Generating Business Applications from Executable Models

Rafael Chaves (@abstratt)
rafael@abstratt.com
http://abstratt.com
Motivation
Business applications

Domain knowledge +

Applied technology
How it should be...
Domain knowledge
How it usually goes...
Executable models to the rescue
Executable models are models but...

Precise
(as much as needed)

Complete
(about what matters)
Executable models are programs but...

Focused
(on problem domain concepts)

Oblivious
(regarding technological choices)
Executable models promote a complete separation between business and technology.
Demonstration
Tools

- vanilla UML as executable modeling language
  ○ extended via profiles/conventions
- TextUML - **textual notation** over UML
- Eclipse Xtend - for implementing generators
- Cloudfier - web-based modeling IDE with interpreter/generators

*General concepts > Particular tools*
Modeling
Test-driven modeling
User-driven validation
Code generation
Structure-based generation

Classes
Attributes
Operations (signatures)
Associations
Other classifiers (enumerations, signals etc)
Default values/derivations
enumeration Severity
    Minor;
    Normal;
    Major;
    Blocker;
    Enhancement;
end;
package shipit;

public enum Severity {
    Minor,
    Normal,
    Major,
    Blocker,
    Enhancement
}
class EnumerationGenerator {

    def generateEnumeration(Enum enumeration) {
        ...
        package «enumeration.packagePrefix»;

        public enum «enumeration.name» {
            «enumeration.ownedLiterals.map[name].join(',,
        }
        ...
    }

    def String packagePrefix(Classifier contextual) {
        contextual.nearestPackage.toJavaPackage
    }

    def String toJavaPackage(org.eclipse.uml2.uml.Package package_) {
        package_.qualifiedName.replace(NamedElement.SEPARATOR, ",."
    }
}
signal ExpenseApproved
  attribute employeeName : String;
  attribute amount : Double;
  attribute description : String;
  attribute expenseId : Integer;
end;
package payment;

import java.io.Serializable;
import java.util.*;

public class ExpenseApprovedEvent implements Serializable {
    private static final long serialVersionUID = 1L;

    public final String employeeName;
    public final Double amount;
    public final String description;
    public final Long expenseId;

    public ExpenseApprovedEvent(String employeeName,
                                 Double amount, String description, Long expenseId) {
        this.employeeName = employeeName;
        this.amount = amount;
        this.description = description;
        this.expenseId = expenseId;
    }
}

```java
def generateSignal(Signal signal) {
    ...
    package «signal.packagePrefix»;

    import java.io.Serializable;
    import java.util.*;

    public class «signal.name»Event implements Serializable {
        «signal.allAttributes.map['''
            public final «it.type.toJavaType» «it.name»;
            '''].join()»

        public «signal.name»Event(«signal.allAttributes.map['''
            'type.toJavaType' «name»''
        ].join(',' ')」) {
            «signal.allAttributes.map['''
                this.<name> = «name»;
            '''].join»|
        }
    }
    ...
```

class Rental

    derived attribute description : String := {
        self.car.carModel.description + " on " + self.started
    };

    readonly attribute started : Date := { Date#today() };

    readonly attribute returned : Date[0,1];

    readonly attribute car : Car;

    readonly attribute customer : Customer;

    derived readonly attribute inProgress : Boolean := {
        self.returned == null
    };

package car_rental;

import java.util.*;

@Entity
public class Rental {
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY) private Long id;

    public Long getId() {
        return id;
    }

    @Column(nullable=false)
    public Date started = java.sql.Date.valueOf(java.time.LocalDate.now());

    @Column
    public Date returned;

    @ManyToMany
    public Car car;

    @ManyToMany
    public Customer customer;

    public String getDescription() {
        return (this.car.carModel.getDescription() + " on ") + this.started;
    }

    public boolean isInProgress() {
        return (this.returned == null);
    }
}
An issue describes a problem report, a feature request or just a work item for a project. Issues are reported by and assigned to users, and go through a lifecycle from the time they are opened until they are resolved and eventually closed.

```java
class Issue

attribute summary : String;

derived id attribute issueId : Integer;

derived attribute issueKey : String := {
    self.project.token + "-" + self.issueId
};

attribute labels : Label[*];

attribute project : Project;

port userNotifier : UserNotifier;

readonly attribute reportedOn : Date := { Date#today() };

readonly attribute reporter : User;

readonly attribute assignee : User[0, 1];

attribute severity : Severity := Major;
```
package shipit;

import java.util.*;

