マイクロコンピュータと
ワークステーション 53－1
（１９８８．１２．１）

マイクロコンピュータ用ソフトウェア開発に向けたＣＨＩＬＬ言語処理システムの実現

佐藤規男　大森圭祐
日本電信電話株式会社　ＮＴＴソフトウェア研究所

複雑で大規模な実時間・超多重組込みシステムである交換機等の通信システムに汎用マイクロコンピュータが用いられることが多くなってきているが、そのソフトウェアを効率よく開発するためには、良い言語の選択と開発支援環境の充実が重要な課題である。そこで、モジュール化や並列処理など優れた機能をもつＣＨＩＬＬ(CCITT High Level Language)の言語処理系を、高機能化・経済化・ネットワーク化の著しいＵＮＩＸ*ワークステーション上に実現した。ターゲットマシンはＭＣ６８０２０等のマルチターゲットを指向しているが、開発マシン有効利用の観点から、まずセルフのコンパイラとデバッガを開発し、クロス用としてはＩＣＥなどの市販クロスサポートを利用できるようにこれをリターゲットしている。

本論文では、１）大規模・実時間の組込みシステムでのＣＨＩＬＬの有効性、２）ＵＮＩＸセルフ・市販クロス環境を利用した処理系構成法、について述べる。

*）ＵＮＩＸは米国ベル研究所の登録商標である。

A CHILL LANGUAGE SYSTEM FOR MICROPROCESSOR EMBEDDED SOFTWARE

Norio SATO and Keisuke OHMORI
Software Engineering Laboratory, NTT Software Laboratories
1-9-1 Kohnan, Minato-ku, Tokyo, 108 Japan

This paper describes: 1) notable features of CHILL for the programming of the large real-time systems; 2) our implementation experience of a CHILL language processor on top of the UNIX* self and a cross environment available in the market.

Firstly, CHILL comprises strong type checking, modularization, and concurency, which are suitable for complex embedded systems. Secondly, microprocessors are more and more often employed for such systems. Thirdly, the recent progress of UNIX workstation environment is remarkable.

A CHILL language processor is, therefore, implemented on Sun-3*. It consists of a UNIX self compiler and a debugger to make the best use of the workstations, and a cross compiler which outputs CLAND-II* MC68020 assembler codes.

*) CHILL is an international standard programming language in the telecommunication area, designed at CCITT primarily for the implementation of complex embedded systems.
*) UNIX, Sun-3, CLAND-II are registered trade marks of Bell Laboratories, Sun Microsystems, and Tektronics Inc, respectively.

1．はじめに

複雑で大規模な実時間組込みシステムである交換機等の通信ソフトウェアの生産性・保守性を向上するためには、潜在バグに対する信頼性やモデル化能力に優れた言語の選択と支援環境の充実が重要である。

ＣＨＩＬＬ(CCITT High Level language)は交換機等通信システムプログラム用に１９７０年代から、ＣＣＩＴＴ（国際電信電話諮問委員会）で各国での交換機開発経験をもとに設計・標準化されてきたプログラミング言語であり、型チェック、モジュール化・分割翻訳、並列処理、例外処理などをもつ高機能な言語である。

ＣＨＩＬＬの処理系は１９８０年代初頭から各国で開発が進みＩＴＴやＳｉｅｍｅｎｓをはじめとする欧米各国の交換機用ソフトウェアに実際使用されてきている。ターゲットマシンは汎用マイクロプロセッサが主流である。開発マシンはＩＢＭ３７０やＶＡＸ*が主流である。

ＮＴＴでは、データ通信用のＤＩＰＳを開発マシンとし、交換機専用プロセッサＤ１０をターゲットとするＣＨＩＬＬ言語処理系を早期に開発し、大局用電話交換機プログラムなどの開発に用いて来ている。この処理系は従来のアセンブラプログラムを置き換える目的から開発したサブセット言語仕様[7]であり、効率重視から最適化コンパイラを用いている。

