7S－1

内包的テンプレートによるソフトウェア記述プロセスの実現

森谷俊之　村尾　洋　榎本　肇
芝浦工業大学

１．はじめに（Tell から Tell Scoops へ）

この研究はヒューマン・インタフェースに重点をおいたソフトウェア開発環境を提供することを目的としているが、その様なアプローチの１つにソフトウェア開発支援環境「Ｔｅｌｌ」がある。

「Ｔｅｌｌ」は、自然言語風の文章で使用する語の定義を行い、ソフトウェア・プロセス中でその制約条件などを箇条書きで記述し、語彙分割法と呼ばれる詳細化手法を用いて逐次記述していくことで、クラス、動的クラス、動作、関数のカテゴリーに基づき、オブジェクト固有の語句の定義を原始的な語句へと分解して定義しながら仕様を詳細化していくことでオブジェクトを実現する。「Ｔｅｌｌ」における自然言語による仕様記述のアプローチは、ソフトウェア記述プロセス間で、表記されたオブジェクトを中心として人間にとって望ましい表現で仕様化を行っている。

しかし、この様な目的を達成した「Ｔｅｌｌ」は、自然言語をそのままあるいは若干の制約の下に、計算機言語として採用することで、仕様記述を行う環境を構築しているが、仕様から並列手続きへの生成過程については十分に検討が進んでいなかった。

本研究では以上の研究成果に基づき、仕様表現で用いられる表現記法を採用し、さらにオブジェクト指向のパラダイムを取り入れた新たな言語のスタイルおよびそれに最適なソフトウェア開発環境「Ｔｅｌｌ　Ｓｃｏｏｐｓ」([Total ELaboration Language Sentence Constraint and Ordered Oriented Programming Supports] 以下 Tell-Scp と略す) を構築した。

２．「Ｔｅｌｌ　Ｓｃｏｏｐｓ」の概要

２．１　仕様の表現

Tell-Scpにおける仕様の記述の単位となる文は、「Ｔｅｌｌ」と同じく単文と制限する。これにより、自然言語の持つ構文解析の複雑性を回避できる。特に Tell-Scp は自然言語風な表現をユーザーの要求するオブジェクトとその詳細をシンボライズするために用いる。従って、Tell-Scpは Syntax Analyzer を持たない。

仕様で記述される単文は、シンボライズされたオブジェクトをモジュールとして捉える。それらの文はウィンド・システムで実現された文推敲用エディターなどのユーザー・フレンドリーなインタフェースによって記述し、ソフトウェア設計の道具立てを行っていくために記述される。

この言語での前述のようなオブジェクトの制約条件を表現する文は、オブジェクトの属性値間関係を構文にある一貫的制約を利用して単文構造で表現し、それを動詞、Prolog でいうところの述語でシンボライズする。

さらに、オブジェクトを表現するために箇条書きで用いられる個々の文の表現には、対象指向的な思考を取り入れ記述する。具体的に言えば、文における動作を行う主体としての主語は特定のクラスに属するオブジェクトであり、そのメソッドを動詞が決定する、という表現形式がこの言語では貫徹されている。

このような自然言語風にシンボライズされた制約条件を、個々に新たなオブジェクトと考えることでオブジェクトを抽出し、内包的記述を続けていく。それに用いられている語および文のもつ記述形式の整合化を行う枠組みを内包的テンプレートを用いて行う。この関係構造は、人間がそのオブジェクトに対して抱いている自然な概念の階層化のメタファといえる。

Tell-Scpは、さらにソフトウェア記述プロセスでの個々の対象記述をモジュールとして扱い、それをTell-Scpのシステムクラスとして登録していく。

２．２　仕様とＭｉｓｓｉｏｎ

この言語においてオブジェクトを実現するための表現は、仕様(Specification)と制御情報(Mission)の２つのカテゴリーに分類される。仕様とは、オブジェクトが何を行うべきか、オブジェクトの制約条件を記述したものである。仕様の実現のため、Tell-Scpは制約条件が真となるように処理を進めていく。つまり、Tell-Scpにおける仕様を表現する文の解釈手順は、Prolog の実行に類似している。しかし、この様な手法は実際の手続きプログラムと比較して実行効率が悪いことは周知のことである。

