受ける，ということになる．

制約体 Substance[@category]は値 Constraint が格納されることを基本とするが，他の値が格納されることもありうる．そのような他の値（例えば NormalRange）から当該 Substance が制約体であることは推測可能なこともあるが，いずれにせよ制約体 Substance は必ず制約 Relation の Conjugator によって参照される．

一方，制約 Relation は多様な Substance を参照しうる．例えば血圧測定では測定方法も制約として表現されるかもしれない．ただ測定体位や測定機器は制約体 Substance というよりも通常の単なる Substance として認識するほうが，Substance の応用性や可用性を損なわないものと思われる．

ちなみに，制約のみを表現するための属性値（追加改変含む）を以下に列挙する．これらは SimpleType に対する facet 定義となっている：

(1) substance.category.Type
・<xs:enumeration value="正常範囲"/>
・<xs:enumeration value="極値"/>
・<xs:enumeration value="制約"/>
・<xs:enumeration value="NormalRange"/>
・<xs:enumeration value="Limitation"/>
・<xs:enumeration value="Constraint"/>
(2) substance.kind.Type
・<xs:enumeration value="制約"/>
・<xs:enumeration value="規則"/>
・<xs:enumeration value="Constraint"/>
・<xs:enumeration value="Rule"/>
(3) relation.category.Type
・<xs:enumeration value="制約"/>
・<xs:enumeration value="Constraint"/>
(4) relation.kind.Type
・<xs:enumeration value="制約"/>
・<xs:enumeration value="Constraint"/>
(5) conjugator.kind.Type
・<xs:enumeration value="制約"/>
・<xs:enumeration value="Constraint"/>

なお上記は未だ過不足について正規化されていない．

C．12．2 表現可能性

前項の列挙事項等に基づきながら個々の類型の表現可能性を検討する．

ドメイン親和性

ドメイン親和性は，個々の Substance が属する domain/subdomain に応じた Substance 間にて連結親和性（または拒否性）を規定する制約である．

domain/subdomain に関しては，「B．1」「C．6．2」「C．7．2」を参照．

定義と記述特性とにより domain/subdomain は Substance[@category and @kind]で規定される．したがってドメイン親和性は，「その」ようなアトリビュートを有する抽象的かつ包括的な

クラスを表すSubstance間における結合制約を表現するような形式で記述されねばならないことになる．

よってスーパークラスの概念が必要となるが，これは既に導入済みである：一つには構造的に（1）Substanceが重層できること，いま一つに属性値では（2）Substance[@category and @kind]に格納される値のcode schemaによって表現できること．

したがって，ドメイン親和性に関する制約を記述する際に為しておくべきことは，ドメイン自体を表すスーパークラスSubstanceに対してSubstance[@object.ID]と Substance[@oid]を与え，そのような Substance 間における制約Relationを記述し，かつこれを共通基盤として用いることのできる枠組みであることを検証するのみである．

基数（多重度）

基数（多重度）は，あるSubstanceがその配下に配置可能な別の同種Substanceの個数を規定する類型である．

基数は制約 Relation に埋め込む．記述箇所はConjugatorとし，ここに新たなアトリビュートmultiplicityを設けることとする．値の記法はUMLのmultiplicity記法と同様とする．

参照される Substance（instance）が同一種であっても個々の諸属性は異なることがあり，それは個々のSubstance（instance）のDimensionによって表現されることになる．

数値範囲または属性値限定

数値範囲または属性値限定は，極値や正常範囲などの制約類型である．

これを表現するには，まず制約すべきSubstanceごとに，極値を表現する制約体と正常範囲を表す制約体とを用意し，然る後に定義体と制約体とを制約Relationにて連結する．

なおSubstanceには複数個のDimension設置が可能であり，かつDimension[@equivalent]には，例えば "SmallerThan" などの比較叙述詞等も用意されているので，難なく記述できる．

比較と同値

比較は，制約 Relation において相異なる二つのSubstance間の物理量または概念量の比較を表現する類型である．

これは Conjugator/Topology を活用して表現される．Topology/Orientation はアトリビュート direction と coordinate とを有し，かつ比較に必要なアトリビュート値は用意されているので，この記述は容易であろう．

