記　号　処　理 72－2
（1994．1．14）

TAOのパッケージシステム

中村昌志　天海良治　山崎憲一　吉田雅治　竹内郁雄

NTT基礎研究所, NTTヒューマン・インタフェース研究所

実時間記号処理カーネルTAO/SILENTのパッケージシステムについて述べる．言語TAOは記号処理専用マシンSILENT上でOSとしての役割を果たすため，一般的なプログラミング言語の機能の他に，OSの機能を併せ持つ．TAOのパッケージシステムでは，パッケージとシンボルの保護機能およびインクリメンタルリハッシュがOSの機能である．一般的なプログラミング言語の機能としては，パッケージパスによるパッケージの継承がCommon Lispと異なった特徴的なものとなっている．これらの機能を組合わせることにより多様な構造の名前空間を実現することができ，マルチユーザ環境や複数言語の共存などに対応できる．

Package System in TAO

Masashi Nakamura　Yoshiji Amagai　Kenichi Yamazaki
Masaharu Yoshida　Ikuo Takeuchi

NTT Basic Research Laboratories, NTT Human Interface Laboratories

This paper describes the package system of the language TAO on the real-time symbolic processing kernel SILENT. TAO is a programming language, but it also fills the role of an OS. The protection mechanism of packages and symbols, and incremental rehashing are OS functions in the TAO package system. From the viewpoint of a programming language, the package inheritance of TAO with package path is distinct from that of Common Lisp. These functions give TAO the ability to implement various name space structures and the ability to handle the multi-user environment and the multi-lingual environment.

1 はじめに

TAO/SILENTは実時間記号処理カーネルである．ここで提供される言語TAOはLispの中に論理型プログラミング，オブジェクト指向プログラミング，並行プログラミング，実時間プログラミングの機能を融和したマルチパラダイム言語である．

TAOはSILENTの機械語であるので，一般ユーザの便宜を少々犠牲にしても言語の簡潔性と実行効率を上げる方針で設計されている．その効率性のもとで，TAOはマルチパラダイムの様々な機能，OSとしての機能などを兼ね備えた多機能かつ高実行効率な言語となっている．一般ユーザ向き言語はTAOの上にシステムユーザが作る言語族によって実現される．

TAOが機械語であるという性質とOSとしての機能の双方を兼ね備えている点はパッケージシステムにも反映されている．すなわち，TAOのパッケージシステムはCommon Lispなどのパッケージに比べて単純，簡潔にできている．また，OSの資源保護の機能の一部として，保護レベルによるシンボルとパッケージの保護機能を備えている．これら2つがTAOのパッケージシステムの特徴的な点である．

2 TAOのパッケージシステムの概要

まずTAOのパッケージシステムのうち保護機能を除いた部分についての概要，およびシンボルの性質のうちパッケージに関わるものなどについて述べる．

2.1 パッケージの写像機能

シンボル名からシンボルへの写像というパッケージの機能を実現するプリミティブについて述べる．

TAOのプリミティブでは外部名からシンボルへの写像を行なう関数をパッケージ名からパッケージへの写像機能と，シンボル名からシンボルへの写像機能とに分けて提供する．readでは2つの写像機能を合わせて，外部表現からシンボルへの写像機能を実現する．

TAOではシンボル名からシンボルへの写像のプリミティブとして以下の関数を用意している．

(find-symbol *symbol-name package*)
(intern *name package*)

両者はともにシンボル名とパッケージの2つを引数にとりシンボルを返す関数である(internはシンボルの生成ができる)．

パッケージの生成，消去のためのプリミティブは

(make-package *name* [*protection*] [*allowance*])
(delete-package *package*)

である．関数make-packageはパッケージ名からパッケージへの写像の一種になっており，パッケージを得るための一手段である．delete-packageはパッケージを消去する関数である([ ]内のオプションは後述)．

また，パッケージ名から既存のパッケージへの写像のプリミティブとしては

(global-package *package-name*)

がある．global-packageはパッケージ名からグローバルパッケージとして登録されたパッケージを探す関数である．グローバルパッケージでないパッケージをパッケージ名から探す手段はシステムに用意されていないので，このようなパッケージをパッケージ名から得ることは(自分で生成したパッケージを除いて)難しくしてある．後で述べるパッケージの隠蔽にこの性質を利用している．