そこで、Tell-Scpでのソフトウェア記述プロセスは、個々に仕様を推敲し記述を行った上で、仕様にある種の制御情報、つまりどのようなタイミングであるいは順序で行うべきかを記述するMissionがある。制御情報は仕様に単文で記述された制約条件の解釈のシーケンスを制御するための条件や手順を記述しているためにオブジェクトは効率的に実行される。

Mission の記述もまた単文で表現され、用いられている動詞などの語彙を仕様に記述されている単文と参照させ、仕様の制約条件のシーケンスを制御させる。

２．３　Ｍｉｓｓｉｏｎの要素

Mission は３つのカテゴリーから構成されている。

（a）　ｃｏｎｆｉｇｕｒａｔｉｏｎ

Mission で制御を行うために必要な値や状態を取り扱う動的構造体を宣言する。ここで宣言された構造体は、他の仕様及び Mission から情報隠ぺいされる。

（b）　ｏｒｄｅｒ

仕様上に現れるオブジェクトの持つ値の順序指定を行う。

（c）　ｔｉｍｉｎｇ

この仕様の動作前、動作中、動作後であるための条件を記述する。記述される動作条件は、インスタンスの持つ内容だけでなく、オブジェクトを構成するインスタンスや仕様を制約している文やインスタンスの activity によってデータ駆動的に決定させることも可能である。

３．仕様・Ｍｉｓｓｉｏｎとクラス

Tell-Scpでは、仕様と Mission （以下、これらを総称してドキュメントと呼ぶ）がシステムクラスを構成し、通常の動的メソッドも対応する仕様と Mission の統合の結果として生成される。そのドキュメントとインスタンス変数、システムクラスの関係を図１に示す。

さらにTell-Scpでは、システムクラスをその性質によって静的・動的という直交的関係を持たせ、静的なクラスと動的なクラスの２

Realization of Software Descriptive Process by Using Intentional Template.

Shibaura Institute of Technology

つに分類する。静的なクラスでのメソッドは Function として表現され、そこでのドキュメントは単一の Mission しか持たないのに対し、動的なクラスでのドキュメントは Mission を複数持ち、Mission そのものを動作前、動作中、動作後の動的条件に応じて制御できる。動的なクラスでのメソッドは action と呼ばれる。この様な分類は、記述パラダイムとして非常に役立つと考えられる。

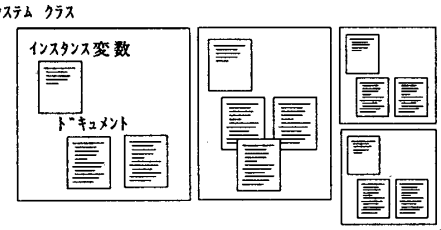

図1　システムクラスとインスタンス変数、ドキュメントの関係

4．「Tell Scoops」の構成

上述のような概念に立脚し、Tell-ScpのユーザーはXウインド上でソフトウェアを開発する。Tell-Scpのソフトウェア記述プロセスの概念と支援環境との関係を示すTell-Scp全体の概念図を図2に示す。

4.1 Prologコードへのトランスレートと実行

「2.2 仕様とMission」から明らかであるようにTell-Scpでの記述表現は Prolog のソースにトランスレートされ、オブジェクトは Prolog インタプリタ上での Prolog ソースの実行により行われる。

図2のように、Tell-Scpは変換プロセッサにより、ドキュメント毎に仕様は上述のような視点に則り、1階論理の Prolog コードにトランスレートされ、Mission はメタ論理述語を利用し、仕様に対してメタとなる Prolog コードを生成する。その様にして作られた Prolog コードもクラス同様の構造で管理される。

実行時には、実行マネージャーが生成された Prolog コードを読みとり、オブジェクト固有のインスタンスを管理しながら、オブジェクトの生成、オブジェクト間のメッセージ・パッシング、そしてオブジェクトの消滅というフェーズでソフトウェア記述プロセスの目的を達成する。