取捨自由度

取捨自由度は独立した類型ではなく，存否，順序と共起，依存と排他の三類型と関わる．よって，これらの類型が表現されるべきXML Schemaの箇所に，この取捨自由度を表現する記述形式を埋め込むべきであろう．【記述箇所1】

存否

存否は，制約 Relation における・ある参照Substance の存否または要不要を規定する類型である．

したがって，Conjugatorの配下に記述されねばならない．また，取捨自由度と連携するように記述されるべきである．【記述箇所2】

なお留意すべきは，制約表現における存否とはSubstance[@existence]とは全く概念が異なることである．

順序と共起

順序は，制約 Relation における Substance の連について，各元の出現順位を規定する類型である．

これは Conjugator/Topology[@path]にて簡単に表現できる．なお取捨自由度と連携して記述されるべきである．【記述箇所3】

共起は，制約Relationにおける複数Substanceの出現同時性を規定する類型である．

これは Conjugator/Topology[@path]の値を一にすることで簡単に表現できる．なお取捨自由度と連携して記述されるべきである．【記述箇所3】

選択

選択は，複数Substance間の共起または順序の制約を規定する類型である．

可被選択Substance数が二個の場合には排他や依存と同値と扱いうるが，可被選択 Substance数が二個を超える場合には，これらとは同値とならない．

しかし，共起または順序と取捨自由度との組み合せと同値となる．したがって，この組み合せ手法で表現することとする．【記述箇所3】

依存

依存は,Substanceの存在条件が別のSubstanceの存在を前提とする制約類型である．このとき，別のSubstanceの存在は，物理的または理念的に前方・共起・後方のいずれの場合もありうる．

この点を考慮しつつ，まず依存に関する種別を挙げると以下の如くである：

・多段依存
・共起元依存
・複数元依存
・多重元依存

この四種別について，四つ事物要素，A，B，C，Dが存在する場合を用いて例示概説する．

多段依存とは，BがAに依存し，CがBに依存し，DがCに依存する場合である．

共起元依存とは，A，B，C，Dが共起する際に，B，C，DがAに依存する場合である．これは，前述した共起とは相異なっている．というのも共起は，依存関係について何ら明示しないゆえ各事物要素間で互いに独立か従属かについて不明瞭である．よって共起は，事実やパタンの表現には充分だが，厳密な制約関係を述べる際には共起元依存を用いるべきである．

複数元依存とは，A，B，C，Dの共起に関わらず，CやDが，AとBもしくはAとBとDとに依存する場合である．よって共起より広範となる．なお共起元依存と同様に，これは前述した順序とは異なっている．なお，被依存元の多重性は前提とせず，逆に暗黙に被依存元の相互独立を前提としている点で次の多重元依存と異なる．

多重元依存とは，A，B，C，Dが共起しない際，DがA-B-Cという連なりに依存する場合である．これは多段依存と複数元依存との組合せにて表現可能のように見える．しかし乍ら，それは厳密性を欠いたうえでの流用に過ぎないのである．というのも多段依存は二者間依存を示すのみであって，連への依存には言及しないからである．

次にCSX ontological XML Schemaの諸特性を考慮しつつ，依存種別の記述形式に関して方向性を定めることとする．

まず記法の選択肢を挙げて，各手法の問題点を述べる．各問題点は主に，複数の元に依存する場合に生じうる（独立/従属であれ連であれ）．一番と二番の二手法は，種々の依存形式を単一制約Relationにおいて簡潔に表現するための候補として考案した．三番と四番の二手法は，単一制約Relationにおける完全表現を諦めた記法である：

・Conjugator[@object.refID]で多重参照する
・依存Substanceを新設する．
・別のRelationで扱う
・被参照複数元を括るSubstanceを生成する

一番の手法を採ると，制約Relation配下でのConjugator[@object.refID]で多重ID参照が発生することになる．そのうえ，object.refID自体が別の属性を取る必要が生じる可能性がある．もしそうなった場合，object.refIDはもはやエレメントConjugatorのアトリビュートとは成りえず，Substance/Substance.codeの如く，エレメントとして扱わざるをえなくなる．

