Vol.2014-IS-128 No.5
2014/6/6

因果関係を導入した制約表現に基づく税務処理システム構成法

矢野 寛将[1]　井田 明男[2]　金田 重郎[1]

概要：業務システムは従来，業務担当者が行っている作業を手続き的に実装してきた．しかし，(1) 業務規則が複数のプロセスに分散して実装される，(2) システムにすでに入力したデータの修正処理では，修正処理パターンの増加に応じて追加的にプログラミングする必要がある，などの問題があった．そこで本稿では，業務規則を制約として宣言的に表現し，その制約表現に基づき業務システムを実装する手法を提案する．具体的には，業務規則を制約として表現した後，入力データと出力データの因果関係に基づいたグラフの構築を行う．評価のため，京都府と京田辺市の地方税税務処理を提案手法に基づき実装を試みた結果，23 種の税務処理（一部の税種について細かい税額計算を除く）について約 5,700 行（ユーザインタフェースとそのサブシステムを除く）で実装できた．この実装は既存システムに比べてコンパクトであり，業務規則も凝集しているため，開発効率と保守性が共に高いと考えられる．

キーワード：業務システム，税務処理，制約表現，制約充足，単方向性，構成法

An Implementation Methodology for Local Tax Processing Based on Constraint Expression with Cause-and-Effect

Abstract: ICT systems for business applications have been implemented widely by procedural approach. However, these implementations have some issues: (1) One business rule is scattered into many processes. (2) Modification of previously inputted data requires additional programming. This paper proposes a new implementation methodology based on constraint expression which is declarative expression of business rules. In proposed methodology, the author regards business rules as constraint and created graph based on case-and-effect relationships between input and output. The author implemented a prototype system with about 5,700 lines in an object-oriented programming language. The system supports 23 kinds of local taxes (but some detailed tax-amount calculation routines are omitted). Our evaluation shows that the proposed methodology provides very compact and well-structured codes compared to conventional applications. So, both development efficiency and maintenance ability are high.

Keywords: Business Systems, Local Tax Processing, Constraint Expression, Constraint Satisfaction, Unidirectional, Implementation Methodology

1. はじめに

業務システムは従来，業務担当者が行っている作業を手続き的に実装してきた．業務フローは図 **1** のような Data Flow Diagram（DFD）[1] を用いて表現され，それぞれのプロセスは手続き的プログラミング言語によって実装される．ユーザはシステムのメニューから業務を選択する．選択された業務は，イベントとしてシステムに通知され，手続き的なプロセスが呼び出される．しかし，手続き的記述に基づくプログラムは，（1）同じ業務規則が複数のプロセスに分散して実装される（図 **2**），（2）システムにすでに入力したデータの修正処理では，修正処理パターンの増加に応じて追加的にプログラミングする必要がある（図 **3**），という問題がある．これらの問題が，業務システムの設計・開発やメンテナンスを難しくしている．

この問題を解決するために，石井らは業務規則を制約と

[1] 同志社大学大学院理工学研究科
Graduate School of Science and Engineering, Doshisha University
[2] 同志社大学理工学部
Faculty of Science and Engineering, Doshisha University

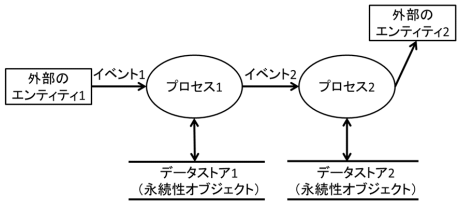

図 1　DFD で表す業務フローの例
Fig. 1　Example of Business Flow in DFD Format.

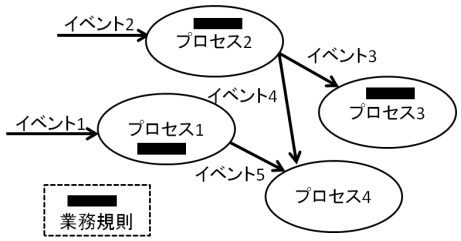

図 2　分散した業務規則
Fig. 2　Distributed Domain Business Rule.

して宣言的に表現できるプログラミング言語を開発し，それを用いてオフィス処理を実現する手法を提案した [2][3]．石井らの手法では，双方向にデータを伝搬させるメカニズムに基づき処理を行う．しかし，双方向にデータを伝搬させるため，ユーザとの対話処理が増大する欠点がある．著者らはすでに単方向にデータを伝搬させるメカニズムを用いれば，対話処理が削減できることを証明している [4]．しかし，単方向性制約伝搬を実現させるアプリケーションアーキテクチャの設計が不十分であった．