しかしながら、近年ＩＳＤＮなどが注目されるに至って通信システムが多様化する動向にあり、そのソフトウェアの作成規模が加速度的に増大しつつある。さらに、ＬＳＩの進歩により機能・性能の高い汎用マイクロプロセッサを使用することが有利になりつつある。このことから新しいソフトウェアの設計概念も試行されている[11-13]。さらに、ここ数年ＵＮＩＸワークステーションの高性能化・低価格化も著しい。そこで、この様な状況に即した支援環境を構築することとし、次のような方向づけを行った。

（１）ＣＣＩＴＴ最新標準ＣＨＩＬＬとする。

現在のサブセットはＤ１０マシンの制限やアセンブラで使用していたプログラミング技術の継承などの歴史的理由から汎用マイクロプロセッサでは継承困難な特殊機能を多く含む。そこで、これを取り除くとともに、メッセージ主体の並列処理などの有用な機能を取入れてフルセットに近い言語機能を実現する。

（２）開発はＵＮＩＸ*上で行う。

クロスのコンパイラ・デバッガだけではなく、セルフコンパイラとシンボリックデバッガを提供することにより、資源制約の少ない開発マシンの作業分担を多くするようにする。当初はマイクロＶＡＸ*-IIで始めたが、ここ２、３年のＵＮＩＸワークステーションの高機能・低価格化・ネットワークの充実からｓｕｎ-３*にリホストすることとした。

クロスサポートはＵＮＩＸに接続可能な市販ＩＣＥを使用しこれと接続できるようにする。

（３）マルチターゲットを考慮する。

ターゲットマシンはアドレス空間の制限の少ない３２ビット汎用コンピュータとし、コンパイラは最適化よりも即応性に重点を置くこととする。

---

*)ＶＡＸはＤＥＣ社の登録商標である。
*)ＵＮＩＸはベル研究所の登録商標である。
*)Ｓｕｎ-３はｓｕｎマイクロ社の登録商標である。

2．ＣＨＩＬＬの特長的機能

ＣＨＩＬＬの構文はＡｌｇｏｌ６８やＰＬ/Ｉの影響を受けており、「procedure A; type T=...; x:integer;」などは「A:PROC(); SYNMODE T=....; DCL x INT;」と記述するが、機能は主にＰａｓｃａｌやＭｏｄｕｌａを参考としている[4]。言語仕様書は勧告Ｚ２００[1]という番号が付けられており、４年に一度改訂される。１９８８年にはＩＳＯ標準としても認定されるべく提案されている。本言語仕様書は極めて体系的かつ正確に記述されているためコンパイラ作成には非常に有用である。

次に特長的機能を述べる。

（１）豊富なデータ型と型チェック機能をもち、ビット詰め機能も有する。また、パラメータや動的割付には可変長データも扱える。

以下、プログラム例に沿って特長を説明する。本例は有名な５人の哲学者問題のＣＨＩＬＬによる記述例である。便宜上に図１、図２、図３分けて示す。

```
1 －ＵＮＩＸ標準入出力インタフェースです。
2
3 c_defs: SPEC MODULE―翻訳単位です。
4         printf:PROC(format CHAR LOC,...)
5                 RETURNS(INT)INTERFACE(C);
6         scanf:PROC(format CHAR LOC,...)
7                 RETURNS(INT)INTERFACE(C);
8         GRANT ALL;
9 END c_defs;
```

