UML-based Approach to Specify Secured, Fine-grained Concurrent Access to Shared Resources:
to appear in ‘Journal of Object Technology’
Jan/Feb Issue of 2007. (www.jot.fm)

Jagdish Suryadevara [BITS]
Shyamasundar R.K [TIFR]
Abstract

In object oriented paradigm, a concurrent system can be regarded as a collection of autonomous active objects which synchronize and communicate through shared passive objects. In this paper, we propose a UML-based approach to specify secured, fine-grained concurrent access to shared resources ensuring data integrity and security. The goal of the approach is to develop the UML specification with precise executional semantics, yet independent of low-level synchronization primitives and implementation environment. The approach is largely inspired from the language constructs of CDL*. A light-weight extension of UML 2.0 meta-model is proposed for the required constructs and semantics. UML protocol statemachine is used to define the access protocol for shared resources and UML activity is used to specify the behavior of methods implementing plausibly concurrent operations. The UML activity construct is extended to support concurrency features; synchronization regions, mutual exclusion and conditional synchronization not supported in current UML2.0 semantic model. The approach can be easily extended to a programming framework of design and coding.
Abstract

- A concurrent system can be regarded as a collection of autonomous active objects which synchronize and communicate through shared passive objects.
- The goal of the approach is to develop the UML specification with precise executional semantics, yet independent of low-level synchronization primitives and implementation environment.
- The approach is largely inspired from the language constructs of CDL*.
- The UML *activity* construct is extended to support concurrency features; *synchronization regions, mutual exclusion* and *conditional synchronization* not supported in current UML2.0 semantic model.
Introduction

- In addition to *semantic variation points*, UML provides standard *extension mechanisms* known as *profiles, stereotypes, and constraints* to support specific domain modeling.
- In spite of several efforts, concurrency in UML remains an active research area, requiring concrete approaches to precise modeling to support the programming activity.
- UML mechanisms to specify concurrency;
  - *isActive* meta attribute of metaclass *Class* to specify whether the object is active or passive,
  - *concurrency* meta attribute of metaclass *Operation* to specify concurrent operation invocation, and
  - *orthogonal regions* of the state machines to specify concurrent activities in an object.
Inadequacies of classical UML semantics to specify fine-grained parallelism

- *readers-writers* example with writer priority
- UML active/passive object paradigm
  - Active object: *run-to-completion* semantics
    - Implicit synchronization
  - Passive object: needs external synchronization mechanisms
- Passive object semantics
  - *Active object* has a thread of control and runs in its address space while passive objects run execute their methods concurrently in the callers’ thread of control
  - *Passive objects* still have a degree of control over the invocations made towards them through the specification of meta-attribute *concurrency* with values *sequential, guarded, or concurrent*
  - A well-defined event-deferral mechanism required
Overview of CDL* constructs

**MODSPEC SHARED reader-writer**

**EXPORT**

- TYPE page : some_type;

**STATE**

- VAR buff: page;
- VAR nr, nw: INTEGER INIT 0;
- VAR writerbusy: BOOLEAN INIT FALSE;

**INVAR** [writerbusy → nr=0];

**PARALLEL** (read, read), (startread, startread),
            (endread, endread),
            (read, startread, endread)

**TRANS**

- ENTRY PROC startread
  
  - [ nw=0 → nr := nr+1 ];
  - ENTRY PROC read(OUT x:page)
    - [ x := buff ];
  - ENTRY PROC endread
    - [ nr := nr-1 ];
  - ENTRY PROC startwrite
    - [ nw := nw+1;
      ¬ writerbusy AND nr=0 →
      writerbusy := TRUE ];
  - ENTRY PROC write(IN x:page)
    - [ buff := x ];
  - ENTRY PROC endwrite
    - [ nw := nw-1;
      writerbusy := FALSE ];

**END MODSPEC SHARED reader-writer;**

**MODBODY reader-writer**

**PROC startread**

  - [ nw=0 → nr := nr+1 ];
  - DO DELAY (nw=0);
  - nr := nr+1

**END startread;**

**PROC endread**

  - [nr := nr – 1];
  - DO nr := nr – 1;

**END endread**

**PROC startwrite**

  - [ nw := nw+1;
    ¬ writerbusy AND nr=0 →
    writerbusy := TRUE ];
  - DO nw := nw + 1;
  - DELAY ( writerbusy AND nr =0);
  - Writerbusy := TRUE

**END startwrite;**

**PROC endwrite**

  - [ nw := nw-1;
    writerbusy := FALSE];
  - DO writerbusy := FALSE;
  - nw := nw-1

**END endwrite;**

**END MODBODY reader-writer;**

**NOTE:** read, write procedures are not included above for want of space
The solution given is essentially the solution that can be described using monitors under the assumption of trustworthiness of the processes. It can be easily seen that if the processes do not call the procedures in the expected order then the program does not satisfy the specification.