本稿では因果関係を導入した制約を用いて業務規則を表現し，その制約表現に基づき税務処理システムを構成する手法を提案する．業務規則は法律の条文構造に倣って記述されているため，税務処理に適応できれば，業務システムにも適応できると考える．提案手法の内訳として，アプリケーションアーキテクチャと，そのアーキテクチャを構成する手順について述べる．構成手順では，因果関係を導入した制約表現から，入力データと出力データの因果関係に基づき条件分岐のグラフを構築する．このグラフに基づき業務処理を行うために，単方向にデータを伝搬させるメカニズムを用いる．法律の条文構造は，要件部と効果部の 2 つの部分から構成される [5]．提案する制約表現は，要件部と効果部から成る業務規則をそのまま変換した表現である．そのため，提案する制約表現は，業務規則とのセマンティックギャップが小さい．したがって，開発効率と保守性が共に高いと期待される．

2. 従来の DFD アプローチの問題

2.1 本質モデル

S.M.McMenamin と J.F.Palmer は，システムの「本質モデル」と呼ばれる考え方を提案し，システムの本質について次のように述べた [6]．

（1）複数のプロセスはそれぞれ，外界から来るイベントに応答することが業務システムの本質である．イベントには 2 種類あり，外部エンティティ *1 が要求するイベントと，時刻に応じて発生するイベントである．

（2）それぞれのプロセスの粒度は，1 つのイベントを処理する内容によって決まる．

この本質モデルに基づくと，ある業務フローに対して適切な DFD[1] を作成できる．DFD は，業務システムを設計・開発するとき，業務データの流れを可視化するために有用であり，広く用いられている．DFD で表現した本質モデルの例を図 1 に示す．外部エンティティ 1 から要求されるイベントは，まずプロセス 1 に処理され，その処理結果は永続性オブジェクト *2 として保存される．次に，プロセス 1 がイベント 2 をプロセス 2 に送信すると，プロセス 2 が動作する．

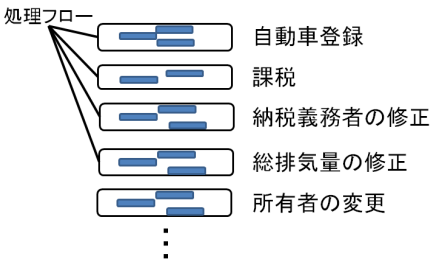

図 3　修正パターンの増加
Fig. 3　An Increase of Modification Modules.

2.2 DFD で表現した本質モデルの問題

しかし，図 1 に示される従来の本質モデルは，次の 2 つの問題がある．1 つ目は，業務規則が複数のプロセスに分散して実装されることである（図 2）．図 2 のように，一般的に 1 つの業務規則は複数のイベント（イベント 1，イベント 2，イベント 3）を持っている．したがって，業務規則は複数のプロセス（プロセス 1，プロセス 2，プロセス 3）にコピーされる．つまり，同じ業務規則が複数のプロセスに分散する．業務規則は社会情勢の変化によって変更が多いため，分散していると保守が難しくなる．

2 つ目は，システムにすでに入力したデータの修正処理では，修正処理パターンの増加に応じて追加的に実装されることである（図 3）．あるデータが修正されたら，他のデータも一緒に修正しなければならない場合が出てくる．業務データはたくさんあるため，その分の修正処理パターンがある．業務データが増えていくと，修正処理パターンも増加するため，業務要求が複雑になっていく．そのため，

*1　DFD におけるデータの移動元または移動先．
*2　オブジェクトを生成したプログラムの実行が終了しても，永続的に存在するオブジェクト．オブジェクトとはクラスとインスタンスから構成される実体．具体的にはファイル．

表 1　制約表（自動車税の場合）
Table 1　Constraint Table(Car Tax.)

番号	現在の日付	自動車の存在	所有者の存在	税の賦課
1	3 月 1 日以降 3 月 31 日以前	○	○	×
2	4 月 1 日	○	○	○（年の税額）
3	4 月 2 日以降 2 月末日以前	○	○	○（年の税額または翌月から月割の税額）
4	すべての日	×	×	×

開発が困難となる．

3. 業務規則の制約表現

石井らは，従来の手続き的記述に基づく業務システム開発の問題を解決するために，業務規則を制約として宣言的に表現できるプログラミング言語を開発した [2][3]．制約表現したプログラムを実行し，業務処理を行った．著者らは，石井らと同様に，業務規則を制約とみなす．

地方税法を例として，業務規則を制約として見る方法と，制約表現を行う理由について述べる．一般的に多くの業務規則は，法律の形式に倣って記述されるため，法律文の例から業務規則一般に適応できる．そのため，地方税法 [7] の一例を以下に示す．

- 自動車を 4 月 1 日時点に所有している住民は，自動車税を払わなければならない．
- 4 月 1 日以後に自動車を購入し納税義務が発生した住民は，その発生した月の翌月からの月割の税額で，自動車税を払わなければならない．