2.2 パッケージパス (package path)

Common Lispではcurrent packageの概念があるため，パッケージ名を省略してシンボル名のみでシンボルを参照することができる．また，current packageに他のパッケージを継承することにより他のパッケージのシンボルもシンボル名のみで参照できる．

このようなCommon Lispの機能に相当する機能の実現のため，TAOはパッケージパスと呼ばれるパッケージのリストを用いた方法を提供する．

パッケージパスを利用してシンボルを探す関数として

(find-symbol-in-path *symbol-name package-path*)

がある．この関数は*package-path*の中でシンボルを探す．返されるシンボルは，*package-path*に並んでいるパッケージを前から順に探索し，最初に見つけたものである．したがって，同じ名前のシンボルが複数のパッケージに存在した場合，最初のもの以外に属するシンボルは無視される．

関数find-symbol-in-pathはシンボルが見つからなかった場合，シンボルを生成できる最初のパッケージを返す．このパッケージにinternすることよりパッケージパス内での新たなシンボルの生成が可能である．

internは指定されたパッケージにシンボルを生成する際に，パッケージパスとの関連を調べない．したがって，パッケージパス内に同名のシンボルが既に存在しても，指定されたパッケージに同名のシンボルがなければ新たなシンボルを生成することになる．

これを利用してパッケージパス内で既存のシンボルの前方に同名の新シンボルを作ることができる．この結果find-symbol-in-pathで既存のシンボルに代わって新シンボルが見えるようになる．これはread-read consistency

の規則を破ることになるのでCommon Lispでは嫌われていた性質であるが，TAOではむしろこの性質を利用することを考えている．

パッケージパス中のシンボルをunintern すると，intern の場合と逆に，後方のパッケージにあった同名のシンボルが見えるようになることがある．この場合にread-read consistency が破れることはCommon Lispでも同様に起きる．

2.3 シンボル

2.3.1 シンボルの表記法

TAOのシンボルはパッケージ名を *package*, シンボル名を *symbol* とすると，パッケージ名とシンボル名をコロンでつないで

package:*symbol*

と表記する．TAOのパッケージシステムではCommon Lispのような外部シンボル (external symbol) と内部シンボル (internal symbol) の区別はないので，コロンは1つだけである．key パッケージのシンボルについてはパッケージ名 key を省略して

:*symbol*

と表記する．

2.3.2 シンボルの大域属性

TAOのシンボルの大域属性は以下の5つである．

- symbol function
  シンボルに大域束縛されている関数
- symbol predicate
  シンボルに大域束縛されている述語
- symbol value
  シンボルに大域束縛されている値
- symbol symtab
  シンボルの属性表 (属性リストに相当)
- symbol package
  シンボルが属しているパッケージ

2.3.3 alias symbol

alias symbol はシステムやアプリケーションが提供する関数 / 述語などをユーザが自分用にカスタマイズしたい場合に使う．システムやアプリケーションのパッケージ内のシンボルは通常は保護されており，ユーザには変更不可能になっている．したがって，ユーザがシステムやアプリケーションの提供する関数 / 述語をカスタマイズする場合には，元のシンボルを別のパッケージにコピーして，そのコピーに対して変更を加えることになる．このようなコピーされたシンボルを alias symbol と呼ぶ．

alias symbol は関数

(make-alias-symbol *symbol package*)

で生成される．これは指定されたシンボルと同名の alias symbol を指定されたパッケージに作る．作られた alias symbol は大域属性のうち，symbol function, symbol predicate が元のシンボルと等しく，symbol value, symbol symtab は空になっている．

2.3.4 virus symbol

パッケージパス中のあるパッケージにシンボルを生成すると，それより後方のパッケージに属している同名のシンボルは (find-symbol-in-path では) 見えなくなってしまう．この性質を利用してパッケージパス中にあるシンボルを隠蔽する道具が virus symbol[1] である．

virus symbol は関数

(make-virus-symbol *name package*)

により名前を指定して作る．find-symbol-in-path がシンボル探索の途中で virus symbol を見つけると探索を打ち切ってしまい，シンボルが見つからなかった場合と同じ値を返す．

virus symbol が属するパッケージには，これを消して同名の新シンボルを生成できるので，パッケージパスの前方のパッケージに既存のシンボルと同名のシンボルを作る場合にも使える．

