```
<csx:dimension
 tude="dateTime"
 unit="anonymous"
 equivalent="Equal"
 measure="2005/03/04 16:29:41" />
   <csx:description uid="PL.0.DS">
     <csx:scope
      domain="Medicine"
      category="診療録.一号様式"
      kind="匿名化.未" />
   ..
   </csx:description>
</csx:infoNode>
```

なお infoNode[@uid]の設計は（C.10）を参照願いたい．そして infoNode[@bearing] には，その塊や文脈を規定する arcScope[@uid] の値が格納される（C.4）の（2）．

この例では ProblemList も DxProblem も，属性 @bearing の値を二つ持っている．というのも，ProblemList を構成する arcScope と，病名変遷を表現する arcScope との双方において，その infoNode の『あり方』を規定するためである．なお区切子は xs:IDREFS に拠って blank である．

次に，病名 infoNode を用意する：

```
<csx:infoNode
 uid="PL.0.PR.0"
 category="Problem"
 kind="Problem"
 bearing="RL.PL.0 TRS.PR.0.1.1">
   ..
</csx:infoNode>

<csx:infoNode
 uid="PL.0.PR.1"
 category="Problem"
 kind="Problem"
 bearing="RL.PL.0 TRS.PR.0.1.2">
   ..
</csx:infoNode>
```

そして，評価や目的や計画を示す infoNode を，必要に応じて，それぞれ用意する．

```
<csx:infoNode
 uid="PL.0.AS"
 category="Problem"
 kind="Assessment"
 bearing="RL.PL.0">
<csx:nodeCode
 CSname="CSX"
 CScode="CSX"
 CSver="0.96.9"
 codeName="問題評価"
 code="問題評価" />
<csx:construe
 NSname="CSX"
 NScode="CSX"
 NSver="0.96.9"> [内容] </csx:construe>
</csx:infoNode>

<csx:infoNode
 uid="PL.0.GL"
 category="Goal"
 kind="Goal"
 bearing="RL.PL.0">
<csx:nodeCode
 CSname="CSX"
 CScode="CSX"
 CSver="0.96.9"
 codeName="治療目的"
 code="治療目的" />
<csx:construe
 NSname="CSX"
 NScode="CSX"
 NSver="0.96.9"> [内容] </csx:construe>
</csx:infoNode>

<csx:infoNode
 uid="PL.0.PN"
 category="Solution"
 kind="Plan"
 bearing="RL.PL.0">
<csx:nodeCode
 CSname="CSX"
 CScode="CSX"
 CSver="0.96.9"
 codeName="治療計画"
 code="治療計画" />
<csx:construe
 NSname="CSX"
 NScode="CSX"
 NSver="0.96.9"> [内容] </csx:construe>
</csx:infoNode>
```

これら infoNode は，@uid="RL.PL.0" を持つ arcScope によって関係付けられることになり，その配下の infoArc は，以下の infoNode[@uid] を持つ infoNode を指し示すことになる（xml 例の一部を記す）：

```
<arcScope   uid="RL.PL.0"
  <infoArc  ref="PL.0"
  <infoArc  ref="PL.0.PR.0"
```

```
<infoArc ref="PL.0.PR.1"
<infoArc ref="PL.0.AS"
<infoArc ref="PL.0.GL"
<infoArc ref="PL.0.PN"
```

C．11．3 人と立場

今期は以下のxml segmentをdescription下に設置する簡便な方策に拠った（C.6.1）（C.13）：

```
<csx:infoNode
 uid="PL.0.PPT"
 category="[Participant]"
 kind="[capacityType]">
 <csx:nodeCode
  CScode="[Directory]"
  CSver="0.9x"
  CSname="PARCEL"
  codeName="第一内科＋感染症グループ＋主治医＋大嶺武史"
  code="[DIRECTORY.ID]" />
</csx:infoNode>
```

C．11．4 検査

検査コードはMEDIS-DC検査コード集と，その規範であるJLAC10に依拠することとした．

以下に定式化における基本設計の概要を示す：

```
infoNode
 @category="Solution"
 @kind="Order.Item"

nodeCode
 @CScode="JLAC10medis"
 @CSver="2003.10"
 @CSname="MEDIS-DC 標準臨床検査マスタ"
 @code="管理番号"
 @codeName="分析物名"
 @priority="1"

nodeCode
 @CScode="JLAC10"
 @CSver="10.??"
 @CSname="臨床検査項目分類コード"
 @code="検査コード"
 @codeName="^分析物名^材料名^識別名^測定法名"

nodeCode
 @CScode="JPReceipt"
 @CSver=""
 @CSname="レセプト電算処理システム用コード"
 @code="診療行為コード"
 @codeName="診療行為名称"
```

さらに，必要ならば様々な属性はdimensionに格納することができる．たとえば：

```
dimension
 @tude="分析物コード"
 @equivalent="Equal"
 @measure="|そのコード|"
 @unit="anonymous"
 @dataType="String"

dimension
 @tude="保険点数（基準）"
 @equivalent="Equal"
 @measure="|その点数|"
 @unit="anonymous"
 @dataType="String"
```

今年度の実装におけるxmlによる直列化の例を以下に掲げる．これは個々の項目を表現するinfoNodeである．本研究では保険点数は必要としないのでnodeCodeは単一としている．

```
<csx:infoNode
 category="Solution"
 kind="Order.Item" />

<csx:nodeCode
 CSname="JLAC"
 CScode="JLAC"
 CSver="10.x"
 codeName="蛋白分画"
 code="3A0200000001232" />
</csx:infoNode>
```

臨床では，幾つかのオーダ項目が一括してエントリされたり参照されたりする．このような一括性はコンテナ infoNode によって表現する．そしてコンテナ infoNode は，上述した個々の項目 infoNode を「包む」ことになる．

```
infoNode
 @category="Solution"
 @kind="[Order.Entry / Order.Refer]"

nodeCode
 @code="[OrderSession.SerialNumber]"

dimension
 @tude="dateTime"
 @unit="anonymous"
 @equivalent="Equal"
 @measure="2005/03/04 16:35"
 @dataType="datetime" />
```

このコンテナ infoNode[@kind]の値で，オーダ

エントリ（Order.Entry）なのかオーダ結果参照（Order.Refer）なのかを判別できるようにしている．

C．11．5 処方

処方薬剤コードは，MEDIS-DC 薬品コード集と，その規範とされているHOTに依拠することとした．なお参照コードはHOT7とした．

定式化の基本設計は，検査とほぼ同様である．まず処方オーダの全体を表現するコンテナinfoNodeを作る：

```
<csx:infoNode
 uid="CTX.1.B.1.BD.CTN.1"
 category="Solution"
 kind="Order.Entry"
 bearing="RL.CTX.1.B.1.BD.CTN.1">

<csx:nodeCode
 CSname="PARCEL"
 CScode="PARCEL"
 CSver="0.9"
 codeName="オーダ.処方"
 code="[OrderSession.SerialNumber]"/>

<csx:dimension
 tude="dateTime"
 unit="anonymous"
 equivalent="Equal"
 measure="2004/01/03 09:12:00"
 dataType="datetime"/>

<csx:description>
 <csx:scope
  domain="Medicine"
  category="処方箋"
  kind="保管"/>
</csx:description>

</csx:infoNode>
```

なお infoNode[@uid]の設計は（C.10）を参照願いたい．そして infoNode[@bearing] には，その塊や文脈を規定する arcScope[@uid] の値が格納される（C.4）の（2）．