二番の手法を採ると，事物要素SubstanceにObject.refIDを格納する情報容器を新設する必要がある．これ自体の改変は容易であるが，Substance内部にRelationを含むことを意味し，semanticsにおいてcontaminationを生じせしめることとなる．

三番の手法を採ると，もとの制約Relationの内部における依存関係が不明確化してしまう危険性を否めない．また，もしRelation間のRelationを記述する必要が生じたならば，CSX Ontology modelについて根底から再考せねばならなくなるだろう．

四番の手法は，これ迄の研究経過と同様の手法である．この場合にも最終的な制約Relationにおける依存関係はヒトには認識しづらいが，CSX Ontology modelおよびCSX Ontology XML Schema modelのsyntaxは，現状のまま使用可能であり，かつ，semantics上のcontaminationを生じせしめることも無い．

上記の事情から，今後，四番の手法に拠って各記法を考案していくこととする．なおその際，一番の手法の併用が必須であれば，これを組み込むこととする．【記述箇所4】

排他

排他は，複数Substance間の相反的な存在禁止制約を規定する類型である．依存と同様，存在検査されるSubstanceは，物理的もしくは理念的に，前方・共起・後方のいずれの場合もありうる．なお留意点は依存と同様である：

・多段排他

・共起元排他
・複数元排他
・多重元排他

したがって CSX ontological XML Schema での諸排他形式の記法も依存と同様となる．【記述箇所4】

C．12．3 輻輳と隔離と

依存と排他には深い考察を要した．

多重グラフ構造を成す対象世界における依存と排他に関する制約表現の困難さは，arc での有向性有無に関わらず，（1）node での多入力・多出力すなわち輻輳と，（2）隔離 node 間における依存と排他の管理とに拠る．

この事情は，一群の nodes を一つの部分的な界（domain/subdomain）に整理したうえで nodes 関係を制約したとしても，同様である．

したがって CSX ontological XML Schema が，敢えて XML の木構造を十二分に活用しない設計，つまり『形式的な単段記法』あるいは Substance を介した Relation の多層性を要求する記述構造，に起因するのではない．

むしろ逆に，記述形式に“究極的”な木構造を採れば，serialization の際には，諸所に分散した事物要素の同一性を徹底管理しなければならなくなる．このとき，親要素と子要素とを持つ Substance の参照は，node 文脈上，限定の範囲が不明確となってしまうことがある．

したがって，対象世界がグラフ構造であることを意識して XML syntax に基づく木構造に依存し過ぎることを避けた戦略は，妥当であったと云えよう．

C．12．4 Archetype

Archetype

Archetype とは，ある Construct を構築する際の原型を意味する．情報システムの場合には，システム設計とシステム構築に資する根源的 entity と，その entity に関わる種々の制約を云う．

制約表現力を獲得した CSX ontological XML Schema は Archetype を記述することができる．

Two model methodology

Information model や reference information model はシステムアーキテクチュアを設計する際に極めて有用であるものの，その domain や realm における大概を説明するのみである．

したがって個々の entity および制約属性まで言及するのは困難か，あるいは RIM 内に細かなデータ構造と制約規則とを同時に持つことになる．後者の場合，RIM はさらに巨大化するとともに情報モデルと知識表現との分離が不十分となるゆえ，維持性が低下することになる．

そこで archetype を活用して IM や RIM およびアーキテクチュアを設計する手法である two model methodology が提唱されることとなった [Beale; DeepThought, Ocean Informatics, OpenEHR].

Archetype Definition Language

Archetype の活用により domain knowledge や business rule は RIM から分離され，双方とも柔軟かつ妥当な維持性を獲得することになる．

そこで ADL (Archetype Definition Language) が提唱されているが，これは XML を敢えて使用していない．その事由は，XML syntax による記述制約の呪縛から逃れるため，としている [Beale; DeepThought, Ocean Informatics].

