8Q-06 ISLISP処理系TISLにおけるネイティブインタフェース

泉 信人　伊藤 貴康
東北大学大学院情報科学研究科

1 はじめに

プログラミング言語 Lisp は人工知能や記号処理の研究のために設計された言語である．言語の発展に伴い多くの方言が生まれ，1997 年に ISLISP が Lisp の ISO 標準 [1]として制定され，1998 年に JIS 標準 [2]として登録された．その後いくつかの ISLISP 言語処理系が開発され，公開されている．

ISLISP はオブジェクト指向機能を備えながらもコンパクトな言語仕様となっている．ISLISP の機能のみではアプリケーションの開発に必要な全ての機能を満たせない場合がある．この機能の実現のたびに言語処理系を拡張することは効率が悪く，外部プログラムとのインタフェースを用意し，外部との相互作用によって必要な機能を実現することがある．

現在の ISLISP の規格には外部プログラムとのインタフェースについては規定されていない．そこで，筆者等によって作成および公開されている ISLISP 処理系 **TISL**[3]に対して，Java のネイティブプログラミングインタフェース JNI[4]を参考に外部プログラムとの相互作用を行うインタフェース **TISL Native Interface(TNI)**を設計し，実装を行った．本稿では，TNI の概要について報告する．

2 TISL ネイティブインタフェース TNI

TISL ネイティブインタフェース TNI を通すことによって，ネイティブコードすなわち機械語によるコードと TISL 上で実行される ISLISP プログラムとの相互作用を行うことが可能になる．

TNI の設計では，ネイティブコードとのインタフェースとなるインタフェースポインタなどは JNI を参考に設計が行われており，インタフェース関数は ISLISP 用に設計が行われている．JNI を用いたプログラミングの経験のあるプログラマであれば容易に TNI を用いたプログラミングが可能である．

外部プログラムは ISLISP 関数とリンクすることによって利用される．外部プログラムは TNI を通すことによって，ISLISP オブジェクトの生成，処理系によるメモリ管理，ISLISP 関数の呼出し，例外処理などを行うことが可能である．

2.1 インタフェースポインタ

TISL におけるネイティブインタフェースはマルチスレッド環境への対応のため二つに分けられる．一つは個々のスレッドへのインタフェースを追加したり削除したりするための **TISL インタフェース**であり，もう一つは実際にネイティブコードとの相互作用を行う **TNI** である．

ネイティブコードは TISL インタフェースまたは TNI を表すインタフェースポインタを通してインタフェース関数を呼出す．TISL インタフェースには 3 つ，TNI には 88 のインタフェース関数が用意されている [3]．インタフェースポインタはインタフェース関数テーブルへのポインタへのポインタとなっており，全てのインタフェース関数は第一引数にこのインタフェースポインタを取る．

2.2 ISLISP 側からネイティブコードの呼出し

ネイティブコードは新しく設計された **deflink** 定義形式によって ISLISP 関数とのリンクが設定される．ISLISP 関数とリンクされる外部関数は第 1 引数として TNI ポインタが渡され，第 2 引数以下で ISLISP からの引数が渡される．戻り値を含めて ISLISP オブジェクトは ISLISP オブジェクトポインタ TISL_OBJECT によって参照渡しされる．

2.3 オブジェクトの参照

ISLISP オブジェクトは全て TISL_OBJECT によって参照渡しされる．処理系はネイティブコードに

A native interface for an ISLISP processor TISL
Nobuto Izumi, Takayasu Ito
Graduate School of Information Sciences, Tohoku University

渡されたオブジェクトがガーベジコレクタによって解放されないようオブジェクトの追跡を行う．ネイティブコードは TNI 関数を通してオブジェクトが必要ないことを処理系に知らせることができる．

オブジェクトの参照は**ローカル参照**と**グローバル参照**の 2 種類に分けられる．ローカル参照は外部関数呼出しの間のみ有効であり，外部関数呼出し終了後に自動的に回収される．グローバル参照は TNI 関数 **new_global_ref** によって作成され，TNI 関数 **delete_global_ref** によって明示的に解放されるまで有効な値となる．ネイティブコードへの引数や TNI 関数の戻り値などはローカル参照である．

2.4 ISLISP オブジェクト

ISLISP の定義済みクラスに対してインスタンス生成やアクセスを行う TNI 関数を用意している．<integer>に対して，生成 **create_integer** や整数取得 **object_get_integer** の TNI 関数が用意されている．

2.5 ネイティブコードからの ISLISP 関数の呼出し

ネイティブコードから ISLISP 関数を呼出す TNI 関数 **function_call** が用意されている．第 2 引数に呼出したい ISLISP 関数に対応する ISLISP オブジェクトを渡し，第 3 引数以下で呼出す関数に渡す実引数を与える．

2.6 ネイティブコードのエラー

処理系はネイティブコードのプログラミングエラーや TNI 関数への不正な引数渡しを検出し ISLISP 例外を発生させることはしない．

2.7 ISLISP 例外処理

TNI 関数 function_call を通して ISLISP 組込み関数 signal-condition や error 等を呼出すことによってネイティブコードの中で ISLISP 例外を発生させることが可能である．

ネイティブコードに制御がある場合には処理系によって用意される例外ハンドラが有効になる．この例外ハンドラは一度制御をネイティブコードに移し，ネイティブコードに例外処理を行う機会を与える．TNI 関数 **get_last_condition** によって発生した例外を取得することができ，TNI 関数 **clear_condition** によって例外状態をクリアすることができる．

2.8 機能拡張の例

TNI の簡単な例として bit 演算を挙げる．

```
TISL_OBJECT   bit_and(TNI* tni,
                      TISL_OBJECT arg1,
                      TISL_OBJECT arg2) {
    tINT i1, i2;
    i1=(*tni)->object_get_integer(tni, arg1);
    i2=(*tni)->object_get_integer(tni, arg2);
    if ((*tni)->get_last_condition(tni))
        return NULL;
    return (*tni)->create_integer(tni, i1&i2);
}
```

上では TNI ポインタの他に二つの引数を取る外部関数 **bit_and** を定義している． **object_get_integer** を通して二つの引数から整数を取出し，**get_last_condition** を通して例外の発生を検出した場合には処理系に例外処理を行わせ，それ以外の場合には **create_integer** を通して演算結果の新しい整数を作成している．この外部関数がライブラリ lib に登録されているとすると，deflink 定義形式を次のように使用する．

```
(deflink bit-and (b1 b2) "lib" "bit_and")
```

上の定義形式で定義された ISLISP 関数 bit-and は他の ISLISP 関数と同様に使用することができる．

3 まとめ

TNI の概要について簡単に報告した．詳細については参考文献 3)を参照されたい．TNI は JNI を参考に設計されており，ISLISP におけるネイティブコードへの標準的なインタフェースのベースとなることが期待される．TNI を使用することにより，処理系の実装と機能拡張の実装が分離できる．

TNI を使用し作成した標準的な拡張機能のライブラリをまとめ，ユーザに提供していくことは今後の課題である．

参考文献

1) ISO/IEC 13816 : 1997, Programming language ISLISP, ISO/IEC (1997).
2) JIS X 3012 : 1998, プログラミング ISLISP, 日本規格協会(1998).
3) 泉 信人, 伊藤 貴康 : TISL, http://www.ito.ecei.tohoku.ac.jp/TISL.
4) Sheng Liang, *The Java Native Interface : Programmer's Guide and Specification*, Addison Wesley (1999)