図１．Ｃプログラムインタフェース記述例

```
1
2  ―ＵＮＩＸは非同期ｉ／ｏはないので、
3  ―ＵＮＩＸ上で並列プログラムを走行させても
4  ―プロセスのスイッチが起こらず、
5  ―各プロセスが順次起動するだけです。
6  ―（例えば、５人の食事問題では１人目が食事中
7  ―のままとなります）
8  ―プロセスの走行時間を意図的に作り出すために
9  ―本パッケージを使います。
10 ―タイミング値は大きいほど再開が遅らせること
11 ―ができます。
12
13 elapse: module spec ―仕様部（翻訳単位です）
14         grant all;
15         consume_time:proc(n int);
16 end elapse;
17
18 elapse: module body ―実現部（翻訳単位です）
19         signal go;
20         dummy:process(cp instance);
21                 send go to cp;
22         end dummy;
23         consume_time: proc(n int);
24                 case n of
25                 (0) : return;
26                 (1) :start dummy(this,1);
27                 (2) :start dummy(this,2);
28                 (3) :start dummy(this,3);
29                 (4) :start dummy(this,4);
30                 (5) :start dummy(this,5);
31                 (6) :start dummy(this,6);
32                 (7) :start dummy(this,7);
33                 (8) :start dummy(this,8);
34                 (9) :start dummy(this,9);
35                 (10):start dummy(this,10);
36                 (11):start dummy(this,11);
37                 (12):start dummy(this,12);
38                 (13):start dummy(this,13);
39                 (14):start dummy(this,14);
40                 else start dummy(this,15);
41                 esac;
42                 receive case
43                 (go): return;
44                 esac;
45         end on(spacefail):return;
46         end consume_time;
47 end elapse;
```

図2．仕様部と実現部の分離記述例

```
1  ―５人の哲学者が食事する問題をシグナルを
2  ―用いてプログラミングした例です。
3  ―各哲学者対応に１プロセスを生成する他、
4  ―フォークを管理するプロセスを設けます。
5  ―このプログラムを走らせると状況が表の形で
6  ―表示されます。
7  ―各シグナルネームにはｔｏオプションを指定しました。
8  ―フォークのプロセスは
9  ―１つだけなのでｓｅｎｄ文のｔｏオプションがなくとも
10 ―シグナルが転送されますが、
11 ―哲学者プロセスは５つですからｓｅｎｄ文に
12 ―ｔｏオプションが必要です。
13 ―なお各プロセスで利用可能なフォークの個数を配列
14 ―で管理します。
15
16
17 dining: MODULE
18
19   FROM c_defs SEIZE printf;
20   ―シグナル定義
21   SIGNAL  free=(INT(0:4)) TO fork,
22           reserve=(INT(0:4)) TO fork,
23           ok TO phil;
24   FROM elapse SEIZE consume_time;
25
26   phil: PROCESS(i INT(0:4));―プロセス定義
27         DO FOR j:=1 TO 100;
28            ― think();
29            printf(c'1B'//'[%d;11H'//C'00', i*2+10);
30            printf(C'09'//'Phil=%d thinking     ?'
31                        //C'0A'//C'00',i);
32            SEND reserve(i);―シグナル送信
33            RECEIVE CASE ―シグナル受信
34            (ok): ―eat();
35                  ――――
36                  printf(c'1B'//'[%d;11H'//C'00', i*2+10);
37                  printf(C'09'
38                         //'Phil=%d    eating ! %d times'
39                        //C'0A'//C'00',i,j);
40                  ――――
41                  consume_time(10);
42                  SEND free(i);―シグナル送信
43            ESAC;
44         OD
45         ON(SENDFAIL,SPACEFAIL,EXTINCT):―例外処理部
46                  ―SENDされなかったとき
47                  ―の処理
48         END;
49         printf(c'1B'//'[%d;11H'//C'00', i*2+10);
50         printf(C'09'//'Phil=%d got stuffed.  '//C'0A'//C'00',i);
51   END phil;
52
53   fork: PROCESS();―プロセス定義
54
55           aquire: PROC(i INT(0:4));
56                 forks((i+1)MOD 5)-:=1;
57                 forks((i-1)MOD 5)-:=1;
58           END aquire;
59           release: PROC(i INT(0:4));
60                 forks((i+1)MOD 5)+:=1;
61                 forks((i-1)MOD 5)+:=1;
62         END release;
63
64         DCL     forks ARRAY(0:4) INT(0:2)
65                                 :=[(0:4):2];
66         DCL     waiting ARRAY(0:4) INSTANCE
67                                 :=[(0:4):NULL];
68         DCL     id INSTANCE,
69                 left,right INT(0:4);
70
71        DO FOR EVER;
72           RECEIVE CASE SET id;―idは送り手識別子
73           (reserve IN i):
74                 IF forks(i)=2 THEN
75                    SEND ok TO id;―シグナル送信
76                    aquire(i);
77                 ELSE
78                    waiting(i):=id;
79                 FI;
80           (free IN i):
81                 release(i);
82                 left:=(i-1) MOD 5;
83                 right:=(i+1) MOD 5;
84                 IF waiting(left)/=NULL
85                       AND forks(left)=2 THEN
86                    aquire(left);
87                    SEND ok TO waiting(left);
88                    waiting(left):=NULL;
89                 FI;
90                 IF waiting(right)/=NULL
91                       AND forks(right)=2 THEN
92                    aquire(right);
93                    SEND ok TO waiting(right);
94                    waiting(right):=NULL;
95                 FI;
96           ESAC;
97         OD;
98
99    END fork;
100        ―
101        printf(c'1B'//'[2J'//C'00');
102        START fork();
103        DO FOR i IN INT(0:4);
104              START phil(i);
105        OD
106        ON(SPACEFAIL):―プロセスが起動出来なかったとき
107        END;
108  END dining;
```

