UTA

Today's Agenda

- Quiz 1 on next Tue.
- Quick Review
- Finish Program Proof
- Introduction to OCL

UTA

Quick Review

- What is the difference between first-order logic and propositional logic?
- What is deductive verification? What is partial correctness?

UTA

Introduction to OCL

- Model Driven Architecture
- Overview of OCL
- OCL By Example
- Summary

UTA

What is MDA?

In MDA, the software development process is driven by the activity of modeling.

The MDA framework defines how to specify and transform models at different abstraction levels.

MDA is under supervision of the Object Management Group (OMG).

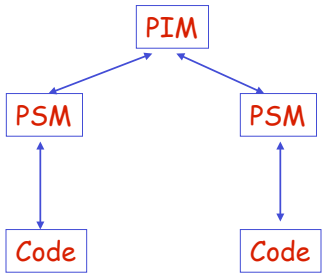
More info. can be found at MDA Explained: The Model Driven Architecture: Practice and Promise, Addison-Wesley, 2003.

The MDA Process

The MDA process consists of three steps:

- Build a model with a high level of abstraction, called Platform Independent Model (PIM).
- Transform the PIM into one or more Platform Specific Models (PSMs), i.e., models that are specified in some specific implementation technology.
- Transform the PSMs to code.

PIM, PSM, and Code



UTA

MDA Elements

- Models are the basis of MDA.
  - Models must be consistent and precise, and contain as much information as possible.
- Modeling languages describe models.
  - These languages must be well-defined to enable automatic transformation.
- Transform. tools do the dirty work.
  - PIM-to-PSM is more challenging than PSM-to-Code.
- Transform. definitions map one model to another.
  - These definitions must be independent of the tools.

UTA

MDA Benefits

- Portability
  - PIMs can be transformed to different PSMs.
- Productivity
  - Developers work at a higher level abstraction.
- Cross-platform interoperability
  - PIMs serve as a bridge between different PSMs.
- Easier maintenance and documentation
  - Maintaining PIMs is much easier than maintaining code.

UTA

Maturity Levels

The maturity level indicates the gap between the model and the system.

- Level 0: No specification
  - Everything is in mind.
- Level 1: Textual description
  - Informal English description.
- Level 2: Text with Diagrams
  - Use diagrams to help understanding.
- Level 3: Models with Text
  - Models have well-defined meaning
- Level 4: Precise models
  - Precise enough to enable automatic model-to-code transformation.
- Level 5: Models only.
  - Code is invisible.

UTA

Introduction to OCL

- Model Driven Architecture
- Overview of OCL
- OCL By Example
- Summary

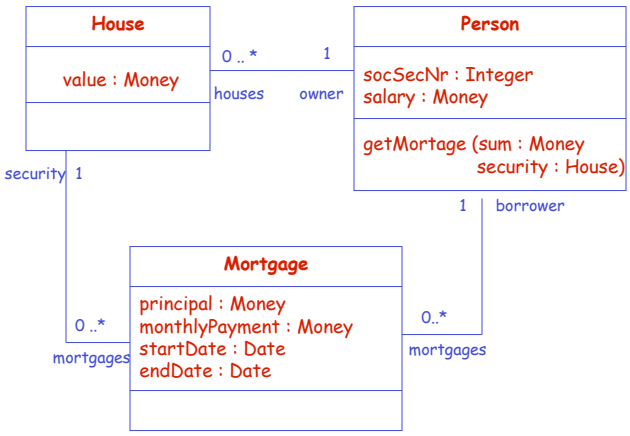
UTA

UML, OCL, and MDA

- UML uses diagrams to express software design.
  - Diagrams are easier to understand, but many properties cannot be expressed using diagrams alone.
- The use of OCL can add additional and necessary info. to UML diagrams.
  - OCL uses expressions that have solid mathematic foundation but still maintains the ease of use.
- Combining UML and OCL is necessary to construct models at maturity level 4
  - The application of MDA relies on Level 4 models.

UTA

Example (1)



UTA

Example (2)

Can we express the following info. on the diagrams?

- A person may have a mortgage on a house only if that house is owned by himself.
- The start date for any mortgage must be before the end date.
- The social security number of all persons must be unique.
- A new mortgage will be allowed only when the person's income is sufficient.
- A new mortgage will be allowed only when the counter-value of the house is sufficient.