4.2 ユーザー・インタフェース

単文を詳細化する過程で語間依存関係を明確にするために、ユーザーフレンドリーなインタフェースが不可欠となる。図2に示されている現時点で用意されている支援環境を以下に示す。

a）ブラウザ

smalltalkにあるようなブラウザは大変強力なものであるが、今回は時間的な制約から単にクラスの構成や特定のクラスのドキュメントの内容を参照できる程度の簡易なものである。

b）文推敲構造化エディター

ドキュメントは、仕様の箇条書きや Mission の記述など、限定されたフォーマットで記述される。このエディターは、それらをカテゴリー毎に記述するためのエディターである。このエディターは単に記述をおこなうだけではなく、仕様と Mission との整合を取るための仕様にある語彙の依存関係を表示するなど、Tell-Scp 上での記述に適した機能も持っている。

c）その他

そのほかに変換プロセッサーや実行プロセッサーなど、図2にあるようなシステムを制御するための機能が必要最小限用意されている。

5．評価

今回のバージョンは、コンカレントに処理を行うシステムの準備段階として、Tell-Scpの仕様としてはコンカレンシーを意識しているにもかかわらず、上述のような実行をシングルプロセスで行うものである。記述プロセスは、オブジェクト単位に独立しているので将来の拡張の準備としては問題ないと考えられるが、制御情報の処理などシングルプロセスで実行する故に、特に複雑なオブジェクト間関係の処理をおこなうことが困難である点や前述の理由でオブジェクト毎の Prolog コードがファイルとして展開されており、実行プロセッサーが実行時に逐次そのファイルを参照する為、実行速度の評価がそもそも不可能であるという点でこのシステムが実用になるには、より一層の研究が必要であろう。

しかし、本研究の主題である自然言語風のシンボリックな記述によるソフトウェア記述プロセスの実現という点では、満足する結果を得ることが出来た。

6．あとがき

このシステムが、人間と計算機との有機的構造を持つインタフェースを備えた総合的ソフトウェア開発環境足り得るためにコンカレントなオブジェクト間のインタラクションは不可欠である。今後は、超並行処理に重点をおきプロセス間インタフェースのプロトコルやその意味ネットワーク上で発生する輻輳問題の検討を深めて行きたい。

最後に、本システムのマン・マシン・インタフェースの製作およびシンタックス・データベースの製作には、芝浦工業大学の飯塚、増山の両氏の協力により行われた。また、本研究は科研費重点領域研究（課題番号 02249207）の一部の援助のもとに行われた。記して謝意を表する。

[参考文献]
(1) Hourai N., 「Integration Method for Specification Process」 1st Int. C.on SI.,1990
(2) Enomoto N., 「Software Development Process from Natural Language Specification」 11th Int. C.on SE.,1989

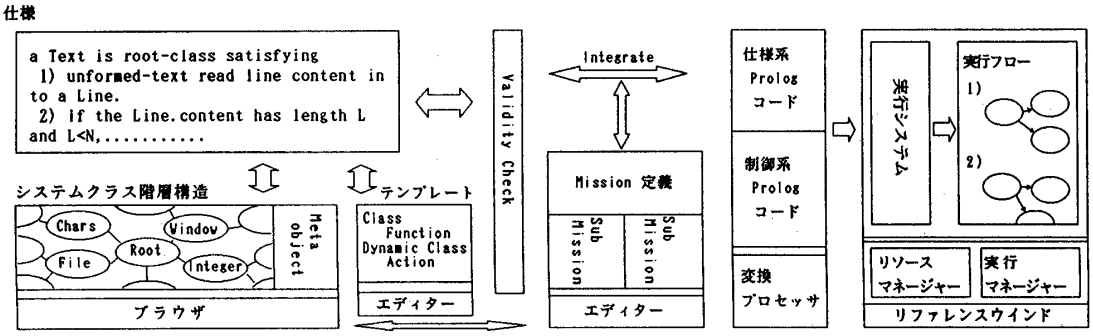

図2　Tell Scoops の概念図