Internet Backplane Protocol – Test Language v 1.0

Alessandro Bassi Xiang Li

I. Introduction

In this paper the IBP-Test Language (IBP-TL) is specified. The IBP-TL is a langauge and a set of tools developed to test the correct functionality of the IBP depot and of the Client Library, and the semantics of the IBP protocol. It cannot be used for any other purpose, as the data used for testing cannot be set and are completely meaningless. For more information about IBP, please refer to [2]; for a complete description to the API, please refer to [1]

II. Syntax

A. Comments

The character % introduce a comment. The comment symbol has to be at the beginning of the line, and the whole line is considered as comment (therefore ignored).

B. Keywords

The Table 1 represents the keywords used in IBP-TL. Despite some of those might be used as varaible names without causing an error, such use is strongly discouraged.

C. Variables

In the IBP-TL space there are only three variables:

- DEPOT *alias host port*
- ATTRIBUTES *alias storage_type reliability duration*
- TIMER *alias ClientTimeout ServerSync*

The first variable class links an alias to an IBP Depot, which is defined as host :port as usual.
The second links an alias to a set of attributes; the storage_type attribute can be any of the canonical IBP types, that is ByteArray (expressed with BA), Buffer (BU), FIFO Queue (FI) or Circular Queue (CQ); the reliability can be any of the canonical IBP reliabilities, that is Stable (ST) or Volatile (VL). The duration can be –1 (permanent), or the number of days that the capability is supposed to live.
The third variable class links an alias to a client timeout and to a server timeout; the values are expressed in seconds.
More information about these IBP internal structures can be found in [1]

D. Declarations

API

Begins an area where a sequence of API calls is made. The area must be closed by an END statement. Cannot be nested.

```
Ex :
API
AL depot T1 1024 A1 C1
ST C1 T1 1024
LD C1 T1 1024 0
MA DEC C1 T1
END
```

CHRON

Starts a timer. This call must be within an API area. The timer is stopped by an END statement. Can be nested.

```
Ex :
API
  CHRON tim
    CHRON storeT
      REPEAT 1000
      ST C1 T1 1024
      END
    END
    CHRON loadT
      REPEAT 1000
      LD C1 T1 1024 0
      END
    END
  END
PRINT tim
PRINT storeT
PRINT loadT
END
```

AL	API	ATTRIBUTES	CHRON
CNG	CP	DEC	DEPOT
DS	END	ERROR	IN
INC	INQ	LD	MA
MC	OUT	PARALLEL	PERFTIMER
PRB	PRINT	PROTOCOL	REPEAT
SET	SLEEP	ST	THREAD
TIMER			

Fig. 1 – reserved Keywords

END

Closes an area.

ERROR

Allows the interpreter to either ignore the errors, or to stop execution. Its syntax is :
ERROR IGNORE
ERROR STOP
The area are closed by an END. Cannot be nested.

NB this directive is not implemented in the actual version of the code

PARALLEL

Starts an area where the code has to be executed in parallel threads. Has to be used with the THREAD directive; other API instructions have to be included between THREAD … END areas. All THREAD areas are executed in parallel; all instruction within any THREAD … END area are executed sequentially. Closed by END. Cannot be nested.

```
Ex :
PARALLEL
  THREAD
  ST C1 T1 1024
  END
  THREAD
  LD C1 T1 1024 0
  END
END
```

PRINT

Prints the variable that follows

```
Ex:
PRINT depot
```

PROTOCOL

Begins an area where a sequence of PROTOCOL calls is made. The area is finished by an END statement. Cannot be nested

```
Ex :
PROTOCOL
OUT 0 FD string DATAVAR NULL
IN 0 FD IBP_OK size
END
```

REPEAT n

Indicates an area to be repeated n times. The area is closed by an END. Can be nested.

```
Ex:
API
REPEAT 10
  AL depot T1 1024*100 A1 C1
  REPEAT 100
    ST C1 T1 1024
  END
  REPEAT 100
    LD C1 T1 1024 0
  END
  MA DEC C1 T1
END
END
```

SET

Begins an area where variables are set. The area is finished by an END statement. Cannot be nested

Ex :
SET
DEPOT depot toto.cs.utk.edu 6714
END

SLEEP n

Stops the execution of the script for n seconds.