次に，対象薬剤を表現する infoNode を作る：

```
<csx:infoNode
 uid="CTX.1.B.1.BD.ITM.1.1"
 category="Solution"
 kind="Prescription">

<csx:nodeCode
 CSname="JPHOT"
 CScode="JPHOT"
 CSver="7.2004"
 codeName="ファモチジン錠２０「サワイ」＋２０ｍｇ１錠＋内"
 code="1147520"/>
</csx:infoNode>
```

用法指示の詳細は，用法指示 infoNode の各dimensionに格納する．

用法指示 infoNode は，本来，arcScope によって薬剤 infoNode と関連させられるべきではあるが，今年度の試作実装においては下記例のように nodoCode[@code]を埋め込む簡便法によった．

```
<csx:infoNode uid="CTX.1.B.1.BD.ITM.1.1.i"
 category="Solution"
 kind="Instruction">
<csx:nodeCode
 CSname="CSX"
 CScode="CSX"
 CSver="0.96.9"
 codeName=""
 code="CTX.1.B.1.BD.ITM.1.1"/>

<csx:dimension
 tude="latitude"
 unit="anonymous"
 equivalent="Equal"
 measure="内服.定時"
 dataType="string"/>

<csx:dimension
 tude="cycle"
 unit="times"
 equivalent="Equal"
 measure="3"
 dataType="string"/>

<csx:dimension
 tude="moment"
 unit="anonymous"
 equivalent="Equal"
 measure="毎食後"
 dataType="string"/>

<csx:dimension
 tude="duration"
 unit="day"
 equivalent="Equal"
 measure="14"
 dataType="int"/>

<csx:dimension
```

```
 tude="amount"
 unit="anonymous"
 equivalent="Equal"
 measure="3"
 dataType="int"/>
</csx:infoNode>
```

ちなみに頓服頓用を指示する dimension では，以下のような例となる：

```
<csx:dimension
 tude="latitude"
 unit="anonymous"
 equivalent="Equal"
 measure="内服.頓服"
 dataType="string"/>

<csx:dimension
 tude="moment"
 unit="anonymous"
 equivalent="Equal"
 measure="疼痛時"
 dataType="string"/>

<csx:dimension
 tude="times"
 unit="times"
 equivalent="Equal"
 measure="6"
 dataType="int"/>
```

そしてこれらの infoNode は arcScope[@uid]が "RL.CTX.1.B.1.BD.CTN.1" のものによって関係付けられることになり，その配下の infoArc は以下の infoNode[@uid]を持つ infoNode を指し示すことになる（xml 例の一部を記す）：

```
<arcScope   uid="RL.CTX.1.B.1.BD.CTN.1"
  <infoArc  ref="CTX.1.B.1.BD.CTN.1"
  <infoArc  ref="CTX.1.B.1.BD.ITM.1.1"
  <infoArc  ref="CTX.1.B.1.BD.ITM.1.2"
```

さらに同様にして，以下の二つのグループが別の arcScope にて関連付けされることになる．

```
CTX.1.B.1.BD.ITM.1.1
    CTX.1.B.1.BD.ITM.1.1.i

CTX.1.B.1.BD.ITM.1.2
    CTX.1.B.1.BD.ITM.1.2.i
```

C．11．6 処置手術

手術処置コードは MEDIS-DC 手術処置コード集に依拠することとした．定式化も検査や処方薬剤と同様である．

```
infoNode
 @category="Solution"
 @kind="[Procedure / Operation]"

nodeCode
 @CScode="ICD9CMmedis"
 @CSver="2004.06"
 @CSname="MEDIS-DC 標準手術処置マスタ"
 @code="[管理番号]"
 @codeName="[マスタ名称]"
```

同様にして，以下のコード体系コードを付加することもできる．複数のコードを併記する際には @priority を用いて，アプリケーションや受信先での処理に資することとする．

```
nodeCode
 @CScode="医科点数表"
 @CSver="2004.04"
 @CSname="医科点数表の解釈"
 @code="[Kコード1]"

nodeCode
 @CScode="JPReceipt"
 @CSver=""
 @CSname="レセプト電算処理システム用コード"
 @code="[レセ電コード]"

nodeCode
 @CScode="ICD9CM"
 @CSver="9.CM"
 @CSname="ICD9CM"
 @code="[当該コード]"
```

C．11．7 外部実体

今年度の試作アプリでは，結果参照機能の例として，簡易な「参照画像の挿入機能」を設けた．ただしこの機能はStand-alone版のみとし，C/S版では binary file はサーバに upload できないよう設定している．

外部実体の参照には infoNode[@rid]を用いる．書式は URI とする．なお今年度の実装では JPEG のみが有効である．

画像参照はオーダ結果参照となるので，infoNode[@kind]の値は Order.Refer となる．

C．11．8 加療過程

診療セッションは，たとえば@uid=" RL.CTX.0" を持つ arcScope によって関係付けられることになり，その配下の infoArc は，以下の infoNode[@uid]を持つ infoNode を指し示すことになる．以下，xml 例の一部を記す：

```
<arcScope uid="RL.CTX.0"
  <infoArc  ref="CTX.0"
  <infoArc  ref="CTX.0.B.0"
  <infoArc  ref="CTX.0.B.1"
```

さらに診療ブロック CTX.0.B.0 は，@uid="RL.CTX.0.B.0" を持つ arcScope により以下の infoNode が纏められる：

```
CTX.0.B.0
    CTX.0.B.0.P
    CTX.0.B.0.HD
    CTX.0.B.0.BD
    CTX.0.B.0.D
```

同様にして段が下がるごとに各々arcScope によって纏められらてていくことになる．

```
CTX.0.B.0.BD
    CTX.0.B.0.BD.ITM.0
    CTX.0.B.0.BD.ITM.1
    CTX.0.B.0.BD.CTN.0
        CTX.0.B.0.BD.CTN.0.ITM.0
        CTX.0.B.0.BD.CTN.0.ITM.1
        CTX.0.B.0.BD.CTN.0.ITM.2
```

なお各々の arcScope@uid は，上記と同様に(C.10) に記したスキーマで附番され，同時に，関係する infoNode の @bearing には，その arcScope@uid の値が，必要なだけ挿入される．

C．11．9 病名変遷

変遷する PL や PR は，@uid に TRS で始まる値を持つ arcScope によって表現される(C.10.2)．以下，xml 例の一部を記す：

```
<arcScope   uid="RL.PL.0"
  <infoArc  ref="PL.0"
  <infoArc  ref="PL.0.PR.0"
  <infoArc  ref="PL.0.PR.1"
  <infoArc  ref="PL.0.PR.2"  ..

<arcScope   uid="RL.PL.1"
  <infoArc  ref="PL.1"
  <infoArc  ref="PL.1.PR.0"
  <infoArc  ref="PL.1.PR.1"
  <infoArc  ref="PL.1.PR.2"  ..

<arcScope   uid="TRS.PL.0.1"
            category="Graph"
            kind="Transition">
  <infoArc  ref="PL.0"
            category="Source"
            kind="Transition.Proceeded">
  <infoArc  ref="PL.1"
            category="Goal"
            kind="Transition.Proceeded">
</csx:arcScope>

<arcScope uid="TRS.PR.0.1.1"
            category="Graph"
            kind="Transition">
  <infoArc  ref="PL.0.PR.0"
            category="Source"
            kind="Transition.Proceeded">
  <infoArc  ref="PL.1.PR.0"
            category="Goal"
            kind="Transition.Proceeded">
</csx:arcScope>

<arcScope   uid="TRS.PR.0.1.2"
            category="Graph"
            kind="Transition">
  <infoArc  ref="PL.0.PR.1"
            category="Source"
            kind="Transition.Proceeded">
  <infoArc  ref="PL.0.PR.2"
            category="Source"
            kind="Transition.Proceeded">
  <infoArc  ref="PL.1.PR.1"
            category="Goal"
            kind="Transition.Converged">
</csx:arcScope>
```

