Z to Java

Stephen Murrell[1] **Robert Plant[2]**

[1]Department of Electrical and Computer Engineering
[2]Department of Computer Information Systems
University of Miami, Coral gables, FL 33124

Tel: (305) 284-1963
Fax: (305) 284-5161
rplant@umiami.miami.edu

Abstract

The commercial demand to create industrial-strength heterogeneous systems on a networked platform for mission critical systems requires that a formal development approach be taken. We describe an initial attempt at creating Java programs from Z specifications.

1. Introduction

Recent work in the area of validation and verification has started to move the focus of the research away from the static and dynamic testing of knowledge-based systems [Gupta.91] to more formal notations in an attempt to rigorously define the properties of the systems. The earlier emphasis on testing was due in part to the relative youth of the field and the need to establish a basic set of techniques from which to develop a special understanding of these systems. The testing approach primarily utilised established techniques used in the verification of conventional systems and adapted them for use in the declarative knowledge-based systems [Preece.92]. The application of conventional techniques focused primarily upon the identification of five anomalies in rule sets, these being circularity, subsumption, redundancy, dead-end rules and inconsistency [Nguyen.85]. This has led to the creation of over thirty five tools that consider this problem [Murrell.97]. It can be seen therefore that this area is now well understood and has a large and extensive literature.

The early work of the validation and verification of knowledge-based systems was principally focused upon rule-based systems of a stand alone nature [Nguyen.85]. However, along with the growing understanding of these system came a need to embed them into more complex environments, hence the need to perform validation and verification on complex heterogeneous systems [Murrell.96]. Again, techniques from the domain of conventional systems testing were considered. These are documented extensively by Miller [Miller.95]. The limitations of these approaches for heterogeneous industrial-strength systems having been understood required researchers to utilise more formal techniques and more rigorous development techniques [Murrell.96]. The research into the application of formal methods to the validation of knowledge-based systems can be split into two primary fields: work in the specificational aspects of systems and the work in the area of logic and languages. We will consider these in more detail in the next two sections.

2. Formal Approaches to the Validation of KBS

Work on the application of formal methods to knowledge-based systems has focused upon several areas. The use of mathematical notations to specify and derive rule-based programs [Roman.93], the specification of production system architectures [Gold.95], the specification of the semantics of a production system [Murrell.95], as well as the application of formal notations to domains such as medicine [Todd.95]. Recent work has built upon these foundations and applies notations such as Z to the problem of specifying heterogeneous systems [Murrell. 96], as well as the application of description logics [Rousset and Hors.96]. The area of complex systems and

the inherent complexity of formal notations has also caused a growth in research in the area of tools to support such developments, including the B-Method (B-Tool & B-Toolkit), Estelle (EDT: Estelle Development Toolkit), EVES (Z/EVES)[1], PROSPECTRA (PROgram development by SPECification and TRAnsformation) [Hoffan.91], VDM Domain Compiler [Schmidt.91], ProofPower, Mural, IPTES Toolset, DST-Fuzz [Woodcock.93]. In order to derive maximal effect from the research there has been a move to standardise[2] the notations and both Z and VDM-SL are undergoing ISO Standardisation.

3. Languages for KBS

The history of AI and knowledge-based systems is intertwined with the history of programming languages, as LISP[3] is at the core of both these two fields of computer science. The adoption of LISP as the default AI language from the early days through to the modern variances of LISP such as CLIPS[4] (C Language Integrated Production System) and OPS5 has also been interspliced with theoretical developments in programming language theory through the logic-based languages such as Prolog [Clocksin.81] and Godel, and the advent of expert systems shells such as VP-Expert, Expert,[5] ART, AIM, CRYSTAL , CxPERT[6]. However, the adoption of knowledge-based systems has been hampered by the unwillingness of industry and commerce to adopt these "esoteric" languages that fall outside of their in-house standards and programming expertise. The most successful of the expert systems implementation languages are C and C++, which are both understandable to the commercial in-house programming staffers and easily interoperates with other systems.

[1] http://www.comlab.ox.ac.uk/archive/formal-methods.html#VDM

[2] http://www.comlab.ox.ac.uk/oucl/groups/zstandards/

[3] http://www8.informatik.uni-erlangen.de/html/lisp/histlit.html

[4] http://rodau.ed.umuc.edu/~rshecter/inss555/clips/cliphtml/vol1.html

[5] http://knight3.cit.ics.saitama-u.ac.jp/ai/expert.html

[6] http://www.ioe.ac.uk/hgm/expert3.html

4. KBS Through Java

The use of C and C++ as preferred development languages for knowledge-based systems and the consequent use of these systems in a heterogeneous environment has led to their use in an increasing number of critical and operationally dependent systems. Hence the need to utilise formal techniques for specification and then map these specifications to a C/C++ environment. There are currently three tools that perform this activity: The LOTOS[7] Toolbox which supports specification in LOTOS and then via a tool set the development of C code prototypes; RAISE[8] which assists in the development of VDM-SL specifications into C++ , and PET DINGO [Woodcock.93] which support the translation of specifications written in the Estelle language into C++ code. LOTOS Toolbox and RAISE are both commercial products while PET DINGO is in the public domain.