/**
 * An issue describes a problem report, a feature request or just a work item for a project. Issues are reported by and
 * assigned to users, and go through a lifecycle from the time they are opened until they are resolved and eventually
 * closed.
 */
@Entity
public class Issue {

@Id @GeneratedValue(strategy=GenerationType.IDENTITY) private Long id;

public Long getId() {
    return id;
}

ategant

@Inject @Transient private IssueService issueService = new IssueService();

@Inject @Transient shipit.UserNotifier userNotifier;

@Column(nullable=false)
public String summary = "";

@Column(nullable=false)
public Date reportedOn = java.sql.Date.valueOf(java.time.LocalDate.now());

@Column(nullable=false)
@Enumerated(EnumType.STRING)
public Severity severity = Severity.Major;
package "entity.packagePrefix";

"generateStandardImports"
"entity.generateImports"

"entity.generateComment"
"entity.generateEntityAnnotations" public class "entity.name" "entity.generateEntityGenealogy"{

"entity.generatePrefix"

"entity.generateAnonymousDataTypes"

"entity.generateEntityId"

"entity.generateProviders"

"IF" !ports.empty
    "*************** PORTS ***************/

 "generateMany(ports, [generatePort])"
 "ENDIF"

"IF" !signals.empty
    "*************** SIGNALS ***************/

 "generateMany(signals, [generateSignal])"
 "ENDIF"

"entity.generateAttributes"

"IF" !relationships.empty
    "*************** RELATIONSHIPS ***************/

 "generateMany(relationships, [generateRelationship])"
 "ENDIF"
```javascript
/**
 * An issue describes a problem report, a feature request or just a work item for a
 * project. Issues are reported by and assigned to users, and go through a lifecycle
 * from the time they are opened until they are resolved and eventually closed.
 */

var issueSchema = new Schema{
    summary : {
        type : String,
        "default" : null
    },
    reportedOn : {
        type : Date,
        "default" : function() {
            return new Date();
        }()
    },
    severity : {
        type : String,
        enum : ["Minor", "Normal", "Major", "Blocker", "Enhancement"],
        "default" : "Major"
    },
    status : {
        type : String,
        enum : ["Open", "InProgress", "Assigned", "Resolved", "Verified"],
        "default" : "Open"
    },
    resolution : {
        type : String,
        enum : ["Fixed", "WorksForMe", "WontFix"],
        "default" : "Fixed"
    },
    resolvedOn : {
        type : Date,
        "default" : null
    },
```
State machine-based generation

States
Transitions
Triggers
Guards
Entry/exit/do activities
Java enum-based state machine (FSM): Passing in events ...

Aug 30, 2014 - I'm using several enum-based state machines in my Android ... So you want to dispatch events to their handlers for the current state. To dispatch ...

Java Secret: Using an enum to build a State machine | Java ...

Jul 10, 2011 - In this article, I outline some the individual features of enum in Java, and put them together to form a state machine. Enum for Singleton and ...
You visited this page on 2/21/15.

Java Secret: Using an enum to build a State machine ...

Jun 24, 2011 - Java Secret: Using an enum to build a State machine. Overview. The enum is Java is more powerful than many other languages which can ...

Implementing State Machines with Java Enums - Mirko Sertic

Apr 9, 2013 - We model a State Machine using a Java Enum. The Enum concept fits better to the State Machine concept, and Java Enums are much more ...
statemachine Status
    initial state Draft
        (*
            If amount is under automatic approval limit,
            the expense is automatically approved.
        *)
        transition on call(submit) to Approved when { self.automaticApproval };
        transition on call(submit) to Submitted when { not self.automaticApproval };
    end;

state Submitted
    transition on call(approve) to Approved;
    transition on call(review) to Draft;
    transition on call(reject) to Rejected;
end;

state Approved
    entry {
        self.processed := Date#today();
        self.reportApproved();
    }
end;