THREAD

Starts an area where the code has to be executed sequentially. Has to be used with PARALLEL. Closed by END. Cannot be nested. See PARALLEL for more details.

E. API calls

AL

Calls IBP_allocate. The syntax is :
AL *depot timer size attributes capname*

ST

Calls IBP_store The syntax is :
ST *capname timer size*

CP

Calls IBP_copy. The syntax is :
CP *readcapname writecapname timer size offset*
NB : the timer is the same for both source and destination

MC

Call IBP_mcopy. At present, this call is not implemented yet.

LD

Calls IBP_load The syntax is :
LD *capname timer size offset*

MA

Calls IBP_manage. The syntax is :
MA *managecommand capname timer*
Managecommand can be any of the following :
INC – IBP_INCREMENT
DEC – IBP_DECREMENT
CNG – IBP_CHANGE
PRB – IBP_PROBE

DS

Calls IBP_status. The syntax is :
DS *statuscommand depot timer [stablestorage volstorage duration]*
Status command can be any of the following :
INQ – IBP_ST_INQ
CNG – IBP_ST_CHANGE
The last 3 fields are required if the command is CHANGE, ignored otherwise.

NB: There is intentionally no possibility to set the password; IBP-TL will use as a depot password 'IBP'. As this tool is made just for testing, we believe it's not safe to give it the possibility of modifying real allocation space in real depots; therefore, we limit this command to depots which have the standard IBP password.

F. Protocol calls

The PROTOCOL calls are made to test the robustness of the server, in case of a badly-formed IBP message, and the semantics of the protocol itself. In this section only two commands are allowed.

IN

Specifies a communication unit that has to be received from an end point

OUT

Specifies a communication unit that has to be sent to an end point

In a PROTOCOL area, the first sub-command must be OUT; its first parameter shows the type of IBP call, and the number of parameters. As an example, we provide here a description of the API calls decomposed into PROTOCOL calls

IBP-allocate
PROTOCOL
OUT ibp_allocate depot timer size attributes capname
IN data depot timer size attributes capname
END

IBP-store
PROTOCOL
OUT ibp_store capname timer size
IN size ibp_store capname timer size
OUT data ibp_store capname timer size
IN data ibp_store capname timer size
END

IBP-load
PROTOCOL
OUT ibp_load capname timer size offset
IN size capname timer size offset
IN data capname timer size offset
END

IBP-copy
PROTOCOL
OUT ibp_copy source-cap target-cap timer size offset
IN data source-cap target-cap timer size offset
END
Only the first source-cap is valid.

IBP-manage
PROTOCOL
OUT ibp_manage sub-command capname timer
IN OK sub-command capname timer
END

IBP-status
This call has two different impementations, according to the STATUS command :

PROTOCOL
OUT ibp_status INQ depot timer
IN OK INQ depot timer
END

PROTOCOL
OUT ibp_status CNG depot timer stablestor volstor duration
IN size CNG depot timer stablestor volstor duration
OUT data CNG depot timer stablestor volstor duration
IN data CNG depot timer stablestor volstor duration
END

NB. As explained in the previous section, there is no possibility to set the password, as the password is the IBP standard one ('IBP').

G. Examples

i .The following script is equivalent to the actual ibp-smoketest for windows :

```
SET
DEPOT D1 toto.cs.utk.edu 6714
DEPOT D2 titi.cs.utk.edu 6714
TIMER T1 2 2
ATTRIBUTES A1 BA ST -1
END
API
AL D1 T1 1024 A1 C1
ST C1 T1 1024
AL D2 T1 1024 A1 C2
CP C1 C2 T1 1024 0
MA PRB C2 T1
DS INQ D2 T1
LD C2 T1 1024 0
MA DEC C1 T1
MA DEC C2 T1
END
```

ii. The following script is equivalent to the old ibp-quicktest

```
SET
DEPOT D1 toto.cs.utk.edu 6714
TIMER T1 2 2
ATTRIBUTES A1 BA ST 3600*24
END
API
AL D1 T1 1024*1000 A1 C1
REPEAT 1000
ST C1 T1 1024
END
REPEAT 1000
LD C1 T1 1024 0
END
MA DEC C1 T1
END
```

III. TEST SHELL

This shell offers to the user a simple interface to the test driver. Since the purpose of the test driver is very basic, the shell will only accept the following commands.

