# 論理型言語処理系「ＬＯＮＬＩ」における モジュール化機能

3E-7

広瀬 正* 品田 啓子* 竹内 洋一* 加藤 整** 中尾 和夫*
*(株)日立製作所 システム開発研究所 **日立ﾏｲｸﾛｺﾝﾋﾟｭｰﾀｴﾝｼﾞﾆｱﾘﾝｸﾞ

## 1. はじめに

「ＬＯＮＬＩ」(Logic OrieNted Language Inferencer)は、実応用への適用を目的として、他言語プログラム結合機能などを拡張した論理型言語処理系である[1]。汎用大型計算機（ＨＩＴＡＣ Ｍシリーズ）およびワークステーション（２０５０）上で稼動する。いくつかの応用システムの試作を通して得た機能拡張要求の中で、特にモジュール化機能に関して報告すると共に、１つの提案を行う。

## 2. システム記述言語ＬＯＮＬＩ

prologを知識表現言語そのものとしてでなく、より基礎的なシステム記述言語（核言語）としてとらえた時、モジュール化機能に関して以下の問題が生じる。

(1)述語名称の衝突 現状の仕様では述語名称に対して単一の命名空間しか与えておらず、特に大規模プログラム開発時に問題となる。いわゆる「情報隠蔽原理」のための機能がないという問題である。実行制御機能が「cut」など単純なものに限定していることが、多くの述語名称を生む結果となり、この問題に拍車かける傾向にある。

(2)閉世界仮定の制御 メタプログラミング技法を用いること、とくに既存のプログラムにそれを制御するメタプログラムを追加することが、現状の仕様では記述しにくい。これは(1)の問題とも関連するが、記号処理言語としての特質を生かしたプログラミングを奨励するために重要と考えている。

(3)項シンボル管理 論理型言語において印字名称の管理は処理系が自動的に行う機能であるが、その処理コストが大きいので、可能な範囲でのユーザによる制御が望まれる。特に、外部データベースを扱う部分では、印字名称の登録、削除（識別子回収）の頻度が大きく、ニーズも高い。

(4)スコープ範囲情報の有効利用 述語名称の有効範囲が閉じたプログラム部分に限定されていることがあらかじめ判ると、その情報に基いた述語呼び出しの最適化ができる。ユーザの負担にならない形式でなら、性能情報をも記述できるほうがよい。

## 3. 設計方針

以下の点を考慮して設計した。

(1)BASICな機能だけを提供し、できるだけユーザのコントロール範囲を広くとる。

(2)インタープリティブ実行・コンパイルド実行間の連続性（互換性）を保つ。

(3)従来システムの仕様との上位互換性を保つ。

(4)スコープ範囲情報を処理系高速化に利用する。

(5)分散型知識情報処理システムにおける LONLI-LONLIインターフェイスとの協調を図る。

ここで(5)は、ホストーＷＳあるいはＷＳ－ＷＳ分散型の知識情報処理システムの実現を念頭においたものである。可能ならば、これら分散型知識情報処理システムはマシン・ＯＳ非依存に記述したい。LONLI言語で記述したプログラムはこれら分散型知識情報処理システムの構成に柔軟に対応できるものにしたい（図１）。

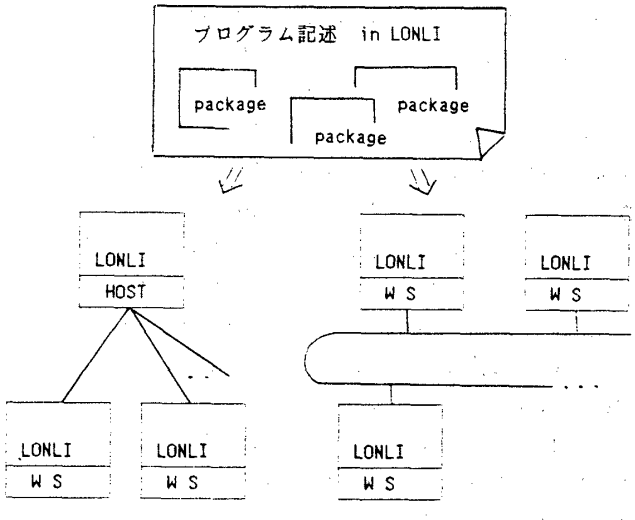


図１ 分散型知識情報処理システム構成

Moduler Functions for Logic Programming Language LONLI
Tadashi HIROSE, Keiko SHINADA, Youich TAKEUCH, Osamu KATO, Kazuo NAKAO
HITACHI, Ltd. HITACH Micro Computer Engneering Ltd.

