データベースシステム 123－9
（2001．1．23）

情報システム開発方法論 Lyee に基づくソフトウェアの XML による記述

穂鷹 良介（東京工科大学）
武田 陽三，戸村 茂昭（ソフトウェア生産技術研究所）

概要

情報システム開発方法論 Lyee は利用者の意図が具体化した単語だけから直接ソフトウェアを定義し、その定義情報から自動的にプログラムを生成する．

Lyee の一つの独創的な点はソフトウェアの定義をまず固め，その後定義に対応するプログラムを生成するところにある．Lyee が何故に応用プログラムの生成に成功しているかという鍵の一つは Lyee で確立したソフトウェアの構造が様々な応用環境にも拘わらず普遍性を具備しているからである．

ソフトウェアの定義手続きはこの構造に単純に沿った (1)スターター定義 (2)スターター定義の自動的な拡充定義 (3)拡充定義に対しての利用者による追加定義 の３ステップから成る．

本論文はまず実務で行われている定義の内，本質に関係ないものを取り去り定義の本質を明確にし，更にソフトウェアの定義はデータモデルそのものであることを主張する．また，読者の理解を助けるために定義を XML 文書の形で示した．

XML representation of software based on information system development methodology Lyee

Hotaka Ryosuke (Tokyo University of Technology)
Takeda Yozo, Tomura Shigeaki
(The Institute of Computer Based Software Methodology and Technology)

Abstract

The information system development methodology "Lyee" automatically generates a program based on the definition of a software obtained from the set of words that materialize the intention of a user.

Lyee adopts a unique strategy in that it fixes the definition of a software first and then generates the corresponding program later. One of the clues why Lyee can successfully generate application programs is that the software structure Lyee establishes has such a universal property that remains invariant in various application environments.

The defining procedure of a software simply follows this software structure in 3 steps, i.e., (1) starter definition (2) automatic generation of extended definition of the starter definition and (3) addition of user's definition over the extended definition.

This paper first eliminates non-essential usual practices adopted in real application development environments and claims that the software definition is best explained by data model. Examples are shown in XML documents to ease readers' understanding.

1．はじめに

情報システム開発方法論 Lyee は意味を捉えるためのモデルであり，既存のソフトウェア開発方法論と全く別の発想で根来文生氏が樹立したものである．Lyee においてはソフトウェアはモデルの記述（あるいは定義）対象であって，Lyee が記述できる世界の一部である.情報システム開発時にはまず概念的な存在としてソフトウェアを確定する．

Lyee においてはソフトウェアとプログラムとは別のもので，ソフトウェアが確定したとき，それが持つ機能を実現するメカニズムとしてプログラムを位置付ける．一定の機能さえあればどのプログラム言語でもソフトウェアの機能を実現するものとして使用することが出来，Lyee 環境では通常，これを自動生成する．アプリケーションとは独立にはじめからソフトウェアの構造がきちんと決まっているためにこのことが可能となる．

Lyee を用いるシステム開発は従ってソフトウェアの定義と自動生成とから成る．

ソフトウェアの定義は更に
（1）Starter Definition の作成
（2）Starter Definition から Extended Definition の自動生成
（3）Extended Definition に対しての定義の追加

に分かれる．Lyee ではソフトウェアの定義を一気に行うのではなく一度入力した情報を普遍的なソフトウェアの構造に合わせて拡充する．次に拡充された構造に対して更に情報を追加する方法を取っているため手続きが 3 ステップと成る．

本論文は Lyee のソフトウェア定義は対象とする世界に存在するデータのモデルを決定していることであるとみなすことが出来ることを主張し，実際にデータモデルを定義する言語として XML を用いて定義する例を示す．ソフトウェアの定義がデータモデルになるということはソフトウェアの開発にデータベースの諸技術が即応用できるということであり，ソフトウェアとデータベースの新たな協調関係が可能であることを意味する．本論文ではソフトウェア定義のデータモデル的な側面に話を限定する．

プログラムの自動生成はプログラム言語ごと（個別の応用プログラムごとではない）にテンプレートを用意することによって可能となる．テンプレートが準備されていればプログラムの自動生成はボタンを一つ押すだけの作業である．テンプレートの開発は興味深い研究対象であるが，本論文ではソフトウェア定義のデータモデル的な側面に話を限定する．