法律文を含む業務規則は，文章から多くの情報を読み取ることができる．つまり，上記の法律文は，次のように解釈できる．

- 自動車を所有している住民は，3 月 1 日以降 3 月 31 日以前の時点でその年度の納税義務はない．
- 住民が 4 月 1 日に自動車を所有していれば，4 月 1 日以降にその年度の納税義務がある．この場合，年の税額を支払わなければならない．
- 住民が 4 月 2 日以降，翌年 2 月末日以前に自動車を所有すれば，その所有の日からその年度の納税義務がある．この場合，納税義務が発生した翌月からの月割の税額を支払わなければならない．
- 住民が 4 月 1 日以降，翌年の 2 月末日以前に自動車を所有していなければ，自動車税を支払う必要はない．

このようなすべてのケースをまとめると，**表 1** の真理値表となる．賦課とは課税することをいう．○は存在している，賦課されていることを示す．×は存在していない，賦課されていないことを示す．

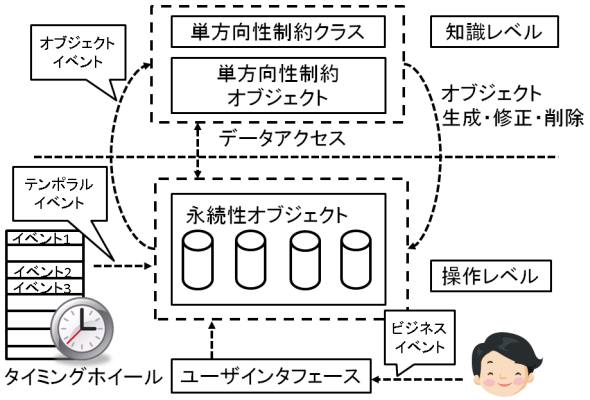

図 4　提案するアプリケーションアーキテクチャ
Fig. 4　Proposed Application Architecture.

業務処理を行うとき，その処理で扱うデータ状態は，真理値表のすべてのケースのうち必ず 1 つの行に一致しなければならないという「制約」がある．これは，業務規則を解釈してすべてのケースを 1 行ずつ洗い出したためである．処理で扱うデータ状態が 1 つの行に一致すれば，業務規則，つまり制約を満たす．したがって，制約を満たせば，業務データが常に正しい値になる．表 1 を例に挙げると，4 つの行が上記の業務規則の制約である．業務規則を満たすためには，1〜4 のうちの必ず 1 つの状態に一致しなければならない．現在が 3 月 31 日であり，かつ住民が自動車を所有していれば，この業務データは 1 番の行のみに一致している．このとき，自動車を所有している住民に，自動車税は課税されない．

システム開発を行うとき，システムの分析者は，表 1 の真理値表のように業務規則からすべてのケースを見つけなければならない．制約表現は，業務規則をそのまま記述した形式であるため，業務規則と制約表現はセマンティックギャップが少ない．したがって，制約表現を用いれば，システムの設計・開発・保守が効率的である．

4. 単方向性制約伝搬を実現するアーキテクチャ

4.1 提案アーキテクチャの概要

従来の本質モデルの問題を解決するために，本稿では新しいアプリケーションアーキテクチャを提案する（**図 4**）．このアーキテクチャでは，単方向性制約伝搬というイベント駆動制御を行っており，この制御にソフトウェアパターンを組み入れている．アーキテクチャの上層は「知識レベル [8]」であり，単方向性制約オブジェクトが，制約という枠組みの中で単方向にデータを伝搬させる．このメカニズムを単方向性制約伝搬と呼ぶ．このメカニズムに基づくと，業務データが制約を満たし正しい値になる．図 4 の単方向性制約クラスには，制約表現から，データ同士の因果関係に基づいた条件分岐を実装する．下層は「操作レベル [8]」であり，永続性オブジェクトを保持している．この永続性

オブジェクトは現実世界のコピーである．この 2 階層構造は，M.Fowler のアナリシスパターンを用いている [8]．

本アーキテクチャではイベント駆動制御を行う．イベントが起こるたびに，常に制約がチェックされ，すべてのデータが正しい値になる．本アーキテクチャは，システム内でイベントが発生すると，制約へ伝搬させる処理を行う仕組みとなっている．システムの「本質モデル [6]」と同様に，イベントに対応することがシステムの本質であるためである．本アーキテクチャでは，イベントは以下の 3 種類がある．

- ビジネスイベント…ユーザは，ユーザインタフェースが提供するメニューからイベントを引き起こす．メニューには，永続性オブジェクトを生成・参照・修正・削除する機能がある．
- オブジェクトイベント…永続性オブジェクトを変更（生成・修正・削除）する．永続性オブジェクトが変更されると，制約に通知される．制約にイベントを通知するための仕組みに，GoF のオブザーバパターン [9] を利用している．永続性オブジェクトが Subject（観察対象）であり，単方向性制約伝搬クラス・オブジェクトが Observer（観察者）である．
- テンポラルイベント…将来的にイベントが起こる場合，制約はそのイベントを日付に紐付けて「タイミングホイール」に事前に登録しておく．登録した日付になれば，登録しておいた新しいイベントが自動的に起動する．