この例は PL.0 から PL.1 に推移し，PL.0.PR.1 と PL.0.PR.2 は PL.1.PR.1 に合流した，ことを表現している．詳細については前年度報告書も参照願いたい．

C．11．10 病名診療行為連関

病名変遷 xml と加療履歴 xml との結合は，@uid に Lk で始まる値を持つ arcScope によって表現される(C.10.2)．以下，xml 例の一部を記す：

```
<csx:arcScope uid="Lk.0.PL.CTX"
              category="Logic"
              kind="Implication">
<csx:infoArc  ref="PL.0"
              category="Reason"
              kind="Cause" />
<csx:infoArc  ref="CTX.0"
              category="Object" />
</csx:arcScope>
```

上記では session 結合が為されている．また arcScope では〔論理；含意〕と意義付けられ，infoScope では〔理由；原因〕と〔対象〕とが示されている．

次に，DxProblem と対応する BlockHeaderItem との同値性が主張されることになる．ここでは arcScope にて〔論理；同値〕と意義付けられ，infoScope には〔源泉〕と〔対象〕とが示されている．

```
<csx:arcScope uid="Lk.0.PR.BH.0.0"
              category="Logic"
              kind="Equivalence">
<csx:infoArc  ref="PL.0.PR.0"
              category="Source" />
<csx:infoArc  ref="CTX.0.B.0.HD.ITM.0"
              category="Target" />
</csx:arcScope>
```

```
<csx:arcScope uid="Lk.0.PR.BH.0.1"
              category="Logic"
              kind="Equivalence">
<csx:infoArc  ref="PL.0.PR.2"
              category="Source" />
<csx:infoArc  ref="CTX.0.B.0.HD.ITM.1"
              category="Target" />
</csx:arcScope>
```

```
<csx:arcScope uid="Lk.0.PR.BH.1.0"
              category="Logic"
              kind="Equivalence">
<csx:infoArc  ref="PL.0.PR.1"
              category="Source" />
<csx:infoArc  ref="CTX.0.B.1.HD.ITM.0"
              category="Target" />
</csx:arcScope>
```

なお連番は，病名変遷 xml と加療履歴 xml との診療セッション番号，加療履歴 xml の診療ブロック番号ならびにその BlockHeaderItem 番号と一致している．

実例については資料を御参照願いたい．

C．12 実装アーキテクチャと基幹クラス

本節の(C.12.1)から(C.12.4)までは前年度の報告書を一部は改変しながら再掲した．というのも，このデザイン(C.12.2)と実装なくしては今年度の参照実装は実現しえず，また再掲なくしては本報告書の理解も困難だからである．

本節の(C.12.5)以降に整理と見直しを要約した後に，今年度の再設計と再構築の方針を述べる．これらは山田が担当し，実装した．

C．12．1 共起表現と結合選択性

CSX model の応用可能性は，単に病名やプロブレム変遷表現のみに限らず，(D.2) や (E) に挙げた重要事項に関わる各種情報を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の制約や許容に関わる表現は必須となるが，Tree pane では，これらを実現できない (B.13.4)．

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，Tree pane においては，合流（convergence）表現も擬似的でしかない．

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 infoArc 数のみならず，「結合の意義とその可否」についても，多様とならざるをえない．よって唯一つの “anchor” 属性によって結合可能性が規定されてしまう画面コントロールは，本質的に機能脆弱と云わざるをえないことになる．

これらの点を解決するために，前年度に Graph pane を開発したので，今年度の参照実装では，これを採用する．

C．12．2 Graph pane

Graph pane は病名/プロブレム変遷のみならず，グラフ構造上にある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た Microsoft Windows .NET Framework 2003 上の画面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，その支配下において管理することとなる．

そして表示項目とは事物要素のみならず，その属性をも扱いうる．また前述した共起性表現や domain や subdomain 間の結合選択性のサポートをも実現可能となっている．

よって Tree pane での anchor (B.13.4) は，もはや存在しえない．替わりに用意すべきは関係線と結合とを表す Nexus と Ligand のみである．ただし此処には実は infoNode の結合選択性を示す Receptor が存在する，とした．

なお Ligand や Nexus は形状や lock や unlock などの諸属性を有しており，programmable あるいは end-user editable 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．

結合と表示

- 表示項目は ViewLabel に格納され表示される．
- ViewLabel は infoNode 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 Ligand 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

C．12．3 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arcScope[@category and @kind]およびinfoArc[@category and @kind]を決定する必要がある．

加えてinfoArc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設計との間の，視点の不整合から生じている．つまり頂点管理か，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arcScopeの存在が必要条件となる．

arcScope ならびに arcScope[@category]とarcScope[@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 infoArc[@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，将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している，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infoNodeの・categoryとkindで決定されるdomainや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．

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arcScopeの管理下に置かれてinfoArcとされる．また，一つのarcScope 内のinfoArcには唯一つの「親」infoArcが存在しうるという整理，換言すれば全ての infoArc[@kind]の決定が為されうるのである．

なおLigand - Nexus - LigandとinfoArcとのmappingを管理するために，AppLogic tierにAppLigamentとAppConjugatorとを用意した．そしてAppConjugatorにはreplicaが置かれ，相応と整合に貢献することとした．

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：

・arcScopeとinfoArc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，arcScope [@category]，arcScope [@kind]，infoArc [@category]は画面製作者の管理下にあって自明であり，infoArc [@kind]は infoNode [@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．12．4 論理層

Graph paneが求めるクラス構造は複雑化ゆえ，論理アーキテクチュアを背景したクラス構築とそれらの配置，そして必要な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．

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

このデザイン・パタンを採用したのは，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．

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．なおBizItemとは，実際のところinfoNodeのCollectionである．

BizIndexとは，多次元空間におけるinfoNode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（本研究ではCSX model）を適用するための課題を担う．今回の実装ではCSX model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 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．

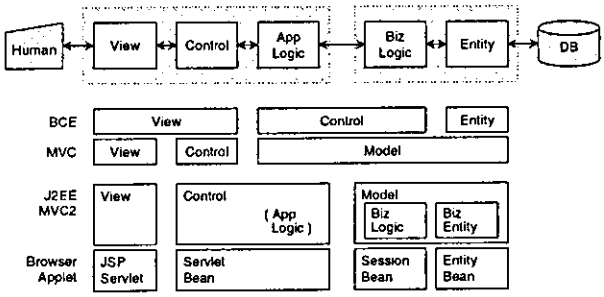
インターフェイスの役割を果たすクラスは，AppItem（とItemAllignMap）とAppLigament，またBizItem，BizRelation，BizIndexである．

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も含むこととなりうるが，LigandやNexusにおいても表現可能である．

C．12．5 前年度モデルの整理

上記の5階層モデルと他のアーキテクチュア・デザイン・パタンとを比較してみる．

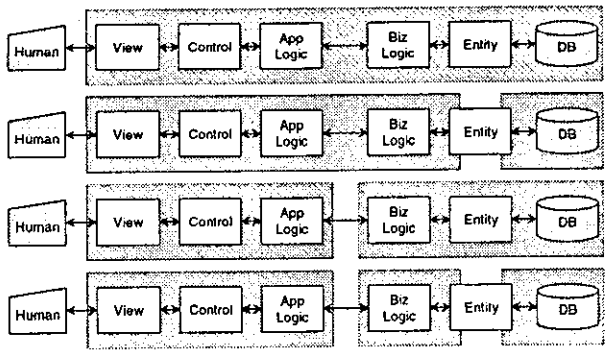


層分割視点は以下の三点に拠っていることが理解できる：

- 想定するdeploymentの差異，もしくはDB serverやApplication serverに関する2階層または3階層モデルに対する認識
- BizLogic tierの重視の程度
- Client applicationの製造コストの意識とAppLogic tierの配置