3 パッケージとシンボルの保護機能

3.1 保護レベル

TAOは保護レベルによりOSの資源保護の機能を実現している．利用者 (OSを利用するプロセス) は0～3の4レベルに分けて保護される．それぞれの保護レベルに対応する利用者 (プロセス) は表1のとおりであり，下位レベルの (0に近い) 利用者が上位レベルの (3に近い) 利用者の資源を操作することは禁じられている．

レベル	プロセス	対応するユーザ
3	システムコア	(SILENT 開発者)
2	言語族, ネットワーク等	システムプログラマ
1	AP, ツール類	AP プログラマ
0	ユーザプロセス	一般ユーザ

表 1: 保護レベルとプロセス, ユーザとの対応

[1] virus という名前は virtual irusu(居留守) に由来する．

3.2 パッケージとシンボルの保護方法

TAOでは，パッケージもシンボルも保護レベルを利用した保護方法をとる．例えばユーザプロセスが不用意にシステムのパッケージやシンボルを壊さないように，システム側とユーザ側で保護レベルに差をつけ，下位レベルのプロセスから上位レベルのシンボルやパッケージを操作できないようにしている．

シンボルの中にはいくつかのレベルのプロセスが共有するものがある．例えばシステムがユーザに提供するシステム関数のシンボル(例えばcarやcons)はユーザもアクセスできなければならない．しかし，ユーザプロセスがシステム関数を書き換えることは防ぎたい．これに対処するため，アクセスの制限と他の操作の制限は独立して設定できるようにしている．

一方，同じ保護レベル同士の場合，例えば一般ユーザAのプロセスが一般ユーザBのシンボルやパッケージを壊し，Bのプロセスも Aのシンボルやパッケージを壊す場合は，保護レベルだけでは防げない．そこで隠蔽を利用して保護を実現する．すなわちAのプロセスからはBのシンボルとパッケージが見えず，Bのプロセスからは Aのシンボルとパッケージが見えないようにしておけばよい．

以上の2つ(保護レベルと隠蔽)により，TAOのシンボルとパッケージは保護される．

3.3 パッケージに対するプロセスの権限

シンボルとパッケージを保護するためには，これらに対する操作を制限しなければならない．TAOでは効率性のため，制限の形態をパッケージ毎に定めることにした．例えばパッケージ毎にアクセス許可/不許可ビットを設け，そのパッケージに属するシンボルへのアクセス権限の判定にはこのビットを使うという方法である．この方法では，あるシンボルを「アクセス不許可」にするためには，それが属するパッケージのシンボル全てを「アクセス不許可」としなければならない．

以下では，効率性を追求するために権限の種類について検討した結果について述べる．

パッケージ，シンボル，シンボルの大域属性の3つそれぞれについて生成，消去，書き換え，アクセスの4つの権限を考え，この中から明らかに重複するものや無意味なものを除くと以下の7種類が残る．

1. パッケージの生成
2. パッケージの消去
3. パッケージ内のシンボルの生成
4. パッケージ内のシンボルの消去
5. パッケージ内のシンボルへのアクセス
6. シンボルの大域属性へのアクセス
7. シンボルの大域属性の書き換え

これらの7つの権限それぞれについて，許可/不許可を決めることにより制限の形態が決まる．TAOでは，このうちパッケージ生成は常に可能なので省略する．またシンボルへのアクセス(名指し)はシンボルの大域属性へのアクセスと合わせて使われることがほとんどなので，1つの権限にまとめる．パッケージ消去とシンボル消去も1つの権限にまとめる．この結果，必要な権限は以下の4つにまとめられる．

- accessibility (アクセス可能性) そのパッケージのシンボルを名指しできる．この権利がないのにfind-symbolを行なうとエラーとなるが，find-symbol-in-pathでは無視されるだけである．また，シンボルの大域属性にもアクセスできる．
- creatability (生成可能性) そのパッケージにシンボルを生成(intern)できる．これがないのに，internを行なうとエラーとなる．
- modifiability (属性変更可能性) そのパッケージのシンボルの大域属性を変更できる．なお変更できる大域属性はsymbol packageを除いた以下の4つである．

  symbol function, symbol predicate, symbol value, symbol symtab
- deletability (消去可能性) そのパッケージのシンボルおよびパッケージ自身を消去(unintern)できる．なお，uninternされたシンボル(どのパッケージにも属さないシンボル)はaccessible(ただし，もちろん名指しはできない)かつmodifiableとなる．