この 3 種類のイベントが起こるたびに，常に制約がチェックされる．

4.2 単方向性制約クラスの記述内容

すべての業務規則は，それぞれに対応した 1 つの単方向性制約クラスに書かれている．例えば，地方税法 [7] には，4 月 1 日に自動車を所有している住民に対して課税するという業務規則がある．この業務規則を，因果関係を導入した制約として表現すると，**表 2** になる．図 4 の単方向性制約クラスには，表 2 を基に，条件分岐文を記述する．石井の研究 [2][3] では，制約表現ができる言語が開発された．しかし，本研究の実験では制約表現の言語は開発しておらず，表 2 の制約はメソッド（UML における操作）として実装している．

単方向性制約伝搬オブジェクトは，プライマリインプット（PI）の値からプライマリアウトプット（PO）の値を決定する．本稿ではこのことを，PI から PO へデータが伝搬すると呼ぶ．PI と PO はそれぞれ，制約という閉じた表現の中での入力データと出力データを意味する．具体的に，3 月 31 日に自動車がシステムに登録される状況を考える．4 月 1 日より前のとき，PI が表 2 の 1 番に一致するため，タイミングホイールに登録し，4 月 1 日に制約をチェックするように設定する．4 月 1 日になると，PI が表 2 の 2 番に一致するため，PO では自動車税を生成し税額を計算する．4 月 2 日以後になり，ユーザが入力に誤りを修正するためビジネスイベントを起こした場合，PI が表 2 の 3 番に一致するため，PO では自動車税の再計算を行う．

表 2 因果関係を導入した制約表現（自動車税の場合）
Table 2 Example of Constraints(Car Tax.)

番号	プライマリインプット（要件部）				プライマリアウトプット（効果部）
	現在の日付	自動車の存在	所有者の存在		税の賦課 [アクション]
1	3 月 1 日以降 3 月 31 日以前	○	○	→	× [**4 月 1 日に制約をチェックするように設定**]
2	4 月 1 日	○	○	→	○ [**生成し，税額を計算**]
3	4 月 2 日以降 2 月末日以前	○	○	→	○ [**税額を再計算（存在しなければ，生成・更新）**]
4	すべての日	×	×	→	× [**税の賦課が存在すれば削除**]

4.3 因果関係を導入した制約表現と伝搬メカニズム

石井らの提案 [2][3] において，システムは双方向にデータを伝搬させるメカニズムに基づき，業務データが制約を満たすようになっていた．つまり，出力オブジェクトのデータ状態が変更されたら，石井の制約充足エンジンは，PO を PI へ逆方向に伝搬する．制約伝搬では，業務データが真理値表の複数の行に一致した場合，ユーザに質問を行う．この質問に対するユーザからの応答に基づき，システムはデータを変更し，制約を満たす．石井らの制約プログラミング言語は，記述に対し高い柔軟性を持っていたが，業務処理を行うとき，ユーザとの対話が多く必要であった．双方向伝搬であるために複数の行に一致する回数が多くなり，質問数が多くなったためである．表 2 を例に挙げると，PO から PI を一意に決定できないことがわかる．

そこで，本稿では制約表現に対し因果関係の導入を行う．表 2 のように，事前に PI と PO を決定しておく．「PI から PO」への単方向のみの伝搬に制限し，単方向にデータを伝搬させるメカニズムに基づき，業務データが制約を満たすようにする．法律文は，要件部と効果部に分けられた論理構造である [5]．地方税法 [7] の一例を以下に示す．

- 自動車を 4 月 1 日時点に所有している住民は，自動車税を払わなければならない．

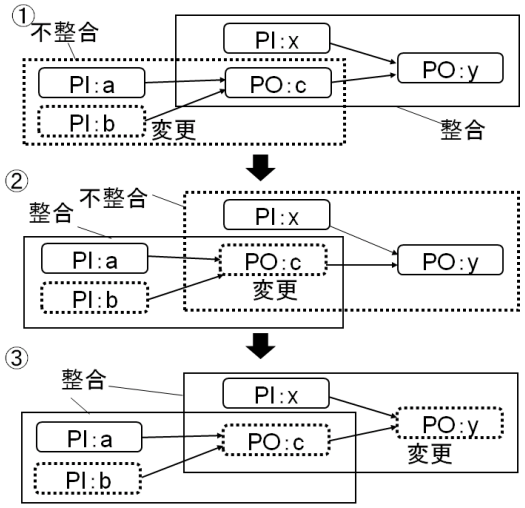

図 5　単方向性制約伝搬の原理
Fig. 5　A Principle of Unidirectional Constraint Propagation.