まず三点目については，我々としては，5階層モデルでのControl tierとAppLogic tierの分離は妥当ではないであろう，と考えている．

そのうえでdeployment候補を挙げると下図のようになる．



上段はStand-alone，他はC/Sモデルであり，うち最下段は3階層アーキテクチュアが前提されている．下の二つはモジュール間protocolの規定を要求することになる．なお最下段は，サーバの分散性が向上することになる．

どのdeploymentを選択するべきかについては実装内容と実装環境に依存するので一概に是非を謂うことはできない．

ところで，今年度の試作アプリケーションは参照実装と位置づけている．したがって前年度の5階層モデルの思想を尊重踏襲し，どのようなデザイン・パタンにも対応可能としておくこととした．ただし前年度の課題を解決することを前提とする条件において，である．

C．12．6 前年度実装の見直し

前年度の実装作業の後に課題として残された事項は以下の四点である：

A) Task制御とInitiator
B) Tier境界を超えるeventの処理
C) MS Windows開発環境の要請（画面ありき）
D) MS Windows開発環境の要請（namespace）

なお（B.10）に記したとおり，開発環境はMS Windows C# .NET Framework 2003である．上記のそれぞれについて順次，要点を述べていくが，これらの問題または解決策は相互に関連している．

A) Task制御 および Initiator

全てのプログラムは起動者を必要としており，また各tierの全Classについても同様でinstanceを作成するにはInitiatorを必要とする．

そしてInitiatorは，プログラムを起動させたときに最初に動作し，かつ，各Tierの全Classをすべてinstance化しておかねばならない．

というのも，tier間に不必要な親子関係を持ち込まない，つまり各tierのClassの独立性を保つため，である．

B) Tier境界を超えるeventの処理

各tierのサービスは，基本的に関数呼出（Method呼出）によって実現される．なおevent処理等でdelegateを使用しているが，これもmethodを登録しているのであって，本質は関数呼出と同等である．

したがってevent受信側やサービス享受側も，必ず，送信側やサービス提供側の情報を持っておく必要がある．しかもこれは，コード記述における単純な形式上の問題ではない．

そこで，各 tier の Class の「まとまり単位」ごとに，Class の Initiator を準備する必要がある．

各 Tier 内の Class は一般に，互いに連繋してサービスを提供している．このような「まとまり」に親に相当する Class がある場合，それに担当させる．親の役割の担う Class が無い場合には，「まとまり」に対する Manager を用意して，それに担当させることとする．

C) MS Windows 開発環境の要請（画面ありき）

Windows アプリケーション開発環境では，プログラムに指示を与えることのできる「画面」が『親』の地位を占めることになる．

一方，本研究の Tier 分割では，View tier のみ画面を持っているが，しかし View tier は Control tier の制御下に存在するべき構成である．

この状況は Windows アプリケーション開発環境の要請とは矛盾する．よって View tier を起動する module が起動する前に，必要な動作環境を全て整えておく必要がある．

D) MS Windows 開発環境の要請（namespace）

Namespace の解決は DLL を経由して為される．つまり Using で定義するのみでは機能せず，その namespace を持った module（を含む DLL）は，先立って compile されている必要がある．

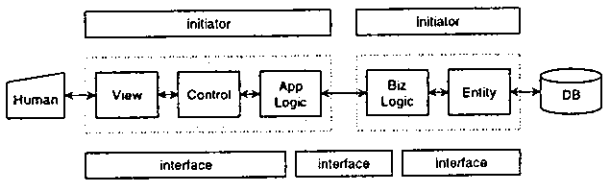
これは『Class 間に親子関係を持たせている』ということであって，各 tier の独立を妨げる要素となってしまう．これを回避するためには，アプリケーションの最下層に Interface を置く必要がある．

すなわち，外部から参照される Class は Interface を継承し，Interface に定義された method や property を持たせることとする．

なお Class 間メッセージ（EventArg）に関わる Class も Class 間で共用となるので，ここに置くこととする．

結論

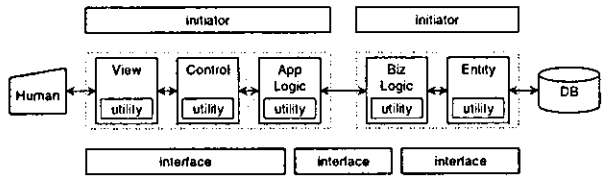
アーキテクチュアは下図の如くした：



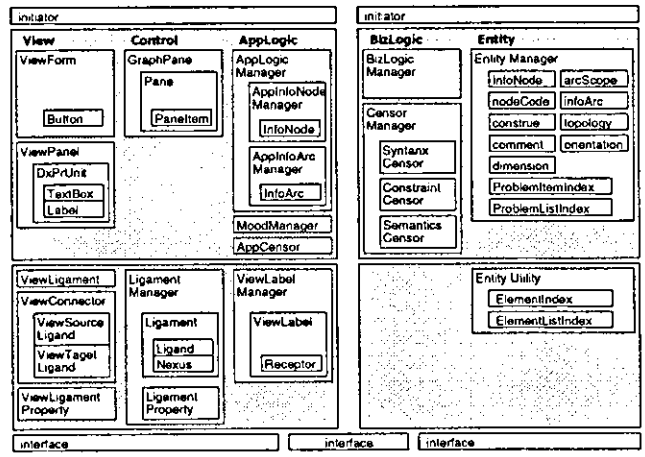
つまり最初に起動されるプログラムが全ての Initiator に対する責任を負い，かつ Interface 構造を導入することで，各 tier の独立性を保持しつつ伝播と連携とを促進することとした．

C．12．7　実装クラスの拡充

さらに各 tier において共用性の高い Class は DLL で供給していく構想を踏まえて，Utility Class として扱うこととした：

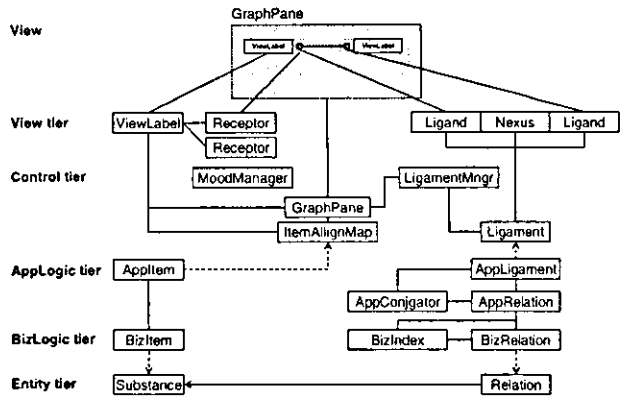


そのうえで (B.13.4)(B.13.5) および (C.6.1) から要請される Class を以下の如く配置した：



今年度の参照実装では，この内部 Class を基にして各アプリケーションを構築していく．

なお下図は再掲であり CSX model の Class 名は旧版のままとなっている．



C．１３　役柄配役立場モデルの適用

C．１３．１　必要性

C.13.1.1　権限根拠の管理

現状のほとんど全ての業務システムは「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，真に権限を「管理」しているなどとは云い難い．