保護レベルによる階層化の意味を考えると，低い保護レベルのプロセスが高い保護レベルのパッケージに対する破壊的操作の権限であるcreatability, modifiability, deletabilityを持つべきでない．一方，accessibilityはパッケージの用途によって異なるはずである．例えば，システム内部専用に使うパッケージは低い保護レベルのプロセスにaccessibilityを与えるべきでない，しかしシステム関数など，システムが外部に提供するものを含むパッケージについては，低い保護レベルのプロセスにもaccessibilityを与えたい．

したがって，creatability, modifiability, deletabilityの3つとaccessibilityは別々に権限を判定としなければならない．権限の判定の簡略化，効率化のため，上記の

3つは保護レベルによってひとまとめで制限する．accessibility については保護レベルの低いプロセスも持ち得るよう，他の権限とは独立に設定できるようにする．

ほとんどのパッケージはこのような2種類の権限の設定により対処できるが，パッケージによっては対処しきれないものもある．keyパッケージのように，低い保護レベルのプロセスに creatability を与えつつ，どのプロセスにも deletability を与えたくないことがある．このような場合に対応するため，TAO は後述する消去不能パッケージを用意している．

3.4 パッケージとシンボルに対する権限の判定

個々のパッケージは保護に関する以下の3つの特性を持っている．

- 保護レベル
  protection level 0,1,2,3
- アクセス許容度
  access allowance 0,1,2,3
- 消去不能性
  non-deletability 0,1

各プロセスの保護レベルとこれらを合わせて，各プロセスの持つ権限を判定する．それぞれの権限の具体的な判定法は以下のとおりである．

accessibility の判定
パッケージの accessibility に関しては保護レベル Pl とアクセス許容度 Aa の2つの特性で許容範囲を指定する．次の不等式を満たすプロセスにパッケージへの accessibility がある．

$$\text{プロセスの保護レベル} \geq Pl - Aa$$

deletability, creatability, modifiability の判定
次の不等式のように，プロセスの保護レベルがパッケージの保護レベル Pl 以上であれば，そのプロセスにはそのパッケージに対する deletability, creatability, modifiability がある．

$$\text{プロセスの保護レベル} \geq Pl$$

ただし，すぐ後に述べる消去不能パッケージ (non-deletability が1のパッケージ) は，これらの権限のうち deletability が制限されるとともに，パッケージパスに入れることによる略記法が使えない．

3.5 パッケージとシンボルの保護と消去不能性

3.5.1 消去不能性による保護

key パッケージのように特別な保護を必要とするパッケージは，消去不能パッケージ (non-deletability が1のパッケージ) にする．消去不能パッケージは以下のような保護を受ける．

まず，消去不能パッケージに対する deletability を持つのは保護レベル3のプロセスだけである．したがって，システムのプロセスのみが deletability を持つことになる．次に，消去不能パッケージのシンボルは略記することができない．すなわち，パッケージパス内に消去不能パッケージが入っていても，関数 find-symbol-in-path は消去不能パッケージ内を探索しない．以上の2点以外については保護レベル，アクセス許容度による保護をそのまま受ける．

3.5.2 消去不能性による保護の例

システム関数 / 述語の保護
略記法によりシンボルへのアクセスをするプログラムは，パッケージパスの状態によって同じシンボル名から異なったシンボルを参照することになる．ユーザがこの性質を利用した巧妙なプログラムを書くことは可能だが，思わぬシンボルを参照してしまうバグの原因となることも多いと予想される．特にシステム内部用関数 / 述語にこのようなアクセスをすることはバグである可能性が極めて高い．このようにバグの原因となり易い不用意なアクセスを防ぐため，TAO ではシステム内部用関数 / 述語などを，消去不能性によって略記法アクセスから守っている．したがって，消去不能パッケージのシンボルはいわゆる (Common Lisp で言うところの) qualified name で参照しなければならない．

key パッケージ
key パッケージは，あらゆるプロセスがキーワードシンボルを生成できるよう creatability を与えねばならないが，キーワードおよび key パッケージ自身がうっかりと消去されないようにしなければならない．そこで，パッケージの保護レベルを0にして全ての保護レベルのプロセスに creatability を与えつつ，消去不能性を付与することにより deletability を持つプロセスを保護レベル3のものに限定している．