UTA

Example (3)

OCL can be used to express those info.:

```
context Mortgage
inv: security.owner = borrower
```

```
context Mortgage
inv: startDate < endDate
```

```
context Person
inv: Person::allInstances() -> isUnique (socSecNr)
```

```
context Person::getMortgage(sum: Money, security: House)
Pre: self.mortgages.monthlyPayment -> sum() <= self.salary * 0.30
```

```
context Person::getMortgage(sum: Money, security: House)
Pre: security.value >= self.mortgages.principal -> sum()
```

Why these info.?

- Avoid any potential misunderstandings
  - Not everyone is aware of these constraints
  - People may make different assumptions.
- Enable automatic model analysis/transform.
  - Computer has no "intuition".
  - Software tools are possible only if the model contains complete information.
- Document your design decisions.

Characteristics of OCL (1)

- OCL is a constraint and query language
  - A constraint is a restriction on one or more values of a model.
  - OCL can be used to write not only constraints, but any query expression.
  - It is proved that OCL has the same capability as SQL.

```
context: Flight::availableSeats () : Integer
body: plane.numberOfSeats – passengers -> size ()
```

UTA

Characteristics of OCL (2)

- OCL has a formal foundation, but maintain the ease of use.
  - The result is a precise language that should be easily read and written by average developers.
- OCL is strongly-typed
  - This allows OCL expressions can be checked during modeling, before execution.
  - What is the benefit?
- OCL is a declarative language
  - OCL expressions state what should be done, but not how.

UTA

Introduction to OCL

- Model Driven Architecture
- Overview of OCL
- OCL By Example
- Summary

UTA

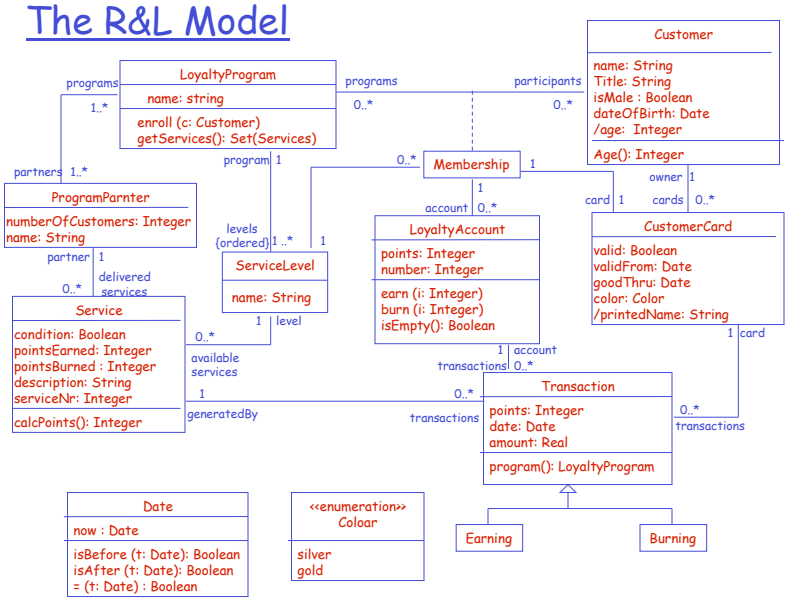
The R&L System

The "Royal and Loyal" system handles loyalty programs for companies to offer customers various kinds of bonuses.

Examples of the bonuses are bonus points, air miles, reduced rates, service upgrades and so on.

UTA

The R&L Model



UTA

Main Classes

- LoyaltyProgram – one instance per each loyalty program
- ProgramPartner – a company which participates in a loyalty program
- Customer – a person who enters a loyalty program
- CustomerCard – information on an ID card
- LoyaltyAccount – keep track of the points
- ServiceLevel – administer the level of services
- Transaction – records transaction information

UTA

The S&G Program

The program has four partners: a supermarket, a line of gas stations, a car rental service, and an airline:

- At the supermarket, the customer can use bonus points to purchase items. The customer earns five bonus points for any regular purchase over $25.
- The gas stations offer a discount of 5 percent on every purchase.
- The car rental service offers 20 bonus points for every $100 spent.
- The airline offers one bonus point for each 15 miles for every flight that is paid normally.

