Tool Support for Event-B Code Generation

Andrew Edmunds and Michael Butler

November 18, 2009

Abstract

The Event-B method is a formal approach to modelling systems, using refinement. Initial specification is done at a high level of abstraction; detail is added in refinement steps as the development proceeds toward implementation. In previous work we developed an approach to bridge the gap between abstract specifications and implementations using an implementation level specification notation. In this paper we present details of the tool support for our notation and some of our experiences using the tool.

1 Introduction

The Event-B method [1] is a formal approach to modelling systems, with tool support. The modelling approach uses an event based view of how a system evolves atomically, from one state to another. A system's state is modelled using sets, constants and variables; and updates to state are described in the bodies of guarded events. System properties are specified in invariants and proof obligations are generated, which should be discharged in order to prove that the event actions do not violate the invariants. When modelling a software system, an Event-B model will be refined to a point where we are ready to provide information about the implementation. Consideration is given to how tasks may be performed by executing processes, and how the processes may interleave. In our paper [3] we introduce an intermediate specification language, Object-oriented, Concurrent-B (OCB), which we use to link Event-B models and object-oriented implementations, see Figure 1. Early work has targeted Java [4] as the implementation language, since it is often used to implement concurrent systems, however our work is not limited in any way to this target.

OCB is used to specify implementation details for an Event-B Development. Our system may consist of a number of processes which can perform tasks, and objects which can be shared between the processes. The processes require mutually exclusive access to the shared data; and we limit access to shared data using atomic procedure calls. The behaviour of processes is specified using non-atomic clauses, which may interleave, and labelled atomic clauses. Events of the abstract development are refined by the labelled atomic clauses in the OCB specification. Actions of the labelled atomic clauses are limited to assignment to

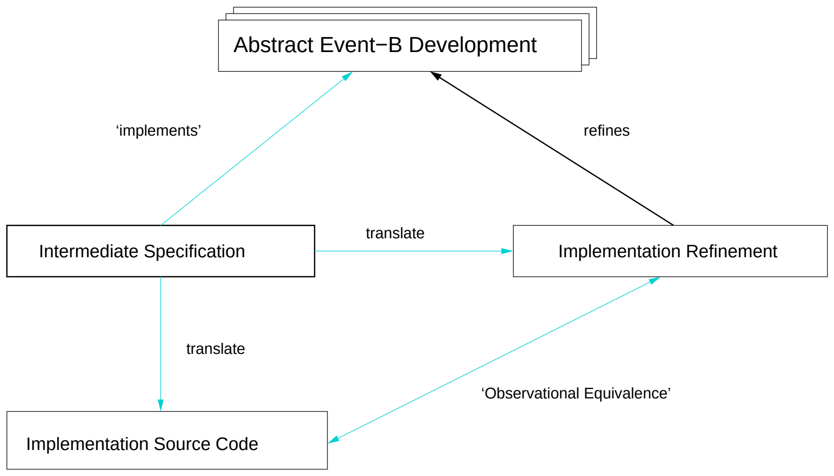

Abstract Event−B Development
'implements'
refines
Intermediate Specification
translate
Implementation Refinement
translate
'Observational Equivalence'
Implementation Source Code

Figure 1: Overview of a Development

local data, a procedure call, or a constructor call. OCB processes are specified using the *ProcessClass* construct, and shared objects are specified using the *MonitorClass* construct. Fragments of a textual version of the constructs are shown in Figures 2 and 3.

2 Tool Support

The tool support for Event-B is based on Eclipse [6], and we use the Eclipse platform's extensibility mechanism to plug-in our support for code generation. The OCB syntax is embodied in an Eclipse Modelling Framework (EMF) model which we call the OCB meta-model. During the development of the OCB meta-model we found that the syntax mapped easily to the EMF model. We added

```
ProcessClass Proc{
 // Attribute Declarations
...
 // The Constructor Procedure
 Procedure create(...){ ...}
 // The Process Behaviour
 Operation run(){
   p1:  // Labelled Atomic Clauses
   p2: ... }
}
```

Figure 2: ProcessClass Specification

```
MonitorClass Channel{
 // Attributes Declarations
...
 // The Constructor Procedure
 Procedure create(){ ... }

 // A Procedure Definition
 Procedure n(...){
   when(<guard>){<updates>}}
...
}
```