そのようなシステムにセキュリティ強化用のハードウエアやinfrastructureを付加しても，本人確認等が多少強化されるに過ぎず，権限の管理には至ることができないのである．

現実には単一の組織内においてさえ（B.7）に記した状況があって以下のような事象が発生している：

・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
・現場の状況に応じた立場の選択
・権限の委譲と移譲

この複雑な現実に即して適切に権限管理するためには，権限の根拠を明確化する機構が必須となる．換言すれば権限根拠を直接的間接的に生じせしめる様々な情報塊を洗い出し，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 すなわち 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．

C.13.1.2　根拠の固さ と 場

権限管理では，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．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

ところで日常業務においては現場という『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．

短寿命な権限根拠を，システム管理者によって管理することは実施的に不可能である．

したがってParticipant自身の宣言と事後の監査により“behavior”の妥当性を検証し，また厳格な運用規則を適用する手法の援用も必須である．適切な行動を執りうる権限を付与することは，奇妙なアクセス制限に優先する．

C．１３．２　モデル内の要素

3C model自体については（B.13.6）や前年度の報告書などを参照されたい．以下に，その構成要素を挙げる：

・行為者
・役柄（Character）
　・サービス実施者（Participant）
　・サービス消費者（Consumer）
　・親類縁者知人など（kithKin）
・組織単位（Party）
・組織単位縦列（Fleet）
・配役（Cast）
・行為場（actField）
・組織で規定される役割（roleInHosp）
・場において要求される役割（roleToPatient）
・権限根拠（PrivilegeBasis）
・立場（Capacity）
・行為点（actPoint）
・所作（action）
・権限（Privilege）

つまり 3C model を完全に実装するには，先ずこれらの要素を全てクラス化する必要がある．

そのうえで対象業務に要する付帯Classを準備し，さらに業務の利便に考慮した機能を付加していく必要がある．

上記の要素に認証Certificationを加えただけでも，大雑把に見積もっても以下の如くなる．

・actField（actPoint+,Fleet+）
・actPoint（actCharacter+, Location+, Time）
・actCharacter（Character,Capacity）
・Capacity（PrivilegeBasis）
・PrivilegeBasis（License*,Role+,..）
・Role（roleInHosp+,roleToPatient*）
・Directory（Character+,Fleet+,Party+）
・Character（Person,Fleet）
・Fleet（Party+）
・Pearson（License*,Insurance*,..）
・Certification（Person,key,pass）

C．１３．３　交換等に要する拡張

加えて，たとえばCSX modelを交換に用いようとした場合には以下のClassが必要となる．

・Case（actField+ | actPoint+）
・Holder（actPoint+）
・Concerned（actPoint+）

簡約すれば，クラスConcernedは現場における

所作への直接的な関わり，クラス Holder とは文書の作成や認証や発行への関わり，クラス Case とは当該事例に関する関与者等の広がり，を表現することになる．

これだけの量の試作実装を，本研究の主主題と同時に実施することは不可能なので，参照実装では，範囲を絞って成果の一部を適用することとした．

C．13．4 試作アプリでの対象範囲

応用した要素や考え方は以下のみとした：

1) Fleet の構成
2) Fleet への Consumer の登録
3) Fleet への Participant の所属
4) Capacity の宣言
5) Capacity に応じたアクセス可能範囲の制限

とはいえ，これだけでも，

・3C model を適用しても操作負荷は増大しない
・立場に拠って付与される権限が変化しうる

ことの実感による理解には十分と思われる．

C．13．5 実装クラス

実装した Class は以下の通りである：

・certification
・participantDir
・consumerDir
・organizationDir (organization, capacity)
・party (department, group)

いずれも Schema は xsd または well-formed xml にて記されており構造は単純である．とはいえ 3C model に即して，次のような特徴がある：

・certification に登録されている Person は，Participant にも Consumer にもなりうる．
・Party の再帰構成によって Fleet を構築する．当然ながら種別 (organization, department, group) の階層秩序性は保持している．

クラス図の概要は (C.16.5) に記した．

C．13．6 登録ルール

上記ほか，以下の小さなルールを用意した：

・UserA が作った paticipantA の partyID には，UserA の操作により，UserA の作った hospitalA, departmentA, groupA のうち，各々，自身より下位の partyID のみを collect できる．
・UserA の作った consumerA は，UserA の操作により，UserA の作った fleetA に collect できる．
・UserA の作った consumerA は，UserA の操作により，UserB が作った fleetB に collect できる．
・UserB は，fleetB に collect されている consumerA を，その fleetB における consumer collection から削除できる．

このルール実装によって，簡易ではあるものの，Consumer の紹介など地域連携をも模倣できるようになっている．

すなわち，場の形成と権限に管理に関わる情報モデルについて，その一部を実装した．

C．13．7 ログインと記録

参照実装へのログインの際，end-user は以下を尋ねられることになる：

・account
・password
・organization
・department
・group (optional)
・capacity

後三者は participant 氏名と共に，(C.11.3) の如く記録されることになる．

C．14 参照実装の全体構成

実装アーキテクチャと基幹クラスは山田が，病名 Composer は尾藤が，他の C#アプリ等は与那嶺と大嶺が，サーバの構築は山本（Soliton）が実装し，設計等は廣瀬と協同して行った．

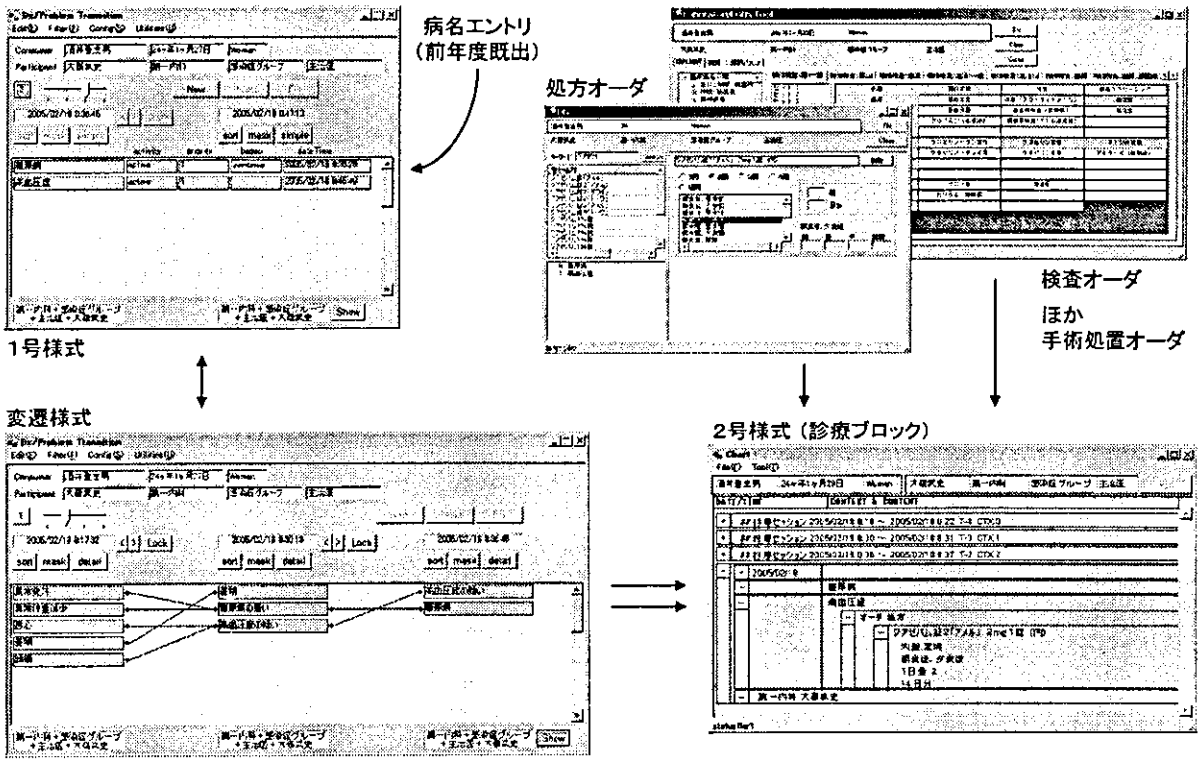
C．14．1 俯瞰

C.14.1.1 診療システム

診療システムは以下のようなイメージであり，またそのような流れで用いる．

1) まず，所属と立場を明らかにしてログインする．

2) 患者一覧から患者を選択すると１号様式画面（Dx/Problem Transition）が展開される．
この画面は，病名やプロブレムの変遷を閲覧編集するためにも用いられる．