3.6 グローバル / インターナルパッケージと隠蔽

システムが一般ユーザに提供するシステム関数 / 述語などのパッケージは，一般ユーザから見えるようにグロー

バルパッケージとして登録しておく．グローバルパッケージとして登録されたものは関数

(global-package *package-name*)

により得ることができる．ただしパッケージを得ることができるのは，そのパッケージに対する accessibility を持つプロセスだけである．

インターナルパッケージはシステム内部用のパッケージであり，パッケージ名からパッケージを得る手段はない．システム内部のみで使用し，ユーザに見せないようなパッケージはGC対策のためインターナルパッケージとして登録しておく．

3.6.1 TAO のシステムパッケージ

TAO で用意するシステムパッケージは以下の表の通りである．これらのうち bas, key, message はグローバルパッケージであり，その他のものはインターナルパッケージである．

パッケージ	保護レベル，アクセス許容度，消去不能性	パッケージの内容
h/w	3，0，1	ハードを直接操作する関数など
sys	3，0，1	特権的システム関数 / 述語など
basi	3，0，1	システム内部用基本関数など
bas	3，3，0	基本関数 / 述語など
key	0，0，1	キーワード
message	0，0，1	メッセージセレクタ
level2	2，0，1	保護レベル 2 の setuid
level1	1，0，1	保護レベル 1 の setuid
level0	0，0，1	保護レベル 0 の setuid

表 2: TAO のシステムパッケージ

4 パッケージシステムの使用例

4.1 保護レベルとアクセス許容度

パッケージパスにより，一般ユーザ (User) のパッケージはアプリケーション (Application)，言語 (Language) のパッケージのシンボルを，アプリケーションは言語のシンボルを参照する．一般ユーザ，アプリケーション，言語の保護レベルは (パッケージ，プロセスとも) それぞれ 0, 1, 2 である．また，アクセス許容度はアプリケーション，言語のパッケージについてはともに 1，一般ユーザのパッケージについては 0 である．

ここで，不自然な例であるが，ある言語のパッケージのアクセス許容度が 1 であると[2]，一般ユーザのプロセスにはこの言語のパッケージへの accessibility がないことになる．このとき，一般ユーザは言語のパッケージ内のシンボル car, cons, throw などにアクセスできないことになる．しかし，アクセス許容度 1 のアプリケーションのパッケージ内のシンボル telnet, edit, grep などへはアクセスできる．

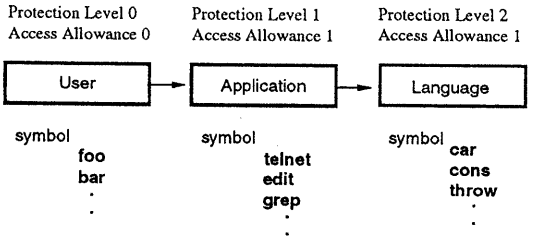

図 1: 保護レベルの異なるパッケージ

4.2 隠蔽

TAO/SILENT はマルチユーザ環境で用いられるので，複数の一般ユーザ (User 1 〜 User 3) が同じ言語のパッケージ (Language) を使うことがある．ここで一般ユーザのプロセスおよびパッケージは全て同じ保護レベルであるが，互いに自分のパッケージを他人に触られたくない．

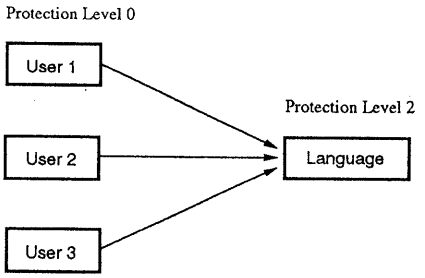

図 2: マルチユーザ

他ユーザのパッケージに触るためには，そのパッケージを何らかの方法で得る必要があるが，ここでは各ユーザのプロセスは自分のパッケージしか知らない．また，言語のパッケージ Language は保護レベルが 2 なので，ユーザプロセスは Language のシンボルに束縛されている関数を書き換えて他ユーザのパッケージを得るようにすることもできない．したがって (グローバルパッケージとして登録しない限り) 互いのパッケージを直接得る