XML も XML schema も，それらが根源的に持たざるをえない syntax に拠る記述制約は小さくないものの，Beale が参照した枠組みは RDF (Resource Description Framework)，OWL (Web Ontology Language)，OCL (Object Constraint Language) であって，CSX ontological XML Schema ではない．

CSX ontological XML Schema での制約表現

CSX ontological XML Schema は「C．12．3」で挙げたごとく『形式的な単段記法』による利点を有しており，また XML の可搬性を活用することができる．

よって研究者は CSX ontological XML Schema による制約表現を試み，これによる規則や知識の表現を企図したのである．

本年度の成果は preliminary であるものの既に設計指針は獲得したものと思われるゆえ，この成果は意義深いものと思われる．

C．12．5 preliminary である事由

記述用例の蓄積による検証に基づいた必要な XML Schema 改訂に関する検討が未然だからである．属性値を SimpleType facet として規定する xsd：fcet.atst.csx.xsd 外の改訂は僅少であろう．

ただし「C．１２．２」において【記述箇所#】と記した部分の表現形式および記述箇所には考慮を要する．

記法は可及的に統一する方針だが，事実やパタンの記述と制約の記述とでは，若干の差を許容することはありうる．この場合，framework を示すアトリビュートも設けることがありうる．

なお，現状の xsd を変更せずに表現しうる制約類型（基数，値範囲，比較）について試作した archetype サンプルならびに instance サンプルは，資料【資料１６】に掲げてある．

D．考察

D．１　応用可能性

本年度の成果とその発展系から得られる応用可能性を簡潔に述べていく．

D．１．１　病名/プロブレム変遷

厚生科学研究（H12-医療-009）の分担研究成果ほか研究者の研究成果と実装実績を発展応用させつつ本研究課題を遂行したところ，病名/プロブレム変遷の記述は勿論，これを具現する試作実装にも成功した．したがって変遷記述に関する CSX ontological XML Schema の有用性検証は完遂した．

今日，病名/プロブレム変遷に関する汎用的な記述モデルおよび実装による検証は他に無いなか，本研究成果は貴重である．

D．１．２　GraphPane と論理層

上記遂行のために CSX ontological XML Schema を扱うアプリケーションアーキテクチュアの一般モデルを導出した．これは CSX XML Schema に限らず，ontology を下敷きとした XML または XML Schema に基づく xml の応用にも参考になると思われる．

加えて，GraphPane は ontology に基づくデータ構造の GUI への描出のみならず，ほかの様々な用途にも応用しうる．

二十一世紀は知の時代と云われるなか，今後の情報システムは単に業務遂行できれば良いというものではなく，さらに事物要素の蓄積ではなく，むしろ関係要素の集積こそ重要である．このような状況に鑑みても，GraphPane の如きツールの応用可能性は広いと云えよう．

D．１．３　役柄-配役-立場モデル

３Ｃ概念モデルは，次に挙げる優位性を有しうる：PKI/PMI への対応可能性，場の記述形成能力，場における立場の表現，権限根拠明確化能力，場の状況の反映能力，現実的管理コスト可能性，現実世界再現性，現実世界再現即応性，ブリーチ回避性．

これらの優位性を獲得可能なのは security と confidentiality の観点，あるいは権限根拠を生成獲得する要素という観点から，事物要素を細分離したうえで再統合を試みたからだろう．

そして獲得したドグマは次の如くである：権限管理とは権限根拠の管理のことである．これを逆に云えば，楽観的で単純かつ自動的な従来の権限付与は・たとえ如何ほどに強固に認証したとしても権限管理とは云い難い．

このドグマは極めて重要であるにも関わらず，看過もしくは無視され続けてきた．その事由は幾つかあろうが，今後はこの点に留意しつつ，privacy と confidentiality とを保持する情報システム機構を考案していく必要があろう．

したがって応用可能性は広汎であり，実実装における権限管理機構の設計は勿論のこと，CP/CPS（Certificate Policy / Certificate Policy Statement）における権限管理面ほか，各種セキュリティポリシの考案に活用可能な考え方である．

なお，実装技術については RuleBase の採用が妥当と思われる．ただし委譲や移譲については token などの他のメカニズムも必要となろう．

D．１．４　パタンと単純規則

CSX ontological XML Schema は，本質的には，事実表現ならびにパタン表現を為すのみだが，後者は制約や規則をも含むことがある．しかもこの XML Schema は物理量のみならず概念量も扱え・かつ・絶対関係のみならず相対関係をも記述できる．

