Fastest: a Model-Based Testing Tool for the Z Notation

Maximiliano Cristiá
Flowgate Consulting and CIFASIS
Rosario – Argentina
mcristia@flowgate.net

Pablo Albertengo
Flowgate Consulting
Rosario – Argentina
palbertengo@flowgate.net

Pablo Rodríguez Monetti
Flowgate Consulting and UNR
Rosario – Argentina
prodriguez@flowgate.net

***Abstract*—Fastest is a model-based testing tool for the Z notation providing an almost automatic implementation of the Test Template Framework. The core of this document is an example showing how to use Fastest to automatically derive abstract test cases from a Z specification.**

I. Model-Based Testing

Model-based testing (MBT) is a well-known technique aimed to test software systems from a formal model [1], [2]. MBT approaches start with a formal model or specification of the software, from which test cases are generated, as shown in Fig. 1. These techniques have been developed and applied to models written in different formal notations, such as Z [3], finite state machines and their extensions [4], B [5], algebraic specifications [6], and so on.

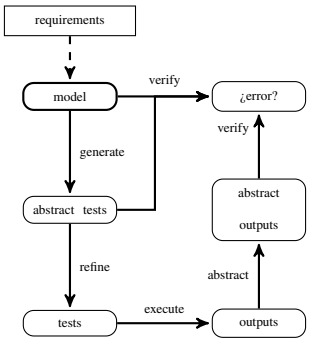

Figure 1: The MBT process.

The fundamental hypothesis behind MBT is that, as a program is correct if it verifies its specification, then the specification is an excellent source of test cases. As shown in Fig. 1, once test cases are derived from the model, they are refined to the level of the implementation language and executed. The resulting output is then abstracted to the level of the specification language, and the model is used again to verify if the test case has detected an error.

II. The Test Template Framework

The Test Template Framework (TTF) described by [3] is a particular MBT theory specially well suited for unit testing. The TTF uses Z specifications [7] as the entry models. Each operation within the specification is analysed to derive or generate abstract test cases. This analysis consists of the following steps:

1) Consider the valid input space (VIS) of each Z operation.
2) Apply one or more testing tactics in order to partition the input space.
3) Generate test objectives (specifications).
4) Prune inconsistent test objectives.
5) Find one abstract test case from each remaining test objective.

One of the main advantages of the TTF is that all of these concepts are expressed in the same notation of the specification, i.e. the Z notation. Hence, the engineer has to know only one notation to perform the analysis down to the generation of abstract test cases.

III. Fastest

Fastest implements the TTF allowing users to automatically produce test cases for a given Z specification. In [8] we show how Fastest semi-automates steps 1, 2 and 4 of the previous list; on the other hand, SEFM 2010 has accepted another paper written by us where we show how step 3 has been semi-automated too[1]. As far as we know, Fastest is the first MBT tool for the Z notation; surely it is the first one in implementing the TTF.

As is noted in [5], the TTF method was "quite widely known and referenced since" its first international publication. However, perhaps its lack of tool support made the MBT community to lose interest in it. Actually, the TTF's original authors only implemented Tinman [9] but, as Legeard in his colleagues say, "it was aimed primarily at providing organization support with little support for manipulating predicates". In effect, these authors conclude, after applying the TTF to a case study, that "TTF generation is a manual process, requiring extensive expertise at manipulating and simplifying schemas." Moreover, they found that the Z/EVES theorem prover [10] was not useful at performing those tasks since "in this case study the predicates were too difficult for the automatic simplification commands of Z/EVES." Finally, Legeard, Peureux and Utting conclude,

[1]The accepted paper is "Pruning Testing Trees in the Test Template Framework by Detecting Mathematical Contradictions".

among other things, that "a higher level of automation of the reasoning support would be useful" for the TTF, and that "BTT[2] is better designed for automation than TTF".

Our intention is to use this tool demonstration to show that these conclusions are at least doubtful.