3) 必要ならば病名エントリツール（Dx/Problem Composer）で病名を構築して登録する．

4) １号様式画面での編集を終えたなら，１号様式画面を確定する（FIX ボタン）．

5) 診療対象とする病名を２号様式画面（Chart）に送信して２号様式画面に診療ブロックを形成する．

6) 検査オーダツール（LaboTest EntryTool）や処方オーダツール（Prescription EntryTool），あるいは手術処置エントリツール（Procedure/Operation EntryTool）で必要な項目を選択して，２号様式画面の診療ブロックに送信する．

7) 確定すれば（FIX ボタン），病名変遷ならびに病名診療行為連関が確定し，XML document として保存される．

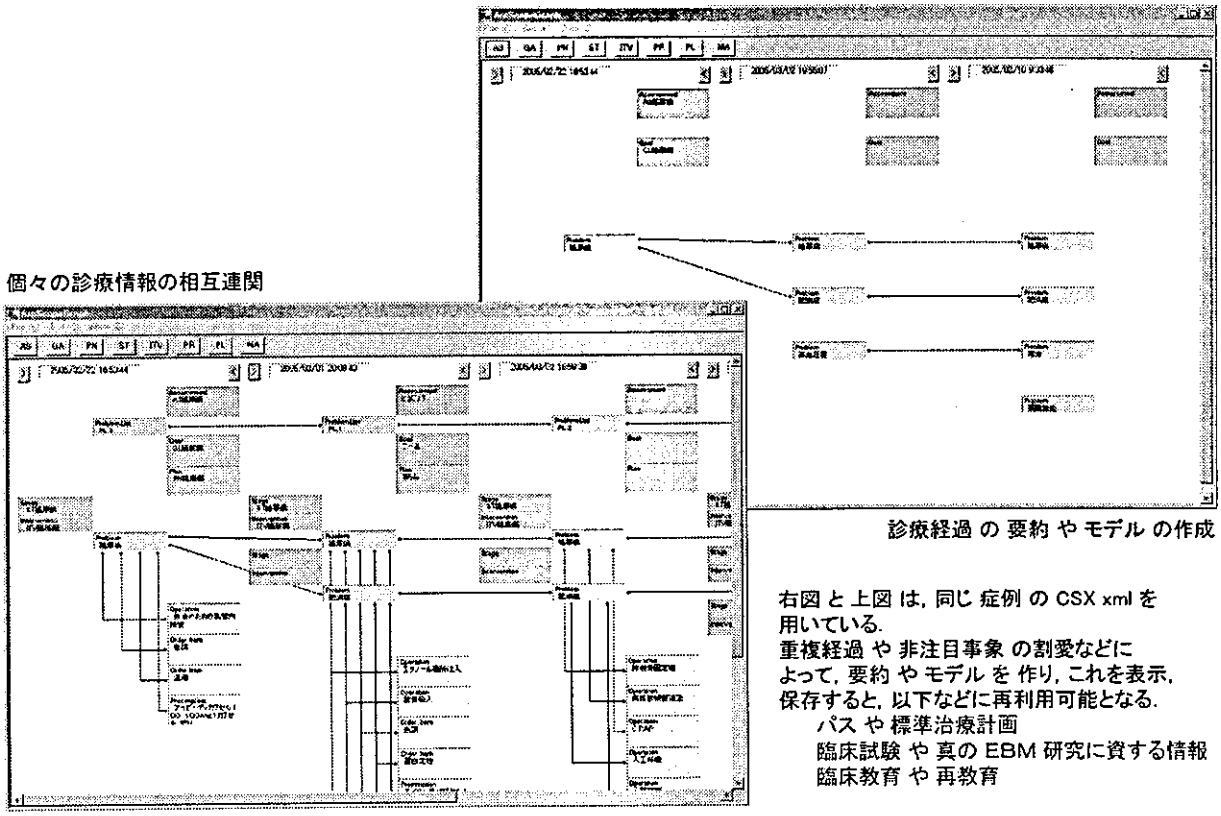
このシステムは Stand-alone 版と C/S 版とを作成した．

C.14.1.2 生成と抽出要約のツール

先の診療システムから出力された診療履歴ファイルを読み込んで，診療過程の要約をビジュアルに作成したり，あるいはCSX model に即した様々な情報塊を作成したりするためのツール TheTOOL である．

なお自然言語を処理するのではなく，意味を担った情報塊 node を探索したり取捨選択したりして，情報塊 node が織り成すグラフ構造に“要約”を施す，つまり臨床医学の観点から重要なノードとそれらの関連のみを抽出するためのツールである．

下図では，左下の前方画面に「診療履歴ファイルを読み込んだ直後の表示例」を，また右上の後方には「抽出と要約後の表示例」を示している．



このシステムはStand-alone版のみを作成した．

C．14．2 構成と機能

C.14.2.1 診療アプリケーション

本研究では以下の参照実装を行った：

Welcome
- ログイン機能
- 所属組織自動絞込機能
- 診療科部選択機能
- 診療グループ選択機能
- 立場選択機能

Patient List
- 選択可能患者一覧機能（自動絞込）
- 患者属性一覧機能
- 所属組織変更機能
- 立場変更機能

選択可能患者再検索機能
患者選択機能
患者情報転送機能（→ Platform）
職員情報転送機能（→ Platform）
アプリ終了機能

Platform
患者情報受信機能
職員情報受信機能
患者属性表示機能
職員属性表示機能
診療セッション中止機能
診療セッション suspend 機能
診療セッション resume 機能（自動処理）
患者情報転送機能（→Transition/Chart）
職員情報転送機能（→Transition/Chart）

Dx/Problem Transition
患者情報受信機能
職員情報受信機能
患者属性表示機能
職員属性表示機能
一号様式病名欄表示機能
新規プロブレムリスト構築機能
病名自動複写機能
変遷自動形成機能
Dx/Pr Composer 呼出機能
構築病名受信機能
病名プロブレム一覧機能
病名プロブレム変遷一覧機能
変遷表示 pane 数切替機能
変遷表示スライド機能（連動）
変遷表示スライド機能（非連動）
変遷表示スライド機能（ロック）
病名属性編集機能
Activity, Priority, Basso
病名属性編集機能
変遷編集機能
変遷属性編集機能
評価入力編集機能
評価表示機能
評価複写機能（隣接表示から）
目標入力編集機能
目標表示機能
目標複写機能（隣接表示から）
計画入力編集機能
計画表示機能
計画複写機能（隣接表示から）
病名プロブレムリスト変遷履歴記録機能
病名転送機能（→ Chart）
診療ブロック選択機能
診療ブロックヘッダ追加機能