以下第 2 章で Lyee によるソフトウェアの定義作業の全体の流れについて概観する．第 3 章では何故に定義作業が 3 ステップとなるか，その背後にある網羅的なソフトウェア記述要素の必要性，Lyee で重視する同期構造とデータベースの共用制御の関係について説明する．第 4 章ではソフトウェア定義作業の第 1 ステップについて，第 5，6 章ではそれぞれ，第 2，第 3 ステップについて簡単な例題を用いて説明する．第 7 章では引き続き検討されなくてはならない将来の研究の方向について私見を述べる．

2．ソフトウェア定義作業の概略

Lyee によるソフトウェア定義作業は図-1 に示すように（1）Starter Definition の作成，（2）Starter Definition から Expanded Definition の自動生成，（3）Extended Definition に対しての追加定義の 3 ステップから成る．ここでは GUI を利用して利用者がコンピュータと会話を行いながら情報処理を行うケースを想定する．

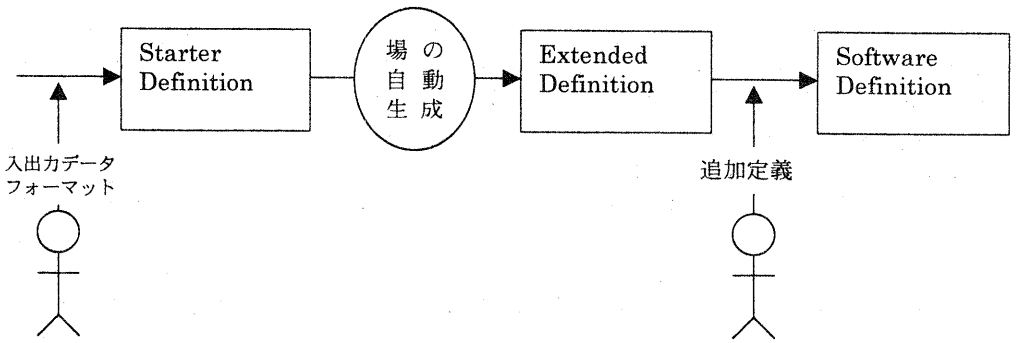

図 - 1　ソフトウェア定義のステップ

3．プログラムの自動生成を可能にするソフトウェア構造

3．1　Lyee のソフトウェア構造

Lyee のソフトウェア構造の全体図を図 - 2 に示す.