図3．プロセスとシグナルの記述例

（２）モジュール間にまたがる型チェックを行う分割翻訳ができる。

概念的にはＭｏｄｕｌａ[5]のモジュールと似ている。翻訳単位は、「モジュール仕様部」、「モジュール実現部」、および両者合体した「モジュール」である。図１におけるc_defsはモジュール仕様部である。Ｃプログラムを仮想本体部とする単なるインタフェースを表すだけで対応する実現部はないのでspec moduleと記述する。図２のelapseはモジュールを仕様部と実現部に分離記述した例である。前者は本体をもつのでmodule specと記述し後者はmodule bodyと記述する。実現部の翻訳時には両者の一致をチェックする。図３のdiningは仕様部と本体部に分離する必要がないため単にmoduleで記述している（分離記述しても支障はない）。

図２の１４行目のgrantと図３の１９行、２４行目のsiezeはＭｏｄｕｌａのexport/importに当たる。

（３）プロセスおよびプロセス間通信からなる並列処理（タスキング）を有する。

図３におけるモジュールdiningのｐｈｉｌ（２６行－５１行）やｆｏｒｋ（５３行－９９行）がプロセスの定義であり、末尾（１０１行以下）に記述してあるようにｓｔａｒｔにより明示的にプロセスを生成する。プログラム全体はモジュールの並びであるが仮想的な一つのプロセスとなりモジュールに直接記述された実行文が実行される。この例では、オブジェクトモジュールelapse.oとdining.oをリンクするときにスタートアップルーチンが作られ、これがそれぞれの初期設定部を順次起動する。

プロセス間通信手段は次のようにコンセプトがいくつかあり、要求されるシステムの効率・用途から使い分ける。もちろん全部同時に使用して矛盾はない。ただし本論文では特に汎用性の高い④のシグナルについて述べることとし、①②③については詳しい説明を省略する。（[1]の他,[9],[10]を参照。）

①イベント変数：プロセスの待行列でありここで待ち状態にあるプロセス１つをｃｏｎｔｉｎｕｅ操作で再びアクティブにする。

例）DCL e EVENT;--イベント変数の宣言
　　DELAY e;--実行したプロセスが待ちになる。
　　CONTINUE e;
　　--待ちプロセスを一つアクティブにする。

②排他アクセスモジュールであるリージョン：Ｈｏａｒｅのｍｏｎｉｔｏｒと類似の機能である。通常イベント変数と組合せて資源管理に用いる。

③同期を伴うデータ転送用バッファ変数：リングバッファでありプロセスはデータの出し入れをバッファ変数を介して行う。空きまたは満杯であると待ち合わせが生じる。

④プロセス間で直接メッセージの転送を行う手段であるシグナル：相手プロセスの状態（アクティブか待ちであるか）に拘らずデータ転送が行われる非同期転送であるためリアルタイムシステムでは適した場面が多い。（ただし、非同期の必要性という点では本例題は余り適当ではない。）