この文の場合，効果部は「住民は自動車税を払わなければならない」という部分である．住民は自動車税の納税義務者となる．一方，要件部は「自動車税を 4 月 1 日に所有している住民は」である．法律文あるいは，それと同じ文構造の業務規則は，文章の構造が単方向になっている．そのため，業務規則と，因果関係を導入した制約表現は，セマンティックギャップが少ない．したがって，制約表現に対し因果関係を導入し，データの伝搬方向を単方向に制限すれば，対話処理を削減できる．

4.4　単方向性制約伝搬の原理

単方向性制約伝搬のメカニズムでは，制約に入ってくる値を連鎖させている閉路を持たない有向グラフ（Directed Acyclic Graph）構造になっている（**図 5**）．すなわち，それぞれの制約には，PI/PO の関係が定義されていて，PI/PO の関係を見ると，半順序の関係となっている．すべてのイベントは，上流にある PI から下流にある PO へ単方向に移動する．図 5 のネットワークは循環がないため，最終的にイベント伝搬は停止する．半順序関係を持っているため，イベントの伝搬に基づき，すべての値が正しい値になる．

単方向性制約伝搬に基づき業務データの整合性を保つ原理を説明する．単方向性制約伝搬では，原則として

- プライマリインプット（PI）: 現実世界で生じているデータの値である．ユーザのみが変更できる値である．
- プライマリアウトプット（PO）: PI から制約によって生じるデータの値である．制約によってシステムのみが変更する．

とする．データ間の制約関係が順に整合性を保つ様子を，以下①，②，③ の順に図 5 を用いて説明する．制約関係は 2 つあり，1 つ目はデータ (a，b，c)，2 つ目はデータ (c，x，y) である．データ間の制約関係に整合性があれば実線の枠，整合性がなければ点線の枠で示す．

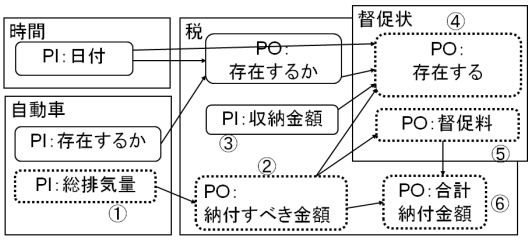

図 6　単方向性制約伝搬の具体例
Fig. 6　Example of Unidirectional Constraint Propagation.

① まず，データ b を変更する．すると，データ (a，b，c) の制約関係は不整合となる．このとき，変更すべきデータは PO つまり，データ c である．
② PO であるデータ c を変更し，データ (a，b，c) の制約関係を整合にする．また，c の変更により，データ (c，x，y) の制約関係は不整合となる．
③ PO であるデータ y を変更し，データ (c，x，y) のデータ間の制約関係を整合にする．最終的にすべてのデータ間の整合性が保たれる．

単方向性制約伝搬の具体例として，地方税法の自動車税の例を**図 6** に示す．自動車税の場合，自動車の総排気量・車種などから税額を決定する．① 総排気量を更新すると，総排気量（PI）と納付すべき金額（PO）の制約の整合性がなくなる．整合性を保つため② 納税すべき金額を更新する．このように制約が単一方向に伝搬することで業務データの整合性を常に保つ．

5.　税務処理プログラム構成の手順

因果関係を導入した制約表現に基づき，税務処理をオブジェクト指向言語のプログラムとして実装する手順を下記に示す．下記の手順は，実験で実装したシステムを作成したときの手順を基に作成している．

- **STEP1 データクラスの作成（設計）**
  業務で取り扱うデータオブジェクトの関連・属性・操作を表すため，データクラスのクラス図を作成する．地方税の税務処理のデータクラスを例に上げると，**図 7** 左の「データ」の部分がデータクラスの一部である．
- **STEP2 制約表の作成（設計）**
  業務規則から制約表を作成する．表のセルの内容は，数値，真理値，計算方法のいずれかとする．表 1 が制約表の一例である．
- **STEP3 表に対し因果関係の決定（設計）**
  その制約表においてプライマリインプットとプライマリアウトプットを決定する．すると，因果関係を導入した制約表が作成される．表 2 が因果関係を導入した制約表の一例である．
- **STEP4 データクラスを実装（プログラム実装）**
  データクラスのクラス図を，そのままプログラムの