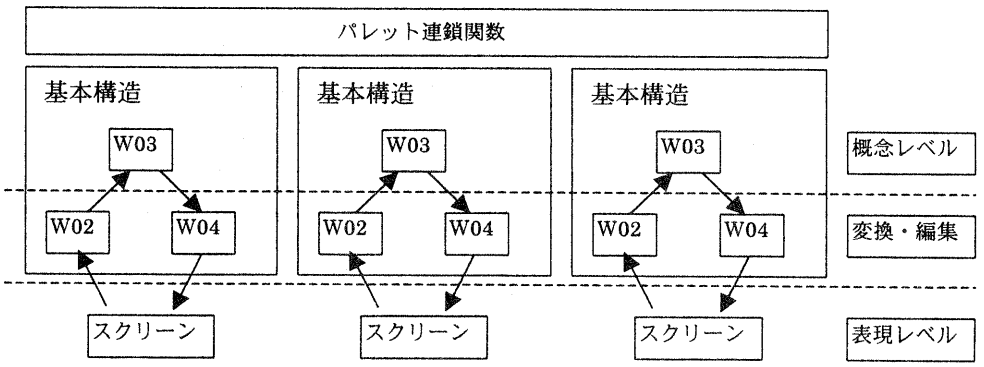

図 - 2　Lyee ソフトウェア構造

W02, W03, W04 のおのおのはパレットと呼ばれるソフトウェア構造の一部でありそれぞれデータ領域を有しており，この 3 個は基本構造という単位を構成する．非常に大雑把にいうとスクリーンに入力されたデータは W02 に対応してデータ項目毎に実装されるプログラムに渡されて内部処理に耐えるように変換され，W03 に対応してデータ項目毎に実装されるプログラムの入力データと成る．そこで指定された自己データの生成がなされ，その生成データが再び W04 に対応してデータ項目毎に実装されるプログラムに渡され編集の後スクリーンに表示される．各パレットに対応してデータ項目毎に実装されるプログラムでは利用者の意図が具体化した単語がスクリーンに定義された各データ項目をそれぞれの役割に応じて処理し，それらの処理をパレット連鎖関数が一括制御する．その一連の処理は利用者が意図するところを具体化した単語の意味を捉える。

プログラムの自動生成を可能にするために Lyee はプログラムとデータのそれぞれの側面に普遍的な規律を導入する．まずプログラムの側面では同期構造と呼ばれている構造的には非常に単純

な（しかし，その働きは深遠な）要素の組み合わせだけを認める．プログラムが単純かつ規則的な構造をしているために，それを生成するための定義（ソフトウェア）も単純かつ規則的なものとなり，定義からプログラムを生成する手続きが可能なものとなる．

データの側面では利用者の意図が具体化した単語のすべてをあらかじめ準備してしまうという方法を取る．すべてをあらかじめ準備ということはデータの“相”のようなものの変種を尽くすことと、入出力インタフェース上のすべてのデータの同期を取るという二つの観点からなされる．

3．2　データの相

図 - 2を拡大して情報処理を行うときにデータがどのように用いられるかを詳細に検討してみる．

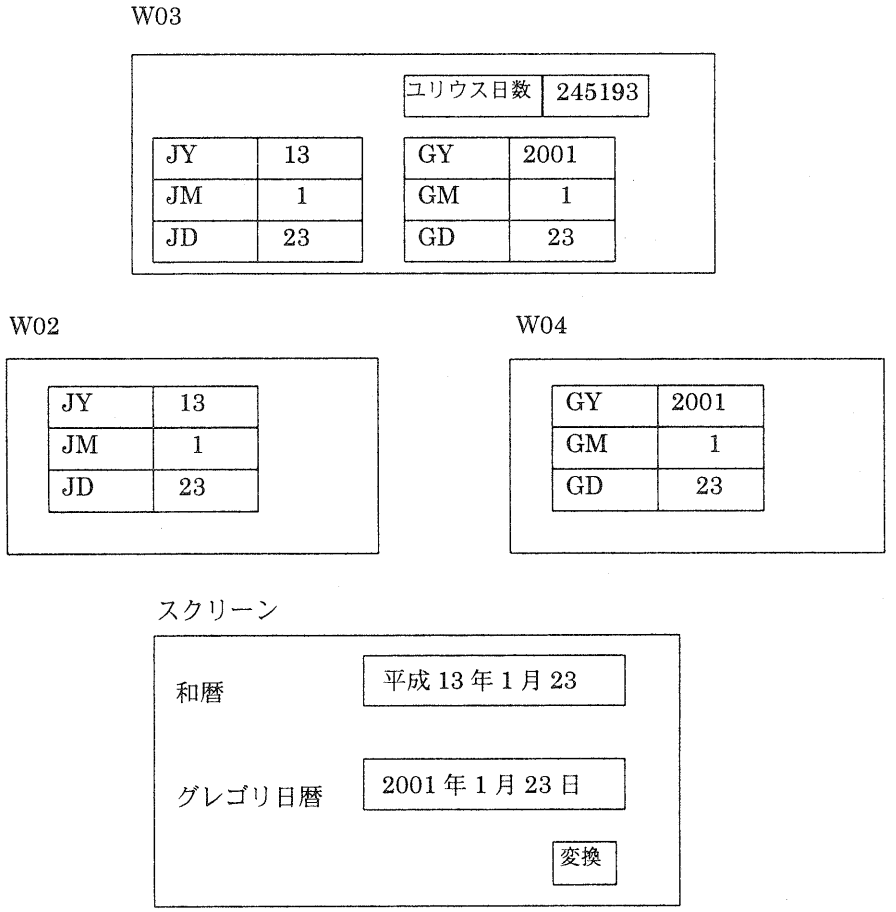

図 - 3　使用される可能性のあるすべてのデータの用意

ここではここで定義されるプログラムがどのように動作するかという点から説明を行う．言い換えるとこのような動作を行うプログラムが生成できるようにソフトウェアの構造を定義するのがここでの目的である．