図２のモジュール実現部elapseにおけるプロセスdummyおよびconsume_timeを呼び出すプロセス（＝phil）はシグナルgo（１９行目）で交信する（２４行目と４２行－４４行）。図３のモジュールdiningにおける５つのプロセスphilとプロセスforkの間はシグナルreserve（２２行目）とfree（２１行目）により交信を行う。（３２行－４３行および７２行－９６行）

割込み処理はプリエンプションを伴うスケジューリングを行えば簡単に並列処理の枠内で実現できる。このため勧告Ｚ２００ではわざと明確には規定していない。我々のスケジューラ（実行時ルーチン）では組込みのイベント変数を提供している。つまり、割り込みベクタ対応に実行時ルーチンの中にイベント変数を用意しておき割り込み発生時に対応するイベント変数をcontinueにより待ちプロセスをアクティブにする。このプロセスの優先度を割り込まれたプロセスの優先度より高くしてけば後者はペンディングされる。

例）
```
system:SPEC MODULE
    DCL vectors(0:255) EVENT;
      --ﾊｰﾄﾞｳｪｱがcontinueするｲﾒｰｼﾞ
    GRANT vectors;
END system;
xxx:PROCESS()
    DO FOR EVER;
      DELAY vector(65);--割込み待ち
      /*割込み処理*/
    OD;
END xxx;
```

（４）例外処理により正常処理・異常処理の記述が簡単である。値や配列添字の範囲違反などが起こると例外処理部が実行される。例外処理部はネスト構造に対応しており、また、手続き呼び出しでも引き継がれる。ＣＨＩＬＬでは冗長なチェックコードをサプレスできるように例外処理部の有無が翻訳時にわかるようなしかけに設計されている。

図２の例におけるモジュールelapse（４５行）

や図３のdining（１０６行目）の「ON(spacefail):･･･」はエリア不足によるプロセス生成失敗時の処理である。
（５）その他、ファイルｉ／ｏ・テキストｉ／ｏ、非同期タイミング待ちなど我々の対象とする通信システムでのニーズが明確でないことから実現していない。入出力は図１のようにライブラリを呼び出すととしている。

３．支援環境の構成

３．１　セルフ系・クロス系の分担

ＣＨＩＬＬの用途は主に組込みシステム用言語であることから実機上で動く機械語を生成するクロスの処理系が最終製品であるが、単体試験位までは極力アクセス容易な開発マシンを使い、ハードウェアとの結合段階で実機試験に移行する方が能率が良いと考えられる。
このためには、開発マシン上で動くターゲットマシンのシミュレータを開発するというアプローチもあるが、１）開発マシン上で実行できるコードを出すコンパイラを作る方が実現容易であり使いやすいこと、２）割込み処理の有無は言語仕様には影響しないため言語仕様はセルフ・クロス共通にできること、などの理由から、まずセルフの処理系を開発し、次のステップでＩＣＥへのリターゲットを行うことにした。

処理系のシステム構成を図４に示す。新規に作成した処理系はセルフコンパイラ・セルフデバッガおよびクロスコンパイラである。
アセンブラ・リンケージエディタについてはセルフ処理系ではＵＮＩＸのものを起動している。
クロス処理系としてはＵＮＩＸと出来ればＬＡＮ経由でＩＣＥを結合したエミュレート試験、さらにターゲットへのダウンロードができる市販ＩＣＥとの結合を行っている。具体的にはソニーテクトロニクス社のＣＬＡＮＤＳ*を用いてＣとの共用支援系とした。同社のオブジェクトモジュール形式はＰａｓｃａｌ系言語との整合もよいことから、さしむき同社の許諾をうけてアセンブラレベルでのデバッグ情報も合わせることにより、アセンブラ・リンケージエディタ・ダウンローダ・シンボリックデバッガを流用することにしている。