state Rejected
    entry { self.processed := Date#today(); }
    transition on call(reconsider) to Submitted;
end;
public enum Status {
    Draft {
        @Override void handleEvent(Expense instance, StatusEvent event) {
            switch (event) {
                case Submit:
                    if (instance.isAutomaticApproval()) {
                        doTransitionTo(instance, Approved); break;
                    }
                    if (!instance.isAutomaticApproval()) {
                        doTransitionTo(instance, Submitted); break;
                    }
                    break;
                default: break; // unexpected events are silently ignored
            }
        }
    },
    Submitted {
        @Override void handleEvent(Expense instance, StatusEvent event) {
            switch (event) {
                case Approve:
                    doTransitionTo(instance, Approved); break;
                case Review:
                    doTransitionTo(instance, Draft); break;
                case Reject:
                    doTransitionTo(instance, Rejected); break;
                default: break; // unexpected events are silently ignored
            }
        }
    },
    Approved {
        @Override void onEntry(Expense instance) {
            instance.processed = java.sql.Date.valueOf(java.time.LocalDate.now());
            instance.reportApproved();
        }
    }
}
void onEntry(Expense instance) {
    // no entry behavior by default
}
void onExit(Expense instance) {
    // no exit behavior by default
}
/** Each state implements handling of events. */
abstract void handleEvent(Expense instance, StatusEvent event);
/**
 * Performs a transition.
 * @param instance the instance to update
 * @param newState the new state to transition to
 */
final void doTransitionTo(Expense instance, Status newState) {
    instance.status.onExit(instance);
    instance.status = newState;
    instance.status.onEntry(instance);
}

public enum StatusEvent {
    Submit,
    Approve,
    Review,
    Reject,
    Reconsider
}

public void handleEvent(StatusEvent event) {
    status.handleEvent(this, event);
}
private def doGenerateStateMachine(StateMachine stateMachine, Class entity) {
  val stateAttribute = entity.findStateProperties.head
  if (stateAttribute == null)
    return ''
  val triggersPerEvent = stateMachine.findTriggersPerEvent
  val eventNames = triggersPerEvent.keySet.map{it.generateEventName.toString}

  public enum <<stateMachine.name>> {
    <stateMachine.vertices.map[
      generateState(it, stateMachine, entity).toString.trim
    ].join(',
    generateBase-Methods(stateMachine, entity, stateAttribute)>
  }
def generateBaseMethods(StateMachine stateMachine, Class entity, Property stateAttribute) {

    void onEntry(<entity.name> instance) {
        // no entry behavior by default
    }

    void onExit(<entity.name> instance) {
        // no exit behavior by default
    }

    /** Each state implements handling of events. */
    abstract void handleEvent(<entity.name> instance, <stateMachine.name>Event event);

    /**
     * Performs a transition.
     * @param instance the instance to update
     * @param newState the new state to transition to
     */
    final void doTransitionTo(<entity.name> instance, <stateMachine.name> newState) {
        instance.<stateAttribute.name>.onExit(instance);
        instance.<stateAttribute.name> = newState;
        instance.<stateAttribute.name>.onEntry(instance);
    }

    ...
}
def generateStateEventHandler(Vertex state,StateMachine stateMachine, Class entity) {
    
    @Override void handleEvent(«entity.name» instance, «stateMachine.name» Event event) {
        IF (!state.outgoings.empty) {
            switch (event) {
                state.findTriggersPerEvent.entrySet.generateMany[pair |
                    val event = pair.key
                    val triggers = pair.value
                    ...
                    case event.generateEventName:
                        triggers.map[generateTrigger].join()
                        break;
                    ...
                ]
            }
        }
        default : break; // unexpected events are silently ignored
    }
    ELSE
        // this is a final state
    ENDIF
}
}
def generateTrigger(Trigger trigger) {
    val transition = trigger.eContainer as Transition

    «IF (transition.guard != null)>>
    if («transition.guard.generatePredicate») {
        «transition.generateTransition»
        break;
    } «ELSE» «transition.generateTransition» «ENDIF» «ENDIF»
}

def generateTransition(Transition transition) {
    «IF (transition.effect != null)>>
    «(transition.effect as Activity).generateActivity» «ENDIF» doTransitionTo(instance, «transition.target.name»);
}
Activity-based generation

Activities
Actions
Input and output pins
Object flows
Available actions

- read/write variable
- create/destroy object
- read self ("this")
- read/write attribute
- create/destroy link
- call operation
- structured node (blocks)
- test identity
- conditional (if/switch)
- loop (for/while)
- raise exception
- send signal
operation deposit(depositedAmount : Decimal) {
    this.balance = this.balance + depositedAmount;
}
Plain Java

def generateCreateObjectAction(CreateObjectAction action) {
    'new «action.classifier.name»()' 
}

def generateAddVariableValueActionAsAssignment(AddVariableValueAction action) {
    '«action.variable.name» = «generateAction(action.value)»'
}

def generateTestIdentityAction(TestIdentityAction action) {
    '«generateAction(action.first)» == «generateAction(action.second)»'.parenthesize(action)
}