スクリーンに和暦が入力された状態で変換ボタンが押されたときにそれをグレゴリ日暦に変換する処理を考えてみる（詳細ロジックは割愛する）．スクリーン上には変換の際考慮しなくてはならない変数（データ項目）が和暦の年，月，日とグレゴリ日暦の年，月，日の 6 個がある（実際は

変換ボタンも変数の一種として扱うがここでは話を簡単にするために無視する)．入力された和暦の年月日はスクリーンのような色々物理的な表現形式に制約されているものであるから，それをLyee の普遍的な方式で扱えるように形式を整えて W02 にコピーされる．この段階ではまだ文字コードなどがインタフェースに依存したものとなっているので更にそれを応用独立な（概念）データに変換して W03 に渡す．

W03 では概念的な演算を施して結果を上記と逆の方向に W04 経由でスクリーンに返して一件落着となる．必要ならば演算の途中で用いるデータ項目を図 - 3 のユウリス日数のように追加しておく（これは 6 章で述べる手続きである)．

微妙な取り扱いについては十分説明できなかったが，この例の示すように単に年というデータも場合場合に応じて使い分け，それに対して利用者の仕様を表現する必要があるので入出力インタフェースに現れたデータ項目を基に使用される可能性のあるデータ項目はすべて，実際に使う使わないに拘わらず網羅的に生成しておく．これはユーザの仕様を取り入れる前に可能なことなので自動生成ができる．言い換えればあるデータ項目は利用者から見れば一つに見えるが厳密に考えるといくつかの「相」を有していて，ここではその相別にデータを準備して後に（6 章で）ユーザの仕様を書くときの準備をする．

3．3　データの同期

厳密な事務計算では，データの入力から一件落着して出力がされるまでの処理は論理的には一瞬間に起きた処理と等価でなくてはならない．データベースの言葉で言えば，ここにはデータベースは必ずしも存在しないのだが，入力に対して出力を得る処理は全体が一括して実行されるかあるいは全体が一括して実行されないかの原子的なトランザクションを構成しなくてはならない．

たとえば，入力が受け付けられて出力がなされるまでに利用者が勝手に再度入力データを変更するようなことがあったとすれば，出力は一意ではなくなり厳密な事務処理がなされたことにはならない．その意味で，排他制御という言葉は使われていないが，一連の処理の中で使われているデータは処理が終わるまでは他の如何なる処理によっても変更されていないことが保証されていなくてはならない．データベースでいうところの一種の共用制御があると考えてよい．

Lyee ではこの概念を同期と言っているが，考えていることはデータベースの共用制御と全く同じである．ここでは紙面の都合上示すことが出来ないが，基本構造がいくつも協力して処理を行うとき，その中の一つがデータベースを扱うときには当然そこではデータベースの共用制御がされることになる．

3．4　ソフトウェアの 3 ステップの定義

以上述べたように Lyee ではまず入出力インタフェースを利用者自身が定義するのが第 1 ステップ，次にデータ項目の相を考慮して可能性のあるデータ項目を人手を介さずに自動生成するのが第 2 ステップ，あとはデータ項目固有のデータ生成式並びにその生成式を成立させるデータ項目を補い出力データを入力データから連鎖反応のように作り出していくロジックを定義データとして投入するのが第 3 ステップである．特に第 3 ステップの仕様を追加するときには第 2 ステップで記述の為に必要なデータが用意されていなくてはならないから，ユーザは第 2 ステップの定義拡充の結果を予想しない限りこれを 1 個のステップで記述することは難しい．

4　Starter Definition の作成

Lyee でソフトウェアの定義をするときにはまず基本となる入出力インタフェースの定義を行う．対応する定義は定義体といわれる．ここでは簡単のためすべての入出力インタフェースは図 - 2 のようなスクリーンであると仮定している．