The advent of the internet and its impact upon industry and commerce as a vehicle for electronic data interchange and electronic trading has led to the increasing need to have access to information and knowledge both intra- and inter-organisationally. This is fueling the adoption of Java [Deitel.97] as a nascent standard medium for the exchange of executable objects.

For these reasons we chose Java as the target language for the creation of executable extensions of Z specifications.

5. The System

The originally intention for Z was not to produce executable specifications, but most industrial users of Z do tend to adopt a specification style which makes schemas appear almost to be programs, with pre-conditions first, the "assignments" to output variables, and finally some post-condition testing. Our system relies to a certain extent on the procedural nature of specifications, but is capable of reordering components when necessary. This makes

[7] http://wwwtios.cs.utwente.nl/lotos/

[8] http://www.ifad.dk/

translation into a practical programming language an achievable goal.

For user convenience, we chose to accept specifications in the Latex Z style. These specifications are mechanically translated into Java applets.

6. An Illustration

For reasons of space, we take as an example the exceptionally simple specification of a counter limited to the range [0,max] with three access operations: *Initialise* sets the counter to zero; *Increase* adds some increment to the counter; *Read* returns both the value of the counter and the space remaining for further increments.

Counter

$n: \mathbb{N}$
$max: \mathbb{N}$

$n \geq 0$
$n \leq max$

Initialise

$\Delta Counter$

$n' = 0$
$max' = 10$

Increase

Δ Counter
$inc?: \mathbb{N}$

$n + inc? \leq max$
$n' = n + inc?$
$max' = max$

Read

Ξ Counter
$val!: \mathbb{N}$
$space!: \mathbb{N}$

$val! = n$
$space! = max - n$

For this style of specification, a single class definition is created; it has two integer members (n and max), three public methods, corresponding to the three schemas that include Δcounter or Ξcounter, and one private method which tests the validity of the state of a counter according to its predicates. Each of these four methods returns a boolean result, indicating success or failure. In addition to the three "active" methods for Initialise, Increase and Read, three "passive" methods are also created, for testing whether the operation is *in-domain* or not.

```
public class counter
{ public int n;
  public int max;

  public boolean valid()
  { if (!(n>=0)) return(false);
    if (!(n<=max)) return(false);
    return(true);}

  public boolean can_initialise()
  { return(true); }

  public boolean initialise()
  { if (!can_initialise())
                return(false);
    n=0;
    max=10;
    return(valid()); }
```

Initialise is always in domain (it has no preconditions), so can_initialise() always returns true. Initialise simply sets the class members as directed, and test, the validity of the result.

```
 public boolean
           can_increase(int inc)
 { if (!(n+inc<=max))
                    return(false);
   return(true); }

 public boolean increase(int inc)
 { if (!can_increase(inc))
                    return(false);
   n=n+inc;
   return(valid()); }
```

Inc? is syntactically marked as an input parameter. The translation algorithm sorts the predicates of a schema into dependency layers: predicates that contain only input variables and

constants become part of the in-domain test. Predicates containing input variables, constants, and "assignments" to new variables are translated first, making more variables available to allow translation of other predicates. This simple method is successful for most real industrial Z specifications, but can not handle "subtle" assignments (e.g. `n:N; n>3; n<5`, or `6=n+2`), a problem that is still the subject of much on-going research. Usually, predicates that appear to be multiple assignments (e.g. `n'=n+1; n'=x`) may be resolved as one assignment and one test.

```
public boolean can_read()
{ return(true); }

public boolean
    read(counter_read_output R)
{ if (!can_read())
                  return(false);
  R.val=n;
  R.space=max-n;
  return(valid()); }
```

Val! and Space! are recognised as output parameters; as Java insists upon passing simple values (such as integers) by value, and class objects by reference, a special dummy class with member representing each of the output variables must be created to force call by reference, allowing output.

```
public class counter_read_output
{ int val;
  int space; }
```

The translator also produces an "Applet" class for testing the executiable specification:

```
public class test_counter
                  extends Applet
{ ..... }
```

Test_counter draws a window with "textfields" for each of the members, and buttons for each of the methods; the buttons are surrounded by editable textfields for each of their input variables, and non-editable textfirelds for each of their output variables; the buttons are not visible when the operations they represent are not in domain. Additionally, "radio buttons" are added to indicate the validity of the schema's private members (See Figure 1).