[2]TAO の上には多数の言語を共存させることができる．それらの言語の 1 つ 1 つにパッケージが付随する．通常はアクセス許容度は 2 だが，アプリケーション作成者専用で許容度が 1 というものも考え得る．

手段はなく，互いのパッケージのシンボルに触ることは難しくなっている．

4.3 alias symbol

Language 3 は，Language 1 のパッケージをパッケージパスに入れて，Language 1 のシンボル全てを使えるようにしている．Language 2 のパッケージからはシンボル mapc だけを使いたく，他のシンボルは使いたくない．

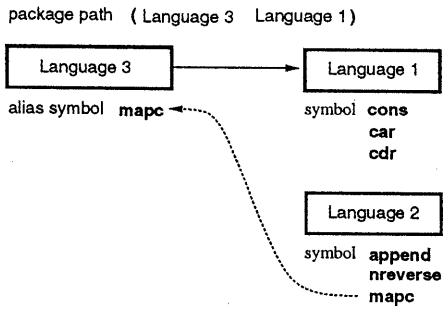

図 3: alias symbol による輸入

このような場合に便利なのが alias symbol によるシンボルの輸入である．Language 3 のパッケージに mapc の alias symbol を作れば，Language 1 のシンボルとシンボル mapc を参照することができる．

4.4 virus symbol

図 4 のようにパッケージパスを設定すると，Language 3 に virus symbol cons が存在するため Language 2, Language 1 にシンボル cons が存在しても見えない．virus symbol が存在しなければ，Language 1, Language 2 のシンボルは symbol cons も含めて全て見えるはずである．このように同じパッケージパスから見えるシンボルのうちある特定のものだけを隠し，他のものは見せておく場合に virus symbol を使う．

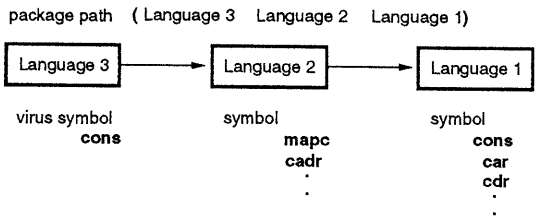

図 4: virus symbol による隠蔽

5 インクリメンタルリハッシュ

TAO/SILENT はロボット制御などに使うことを考慮しているため，ミリ秒以下のリアルタイム性が求められる．リアルタイム性を保つため，TAO では個々のプリミティブを効率的なものにすると同時に，パッケージのシンボル増加時のリハッシュに，次に述べるようなインクリメンタルリハッシュを採用している．

TAO のパッケージ内のハッシュテーブルには通常使用している hashtable と，リハッシュの際に一時的に使用する next-hashtable の 2 つがある．またリハッシュ中であることを示すフラグ rehash と，リハッシュの進み具合を記録する変数 rehash-front が用意されている．

リハッシュの時はフラグ rehash を立てて，新しいハッシュテーブル (のメモリブロック) を next-hashtable に用意し，hashtable から 1 エントリづつ next-hashtable に移動する．ここでエントリとは hashtable 上で 1 つのハッシュ値に対応するシンボルの集合である[3]．エントリの移動は他のプロセスに割り込まれないよう，不可分に行ない，どのエントリまで移動したかを変数 rehash-front に記録しておく．全てのエントリを移動し終れば，next-hashtable を hashtable にする．

リハッシュ中にシンボル探索が起きた場合には，シンボル名のハッシュ値を計算し，rehash-front と比較する．これにより探索しているエントリが hashtable と next-hashtbale のどちらに存在するかが直ちに分る．リハッシュ中のシンボル生成も探索と同様であり，rehash-front より先にある場合のみ古い hashtable に登録される．また rehash 中に rehash が起きた場合にも，このアルゴリズムを少し変更するだけで対応できるが，本稿では省略する．