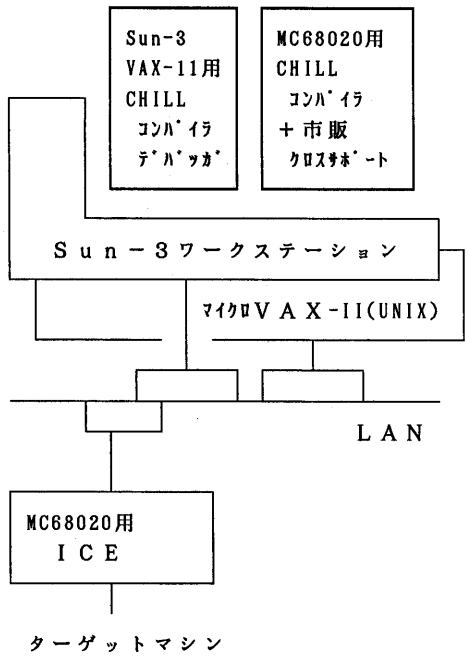

図４．ＣＨＩＬＬ言語処理系の構成

３．２　コンパイラとデバッガの特長

（１）コンパイラ
コンパイラへの要求条件としては、１）チーム開発におけるモジュール間インタフェースの信頼性、２）翻訳時間の短縮、３）オブジェクトコードの性能向上、など相反する条件が要求される。したがって、分割翻訳と中間言語構成、最適化などを考慮したバランスよい設計を主眼に次のように構成した。
①分割翻訳におけるモジュール間の型整合性を保証する。
Ａｄａ*の処理系にならって次のように中間言語ファイルを作る方式とした。仕様部の翻訳で得られる中間言語ファイルを翻訳単位ごとに特定ディレクトリ化に生成する。このディレクトリから関連する仕様部を捜し出してこれを参照しながら翻訳する方式とした。さらに分散化したディレクトリをファイル共用機能を利用して結合することにより大規模な共同開発に適したものにすることができる。

---

*）ＣＬＡＮＤＳはテクトロニクス社の登録商標である。

なお、この中間言語は概念的にはＡｄａの中間言語ＤＩＡＮＡ[6]のＣＨＩＬＬ版である。構造はマシン独立で汎用的であるためドキュメント生成用としても役だっている。

---

*)Ａｄａは米国国防総省の登録商標である。

②翻訳時間短縮と最適化のバランスをとる。

コントロール・データフロー解析に基づいて共通式削除・レジスタ割付けなどを実施しているが、最適化範囲を、さしむき分岐に着目したサブトリーグラフにとどめ、ループや手続き全体といったグローバルな範囲には拡大していない。

また、シンボリックデバッガの使用を考えてメモリ上の変数値とレジスタ上の値と常に一致をとるようなコード生成とした。またプロセスの中断点をまたがるレジスタ引継は行わないためスケジューラでのレジスタセーブは割込み発生時以外は不要になっている。

③翻訳時の所要メモリを適正化する。

ＣＨＩＬＬはそのスコープ規則からマルチパスコンパイラが必須であるため、構文解析・意味解析・中間言語変換・最適化・機械語生成の各フェーズに分割している。ＵＮＩＸ４．２ＢＳＤではオーバレイ構造がとれないため各フェーズを独立なプロセス（＝ロードモジュール）とし、中間言語はファイル経由で引き継ぐこととして順次起動することとした。各フェーズの出力する中間言語は順次ファイルに出力することとして使用メモリが過大になることを防止している。

④即応性・信頼性を高める。

構文解析とコード生成にはジェネレータツール[15][8]を使用することにより即応性を高めている。またコンパイラの検証については自力開発のテストプログラムのほか欧州で開発された検定プログラム約１５００本を併せて使用することにより網羅性を高めている。

（２）デバッガ

デバッガの応答速度、実現容易性、市販クロスツールとの接続の必要から、アセンブラコードにソースファイル・行・シンボル・ブロック構造からなるデバッグ情報を乗せる方式とした。