よってパタンのみで表現できる単純規則等を記述する際にも有用有効である：

・DPC（包括評価）．など

D．１．５　制約と知識表現

とはいえ必要最小限の妥当な制約記述能力を付与すれば，その用途は拡張され知識表現にも応用可能となる．

この点に関する医療分野での応用可能性には，archetype記述のほか，以下が挙げられよう：

・制約的な病名構築
・パスの表現と管理

前者については詳説を要しないだろう．後者については，その表現に関しては，容易に頷じうるだろう．

さて，後者の管理に関する特長についてはCSX ontological XML Schema自体が「分離と構成」能力を有していることにも依存している．なぜなら，意味ある一塊として分離されつつ資源として用意されているパックやスレッドを，ある制約のもとに統合してパスを構築・または・再構築することは，臨床現場におけるパス管理上，極めて有効だからである．

パック/スレッドについては「B．6．2」文献を参照のこと．

D．2 主主題における課題

D．2．1 複数コード体系

事物要素は，複数のコード体系から異なる視点によってエンコードされることがある．これに対応すべくxsdのmaxOccursを改訂する．

D．2．2 制約表現への対応

現時点のxsdは制約に関する表現力は不完全である．よって「C．12」の成果を踏まえつつこれを付与するようxsdを改訂する．

D．2．2 マスクとエイリアス

本年度に発表した研究成果報告において，既に論理設計まで終えたことを示している．これは即座に試作実装可能であるがbudgetとhuman resourceの制約から果たしえなかった．よって次年度に双方の機会が得られたならば，これを実装する予定である．ただ，本研究の主主題においては非本質的ゆえ，試作実装は今後も割愛することもありうる．

D．2．3 ゴールとエンドポイント

ゴールとエンドポイントの記述形式の定式化および症状兆候のそれは，いずれも重要であり本研究主題の延長線上にあるものの，厳密には本研究主題からは若干外れる事項ではある．

しかし乍ら，分担研究報告書にも示されている通りの重要性とともに，EBM推進にも寄与可能であることに鑑み，次年度は，可能な範囲内で達成努力する予定である．

D．2．4 病名/診療行為連関

本年度に発表した研究成果報告において，既に論理設計まで終えたことを示している．これを現状のままで試作実装の段階へ移るのも可能ではあるが「D．2．5」について暫く考察を続けたいと考えたので未了として扱った．

D．2．5 絶対時と絶対順の扱い

まずは比喩から始めよう：聖書には初めに言葉ありき，と記され，次に云々と記されている．してみると“既に”神は存在していたのである．然るに，その神さえ“時”を超えることは敵わなかったようである．というのも“初めに”が存在してはじめて“次に”が存在しえたように見受けられ，そのうえ言葉を発した主体は，神以外に想定しえないからである．これは聖書における最初の真理の提示なのかもしれない．

CSX ontological XML Schemaは事物要素であるSubstanceの子エレメントdimensionに時刻を保持することができる．また関係要素Relationは配下エレメントConjugator/Topologyにて，相対時刻関係を表現することができる．よってシステム時刻も，システム内イベント発生順も保持することが可能である．

然るに上述した比喩に留意するとき，これらを個々のinstanceを表現するSubstance内部の属性として保持させることに，何かしらの心地悪さを感じているところである．よってこれを今暫く考察する．

なお現時点の方向性としては，supervisorでの管理とともにSubstance（instance）にも併記する可能性が高い．

D．2．6 人の扱い

今年度は3Cモデルの定式化が未完なために試作実装におけるヒトの扱いは簡略化されたままとなっている．次年度においては副主題の進捗に応じつつ実装する予定としている．

D．3 副主題における課題

本年度は，Character-Cast-Capacity modelの概念モデルを構築しprimitiveな論理モデルの端諸に至った点である．これをより詳細な論理モデルへと洗練させることが，次年度の副題である．