Figure 3: MonitorClass Specification

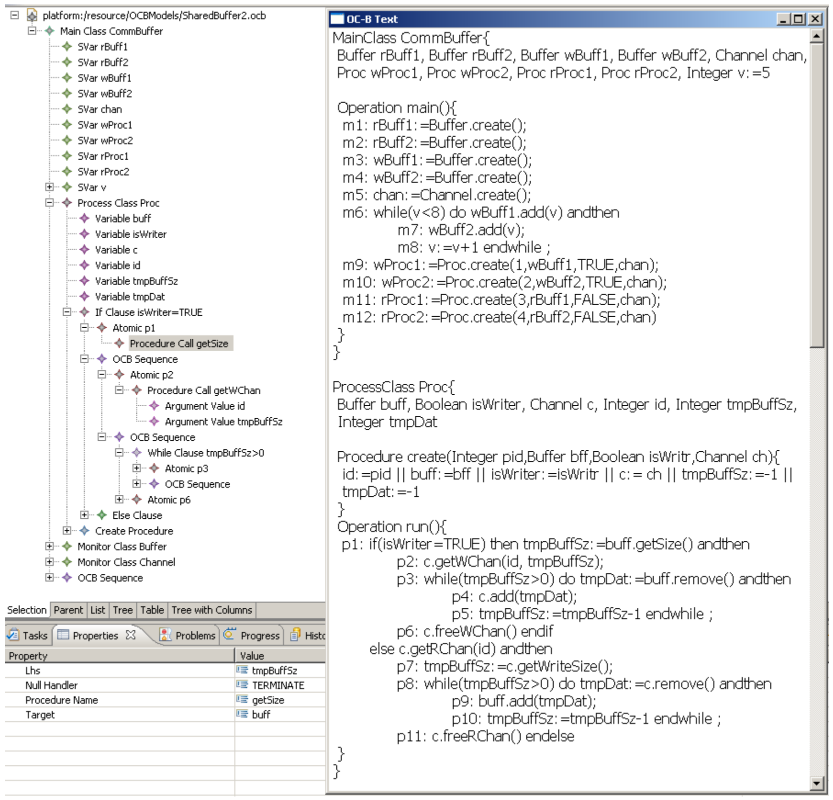
an OCB meta-model plug-in to facilitate creation and editing of OCB models in the Eclipse environment. We developed a plug-in to perform translation of an OCB model to an Event-B model and also to Java source code.

We present a screen-shot of an OCB model, in Figure 4, showing the tree editor view of the model, and a text viewer. We found that the EMF tree editor did not allow us to easily visualize the content of the OCB model, so we added a text viewer.

3 Our Experience

The OCB notation is used to provide an implementation level specification for our abstract development. An OCB specification is subsequently mapped to Java code and back to an Event-B model which refines the abstract development. The text viewer resembles text based code and assisted with the understanding of the model. A full text editor would be a desirable feature in future versions of the tool, as would a diagram based editor similar to UML-B [5] The Java code produced by the translator was suitable for execution with the JVM, and presented no real problems in translation. This was largely due to the similarity between the OCB syntax and the implementation constructs used in Java; and the integration with the Eclipse Java Development Toolkit (JDT). The JDT allows programmatic construction of a Java program.

The translation to Event-B was relatively easy, but its subsequent use presented more problems. A shortcoming of the existing tool lies in the integration of the abstract development and the implementation refinement. The current tool creates an Event-B implementation refinement that must be manually copied into a directory with the abstract development. The tooling issues related to this aspect will be relatively easy to resolve. The specification of the gluing invariants required to link the data of the abstraction to that of the implementation refinement does however require some skill, and we are exploring this area to identify patterns that will simplify the approach. We have found



```
platform:/resource/OCBModels/SharedBuffer2.ocb
  Main Class CommBuffer
    SVar rBuff1
    SVar rBuff2
    SVar wBuff1
    SVar wBuff2
    SVar chan
    SVar wProc1
    SVar wProc2
    SVar rProc1
    SVar rProc2
    SVar v
    Process Class Proc
      Variable buff
      Variable isWriter
      Variable c
      Variable id
      Variable tmpBuffSz
      Variable tmpDat
      If Clause isWriter=TRUE
        Atomic p1
          Procedure Call getSize
        OCB Sequence
          Atomic p2
            Procedure Call getWChan
              Argument Value id
              Argument Value tmpBuffSz
          OCB Sequence
            While Clause tmpBuffSz>0
              Atomic p3
              OCB Sequence
            Atomic p6
        Else Clause
      Create Procedure
    Monitor Class Buffer
    Monitor Class Channel
    OCB Sequence
```