Dx/Problem Composer
病名一覧＋選択機能
病名修飾語一覧＋選択機能
病名修飾語検索機能
病名検索機能
病名修飾語検索機能
病名プロブレム構築機能
構築病名転送機能（→ Transition）

Chart
患者情報受信機能
職員情報受信機能
患者属性表示機能
職員属性表示機能
加療行為履歴一覧機能
加療行為履歴アウトライン機能
病名受信機能
診療ブロック形成機能
各 EntryTool 呼出機能
患者情報受信機能（→ Lab/Px）
職員情報受信機能（→ Lab/Px）
検査オーダ項目受信機能
処方オーダ項目受信機能
手術処置エントリ項目受信機能
受信項目の削除機能
外部ファイル登録機能
登録した外部画像ファイルの表示機能
外部ファイル登録の削除機能
加療行為と病名との関連付け機能
加療行為履歴記録機能

LaboTest EntryTool
病名一覧機能（ツリー）
セット表示機能〔選択機能は現状未実装〕
検査項目メニュ表示機能
検査項目選択機能
検査項目検索表示選択機能（3 軸, AND/OR）
選択項目転送機能（→ Chart）
診療ブロック表示機能
メニュ＆セットのユーザ設定機能

Prescription EntryTool
薬品検索＋一覧機能
薬品検索選択機能
服薬指示設定機能
定時：一日量，均等・不均等割付
頓用：一回量

処用時期
処用日数
選択設定項目転送機能（→ Chart）
診療ブロック表示機能
診療ブロック選択機能

Procedure/Operation EntryTool
処置手術項目一覧機能（K/J, ICD9-CM）
処置手術項目検索機能
処置手術項目選択機能
頻用項目ユーザ設定機能

本報告書（C.9）から窺い知れる様に，実装の難易度や参照実装の重要性は，画面数や機能数といった単純な計量値から推し量れるものではない．

むしろ（C.4）の如き極めて抽象度が高い情報モデルを用い，そのうえ高度な技術を要求する(C.9)の如き構成を実現した，そのこと自体が重要であり意義深い．

とはいえ本研究における試作診療システムの主要画面は9，主要機能は106を数え，一般的な外来診療を模倣できる環境を提供している．短期の開発期間と限られた予算で此処までを達成できたことは，システム構築においても，何かしら示唆するものが含まれているように思われる．

なお本システムは試作ゆえに，市販電子診療録システムに付随する細かな機能等は割愛しており，本研究主題に即した機能となっている．

C.14.2.2 生成と抽出要約のツール

前述とは別に特殊なツールも開発した．これは，元々は，CSX model に基づいた instance を xml にて直列化しつつ生成・保存・編集するためのツールであった．加えて，グラフ構造の視覚化機能も付加していた．

そこで，この機能を流用しつつ，前述した診療システムから出力された診療履歴 xml ファイルから，診療過程の要約を表示・閲覧・保存する機能を持つツールとした．

以下にその機能を掲げる：

TheTOOL
CSX Ontological XML に基づく
XML instance の読込機能と保存機能

XmlInstance
XML instance の表示機能

EDIT
root element の諸属性の表示
infoNode の生成と削除
infoNode の諸属性の生成編集削除
infoArc の生成と削除
infoArc の諸属性の生成編集削除
arcScope の生成と削除
arcScope の諸属性の生成編集削除
topology の生成と削除
topology の諸属性の生成編集削除
orientation の生成と削除
orientation の諸属性の生成編集削除
dimension の生成と削除
dimension の諸属性の生成編集削除
上記編集削除中の相互連関

InfoNodeGraph
infoNode の子要素の状況表示
infoNode の子要素のグラフ表示

ArcScopeGraph
病名変遷と診療行為連関のグラフ表示
診療セッション単位の送り（順逆）
変遷関係線の再描画
不要 node の非表示・再表示
必要 node の表示ロック
不要 node の非表示ロック
node 探索の設定
包絡する node 種別
探索方向（時間的に順・逆）
隣接する infoNode の属性に
依存した node 探索停止条件
node 探索開始 node の設定
node 探索
その編集結果の表示と保存

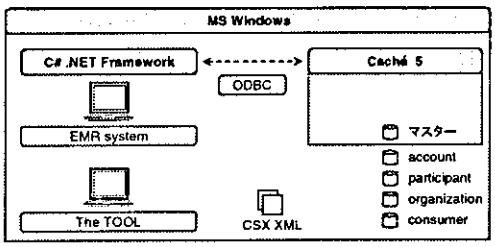
このツールは，本研究班のなかでは TheTOOL と読んでいる．

TheTOOL は，臨床にも臨床研究にも臨床教育においても，極めて有用なツールとなりえよう．

C．15 参照実装（Stand-alone版）

C．15．1 システム構成

方法（B.10）に記した開発環境で下図のように構成した．



Stand-alone版はコード・マスタのみCachéに格納している．その他の各ファイルを配置したディレクトリ構成は以下の通りである（左手がディレクトリ名，右手がファイル名）：

crt	certification.xml
csm	consumerDir.xml
stf	participantDir.xml
org	organization.xml
dat	PT####_PrDx.xml PT####_PrMa.xml PT####_PMLk.xml
ext	PT####_########_##.jpg

各ファイル名は（C.13.5）および（C.16.5）で提示したClassと相応している．

敢えて各ディレクトリに分割配置した理由は，今後にC/S版とした際にも，整理と見通しとを良くしておくためである．

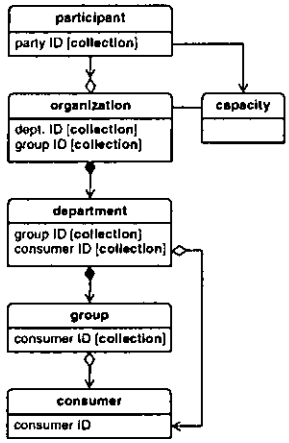
つまり各ファイルは，それぞれ別個のサーバ，もしくはpartitionに管理される可能性が高く，同時に，それが妥当な管理であろうと思われるからである．

C．15．2 視野範囲の限定

C.15.2.1 Cascading Focusing

ログインにおけるFleetと患者の選択の際に，役柄配役立場モデルを適用しながら，順送りに視野範囲を制御するようにした．

すなわち（C.16.5）に示した各ClassへのID collectionの持たせ方は，集約関係を重視するのではなくて，業務フローにおける参照順序を優先する，という手法で解決している．



C.15.2.2 StructCore

MEDIS-DC病名コード集と手術処置コード集は，その拠り所の一つとしてICD10，あるいはICD9-CMや「点数表の解釈」という分類体系を持っている．

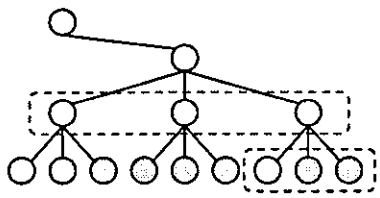
そのようなコード・マスタは，それが依拠する体系の一軸を優先して各コードをツリー表示させるなら，一覧性を保ちつつ視野範囲を的確に限定することが可能となる．

これを実現するための実装Classである“StructCore”は，前年度に尾藤が作成した．

- 上位階層項目のマスタID
- 当該階層項目のマスタID
- 階層項目の連番（平板に展開した際の順）
- 階層項目の区分（0：階層，1：項目）
- 階層項目の名称
- 階層項目の深さ（0-n）
- 階層項目の表示順序（その「踊り場」での順）
- 階層項目の直下の子の数
- 上位が指定する直下の階層項目の表示順序
- 深さ-マスタID-表示順序