紙幅の関係上，以下には，特に困難と思われる

二点と，本研究課題として具現化したい一点を挙げるに留めることとする．なお困難な二点は相互に関連する事項である．

D．3．1 場と立場

役割の継承では，継承における参照元が静的である場合には解決の目処がたちやすい．しかしこれが動的な場合には，実装システムでの参照コストへの配慮は無論のこと，参照元の特定（または限界設定）さえ容易ではない．

そして場における立場とは，少なくとも三要素から成り立っている：

・静的な組織とその役割などの継承
・場における動的な状況
・多元的かつ重層的な文脈における背景

前二者は，これまでの研究成果や実装実績等を普遍化する努力によって，それなりの範囲内でならば解を得られる可能性が高かろう．しかし後者は明らかに類をみない新たな挑戦である．

D．3．2 文脈からの影響

上述の如き挑戦において短期間のうちに成果を得るためには，おそらく，限界設定のための端諸または尺度の導入が必要であろう．

ただ，たとえ端諸または尺度を得たとしても，合理的かつ簡潔なモデルまで昇華させる道程は，けっして近くはないように思える．よって深入りを避けることがありうる．

D．3．3 Tokenの扱い

本研究主題の範囲内における最低限度の権限管理とは，従来の静的権限管理機構にアドオン可能な委譲管理機構を備えること，であろう．

そしてこれはtokenの授受による実現が容易に思われる．すなわち幾種類かのtokenを用意し，その発行と授受とによって権限委譲し，それを根拠として“システム内診療行為”，すなわち診療システム内における各種編集や実施等の権限を与える枠組みである．

E．結論

E．1 EBMと監査

本研究の成果は汎用的であり，理論に根ざした合理的な診療情報システム設計から，緊急性のある課題の解決まで，広い射程を有している．後者について若干を記す：

EBM研究は診療品質の維持と向上に必須だが，臨床現場との間で“ベクトル”が乖離したまま解析が進められたりデータ収集が続けられたりする危険性を否めない．また，一次データの品質はEBM研究の品質を大きく左右することになる．本研究成果とその方向性は価値ある臨床研究への重要な示唆を与えるものである．

我が国の診療情報システムは，システム内での“診療行為”の監査を想定しないものが大半である．本研究成果はAudit Trail（監査記録・証跡）機能の実装を目標とするシステム設計に資するものである．

この点は診療品質にもプライバシー保護にも共通している．これらは全て，「原因や事由に基づいた行為」の連続における場や成果を記録するための情報モデル構築を目標としたことに根ざしている．

E．2 臨床教育と知識表現

論拠性ある行為の経過を記録記述することは，まさに，経験と知識とを蓄積することと同値である．

これは診療品質の監査ばかりでなく，熟練者と初心者との比較を機械処理する可能性を拓くものであり，臨床教育に資するところ大である．

また診療記録形式と知識表現形式を統一的に扱える枠組は知の時代に有用かつ重要である．

E．3 進捗と課題

資料【資料２０】の通りである．

今後の研究は，本研究課題の達成は勿論のこと，「E．1」と「E．2」を強く意識しながら，更なる発展をさせていきたい．

F. 健康危険情報

ない.

G. 研究発表

1) 廣瀬康行. Ontology的分析により構築した記述モデルによる病名やプロブレムの変遷の表現可能性. 医療情報学. 23S：962-966, 2003（2003年11月）
2) 廣瀬康行. 関係者と組織との諸関係を記す 役柄-配役-立場モデル. 医療情報学. 23S：504-507, 2003（2003年11月）
3) 矢嶋研一, 廣瀬康行, 森本徳明, 佐々木好幸, 成澤英明, 尾藤茂. 診療履歴情報とプロブレムの ontology 的リンクモデルと電子カルテシステムへの適用例. 医療情報学. 23S：800-801, 2003（2003年11月）
4) Yasyuki Hirose. Tiny and Compact Meta Meta-information Model. MEDINFO 2004. in printing（2004年9月での発表決定）

H. 知的財産権の出願登録状況

現時点では，ない.