ＵＮＩＸ上にはｓｄｂやｄｂｘなどの既存デバッガがあり、これのもつほとんどの機能はどの言語にも共通な機能であるため、これらをそのまま使用することも考えられる。しかし、ＵＮＩＸではｐｔｒａｃｅシステムコールを用いて容易にデバッガが実現できるため、新規にＣＨＩＬＬ用デバッガの開発を行った。これにより、１）配列上下限の表示やプロセスの状態表示などＣ言語にない機能の拡張、２）変数参照時におけるＣＨＩＬＬ構文による指定、３）マルチウィンドウ機能、など既存デバガにない機能も追加することができた。

４．あとがき

ＣＨＩＬＬは比較的重厚な言語であるがコンパイラの総規模は約８０ｋ行、デバッガ約１０ｋ行であり言語機能の割には大規模にはならなかった。翻訳速度はｓｕｎ－３ではモジュール部で毎分１０００行から２０００行程度である。

実行時ルーチンは並列処理スケジューラ・メモリ管理・文字列データ等の比較や連結などからなるがＣで約２-３Ｋ行程度であり単純である。これを裸のターゲットマシンに乗せることには問題がない。システム設計者により効率向上のため改造することも容易である。何等かの既存ＯＳのもとで走行させる場合には、①ＯＳの１プロセスをＣＨＩＬＬと対応させる方法[11]、②ＣＨＩＬＬのプロセスとＯＳのプロセスを一致させる方法、があるが、①は割り込み処理の扱いに問題があるため、②ができるような拡張性の良いものを選ぶ必要がある。

クロスサポートの市販品は現状ではＣ言語のみを対象としたものがほとんどである。しかし、ロードモジュールとインタフェースをもつダウンローダやリンケージディタ、さらにＩＣＥへのコマンドインタフェースをもつデバッガの大部分はどの言語とも共通であり、将来的にはこの分野での標準化の進展を望みたい。

本言語処理は１９８８年半ばにセルフ系が完成しＮＴＴ研究所にて次期交換ソフトウェアの試行に使用されている。第３四半期にはクロス系が完成し、１９８９年以降は新規実用交換システムの開発に使用される予定になっている。

文献

［１］CCITT:CCITT HIGH LEVEL LANGUAGE (CHILL), RECOMMENDATION Z.200
［２］N.Wirth: PROGRAMMING IN MODULA-2, Springer-Verlag
「３］ANSI:Ada Reference Manual,1983

［４］C.H.Smedema et al: THE PROGRAMMING LANGUAGES Pascal Modula CHILL Ada 1983, Prentice-Halle International
［５］Samuel P et al: A C Reference Manual 1984 Prentice-Halle
［６］細谷僚一監修：ＤＩＡＮＡ入門／言語仕様／応用、１９８６年 啓学出版
［７］松尾、丸山：交換用プログラミング言語ＣＨＩＬＬ、昭和６０年、電気通信協会
［８］佐治、三橋、木村、保坂：コンパイラコード生成ジェネレータの実現法、情報処理学会第３０回全国大会 ４Ｑ－９（１９８５）
［９］丸山、佐藤：交換プログラム記述用並列処理言語の実現、２６－３、昭和６０年５月、情報処理学会論文誌
［10］佐藤：ＣＣＩＴＴプログラミング言語ＣＨＩＬＬ概要、昭和６３年６月、「交換・通信ソフトウェア仕様記述言語の標準化と技術展望：専門講習会」講習会資料、電子情報通信学会
［11］田中、小柳、佐藤：ＣＴＲＯＮへのＣＨＩＬＬ並行プロセス適用方式の一考察、昭和６３年電子情報通信学会秋季全国大会
［12］渡部、小柳、丸山：実時間オブジェクト指向交換プログラム構成の考察
［13］甲斐、丸山、小柳、渡部：オブジェクトモデルによる交換プログラム構成、昭和６３年１月、電子情報通信学会、ＳＥ８７－１４７
［14］ＵＮＩＸ４．２ＢＳＤユーザズマニュアル
［15］Expert Software System（ベルギー）：ＬＬ（１）パーザジェネレータＭｉｒａユーザズマニュアル
［16］テクトロニクス：ＣＬＡＮＤＳ、ＭＶシステムマニュアル