Ada COMPILER SYSTEM SSAdaP

Jan Holý
Kancelářské stroje, o.p.
Prague, Czechoslovakia

ABSTRACT

SSAdaP (Support System for Ada Programmers) is a programming environment for creating, compiling (and cross-compiling), linking and symbolic debugging of Ada programs. It is universal, multi-host and multi-target system, originally developed under VAX/VMS and spread to other computers and operating systems. At present, versions for two hosts and targets are available - VAX/VMS and IBM/OS. Versions for UNIX on VAX and IBM mainframes are in development and others are prepared.

Naturally, accepted language corresponds to Ada ISO/ANSI standard. We tried to avoid even allowed restrictions on language, which is true for representation specifications. Generics processing is solved without often used simplifications, too. Package for unlimited precision arithmetic is supported.

System SSAdaP has been developed in SzKI Budapest (Hungary) in cooperation with Kancelářské stroje Prague. While this is the first and only implementation of full Ada language in Eastern Europe, it is supposed to be spread widely.

USING SSAdaP

Installing SSAdaP

System SSAdaP is distributed on magnetic tape. User can choose:

- system wide installation
- single user installation
- installation for group

Generating Empty Library

```
$ genempty
directory {[.LIB]}: user_library
Define User : user_name
Give target computer {IBM,VAX} [VAX] : target_computer
Define User :
:
```

Directory specified as *user_library* must be already created.

Users

Users in the SSAdaP library are hierarchicaly ordered. The top level super user is called Ada and has special privileges. Each user can create his own subordinates. There are 3 predefined categories of users:

- beginner
- advanced
- master

Versions and Configurations

Various versions of compilation units can be stored in the library upon user's demand. Compilation units can be also grouped into sets called configurations.

Invoking SSAdaP

```
$ ssadap
::connect user_library
::login user_name user_password
```

Compilation and linking phases

phase	unit after the phase
syntactic analysis	entered; preliminary
semantic analysis	accepted
abstract code generation	transformed
concrete code generation	compiled
linking program	linked

Executing Linked Program

Program is executed in operating system environment. You can either leave SSAdaP, or use the **host** command.

SSAdaP COMMANDS

Command Syntax

command param1 param2 .../option1/option2/...

Commands can be labeled (label is an identifier):

label: command ...

Options alter the behaviour of the system:

/option_name=option_value

or in shortened form:

/option_value

Hyphen (-) as the last character indicates the continuation of the line.

Comments start with exclamation mark (!), they continue till the end of line.

Command procedures

Command procedures are sequences of commands. They can contain formal parameters, which are refered as 'p1', 'p2' etc. Procedure is invoked by its name and eventual actual parameters.

COMMANDS OVERVIEW

Entering and Leaving SSAdaP

CONNect - library
LOGin - user
LEave - user environment without leaving library
Bye - exit SSAdaP
Visit - login another user temporary (only for reading)
HOst - enter operating system environment without leaving SSAdaP (return back using DCL command **$logout**)

Program Translation

COMpile - entire compilation of source file
ENter - syntactic analysis
SEMantic - semantic analysis
ANalyse - syntactic & semantic analysis
ABstract - abstract code generation
CONCrete - concrete code generation
GENeric - generation of instance bodies
CONtinue - completion of interrupted compilation
RECompile - all obsolete units
LINk - main program

Informations

DIr - enlists library contents
SHow
- **Complete** - all units related to given main program
- **Depend_on** - all units depending on given unit
- **ON_error** - value of the on_error system variable
- **OPtion** - actual values of options
- **System** - current database & users
- **USED_by** - all units using given unit (in with clause)
- **USER** - LOGIN and VISITed users
- **Variables** - actual values of variables

LISt - source text & errors or cross references listing
HIstory - user commands stack
HElp - the most useful command
REAd - messages (created by SENd) and delete them
Tree - list semantic tree (for developers and distributors only)

A2 - list A2 intermediate code (for developers and distributors only)

SOUrce - copy source file from the database into an external file

Reductions

DElete - library entities
FORget - history elements (see HIstory)
PURge - clearing up database, deleting unidentified units

Modifications

SET - option values
CReate **Definition** - create configuration
Proc - create command procedure
User - create new subordinate user
EValuate - name of the definition
SENd - message to the user (use REAd for reading)
COPy - configuration or command procedure
EDit - invoke operating system editor
CHange **PAssword** - of user
PRedefinedness - for super user only
Protection - of unit
VAlidity - set the expiration of SSAdaP system (for super user only)
VIsibility - of unit
REName - command procedure or definition set (configuration)
SORt - units as a perfect order of compilation

Control Statements

These commands are used namely in command procedures.