以上

A．fcet.elct.csx.xsd : elements and complex types

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    targetNamespace="http://www.hosp.u-ryukyu.ac.jp/medi/csx/0.91"
    xmlns:csx="http://www.hosp.u-ryukyu.ac.jp/medi/csx/0.91"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    finalDefault="#all"
    version="0.91">
<xs:include schemaLocation="fcet.atst.csx.xsd"/>

    <xs:complexType name="substance.Type">
        <xs:sequence>
            <xs:element ref="csx:substance.code"          minOccurs="1"/>
            <xs:element ref="csx:substance.Construe"      minOccurs="0"/>
            <xs:element ref="csx:dimension"               minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="csx:comment"                 minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="object.ID"                    type="xs:ID"                        use="required"/>
        <xs:attribute name="occurrence"                   type="csx:occurrence.Type"          use="optional"/>
        <xs:attribute name="existence"                    type="csx:existence.Type"           use="optional"/>
        <xs:attribute name="oid"                          type="xs:token"                     use="optional"/>
        <xs:attribute name="substance.category"           type="csx:substance.category.Type"  use="required"/>
        <xs:attribute name="substance.kind"               type="csx:substance.kind.Type"      use="optional"/>
    </xs:complexType>

    <xs:complexType name="relation.Type">
        <xs:sequence>
            <xs:element ref="csx:conjugator"          maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="object.ID"                    type="xs:ID"                        use="optional"/>
        <xs:attribute name="occurrence"                   type="csx:occurrence.Type"          use="optional"/>
        <xs:attribute name="relation.category"            type="csx:relation.category.Type"   use="optional"/>
        <xs:attribute name="relation.kind"                type="csx:relation.kind.Type"       use="optional"/>
```

```
    </xs:complexType>

    <xs:complexType name="conjugator.Type">
        <xs:sequence>
            <xs:element ref="csx:topology"                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="object.refID"                     type="xs:IDREF"                                  use="required"/>
        <xs:attribute name="conjugator.category"              type="csx:conjugator.category.Type"              use="optional"/>
        <xs:attribute name="conjugator.kind"                  type="csx:conjugator.kind.Type"                  use="required"/>
    </xs:complexType>

    <xs:element name="substance.code">
        <xs:complexType>
            <xs:attribute name="codeSystem.Name"              type="xs:normalizedString"                       use="optional"/>
            <xs:attribute name="codeSystem.Code"              type="csx:codeSystem.Code.Type"                  use="required"/>
            <xs:attribute name="codeSystem.Version"           type="xs:normalizedString"                       use="required"/>
            <xs:attribute name="code.Name"                    type="xs:normalizedString"                       use="optional"/>
            <xs:attribute name="code"                         type="xs:normalizedString"                       use="required"/>
        </xs:complexType>
    </xs:element>

    <xs:element name="substance.Construe">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:string">
                    <xs:attribute name="nameSpace.Name"       type="xs:normalizedString"                       use="optional"/>
                    <xs:attribute name="nameSpace.Code"       type="csx:nameSpace.Code.Type"                   use="optional"/>
                    <xs:attribute name="nameSpace.Version"    type="xs:normalizedString"                       use="optional"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>

    <xs:element name="comment">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:string">
```

```
                <xs:attribute name="comment.category"      type="csx:comment.category.Type"    use="optional"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="topology">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="csx:orientation"  minOccurs="1"/>
            <xs:element ref="csx:dimension"    minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="path"              type="xs:int"                       use="optional"/>
    </xs:complexType>
</xs:element>

<xs:element name="orientation">
    <xs:complexType>
        <xs:attribute name="direction"         type="csx:direction.Type"       use="required"/>
        <xs:attribute name="coordinate"        type="csx:coordinate.Type"      use="optional"/>
    </xs:complexType>
</xs:element>

<xs:element name="dimension">
    <xs:complexType>
        <xs:attribute name="tude"              type="csx:tude.Type"            use="required"/>
        <xs:attribute name="unit"              type="csx:unit.Type"            use="required"/>
        <xs:attribute name="equivalent"        type="csx:equivalent.Type"      use="optional"/>
        <xs:attribute name="measure"           type="xs:normalizedString"      use="required"/>
    </xs:complexType>
</xs:element>

</xs:schema>
```