Property	Value
Lhs	tmpBuffSz
Null Handler	TERMINATE
Procedure Name	getSize
Target	buff

```
MainClass CommBuffer{
 Buffer rBuff1, Buffer rBuff2, Buffer wBuff1, Buffer wBuff2, Channel chan,
 Proc wProc1, Proc wProc2, Proc rProc1, Proc rProc2, Integer v:=5

 Operation main(){
  m1: rBuff1:=Buffer.create();
  m2: rBuff2:=Buffer.create();
  m3: wBuff1:=Buffer.create();
  m4: wBuff2:=Buffer.create();
  m5: chan:=Channel.create();
  m6: while(v<8) do wBuff1.add(v) andthen
            m7: wBuff2.add(v);
            m8: v:=v+1 endwhile ;
  m9: wProc1:=Proc.create(1,wBuff1,TRUE,chan);
  m10: wProc2:=Proc.create(2,wBuff2,TRUE,chan);
  m11: rProc1:=Proc.create(3,rBuff1,FALSE,chan);
  m12: rProc2:=Proc.create(4,rBuff2,FALSE,chan)
 }
}

ProcessClass Proc{
 Buffer buff, Boolean isWriter, Channel c, Integer id, Integer tmpBuffSz,
 Integer tmpDat

 Procedure create(Integer pid,Buffer bff,Boolean isWritr,Channel ch){
  id:=pid || buff:=bff || isWriter:=isWritr || c:= ch || tmpBuffSz:=-1 ||
  tmpDat:=-1
 }
 Operation run(){
  p1: if(isWriter=TRUE) then tmpBuffSz:=buff.getSize() andthen
            p2: c.getWChan(id, tmpBuffSz);
            p3: while(tmpBuffSz>0) do tmpDat:=buff.remove() andthen
                      p4: c.add(tmpDat);
                      p5: tmpBuffSz:=tmpBuffSz-1 endwhile ;
            p6: c.freeWChan() endif
       else c.getRChan(id) andthen
            p7: tmpBuffSz:=c.getWriteSize();
            p8: while(tmpBuffSz>0) do tmpDat:=c.remove() andthen
                      p9: buff.add(tmpDat);
                      p10: tmpBuffSz:=tmpBuffSz-1 endwhile ;
            p11: c.freeRChan() endelse
 }
}
```

Figure 4: Comparison between OCBText and Tree Editor

that our initial approach gives rise to a great deal of fine-grained atomicity, and that this increases complexity during the proof activity. We expect to be able to rationalize the approach to reduce the complexity in a number of areas.

We found that the large size of the implementation refinements made proof slow and difficult. To overcome the problems caused by large models we intend to use model decomposition similar to that of [2]. In this way we can separate the specification of processes and shared objects into separate machines, which can subsequently be refined independently.

One problem that we have is related to the methodology - caused the large abstraction gap between the abstract development and the implementation specification. We found that deriving the OCB specification from an abstract development could be difficult, but we envisage a number of strategies that may be useful in easing the transition between the abstract development and the implementation level specification. We have looked at the use of Event Refinement Diagrams of [2] to visualize the relationships between the abstract events and labelled clauses of the implementation. We also expect to be able to mitigate the problem by introducing OCB at a higher level of abstraction. In this way we hope to introduce implementation details in a more gradual way, using refinement within the OCB specification level itself.

We have also looked at generating Ada and C source code. Our preliminary investigations suggest that this should be feasible with some changes to the intermediate specification notation.

4 Conclusion

We gave an overview of Event-B and the OCB notation, which we use to specify implementations for concurrent processes sharing data. We described the tool support required for our code generation approach, described our experience using the tools, and provided some ideas about future work in this area.

References

[1] J.R. Abrial. Event based sequential program development: Application to constructing a pointer program. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME*, volume 2805 of *Lecture Notes in Computer Science*, pages 51–74. Springer, 2003.

[2] M. Butler. Decomposition Structures for Event-B. In *Integrated Formal Methods iFM2009, Springer, LNCS 5423*, volume LNCS. Springer, February 2009.

[3] A. Edmunds and M. Butler. Linking Event-B and Concurrent Object-Oriented Programs. In *Refine 2008 - International Refinement Workshop*, May 2008.

[4] J. Gosling, B. Joy, G. Steele, and G. Bracha. *Java Language Specification - Third Edition.* Addison-Wesley, 2004.

[5] C. Snook and M. Butler. UML-B and Event-B: an integration of languages and tools. In *The IASTED International Conference on Software Engineering - SE2008*, February 2008.

[6] The Eclipse Project. Eclipse - an Open Development Platform. Available at http://www.eclipse.org/.