More filename: With this command, the user can view the test script file. The shell will invoke a "more" command to show the file to the user.
Vi filename: With this command, the user can edit the test script file. The shell will invoke a "vi" command to edit the file.
Run filename: Run the test script file to do the test. This is the part of the software we developed. We will discuss the detail design of this part in the following paragraphs.
Exit: Quit from the test shell.

When the test shell is invoked, the prompt is set to IBP-TL>

IV. INTERPRETER

To run a test, we need to make the script file as the input for the interpreter. The output of the interpreter will be used as PRINT (or error) information during the execution.
To interprete the test script file we need to load the whole file into memory. Since any line can hold one and only one command (or comment line), we load the file line by line.
The interpreter will then use a state machine to interpret and execute the script file.

SET area:
Most of the variables are defined in the SET field. In this field, DEPOT, ATTRIBUTES, TIMER can be used to define variables. All variables are recorded in a variable table.

API area:
This area is composed of sequence of API calls. There are seven IBP API calls and some other calls, such as REPEAT, CHRON, SLEEP and PRINT. In this area, most of the commands will be interpreted and executed sequentially. They will be interpreted into the relative IBP calls, and the parameters defined in SET area can be found in the variable table by comparing their names.

However, there are three exceptions in this area, CHRON, REPEAT and PROTOCOL.
For CHRON, we need to invoke a special function. First, it defines a new timer in the variable table and starts it. Then, it interprets and executes the commands in its field sequentially. When it meets the relative END, it stops the timer and quit from the function.
For REPEAT, we also need to call a special function. This special function first remembers the line where the loop started. Then, it interprets and executes the commands in its field sequentially. When it meets the relative END, it comes back to the line where the loop started. After repeating the number of times set in the REPEAT line, the function quits.
It is interesting to notice that those two calls can be nested.
For PROTOCOL, the interpreter fills the Communication Units and executes them, skipping the IBP Client Library. This way, the semantic of the communication and the robustness of the server can be tested.

END:
When the file is over, the interpreter will stop, and the CPU control will be returned to the Test Shell.

V SOFTWARE STRUCTURE

A. Important Data Structures

i. Variable:

```
Struct
{
  char    *name;
  int     type;
  void    *value;
}*variable;
```

This is the structure that represents an IBP-TL variable. There is a global array of this structure for all the variables used in one test script file. When the API uses a variable defined in the SET area, th einterpreter needs to search the whole variable table to find the variable which has the same name; but, as the number of variables is normally rather small, this has no (or extremely little) performance drawbacks. The types of the variables will be defined later.

ii. Line Command

```
Struct
{
  int     argc;
  char    *argv;
} *line_cmd;
```

This is the structure used to represent a command line. There is a global array of this structure for the whole script file. We have to read the whole script file into memory, to allow loops.

And there are some other data structures for global variable, which may be decided later.

B. Some Illustration of the Test Driver

We don't check the grammar of the script file at first. If we find any mistake of the syntax while executing the file, the execution will stop at the current line, unless otherwise defined with the ERROR directive. The test script can be only accepted as a file. The test shell can't recognize the test language, which can only be recognized by the interpreter.
At this time, some functions are not yet implemented (like MCOPY and ERROR), and will be added to the test driver at a later date.

REFERENCES

1. Bassi, A., Beck, M, Plank, J, Wolski, R. Internet Backplane Protocol : API 1.0, University of Tennessee, Knoxville, 2001.
2. Plank, J., Beck, M, Elwasif, W, Moore, T, Swany, M, Wolski, R, The Internet Backplane Protocol: Storage in the Network. in *NetStore99*, (Seattle, WA, 1999).