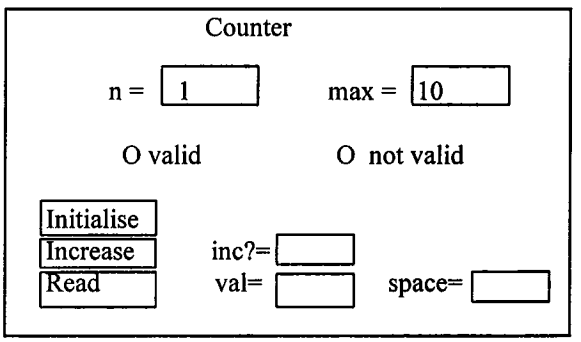

Figure 1. Counter

Additionally, the translator may produce another "applet" class that draws the originally provided Z specification for reference during testing.

7. Conclusions

The paper has shown in outline the process of deriving an executable program from a Z specification. The example use to illustrate the process was extremely simple, but the same principles apply to larger scale problems. One particular benefit is the ability to experiment with a specification remotely, in line with the aims of the ARPA CAETI[9] project, in particular this work is related to that of Gamble and Murrell under this project [Murrell. 96a].

The code produced by this mechanical translation is not particularly efficient (there is much duplication of code, especially in the domain checking), but that is not the point. We do not intend to create or replace the application itself, but to produce a remotely testable object directly and independently from its specification.

8. References

Clocksin, W.F., & Mellish, C.S., 1981
Programming in Prolog
Springer Verlag

Deitel, H.M., & Deitel, P.J., 1997
How to Program Java
Prentice Hall

[9] http://www.fwl.org/techpolicy/caeti.html

Gold, D.I., & Plant, R.T
"Towards the Formal Specification of an Expert System"
International Journal of Intelligent Systems. Vol. 9, Number 9 August 1994, pp739-768

Gupta, U. 1991
Validating and Verifying Knowledge-Based Systems
IEEE Computer Society Press

Hoffmann, B., & Krieg-Bruckner, B. 1991
"The PROSPECTRA System" In, VDM '91 Formal Software Development Methods, 4th Int. Symposium of VDM Europe, Vol. 1. Editors: Prehn, S., & Toetenel, W.J., Lecture Notes in Computer Science 551., Springer Verlag. pp668-671

Miller, L. 1995
Guidelines for the Verification and Validation of Expert Systems and Conventional Software
EPRI TR-103331, Project 3093-01
SAIC, McLean Virginia.

Murrell, S., & Plant, R.T. 1995
Formal Semantics for Rule-Based Systems
Journal of Systems & Software, Vol. 29, No 3.

Murrell, S., Plant, R.T., Gamble, R. 1996
"Formal Specification of Rule-Based Systems Through Styles"
AAAI Workshop Notes: 9th Workshop on Validation & Verification of Knowledge-Based Systems
Portland., Oregon, August 1996

Murrell, S., & Gamble, R., and Butler, P. 1996a
"ZtoRefine 1.0 Users Guide"
Tech Report UTULSA-MCS-96-14 Department of Mathematical and Computer Sciences, Univerity of Tulsa. Sept 1996

Murrell, S., Plant, R.T. 1997
"A Survey of Tools for the Validation & Verification of Knowledge-Based Systems"
Decision Support Systems (To Appear 1997)

Nguyen, T.A., Perkins, W.A., Laffey, T.J., Pecora, D. 1985
"Knowledge base verification"
IJCAI pp 375-378

Preece, A.D., Shinghal, R., Batarekh, A. 1992
"Verifying expert systems: A logical framework and a practical tool"
Expert Systems with Applications 5:421-436

Roman, G., Gamble, R.F., Ball, W. 1993
Formal Derivation of Rule-Based Programs
IEEE Trans. on Software Engineering, Vol. 19., No.3., March 1993 pp277-296

Rousset, M.C., & Hors, P. 1996
"Modeling and Verifying Complex Objects: A Declarative Approach based upon description logics"
AAAI Workshop Notes: Verification and Validation of Knowledge-based Systems and Subsystems, Schmolze, J., & Vermesan, A. Portland, Oregon. August 5th 1996

Schmidt, U., & Horcher, H., 1991
"The VDM Domain Compiler AVDM Class Library Generator" In, VDM '91 Formal Software Development Methods, 4th Int. Symposium of VDM Europe, Vol. 1. Editors: Prehn, S., & Toetenel, W.J., Lecture Notes in Computer Science 551., Springer Verlag. pp675-676

Todd, B. 1995
An introduction to Expert Systems.
Technical Monograph PRG-95
Oxford University Computing Laboratory, Oxford, England.

Woodcock, J.C.P., & Larsen, P.G., (Eds) 1993
FME '93: Industrial Strength Methods, First International Symposium on Formal Methods Europe Odense, Denmark, April 1993. Lecture Notes in Computer Science 670. Springer Verlag.