IV. Architecture and Technology

Fastest is a Java application that should run on any platform running Java 1.6 or higher. It is based on the Community Z Tools (CZT) project [12], thus it reads Z specifications compliant with the Z ISO standard [13].

The tool was envisioned as a client-server application. The main reason for thinking of a distributed system came from the realization that calculating abstract test cases from test objectives in large projects could be a hard computing problem, but highly parallelizable as well. Then, we thought of an scalable application using the idle computer power present in a corporate network. A typical Fastest installation, thus, has some client processes and some testing servers.

Users interact with the application through the clients. The user interface of the client software is text-based, similar to command-line applications like Linux's bash, from which users can issue commands. Fastest asks each testing server to calculate a test case for a particular test class. Then, the time to find abstract test cases decreases proportionally with the number of available testing servers. This parallelization is so efficient because each calculation is completely independent from each other; synchronization is only needed when testing servers communicate a result to a client.

V. An Example

In this section we show how to use Fastest to derive abstract test cases for the Z specification described in Fig. 2. The specification is about the behaviour of the savings accounts system of simple bank. Table I summarizes the meaning of each basic type, state variable and operation. We think that this table plus the common knowledge about savings accounts will suffice to understand the model—we assume the reader is familiar with the Z notation.

The goal is, thus, to automatically derive abstract test cases—i.e. test cases written in Z—from this model by using Fastest. These abstract test cases can then be refined into a programming language—but this is out of the scope of this demonstration. As we have said, Fastest implements the TTF so we must follow it to derive abstract test cases. The next sections roughly follows the steps of the TTF as implemented by Fastest. Please, see the user manual for more details.

A. Writing the Specification

The specification must be written in the standard Z LaTeX mark-up [13] using any text editor. We suggest, however, to use Eclipse [14] with the CZT [15] and TeXlipse [16] plugins.

[2]BTT is a similar method for the B notation [11].

B. Launching Fastest

Fastest is executed from a command line issuing the following command from the installation directory:

```
java -jar fastest.jar
```

As we have said, the tool features a text-based user interface which prints a prompt and waits for commands:

```
Fastest>
```

C. Loading the Specification and Selecting Operations

Assuming the LaTeX file containing the specification is located in the installation directory, the specification is loaded with:

```
loadspec bank.tex
```

Once the specification has been successfully loaded the user has to select those schemas that represent the operations to be tested. Each operation is selected by entering a command like this:

```
selop NewClient
```

We want to test, also, the following operations:

```
selop NewAccount
selop Deposit
selop Withdraw
selop CheckBalance
selop AddOwner
```

D. Adding Testing Tactics

Testing tactics are the means proposed by the TTF to partition the VIS of a given operation. The more testing tactics added to an operation the more abstract test cases will be generated. However, it is not only a matter of adding many testing tactics but, better, the most promising ones for each operation. Fastest adds to any operation a testing tactic named DNF [8]. DNF is the first tactic to be applied. The user can add other tactics with, for instance:

```
addtactic NewClient SP \cup
          clients \cup \{u? \mapsto name?\}
```

In this example we add the following tactics, although they might not be the best choice, since we just want to make a demonstration of Fastest's capabilities.

```
addtactic NewAccount SP \notin
          n? \notin \dom balances
addtactic Deposit SP \oplus
          balances \oplus
          \{n? \mapsto balances~n? + m?\}
addtactic Withdraw NR m?
          \langle 10, 1000, 1000000\rangle
addtactic CheckBalance SP \in
          u? \mapsto n? \in owners
```

$[ACCNUM, UID, NAME]$
$MONEY == \mathbb{N}$
$BALANCE == \mathbb{N}$

Bank
$clients : UID \pfun NAME$
$balances : ACCNUM \pfun BALANCE$
$owners : UID \rel ACCNUM$