Following access clause can be added to above specification making it valid without the assumption of trustworthiness of processes:

- ACCESS (startread)(read)(endread), (startwrite)(write)(endwrite);
- ACCESS (startread)(read)*(endread), (startwrite)(write)*(endwrite);

ACCESS clause defines the order in which the visible procedures could be accessed by the invoking processes. The order refers to the access order for each process and not just the general order of procedure invocation on the shared data.

The second ACCESS clause above allows indefinite number of reads or writes after acquiring the resource. The access clause has wider perspective than that can be inferred from the above example, particularly for serial access devices.
Where as many of the related works in UML are centered around refining the active object semantics, our work aims at refining the semantics of passive object to model a shared resource, a central entity for synchronization and communication. 

Under the proposed semantics, a shared resource is represented by an object which is externally passive and internally active. This choice presents high concurrency and protection of the integrity against concurrent calls. Also, this is essentially nothing but represents UML passive object semantics.

The internal activism corresponds to refinement of UML semantics regarding meta-attribute concurrency which is not clear in UML.
Extended UML meta-model fragments

An extended meta model fragment defining SynchNode and associated SynchHandler to model synchronization regions to specify conditional synchronization, mutual exclusion.
Specification of Readers-Writers problem

(i) Object diagram

(ii) associated AccessOrder
Extended UML 2 activity specification of readParKind operations:

(i) startread()  
(ii) read()  
(iii) endread()  

Extended UML 2 activity specification of writeKind operations:

i) startwrite()  
ii) write()  
iii) endwrite()
Related works

- To specify concurrency some UML methods and tools (RoseRT, Rhapsody, Accord) extend the semantics of UML active object model defining non-preemptive execution model (only one message processed at a time) or preemptive execution model (internal concurrency) with a controller for active objects. Also the internal concurrency of passive objects is controlled through protection mechanism like semaphores or through encapsulation inside an active object.

- Shane Sendall and Alfred Strohmeier proposed an approach to specify concurrent operations through operation schema calculus based on OCL. These schemata are declarative specifications of fine-grained concurrent operation behavior. Pre, post, invariant, shared resources, signals, and exceptions can be specified in operation schemas. The approach results in clear analysis model but the implementor has to obtain the necessary information from careful study of declarative OCL specification of operation schemas. The approach also uses protocol state machines to define temporal ordering of operations but constructing them requires complex state machine composition rules.

- Charles Crichton et al. proposed a pattern for concurrency in UML. The approach is based on modeling attribute states through state machine and operations states through activity diagram. The analysis model thus obtained can be converted to a formal process model (CSP) for validation of the design decisions using formal methods tools.
- **Sebastien Gerard et al.** describe **ACCORD/UML Methodology** for modeling real-time systems. The approach defines a real-time object paradigm (RTO). In addition to attributes, and operations, a real-time object consists of a mailbox, and a controller. The local controller is responsible for mailbox management, scheduling constraints handling, concurrency constraints handling, and thread management. The functionality is similar to that of an operating system scheduler. The behavior of an RTO can be described by a simple protocol kind of a state machine with no orthogonal composite states, no actions or activities in a state. Operations are classified as read, write, and parallel type and concurrent execution is allowed as per 1-writer/N-readers protocol.

- **Iulian Ober** proposed an approach to integrate an existing concurrent object model, named ATOM, with UML object model. The proposed extension redefines active/passive object semantics to eliminate involved inconsistencies. Passive objects cannot have statemachines. Active objects are quasi-concurrent: an executing method can explicitly yield the control for example while it is waiting for an event. Method invocation is de-linked from the associated statemachine and only signals are processed by the statemachine. The statemachine runs quasi-concurrently with the methods and is informed of methods start and end events.

- **Masaaki Mizuno et al.** proposed a unified-process based aspect-oriented methodology. The approach defines steps to weave synchronization code into final solution. This is a semi-formal approach based on coarse-grained solution with formal constructs and global invariants (GI). As finding suitable global invariants is not an easy task, GI patterns are defined to compose complex GIs.
**Conclusion**

- UML 2 activities, with Petri net like semantics, is an expressive formalism to specify complex control/data flow and hence suitable to model procedures and processes.
- In this paper, we have defined a UML-based approach to specify fine-grained concurrent access to shared variables without using low-level synchronization primitives and thus avoiding unnecessary mutual exclusion. This leads to better utilization of resources and overall system performance.
- We have cleanly integrated concurrency constructs with UML’s active/passive object model through a light weight extension of UML2.0 meta model. The proposed constructs of the approach are convenient for programming as well as for establishing the correctness of the specifications.
- We intend to refine the constructs for mapping them into java language and develop a fully functional prototype in multi-threaded java environment.
- As the fundamental approach of the work is based on systematic formal approaches of concurrent programming, we envisage the integration of the work into a formal framework supporting rigorous analysis.
- For this we intend to develop formal semantics for the extended constructs. The approach of this paper can be easily extended to modeling in distributed environment.
Thank YOU