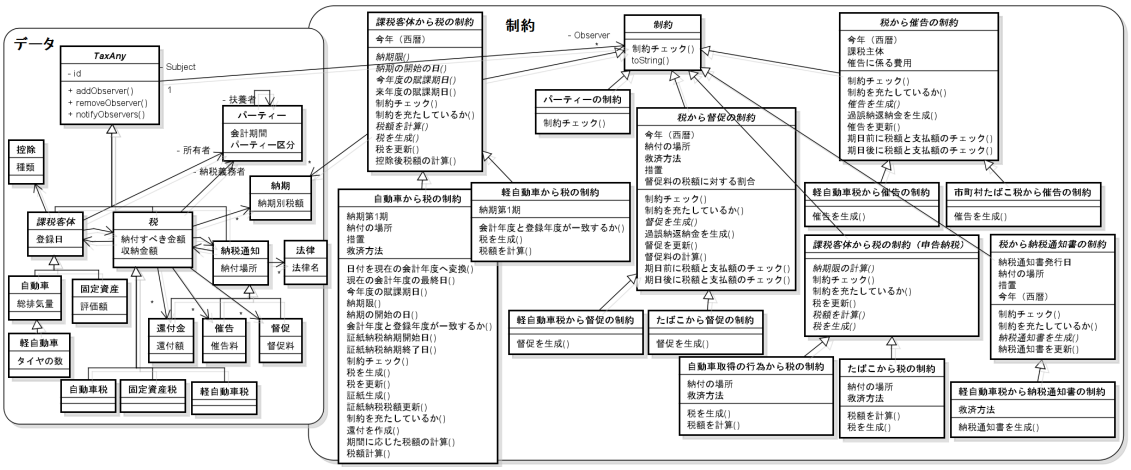

図 7 データと制約のクラス図（一部省略）
Fig. 7 Class Diagram of Data Object and Constraints(Details are omitted.)

クラスとして実装する．ただし，すべてのデータクラスは，オブザーバパターンの Subject クラス（例えば TaxAny クラス）に必ず継承する．また，データクラスはそれぞれ複数の属性を持っており，属性それぞれに対して setter メソッド，getter メソッドを持っている．さらに，setter メソッドの最後の行には必ず，Observer クラスの制約チェックメソッドを呼び出す処理を記述する．

- **STEP5 制約クラスを実装（プログラム実装）**
  1 つの制約（例えば，表 2）に対して，1 つのクラスを実装する．ただし，それぞれの制約クラスはオブザーバパターンの Observer クラス（例えば Constraint クラス）に継承する．制約クラスは checkConstraint() メソッド（制約チェックメソッド）が必ず記述されている．
- **STEP6 制約表を基に，制約に条件分岐を実装（プログラム実装）**
  因果関係を導入した制約表を基に，クラスの中の制約チェックメソッドに，条件分岐を作成する．ただし，特定の計算モジュール，及びプライマリアウトプットを作成・更新するモジュールは，制約チェックメソッドから，他のメソッドまたは他のクラスのメソッドへ分離する．図 7 右の「制約」の部分が，制約クラスとして最終的に作成されるクラス図の一部である．

6. 実験

6.1 地方税税務情報処理システムの実装

提案手法を用いて，京都府と京田辺市の条例 [10][11] が定める 23 種 [*3] すべての地方税について，基本的な税務処理が 1 人によって 4 ヶ月ほどで実装できた．実装言語は Scala と Java である．自動車税は GUI（Graphical User Interface），その他の税はコンソールを用いて，基本的な税務処理が正確に動作したことを確認した．その結果，本実験のシステムは対話処理がなく税務処理が実現できた．したがって，単方向性制約伝搬メカニズムによって，対話処理が削減できるとわかる．税務処理は具体的に，

- 税の賦課・督促・催告・還付の計算（23 種の税）
- 納税通知（7 種の税）

を適切な日付に行った．ただし固定資産税・住民税・事業税について税額の正確な計算を行っておらず，税額の値（正確には課税標準の値）を定数としている．また，本システムには実装できていない処理がある．自動車税を例に挙げると，4 月 1 日の賦課日以降に生じた新規改造などによる登録変更処理は実装できていない．この処理は，4 月 1 日以前の初期登録時に登録に誤りがあった場合の修正処理として実行されてしまう．

6.2 実装したシステムの構成

実装したシステムの構成を**図 8** に示す．次の 5 つのサブシステムで構成される．

- ユーザインタフェースサブシステム…ユーザインタフェースからユーザの入力情報を受け取り，その入力情報をオブジェクト管理サブシステムに揮発性オブジェクトとして保存する．またはその入力情報を基に，オブジェクトを参照・削除する．またはその入力情報を基に，オブジェクトの属性を更新する．
- オブジェクト管理サブシステム…自動車や所有者などのデータを揮発性オブジェクトとして保存する場所．オブジェクトの参照・保存・削除の機能を提供する．
- 制約チェックサブシステム…入力された揮発性オブジェクトの属性を見ながら，制約に基づきオブジェクトの生成や属性の更新を行う．将来的に制約チェックを行う場合に，タイミングホイールサブシステムに保

*3 地方消費税は評価対象から除いた．当該都道府県の地方消費税は国が徴収する，かつ他の 46 都道府県への支払額計算のみのためである．実装した税種は，道府県税は 17 種，市町村税は 6 種．

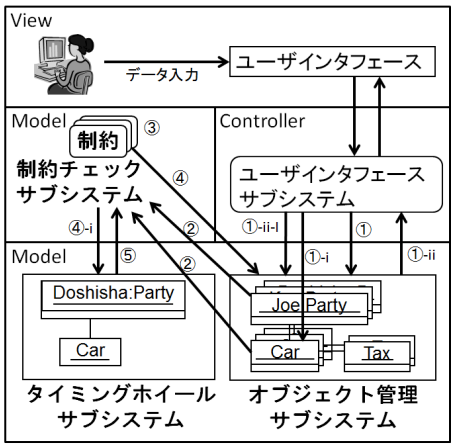

図 8　実験で実装したシステムのブロック図
Fig. 8　Block Diagram of Prototype System.