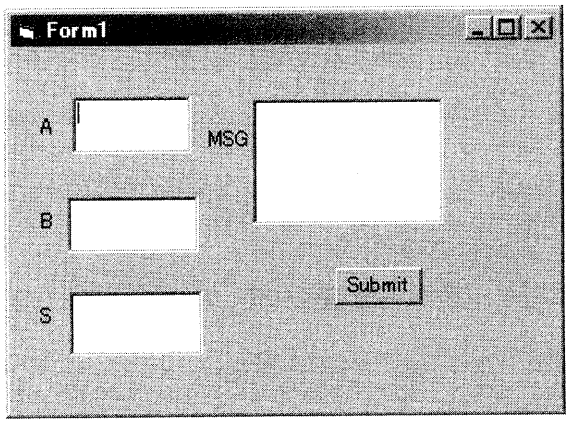

図 - 2　入力スクリーン

図 - 2 に対応する定義体（一部省略）の XML 記述を下に示す.

```
<定義体明細一覧>
        <項目>
                <項目 ID>A</項目 ID>
                <桁数>6</桁数>
                <属性>X</属性>
                <入出力区分>1</入出力区分>
                </項目>
        <項目>
                <項目 ID>B</項目 ID>
                <桁数>6</桁数>
                <属性>X</属性>
                <入出力区分>1</入出力区分>
        </項目>
         （中略）
</定義体明細一覧>
```

このために各スクリーンがどの W02, W04 に所属するかを更に指定する. W03 はインタフェースをもつ W02, W04 に所属するスクリーンを継承する. この記述を XML で以下のように記述する.

```
〈パレット一覧〉
    〈パレット〉
        〈パレット関数 ID〉frmCalcW02〈/パレット関数 ID〉
        〈T0-T1 区分〉0     T0〈/T0-T1 区分〉
        〈パレット区分〉2    W02〈/パレット区分〉
        〈パレット所属定義体一覧〉
            〈定義体 ID〉frmCalc〈/定義体 ID〉
        〈/パレット所属定義体一覧〉
    〈/パレット〉
    〈パレット〉
        〈パレット関数 ID〉frmCalc04〈/パレット関数 ID〉
        〈T0-T1 区分〉0       T0〈/T0-T1 区分〉
        〈パレット区分〉4      W04〈/パレット区分〉
        〈パレット所属定義体一覧〉
            〈定義体 ID〉frmCalc〈/定義体 ID〉
        〈/パレット所属定義体一覧〉
    〈/パレット〉
    〈パレット〉
```

```
        〈パレット関数 ID〉W03Plt〈/パレット関数 ID〉
        〈T0-T1 区分〉0      T0〈/T0-T1 区分〉
        〈パレット区分〉3    W03〈/パレット区分〉
    〈/パレット〉
    〈パレット〉
        〈パレット関数 ID〉PltCntrlFt〈/パレット関数 ID〉
        〈T0-T1 区分〉0      T0〈/T0-T1 区分〉
        〈パレット区分〉T    〈/パレット区分〉
    〈/パレット〉
〈/パレット一覧〉
```

もう一度 Starter Definition 作成の為にユーザが投入する情報をまとめると

（1）ユーザの要求が表現される入出力インターフェースの仕様（具体的にはスクリーンで値の入力あるいは出力が行われる項目の記述

（2）パレット（W02，W04）の定義とそこにどの入出力インタフェースが所属するかの指定

の 2 種の情報である．

5　Extended Definition の作成

これは自動的に行われる．図 - 3 の場合スクリーンにある変数を JY，JM，JD，GY，GM，GD としたとき，Extended Definition は座標軸を screenJY，screenJM，screenJD，screenGY，screenGM，screenGD（以上は図 2 の表現レベルに対応），screenW02JY，screenW02JM，screenW02JD，screenW04GY，screenW04GM，screenW04GD（以上は図 2 の変換・編集レベルに対応），W03JY，W03JM，W03JD，W03GY，W03GM，W03GD（以上は図 2 の概念レベルに対応）用意したことに成る．1 つの各スクリーンデータ項目ごとに 4 次元の空間を用意している．仮に次のステップで W03 ユウリス日数なるデータ項目が一個追加されると全体の空間はそれも一つの座標軸となる 19 次元空間になる．要するに情報処理はこのような空間（これら座標軸の直積集合）の中の点を用いて行われ，この空間からはみ出した点を考慮する必要はない．