Plain Javascript

def generateCreateObjectAction(CreateObjectAction action) {
    '{}
}

def generateAddStructuralFeatureValueAction(AddStructuralFeatureValueAction action) {
    val target = action.object
    val value = action.value
    val featureName = action.structuralFeature.name

    '«generateAction(target)»[«featureName»] = «generateAction(value)»'
}
operation rent(car : Car);
begin
  var rental;
  rental := new Rental;
  link RentalsCustomer(customer := self, rentals := rental);
  link RentalsCar(car := car, rentals := rental);
  send CarRented() to car;
end;

def public void rent(Car car) {
  Rental rental;
  rental = new Rental();

  rental.customer = this;
  this.rentals.add(rental);

  rental.car = car;
  car.rentals.add(rental);
  car.handleEvent(Car.StatusEvent.CarRented);
  persist(rental);
}
operation rentalHistory();
begin
  var car, customer;
  begin
    car := Examples#newCar();
    customer := Examples#newCustomer();
    customer.rent(car);
    Assert#areEqual(1, customer.rentals.size());
    customer.finishRental();
  end;
  begin
    customer.rent(car);
  end;
  begin
    Assert#areEqual(2, customer.rentals.size());
  end;
end;

public void rentalHistory() {
  Car car;
  Customer customer;
  car = Examples.newCar();
  customer = Examples.newCustomer();
  customer.rent(car);

  assertEquals(1L, customer.rentals.size());
  customer.finishRental();
  flush();

  refresh(customer, car);
  customer.rent(car);
  flush();

  refresh(customer);

  assertEquals(2L, customer.rentals.size());
  flush();
}
public void availableUponReturn() {
    Car car;
    Customer customer;
    Rental rental;
    car = Examples.newCar();
    customer = Examples.newCustomer();
    flush();

    refresh(car, customer);

    assertTrue(car.isAvailable());

    customer.rent(car);
    rental = customer.getCurrentRental();
    flush();

    refresh(car, customer);

    customer.rent(car);
    refresh(car, customer);

    assertTrue(!car.isAvailable());

    customer.finishRental();
    flush();

    refresh(rental);

    assertTrue(rental.car.isAvailable());
    flush();
}
static query findByStatus(status : Status) : Expense[*];
begin
    return Expense extent.select(({e : Expense) : Boolean {
        e.status == status
    }));
end;

public Collection<Expense> findByStatus(Expense.Status status) {
    return this.findAll().stream().filter(e -> (e.status == status)).collect(Collectors.toList());
}
Lessons learned
Knowing the target platform matters

- Better to know what the destination looks like before you start
- At a minimum, have fully-functional handwritten example code
Pick your generator language wisely

- **Java**
  - full power, but awkward with collections (improved in 1.8), no string interpolation

- **StringTemplate**
  - standalone templates, but less control

- **Groovy**
  - graph navigation/transformation, string interpolation

- **Xtend**
  - plus static type checking, IDE support
Testing the generator

- have sample applications covering all modeling language features
  - use a compiler/lint to check for syntactical errors

- have tests in your sample apps
  - generate then run tests to detect runtime/integration problems
Generator Reuse

- within a platform (Plain Java, JPA-enabled)
  - share base templates, differ on details

- across target platforms
  - reasoning about models is often the same

- design reuse helps too
  - second generator is much easier
References

Blog
http://abstratt.com/blog/

Executable UML
http://www.executableumlbook.com/
http://www.omg.org/spec/FUML/
http://www.omg.org/spec/ALF/

Cloudfier/TextUML
http://abstratt.com/textuml
Status

● JavaEE generator under active development (open-source EPL) (dev. started on Feb. 18)
● MEAN generator in the back burner
● Looking for sponsors / contracts to further develop the generators
● Cloudfier is still alpha, you are welcome to try it
Evaluate the sessions

Sign in: www.eclipsecon.org
Generating Business Applications from Executable Models

Rafael Chaves (@abstratt)
rafael@abstratt.com
http://abstratt.com