Assignment statement:

variable := *expression*

Variables can be indexed or sliced. Values of these two predefined variables can be set only by SSAdaP:

EOF - set in function FGet
EXSTAT - set after each command execution (error status)

Expression value is always string.

Following string manipulation predefined functions are available:

$change - parts of string
$date - returns the actual date
$find - substring
$is_alpha - check string for all letters
$is_digit - check string for all digits
$is_space - check string for all spaces
$length - of string
$lower - conversion to lower case characters
$match - compare strings
$time - returns the actual time ("12:34:56")
$upper - conversion to upper case characters

Conditional statement:

if *condition* **then** *command*

Error condition statement:

ON_error *command*

Specified *command* is executed in case of error occurance in subsequent commands.

Goto statement:

goto *label*

Text Files I/O

FCReate
FOPen
FGet
FPut
FCLose

Terminal I/O

GET
PUT

DEBUGGING PROGRAMS

SADE (Symbolic Ada_DEbugger) is a tool for tracing program activities and restricted data modifications on the source Ada language level. **Debug** option must be set to non-zero value for compilation and linking.

Debugger Commands

Break	- set breakpoint
CANcel	- cancel breakpoint
Deposite	- modify object value
EXAmine	- object value
EXIt	- debugger SADE; **CTRL/Z** may be used instead
Go	- continue program execution
Help	- again the most useful command
Input	- set debugger input device
Line_mode	- set line mode
Output	- set debugger output device
Refresh_screen	- **CTRL/W** may be used instead
SCope	- set scope
Screen_mode	- set screen mode
Scroll	- screen up, down, left, right
SHow	**Breakpoints**
	Calls
	Compilation_units
	Scope
	Tasks
Spawn	- execute DCL command
STep	- program execution control

IMPLEMENTATION DEPENDENT FEATURES

Standard Pragmas

Pragmas CONTROLLED, SHARED, STORAGE_UNIT, MEMORY_SIZE a SYSTEM_NAME are ignored. The effect of the last two mentioned pragmas can be obtained using SSAdaP runtime system commands. STORAGE_UNIT is preset to 8 and cannot be modified. Pragma INTERFACE provides binding to MACRO32 and VAX C at present.

Implementation Dependent Pragmas

```
pragma ALIGNLESS (type_simple_name);
-- objects of the given scalar type can be placed on any
-- alignment, inclusiv bit alignment

pragma BORDER;
-- separates compilation units explicitly
```

Data Types Representation

type	byte	remark
enumeration	1 4	byte in the range 0..255 or -128..127 longword in other cases
SHORT_INTEGER	2	word
INTEGER	4	longword
LONG_INTEGER	8	quadword
SHORT_FLOAT	4	F-FLOATING
FLOAT	8	G-FLOATING
LONG_FLOAT	16	H-FLOATING
access	4	longword
task	4	longword
fixed	4	longword
length	4	longword

SSAdaP IMPLEMENTATION AIMS

The main part of this system has been written in Ada. Except runtime system and debugger, which have been written in macroassembler, only few hundreds of lines are in it from more than 200 000 lines. This allows the high-level portability to other computers and operating systems. In UNIX versions, macroassembler is beeing replaced by C language.

BIBLIOGRAPHY

[1] Vladík, J.: Ada/EC/SM - uživatelská příručka pro SM 52/12 s VOS 4; Kancelářské stroje Praha 1989

[2] SSAdaP Users Manual; SzKI Budapest 1989

[3] SSAdaP System Programmers Manual; SzKI Budapest 1989