## 4. 2レベルのモジュール化（packageとmodule）

モジュール化レベルとして次の2レベル持たせる。

package ： 項シンボル管理の単位。assert/retractの有効範囲。

module ： 述語名称のスコープ範囲。

packageとmoduleは、図2に示す様に、相補的な関係を持つ。moduleは外部からの不当な干渉を防ぐための枠組みである。特別に許された述語名称（public宣言）に対する呼び出しだけが許される。packageは外部への不当な干渉を発生させないための枠組みである。特別な手続き（inter_package_call）を用いないと外部に存在するpackageをアクセスできない。

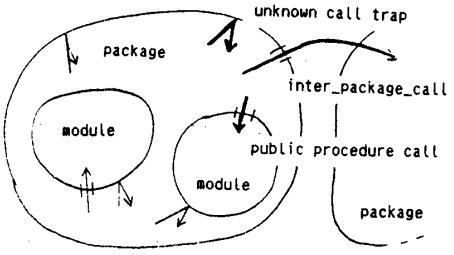


図2 package と module

1つのpackage内は従来システムと同一仕様である。package単位に分散したシステム上に配置できる。package単位に項シンボル管理を行う。つまり、package内では、同一の言葉が使われることを意味する。一方、package間の交信には翻訳が必要である。処理系はこの翻訳機能を提供する。packageが異なるシステム上に配置された場合は、さらにシステム間更新機能が提供される。moduleの主な機能は「スコープの限定」である。これらの機能を分離してユーザに提供することで、システム記述言語（処理系）として必要なきめの細かい機能提供が図れる。

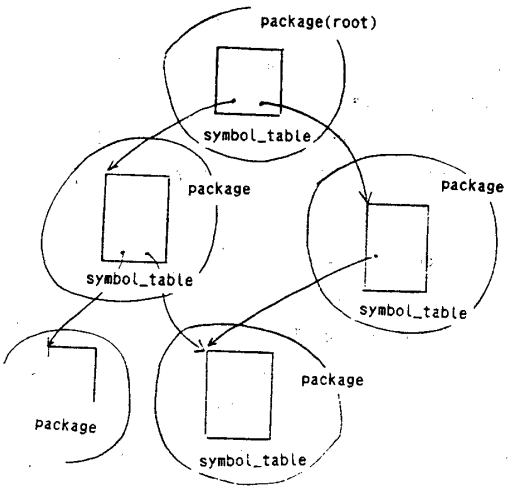


図3 package 構成

## 5. インターフェイスとインプリメンテーション

pcakage管理は表1に示す組み込み述語で行う。システム立ち上げ時にrootという名称のpackageが生成され、それへ制御が移される（"current package"とする）。ユーザは必要に応じてpackageを生成し（create_package）制御を移す（change_package）。package間の関係は基本的にはtree構造だが、network構造とすることもできる（ link_package)（図3）。packageにまたがる呼び出しはinter_package_call述語で行う。通常は、オペレータ宣言機能を用いて<package_name>:<predicate_call>と記述している。package内に存在しない述語呼び出しが発生すると、そのpackage内のunknown_call(X)述語が呼び出される。

module管理は表2に示す組み込み述語で行う。public宣言された述語だけがmoduleの外から呼び出せる。従来仕様との互換性を確保するため、"open"/"close"という概念を導入した。"open"はそのmoduleのすべての述語に対してpublic宣言がなされていると見なす。従来仕様のconsult,compileはそれぞれopen_consult,close_compileと等価である。moduleの指定単位は物理的単位"file"である。1 file内で"open"/"close"の制御を行うために述語open_module/close_moduleがある。

| | | |
|---|---|---|
| create_package(<package_name>) | : | 生成 |
| change_package(<package_name>) | : | 制御移動 |
| link_package(<package_name>) | : | リンク |
| inter_package_call(<package_name>,<predicat_call>) | : | 呼び出し |
| current_package(<package_name_list>) | : | 状態表示 |
| current_child_package(<package_name_list>) | : | 状態表示 |
| unknown_call_trap | : | 未定義割込 |
| remove_package(<package_name>) | : | 消去 |

表1 package管理組み込み述語

## 6. おわりに

論理型言語の試用経験から得たモジュール化機能に関するいくつかの問題点を示し、システム記述言語処理系の立場から1つの提案を行った。提案したpackage機能は、多重世界機能[2]などと似ているが、packageをそのまま知識表現でいう"world"や"frame"として用いることを期待しているものではない。packageは（論理的な意味を含めて）分散型知識情報処理システムのためのシステム構成要素である。

| | | |
|---|---|---|
| open_consult(<file_name>),consult(<file_name>) | : | open読込 |
| open_compile(<file_name>) | : | openコンパイル |
| close_consult(<file_name>) | : | close読込 |
| close_compile(<file_name>),compile(<file_name>) | : | closeコンパイル |
| open_module | : | open制御 |
| close_module | : | close制御 |
| public(<predicate_name>/<arity>) | : | 外部宣言 |

表2 module管理組込み述語

参考文献 [1] 広瀬、他：論理型基本処理系「LONLI」の機能と効果、情報処理学会第29回大会 4P-10(1984)
[2] 中島、他：prolog/KR からUranusへ、情報処理学会知識工学と人工知能36-2(1984)