表 3　作成したすべてのクラスの種類別ステップ数
Table 3　Number of Lines for Prototype System.

制約	データ オブジェクト	オブジェクト管理 サブシステム	DAO*4 クラス	合計
2,776 (749)	1,342 (552)	557*5	993	5,668

存する．

- タイミングホイールサブシステム…日付と紐付けて揮発性オブジェクトを保存する場所．日付が変わると，現在の日付に紐付けて保存されているオブジェクトすべてを制約へ伝搬させる．

6.3　クラス図とステップ数

本実験で作成したデータクラスと制約クラスの関係を表すクラス図を図 7 に示す．ただし，図 7 では簡略化するためクラス・属性・操作を一部省略している．作成した全クラスの総ステップ数は，図 8 のユーザインタフェース（コード自動生成）とユーザインタフェースサブシステム（自動車税では約 300 行）を除き 5,668 行となった．

作成したクラスのステップ数を種類ごとに測定した．その結果を**表 3** に示す．ただし，ユーザインタフェースとユーザインタフェースサブシステムのステップ数は除く．表 3 の制約・データオブジェクトの括弧内は，制約・データオブジェクトのうちスーパークラス，または日付の計算を行うために共通的に利用するクラスのステップ数である．

6.4　複雑な修正処理の実現

自動車税の税務処理のために作成した Graphical User Interface（GUI）を**図 9** に示す．このシステムを用いて，

*4　DAO（Data Access Object）パターンのクラス．オブジェクト管理システムにアクセスするために使用するクラス．
*5　タイミングホイールサブシステムを含む．タイミングホイールサブシステムは，93 行が新たに追加したソースコードであり，その他はオブジェクト管理サブシステムの一部のコードを再利用した．

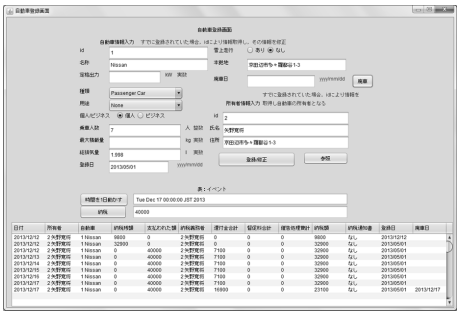

図 9　自動車税税務処理システム GUI
Fig. 9　Experimental User Interface for Car Tax.

自動車の総排気量や登録日，日付などの入力データを変化させて修正処理を確認した．単方向性制約伝搬によって実現が確認された処理として具体的には，

- 自動車登録日の変更による税額の再計算
- 所有者の変更によって，3 月であれば納税義務者の変更
- 自動車の属性変更による税額の再計算
- 自動車廃車日の変更による還付金金額の変更

などである．ユーザインタフェースサブシステムでオブジェクトの属性を変化させる処理のみによって，税額計算後に入力条件を修正するなどの複雑な処理が，付加コードなしで実現できた．

7.　考察

7.1　提案する制約表現のメリット

因果関係を導入した制約表現のメリットは 4 つある．

- 業務知識とプログラムのソースコードとのセマンティックギャップが小さいため，開発効率・保守性が共に高い．従来の本質モデルでは，真理値表（例えば，表 1）の複数行それぞれが，手続き的なプロセスに分散していた．真理値表が一つのモジュールとして実装できれば，セマンティックギャップが小さくなる．また，業務規則が凝集しているため，社会情勢の変化によって業務規則が変更されても，メンテナンスが容易となる．
- 業務規則の網羅的な記述が容易にできる．制約には，業務規則のまとまった単位をそのまま記述するためである．また，制約の記述が必要十分条件となっているためである．
- モジュラリティを高く記述できる．必要十分条件となっているため，制約を記述するときに，他の制約の記述を予想しながら記述する必要がない．
- システムにすでに入力したデータの修正処理を，プログラムに個別のモジュールとして記述する必要がない．最初に入力されるデータと，すでに入力したデータは，イベント駆動制御という同じ処理の中で，制約を常に満たすように正しい値になる．

7.2 課題

最初に，提案したアーキテクチャが，税務処理以外の業務ドメインに適応できるかが課題である．税務処理は単方向に処理を行う．具体的には，初期データの入力，課税処理，納税通知書の発行，督促状の発行，催告状の発行，必要ならば強制執行を行う．また，地方税法では複数の税を一緒に処理せず，それぞれの税務処理は独立している．さらに，住民が複数の自動車を持っていることにより，割引を受けることはない．税務処理はこのような特徴を持っているため，提案するアーキテクチャと親和性が高い．そのため，税務処理以外の業務ドメインに適応できるかどうかを確認する必要がある．しかし，一般的な業務処理は処理内容が税務処理と似ているため，適用できる可能性が高い．