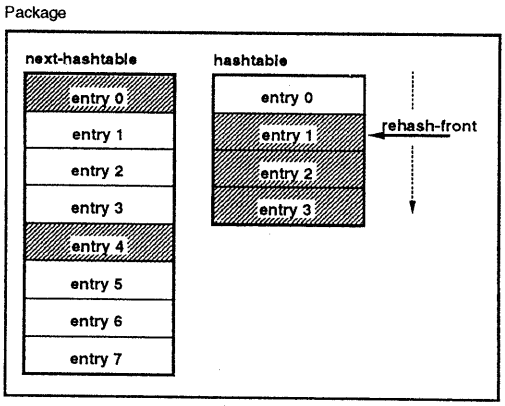

図 5: パッケージ内のハッシュテーブル

[3] チェインハッシュ法を使う．

6 Common Lisp のパッケージとの比較

Common Lisp で提供されているパッケージシステムと比較する.

internal/external シンボル

Common Lisp では 1 つのパッケージ中のシンボルを internal なものと external なものに分けており, external なもののみを外から use-package, import などで参照できるようにしている. TAO では効率性の点から, このような区別を行なう仕組みは設けず, 言語仕様を簡潔化している. 区別が必要な場合には, 言語開発者が internal に使うシンボルと external に開放するシンボルを別パッケージにして対応する (cf. basi と bas).

use-package

Common Lisp で use-package により行なわれていた継承を実現するための機構として, TAO にはパッケージパスがある. 与えられたパッケージパスの中からシンボルを探す関数として find-symbol-in-path が用意してある. find-symbol-in-path はパッケージパスの先頭のパッケージから順にシンボルを探し最初に見つけたシンボルを返す. 一方, Common Lisp で use-package されたパッケージ同士の間には優先順序はない. 継承するパッケージに優先順序がつくという点で, TAO の継承は Common Lisp の継承とは異なっている.

シンボルの探索に find-symbol-in-path を使う場合にはパッケージパスの car 部が Common Lisp の *package* に相当し, cdr 部が Common Lisp の use-package されたパッケージの集合におおよそ相当する.

import

TAO に Common Lisp の import に相当するものはない. シンボルの alias を作る make-alias-symbol という関数があるが, これはシンボルの大域属性のうち symbol function と symbol predicate のみをコピーしその他の属性はコピーしないもので, import とは若干異なっている.

nickname

機械語である TAO には, 言語仕様の簡潔化のため, 用意されていない. これも必要な場合には TAO の上に乗る言語の開発者が作ることになる.

shadow

TAO に Common Lisp の shadow, shadowing-import に相当するものはない. 継承しているパッケージに同名のシンボルが存在する場合にも, どちらのシンボルを優先するかが言語仕様上明確になっているので, Common Lisp のように shadowing-list を用意して優先するべきシンボルを明示する必要がないからである.

7 おわりに

TAO のパッケージシステムについて, 特に保護レベルを中心とした保護機能とパッケージパスによる継承の機構について詳しく述べた. TAO ではパッケージパスによって優先順序付きでパッケージの継承ができるため, Common Lisp のような名前の衝突のない簡潔な継承となっている. また, OS としての機能を果たすため, パッケージとシンボルの保護機能を保護レベルを中心としたシンプルな機構で提供している. これらを用いることにより, マルチーユーザや複数言語の共存に対応し易くなっている.

参考文献

[1] Guy L. Steele Jr.: COMMON LISP The Language 2nd ed., Digital Press,1990.

[2] 竹内, 吉田, 天海, 山崎: TAO/SILENT の実時間性について, 記号処理研究会, 61-1, 1991.

[3] 竹内, 吉田, 天海, 山崎: 新しい TAO の設計, 記号処理研究会, 56-2, 1990.

[4] 天海, 竹内, 吉田, 山崎: TAO/SILENT のソフトウェア・アーキテクチャ, 第 8 回ソフトウェア科学会大会, 1991.

[5] 吉田, 竹内, 天海, 山崎: 新しい記号処理カーネル SILENT の設計, 記号処理研究会, 56-1, 1990.

[6] 天海, 山崎, 竹内: 新 TAO のメッセージ伝達式, オブジェクト指向計算ワークショップ, 1991.

[7] 山崎, 天海, 竹内, 吉田: TAO/SILENT の論理型プログラミング, 記号処理研究会, 64-1, 1992.

[8] 天海, 山崎, 中村, 吉田, 竹内: TAO のコンカレンシ・コントロール, 記号処理研究会, 69-3, 1993.