NewClientOk
$\Delta Bank$
$u? : UID;\ name? : NAME;\ n? : ACCNUM$

$u? \notin \dom clients$
$n? \notin \dom balances$
$clients' = clients \cup \{u? \mapsto name?\}$
$balances' = balances \cup \{n? \mapsto 0\}$
$owners' = owners \cup \{u? \mapsto n?\}$

$ClientAlreadyExists == [\Xi Bank;\ u? : UID \mid u? \in \dom clients]$

$AccountAlreadyExists == [\Xi Bank;\ n? : ACCNUM \mid n? \in \dom balances]$

$NewClient ==$
$NewClientOk \lor$
$ClientAlreadyExists \lor AccountAlreadyExists$

NewAccountOk
$\Delta Bank$
$u? : UID;\ n? : ACCNUM$

$u? \in \dom clients$
$n? \notin \dom balances$
$balances' = balances \cup \{n? \mapsto 0\}$
$owners' = owners \cup \{u? \mapsto n?\}$
$clients' = clients$

$ClientNotExists == [\Xi Bank;\ u? : UID \mid u? \notin \dom clients]$

$NewAccount ==$
$NewAccountOk \lor$
$ClientNotExists \lor AccountAlreadyExists$

DepositOk
$\Delta Bank$
$n? : ACCNUM;\ m? : MONEY$

$n? \in \dom balances$
$m? > 0$
$balances' = balances \oplus \{n? \mapsto balances\, n? + m?\}$
$clients' = clients$
$owners' = owners$

$AccountNotExists ==$
$[\Xi Bank;\ n? : ACCNUM \mid n? \notin \dom balances]$

$IncorrectAmount == [\Xi Bank;\ m? : MONEY \mid m? \leq 0]$

$Deposit ==$
$DepositOk \lor AccountNotExists \lor IncorrectAmount$

WithdrawOk
$\Delta Bank$
$u? : UID;\ n? : ACCNUM;\ m? : MONEY$

$u? \mapsto n? \in owners$
$n? \in \dom balances$
$m? > 0$
$m? \leq balances\, n?$
$balances' = balances \oplus \{n? \mapsto balances\, n? - m?\}$
$clients' = clients$
$owners' = owners$

$NotAnOwner ==$
$[\Xi Bank;\ u? : UID;\ n? : ACCNUM \mid u? \mapsto n? \notin owners]$

$InsufficientFunds ==$
$[\Xi Bank;\ u? : UID;\ n? : ACCNUM;\ m? : MONEY \mid m? > balances\, n?]$

$Withdraw ==$
$WithdrawOk$
$\lor AccountNotExists$
$\lor IncorrectAmount$
$\lor NotAnOwner \lor InsufficientFunds$

CheckBalanceOk
$\Xi Bank$
$u? : UID;\ n? : ACCNUM$
$balance! : MONEY$

$u? \mapsto n? \in owners$
$n? \in \dom balances$
$balance! = balances\, n?$

$CheckBalance ==$
$CheckBalanceOk$
$\lor AccountNotExists \lor IncorrectAmount$

AddOwnerOk
$\Delta Bank$
$u?, t? : UID;\ n? : ACCNUM$

$u? \mapsto n? \in owners$
$t? \mapsto n? \notin owners$
$owners' = owners \cup \{t? \mapsto n?\}$
$clients' = clients$
$balances' = balances$

$OwnerAlreadyExists ==$
$[\Xi Bank;\ t? : UID;\ n? : ACCNUM \mid t? \mapsto n? \in owners]$

$AddOwner ==$
$AddOwnerOk \lor$
$NotAnOwner \lor OwnerAlreadyExists$

Figure 2: A Z specification of the savings accounts of a banking system.