6　Extended Definition に対しての仕様の追加

実は 6 のステップを実行しなくても Extended Definition が出来上がったところでプログラムを自動生成すると「何もしないが動くことは動く」というプログラムを作り出すことが可能である．この状態に対してデータ項目固有のデータ生成式並びにその生成式を成立させるデータ項目等を追加する．

具体的には DTD に Extended Definition に対して以下のように、単語のロジック定義として“空条件”，“自己生成”を追加する．

```
〈!ELEMENT 単語 (単語_作用子区分, 定義体 ID, 単語 ID, 単語種別,
                 領域 ID, 桁数, 属性, 入出力区分, 定義体区分,
                 空条件*, 自己生成*) 〉
```

それにともなって図 - 2 の例題で A，B の和を S に計算するとしてそのロジックを自己生成として

```
〈単語〉
    〈単語_作用子区分〉L〈/単語_作用子区分〉
    〈定義体 ID〉frmCalc〈/定義体 ID〉
    〈単語 ID〉S〈/単語 ID〉
    〈単語種別〉1〈/単語種別〉
    〈領域 ID〉S〈/領域 ID〉
    〈桁数〉6〈/桁数〉
```

```
    〈属性〉X〈/属性〉
    〈入出力区分〉2〈/入出力区分〉
    〈定義体区分〉1〈/定義体区分〉
    〈自己生成〉
        W3_frmCalc_S=val(W03.frmCalc.A)+val(W03.frmCalc.B)
    〈/自己生成〉
〈/単語〉
```

のように追加する.

ここではExtennded Definitionに対して追加された部分だけを示した. 詳しくは参考資料を参照されたい.

以上の定義の結果、上記のソフトウェア定義部分に対応して次のようなプログラム（ここではVisual Basicを指定したとする）が自動生成される.

```
Private Sub L3_frmCalc_S()
 If W03.frmCalc.S <> "" Then Exit Sub End If
 W3_frmCalc_S = Val(W03.frmCalc.A) * Val(W03.frmCalc.B)
 If W3_frmCalc_S = "0" Then Exit Sub End If
 If W03.frmCalc.A = "" Or W03.frmCalc.B = "" Then Exit Sub End If
 W03.frmCalc.S = W3_frmCalc_S
 W03.frmCalc.S_Non = False
 W03_RECALL_FLG = True
End Sub
```

7．将来の研究の方向

今回はLyeeのソフトウェア定義のデータモデル側面について主に考察した. 今後の研究の方向は二つある. 一つはデータモデル的な考察でもう一つはプログラムの自動生成の研究である.

ソフトウェアの定義をXMLというデータモデルで表現できることを示したので, 次は一般のデータベース管理システムを利用してもっと実用的なソフトウェアの記述を行うことが考えられる.

次はこのようにXMLで表現されたソフトウェアの定義から実際にプログラムを自動生成する研究で最初は生成されるプログラム自身をXMLで記述しその構造を考察しプログラム自動生成のメカニズムを論理的に解明することが考えられる. XMLをオブジェクトプログラムに選ぶことによりアカデミックな世界でプログラム自動生成がプラットフォーム独立な環境で行われることが期待される.

謝辞並びにライセンス

Lyeeの研究に関してソフトウェア生産技術研究所株式会社から, 全面的な協力を頂いた事を感謝致します.

参考資料

[1] 穂鷹良介, 武田陽三, 戸村茂昭: 情報システム開発方法論 Lyeeに基づくソフトウェアのXMLによる記述, http://www.teu.ac.jp/media/~hotaka/FinalR4.pdf に本文とソフトウェアのXML記述例がある.
[2] カテナ株式会社：Lyee 一般編, http://www.catena.co.jp/pdf/lyee/ippannhen.pdf
[3] Fumio Negoro: Principle of Lyee Software, IS2000, pp.441-446.