Teaching Parallel Programming Using Model Oriented Programming

Scaffolded Methodology from Prototyping, Design, Implementation, and Testing

Dr. Omar Badreddin, Northeastern University

Motivation

Parallelism is on the rise in diverse domains. As data and processing needs continue to exponentially increase, it has become more urgent to prepare future engineers to design, develop, and test applications that maximize use of parallelism.

At the same time, applications and software become increasingly complex with increasing needs for upfront designs to manage this complexity, and increase software sustainability.

This poster presents a novel design language that treats parallelism as a first class entity, and allows software designers and developers to safely embed papalism.

Approach

This poster presents a novel model-oriented programming languages that treats parallelism and its abstractions as first class entities. This I achieved by embedding elements from UML State Machines, specifically, Composite Concurrent States, and independent thread executions of do activities.

This approach has the following primary objectives:

1) Enable engineers to design for parallelism very early in the design process.
2) Reduce the overhead complexity associated with implementing parallelism efficiently.
3) Reduce the probability of human-errors in the development of parallelism by embedding code generation to support parallel implementation.

Key Parallelism Abstractions

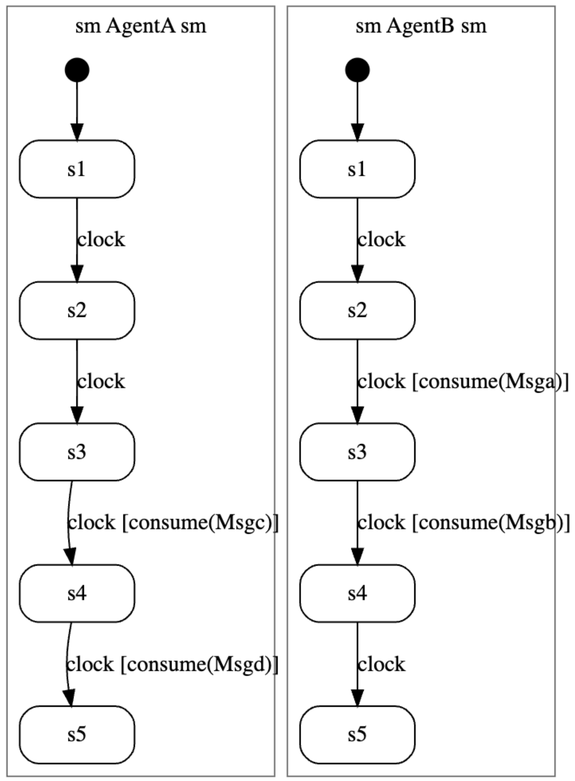
Umple, is a Model-Oriented Programming Language that enhances existing Object Orientation by incorporating key model-level abstractions. These abstractions are inspired from UML Modeling Notations, including Class Diagrams, State Machines, among others.

The following are key parallel abstractions available as first class entities in Umple

1) Do activities: supported as independently running threads.
2) Parallel Regions: supported as two independently executing processes.
3) Communication Machines: supported as two independent objects communicating through message passing or method invocation.

Demonstration Example Using Umple Visual and Textual Notation

Communicating Agents (Visual Syntax)



Communicating Agents (Textual Notation)

```
class AgentA {
  isA Agent;

  sm {
    s1 {
      entry / { send(2, Msga);}
      clock -> s2;
      //after(10)-> s2;
      // -> s2;
    }
    s2 {
      entry / { send(2, Msgb);}
      clock -> s3;
    }
    s3 {
      clock [ consume(Msgc)] -> s4;
    }
    s4 {
      clock [ consume(Msgd)] -> s5;
    }
    s5 {}
  }
}

class AgentB {
  isA Agent;

  sm {
    s1 {
      entry / { send(1, Msgc);}
      clock -> s2;
    }
    s2 {
      clock [ consume(Msga)] -> s3;

    }
    s3 {
      clock [ consume(Msgb)] -> s4;
    }
    s4 {
      entry / { send(1, Msgd);}
      clock -> s5;
    }
    s5 {}
  }

}
```

Additional Parallelism Support

Umple is developed as a fully-fledged programming language with parallelism supported as first class entities in the language. To demonstrate this, Umple compiler, code generator, and all web interface elements are developed using Umple itself.

The following are key parallelism abstraction supported natively in the language:

1) Do Activities: Any state in Umple can include one or more do activities. These activities are implemented as an independent thread executing as long as the state is active. The thread will be interrupted as soon as the state is exited. If later on the state becomes active again, the thread will resume execution.

2) Parallel Regions: Any state, or sub state, can include two or more regions. These two or more regions execute independently of other regions. The language support both forks and joins to synchronize these regions.

State Machine Regions Example

```
class TrackShuttler {
  sm {
    initializing {
      readyToGo -> transferringLoad;
    }
    transferringLoad {
      loaded -> shuttling;
    }
    shuttling {
      reachEnd -> transferringLoad;
      moving {
        nearEnd -> braking;
        accelerating {
          reachedMaxSpeed -> coasting;
        }
        coasting {
          tooSlow -> accelerating;
        }
        braking {
          tooSlow -> coasting;
        }
      }
      ||
      controllingLights {
        lightsOn {
          daylightDetected -> lightsOff;
        }
        lightsOff {
          darknessDetected -> lightsOn;
        }
      }
    }
  }
}
```

Try it out yourself

www.try.Umple.org

Try it out yourself

Presenter: Dr. Omar Badreddin
Associate Professor, Northeastern University

Umple PI: Dr. Timothy Lethbridge
Professor, University of Ottawa
https://www.site.uottawa.ca/~tcl