2 番目に，実験で実装したシステムでは，実装できていない処理があることである．単方向性制約伝搬のイベント駆動制御では，すでに入力したデータの修正処理は，6.4 節に示したように，新たなコードを追加することなしに実現できることがメリットである．しかし，自動車税を例に挙げると，実験で実装したシステムでは，自動車の改造による変更処理が実装できていない．具体的には，4 月 1 日以降に所有者が自動車の総排気量を改造したため，4 月 15 日にシステムに変更登録する．この場合，今年の税額を変更せず，来年度の税額に適応されなければならない．しかし，改造による変更処理は，登録自体に誤りがあった場合の修正処理として実行されて，今年度の税額を修正してしまう．提案する制約表現では，改造による変更処理と，誤りがあった場合の修正処理に区別がつかない．そのため，ここで対話処理が必要となると考える．

今後の発展としては，単方向性制約伝搬のすべての処理は繰り返し構造であるという特性を利用し，XML(Extensible Markup Language) あるいは DSL(Domain Specific Language) を用いて制約を記述することで，ソースコードから分離したい．このようにすれば，ソフトウェア開発者の負担を更に減らすことができると考える．

8. まとめ

本稿では，業務システムのアプリケーションアーキテクチャを提案し，因果関係を導入した制約表現から税務処理システムを構成する手順を提案した．税務処理システムに適応できたことから，業務システムにも適応できる可能性が高い．アーキテクチャでは，単方向性制約伝搬というイベント駆動制御に基づき業務処理を実現する．イベントが起こるたびに，データを制約へと単方向に伝搬させ，データが常に制約を満たす．そのため，データは正しい値になり，業務処理が実現できる．伝搬の実現に，GoF のオブザーバパターンを用いた．アーキテクチャでは，アナリシスパターンを活用することで，操作するデータを「操作レベル」に，制約を満たす処理を「知識レベル」にクラスを適切に分割できる．

提案手法の評価のため，京都府と京田辺市の条例 [10][11] が定める 23 種すべての地方税について基本的な税務処理をオブジェクト指向言語 Scala と Java を用いて実装した．その結果，23 種の基本的な税務処理が，約 5,700 行（固定資産税，住民税，事業税の細かい税額計算は省略）で実装できた．従来の手続き的記述に比べ，コード量が少なくなった．また，個々の業務規則が 1 つのプロセスに凝集した．したがって，開発効率・保守性が共に高い．提案する表現は，要件部と効果部から成る業務規則をそのまま変換した形式になっており，セマンティックギャップが少ないためである．また，すでに入力したデータの修正処理は，個別にコードを追加することなく，制約表現の実装のみで実行できた．これは，提案するアーキテクチャにおいて，イベントが起きるたびにデータを制約でチェックし，システム内のすべてのデータが常に正しい値に保たれるためである．

参考文献

[1] DeMarco, T.: *Structured Analysis and System Specification*, Prentice Hall (1979).
[2] 石井　恵，金田重郎：制約型プログラミングによるオフィス処理の実現，情報処理学会論文誌，Vol. 38, No. 7, pp. 1284–1295（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110002721580/⟩ (1997).
[3] Ishii, M., Sasaki, Y. and Kaneda, S.: A constraint-satisfaction approach to clerical work, *Intelligent Systems and their Applications, IEEE*, Vol. 15, No. 1, pp. 64–72 (online), DOI: 10.1109/5254.820331 (2000).
[4] 桑山浩希，中西義尚，金田重郎：単方向性制約伝搬に基づく業務システム構築法の提案，第 75 回全国大会講演論文集，pp. 771–773 (2013).
[5] 田中規久雄：法律効果規定部の意味機能について，情報処理学会研究報告. 自然言語処理研究会報告， Vol. 98, No. 21, pp. 1–8（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110002934698/⟩ (1998).
[6] McMenamin, S. M. and Palmer, J. F.: *Essential Systems Analysis*, Yourdon Press (1984).
[7] 総務省：地方税法（昭和二十五年七月三十一日法律第二百二十六号），総務省（オンライン），入手先 ⟨http://law.e-gov.go.jp/htmldata/S25/S25HO226.html⟩（参照 2014-1-16）.
[8] マーチン・ファウラー（著），堀内一（監訳），友野晶夫（訳）：アナリシスパターン [新装版]，株式会社ピアソン・エデュケーション (2002).
[9] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional (1994).
[10] 京都府：京都府府税条例　京都府条例第 42 号，京都府（オンライン），入手先 ⟨http://www.pref.kyoto.jp/reiki/reiki_honbun/a3000295001.html⟩（参照 2014-1-16）.
[11] 京田辺市：京田辺市税条例　条例第 22 号，京田辺市（オンライン），入手先 ⟨http://www.kyotanabe.jp/reiki/reiki_honbun/k113RG00000242.html⟩（参照 2014-1-16）.