Term	Meaning
ACCNUM	The set of possible savings accounts numbers
UID	The set of identifiers of individuals (social security numbers, for instance)
NAME	The set of names of individuals
clients u	The name of person *u* as is recorded in the bank
balances n	The balance of savings account *n*
$owners(u, n)$	*u* is an owner of account *n*
$NewClient(u, name, n)$	Account *n* is opened by client *u* whose name is *name*
$NewAccount(u, n)$	Client *u* opens a new account with number *n*
$Deposit(n, m)$	The amount *m* is deposited in account *n*
$Withdraw(u, n, m)$	Client *u* withdraws amount *m* from account *n*
$CheckBalance(u, n, b)$	*b* is the balance of account *n* when client *u* checks it
$AddOwner(u, t, n)$	Client *u* adds *t* as an owner of account *n*

Table I: Meaning of the basic elements of the Z model of Fig. 2.

```
addtactic AddOwner SP \in
          u? \mapsto n? \in owners
addtactic AddOwner SP \notin
          t? \mapsto n? \notin owners
```

E. Generating Test Objectives

Test objectives—or specifications, classes or design—are automatically generated by running the following command:

```
genalltt
```

In doing so, Fastest performs a number of predicate manipulations as Legeard and his colleagues required in [5]. These objectives are structured as *testing trees*. Abstract test cases should be generated only from the leaves of these trees. In other words, each leaf stipulates some conditions under which the implementation must be tested.

F. Pruning Testing Trees

Although Fastest automatically generates test objectives, some of them may represent impossible situations. According to the TTF, each test objective is a set. Then, a test objective represents an impossible situation when its predicate is unsatisfiable. Unsatisfiable leafs should be pruned from testing trees. The automatic pruning strategy implemented in Fastest is executed with the following command:

```
prunett
```

This command, as `genalltt`, performs a series of predicate manipulations but of a different sort. The conception, design and implementation of this command is the subject of the paper published at this conference.

G. Deriving Abstract Test Cases

Deriving an abstract test case from a test objective means to find a vector of constant values satisfying the predicate of the objective. This task is performed with command:

```
genalltca
```

This is a much slower process compared to automatic pruning. When this command finishes some leaves have a child hanging from them representing the abstract test case. Each abstract test case is a Z schema box. The difference between a test objective and an abstract test case is that in the latter each input and state variable is bound to a constant value as exemplified in the following schema:

Deposit_SP_4_TCASE

Deposit_SP_4

$m? = 1$
$balances = \{(accnum0, 1)\}$
$clients = \emptyset$
$n? = accnum0$
$owners = \emptyset$

In this example, Fastest finds automatically all the abstract test cases but eleven. When this happens the user has to check whether the problematic objectives are satisfiable or not. In our case all but four are satisfiable. For those that are satisfiable, the user needs to help Fastest to find the required constants with a command like this one:

```
setfinitemodel CheckBalance_SP_5 -fm
   "owners==\{\{uid0 \mapsto accnum0\}\}"
```

For those that are not, the user has to add an *elimination theorem*; for instance:

ETheorem ArithmIneq4 [const $N, M : \mathbb{Z};\ n : \mathbb{Z}$]
$\mathsf{eval}(N \leq M)$
$n \leq N$
$M < n$

Elimination theorems are at the core of `prunett`; see the paper published by us at SEFM 2010.

H. Summary of Results

Table II summarizes the result of this experiment. As can be seen, Fastest generates 39 abstract test cases automatically from 46 possible test objectives. This high percentage is the consequence of two factors: (a) the high number of inconsistent test objectives removed by the automatic strategy, and (b) the heuristics implemented by `genalltca` to find constants values satisfying a given predicate. These

Operation	Leaves	Pruned Auto	Pruned Man	Remaining	ATC Auto	ATC Man
NewClient	24	15	0	9	9	0
NewAccount	6	1	0	5	5	0
Deposit	24	20	0	5	5	0
Withdraw	20	2	2	16	11	5
CheckBalance	6	0	0	6	4	2
AddOwner	12	8	0	5	5	0
Total	**92**	**46**	**2**	**46**	**39**	**7**
Total pruning time		**3.2s**	**Total ATC derivation time**		**38m40s**	

Table II: Summary of the results. Auto stands for automatically, Man for manually, and ATC for abstract test cases.

results are aligned with our previous experiments, see [8] and the paper to be published at SEFM 2010.