なお最後の二つは，複合インデックスである．

このエンティティは全体として，個々の項目は自らとその親は何者であって“世界”の何処に居り，他者との前後左右の関係は如何なる状況となっているのか，を端的に表現している．



このClassをコード・マスタ毎に付随テーブルとしてCachéに格納することで，ツリー表示やリスト表示を高速に実現することができた．

C.15.2.3 EntryMenu

その一方で，MEDIS-DC検査コード集，あるいは薬品コード集は，一軸で規定されるツリー上に項目を並べあげても実用には即さない．

このことは，分類学（systematic）上の軸なる概念は，必ずしも実用に即すわけではないことを示唆している．いずれにせよ，現場に即したメニュそしてメニュ展開が必須となろう．

然るにシステム実装，特にhuman interfaceの構成の際に規範となるような階層情報または『項目配置』における参考情報などは，今回，入手することはできなかった．

Referenceはいわゆる情報モデルのみならず，このようなソースにも与えられたほうが，開発現場には，より実際的であろうと思われる．

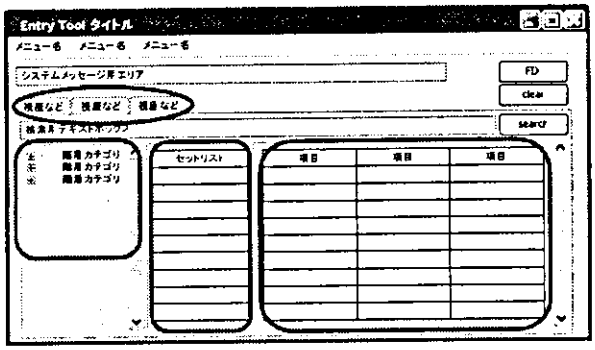
ともかくも，なにかしらの一覧表示機構が必要となったのでEntryMenuデザインを創ることとした．

EntryMenu Design

まずはメニュ構造と展開機構とを，予め想定しておく必要がある．そこで通常用いられている構造と展開を整理しながら一般形と思われるモデルを，将来への発展も見据えつつ構築した．

すなわち，Problem orientedまたはProcedure orientedのような視座が，各人や各診療科部の実情に合致した視野範囲の限定と展開を可能とするようなモデル・デザインである．

これを実現する画面例は以下の如くである．

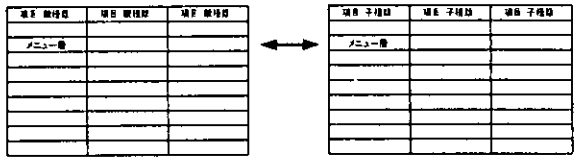


その機能としては次の事項を想定している：

- 選択した視座（病名指向や行為指向など）に応じた階層カテゴリの表示
- 選択した階層カテゴリに応じたメニュおよびセットリストの展開

このうち前者については将に知識を要求する機能なので実現には準備期間を要するものの不可能ではないし重要ではある．

さらに，メニュ項目またはメニュ・タグから，新たなメニュを展開させたり，セット選択にて複数の項目を一括選択したりできる などの，一般的な機能も，当然ながら，想定した．

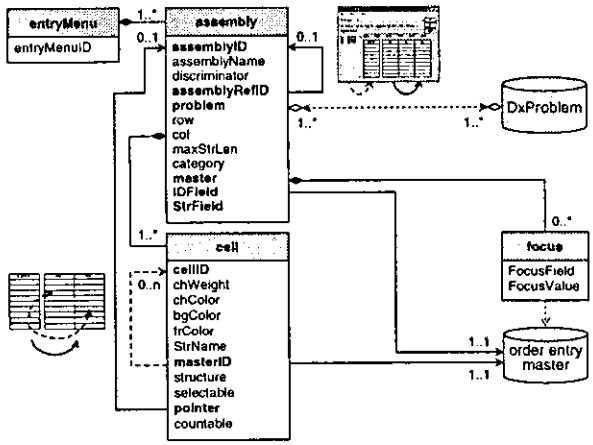


そしてこれら機能を単に実装するのみならず，(a) 参照実装としてモデル化するとともに，(b) end-user による編集の枠組を構築することに努めた．

EntryMenu.class

上記の要件に即して以下のクラス設計した．

クラス assembly は，各『メニュ単位』を表現している．メニュ単位には，通常の選択メニュのほかにセットメニュ（リスト）も含んでいる．セットメニュ（リスト）は，他のメニュ実体を指し示すのでClass diagramとしては自己参照となる．



クラス assembly は対象とするコード・マスタを指し示すと同時に，そのテーブル内の対象とするfieldを指し示す．またメニュの表示上の各種基本属性を規定する（行列の数や幅）．

この条件のもとに，クラス assembly はクラスcellのcollectionを保持する．

クラス cell は，メニュの表示上の属性を保持する．そして登録可能項目ならコード・マスタ項目を指し示し，セットならエントリ登録可能項目（クラスcell）を指し示し，さらにメニュ展開項目ならクラス assembly を指しすこととした．

なお今年度の実装では視座と指向については，そのような知識 ontology が準備されていないため実装の対象外としたが，その実現に要する property は用意しておいた．

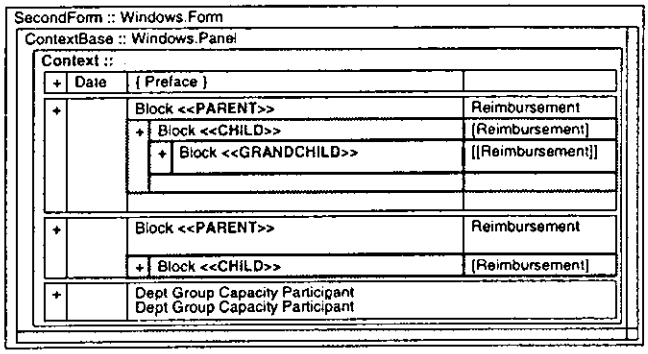
EntryMenu.xsd

このクラス設計に即して EntryMenu.xsd を定式化した．【資料】

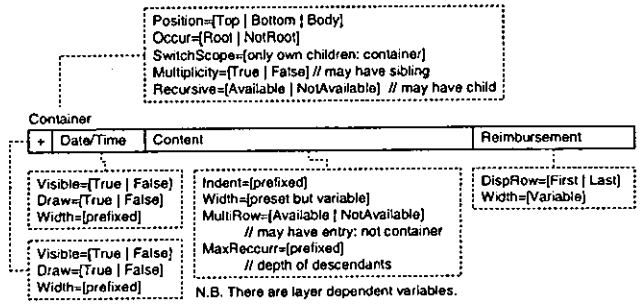
C.15.2.4 Outlined Container

流れや文脈が存在する集積情報において視野範囲を限定する，とは，アウトライン化するということでもある．

よって2号様式画面においてはアウトライン機能を付与した．なお，その際には当然ながら(C.9.2.2) に記した階層的構造にも留意しつつ下図のようにデザインした．



このような階層構造は汎用度の高い module を再帰的に使用して構成することが可能である．よってその概念モデルを設計した（実装については分担報告書（与那嶺）を参照されたい）．



この実装 Class を container と称する．ただし(C.3.4) に記した“container”機能を有する infoNode とは混同されぬよう御留意願いたい．

とはいえ両者は，参照実装においては相補的に利用され機能している．

なお property に格納される値は“独立”ではなく，むしろ

・クラス container の種別に依存
・クラス container が位置する階層深さに依存

している．あるいは，言い方を変えれば，そのような要素を property して，他の要素は基本骨格として再帰性を促したわけである．