VI. Discussion

Fastest provides Z users with an almost automatic implementation of a sound MBT method originally thought for the Z notation. In this regard, Fastest overcomes all of the issues found by Legeard, Peureux and Utting in [5] making their last conclusion doubtful.

Fastest is freely available from [17], including a complete user manual in English. We have added entries to Wikipedia describing Fastest [18] and the TTF [19].

Perhaps there are many Z users out there thinking that MBT is an ideal technique for their daily software engineering work. Meanwhile, they watch other people using good MBT tools for their "younger" notations, but they found no implementation available for their beloved one. We hope Fastest could fill this gap.

References

[1] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[2] R. M. Hierons and et al., "Using formal specifications to support testing," *ACM Comput. Surv.*, vol. 41, no. 2, pp. 1–76, 2009.

[3] P. Stocks and D. Carrington, "A Framework for Specification-Based Testing," *IEEE Transactions on Software Engineering*, vol. 22, no. 11, pp. 777–793, Nov. 1996.

[4] W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes, "Generating finite state machines from abstract state machines," in *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis*. New York, NY, USA: ACM, 2002, pp. 112–122.

[5] B. Legeard, F. Peureux, and M. Utting, "A Comparison of the BTT and TTF Test-Generation Methods," in *ZB '02: Proceedings of the 2nd International Conference of B and Z Users on Formal Specification and Development in Z and B*. London, UK: Springer-Verlag, 2002, pp. 309–329.

[6] G. Bernot, M. C. Gaudel, and B. Marre, "Software testing based on formal specifications: a theory and a tool," *Softw. Eng. J.*, vol. 6, no. 6, pp. 387–405, 1991.

[7] J. M. Spivey, *The Z Notation: A Reference Manual*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[8] M. Cristiá and P. Rodríguez Monetti, "Implementing and applying the Stocks-Carrington framework for model-based testing," in *ICFEM*, ser. Lecture Notes in Computer Science, K. Breitman and A. Cavalcanti, Eds., vol. 5885. Springer, 2009, pp. 167–185.

[9] L. Murray, D. Carrington, I. Maccoll, and P. Strooper, "TinMan - A Test Derivation and Management Tool for Specification-Based Class Testing," in *In Technology of ObjectOriented Languages and Systems (TOOLS)*, 1999, pp. 222–233.

[10] M. Saaltink, "The Z/EVES System," in *ZUM '97: The Z Formal Specification Notation*, J. Bowen, M. Hinchey, and D. Till, Eds., 1997, pp. 72–85.

[11] L. B. P. F. Bouquet F., "Constraint logic programming with sets for animation of B formal specifications," in *CL'00 Workshop on (Constraint) Logic Programming and Software Engineering (LPSE'00)*, London, UK, 2000.

[12] CZT. Community Z Tools (CZT) project. [Online]. Available: http://czt.sourceforge.net

[13] ISO, "Information Technology – Z Formal Specification Notation – Syntax, Type System and Semantics," International Organization for Standardization, Tech. Rep. ISO/IEC 13568, 2002. [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/c021573_ISO_IEC_13568_2002(E).zip

[14] The Eclipse Foundation. Eclipse. [Online]. Available: http://www.eclipse.org/

[15] CZT. CZT Eclipse Plugin. [Online]. Available: http://www.cs.waikato.ac.nz/~marku/czt/eclipse.html

[16] T. Hupponen, K. Karlsson, J. Laitinen, O. Ojala, A. Pirinen, E. Seuranen, and L. Takkinen. TeXlipse. [Online]. Available: http://texlipse.sourceforge.net/

[17] Flowgate Consulting. Fastest. [Online]. Available: http://www.flowgate.net/?lang=en&seccion=herramientas

[18] ——. Fastest in wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/Fastest

[19] ——. Test template framework in wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/Test_Template_Framework