Gold Card - Benefits

Customers with a gold card enjoy additional benefits:

- Every two months, the supermarket offers a free item with an average value of $25.
- The gas stations offer a discount of 10 percent on every purchase.
- The car rental services offers a larger car for the same price.
- The airline offers a business seat for the economy class price.

Gold Card - Conditions

Customers must meet at least one of the following conditions to get a gold card:

- Three sequential years of membership with an average turnover of $5000.
- One year of member with a turnover of $15,000

Initial Values

- A loyalty account will always be initialized with zero points:

```
context LoyaltyAccount :: points
init: 0
```

- A customer card will always be valid at the moment it is issued

```
context CustomerCard :: valid
init: true
```


Derived Attributes

Derived attributes are attributes whose values can be obtained from other attributes.

For instance, the attribute printedName of CustomerCard is determined based on the name and title of the card owner.

```
context CustomerCard::printedName
derive: owner.title.concat(' ').concat(owner.name)
```

Query Operations

Query operations are operations that do not change the state of the system.

For instance, suppose the class LoyaltyProgram has a query operation getServices, which returns all services offered by all program partners:

```
context LoyaltyProgram::getServices(): Set(service)
body: partners.deliveredServices -> asSet ()
```


New attributes and operations

OCL expressions can also define new attributes and operations.

```
context LoyaltyAccount
def: turnover : Real = transactions.amount -> sum ()
```

```
context LoyaltyProgram
def: getServicesByLevel(levelName: String): Set(Service)
       = levels -> select (name = levelName).availableServices ->asSet ()
```

UTA

Invariants

An invariant is a constraint that should be true for an object during its complete lifetime.

Invariants usually represent rules that should hold for the real-life objects after which the software objects are modeled.

UTA

Invariants on Attributes

- A reasonable rule for every loyalty program is to require that every customer is of legal age.

```
context Customer
inv: ofAge: age >= 18
```

- Another example is that in a CustomerCard object, validFrom should be earlier than goodThru.

```
context CustomerCard
inv: checkDates: validFrom.isBefore(goodThru)
```

UTA

Invariants on Associated Objects

Invariants can also express rules for associated objects, which are referred to by rolenames.

```
context CustomerCard
inv: ofAge: owner.age >= 18
```

Question: How do you decide where to put this invariant?

UTA

Using Association Classes

- Association classes do not have rolenames; they are referred to by their class names.

```
context LoyaltyProgram
inv: knownServiceLevel: levels -> includesAll(Membership.currentLevel)
```

- Association classes can also be used as a context.

```
context Membership
inv: correctCard: participants.cards -> includes (self.card)
```

Collections

Often, the multiplicity of an association is greater than 1, thereby linking one object to a collection of objects of the associated class.

OCL provides a wide range of predefined operations to manipulate a collection of objects. These operations are invoked by placing an arrow between the rolename and the operation.

Note that a user-defined operation is invoked using a dot notation.

Collection Operations (1)

- size : returns the size of a collection

```
context LoyaltyProgram
inv: minServices: partners.deliveredServices -> size () >= 1
```

- select : takes an OCL expression as parameter, and returns a subset such that the expression is true for all the elements in the subset

```
context Customer
inv: sizesAgree:
        programs -> size() = cards -> select (valid = true)  -> size ()
```

Collection Operations (2)

- forAll : takes an OCL expression, and returns true if the expression is true for all elements in the collection, and false otherwise.

```
context LoyaltyProgram
inv: noAccounts: partners.deliveredServices -> forAll (
        pointsEarned = 0 and pointsBurned = 0 )
            implies  Membership.account -> isEmpty ()
```

Note that defining a constraint for a class already implies that the condition holds for all instances of the class.

Collection Operations (3)

- collect : returns the set of all values for a certain attribute of all objects in the collection

```
context LoyaltyProgram
inv: totalCountOfCustomers:
        participants -> size ()
           = partners -> collect (numberOfCustomers) -> sum()
```

- Note that the dot notation can be used as an abbreviation for applying the collect operation.

```
context LoyaltyProgram
inv: totalCountOfCustomers:
        participants -> size ();
           = partners.numberOfCustomers -> sum()
```

Collection Operations (4)

- notEmpty : return true when the collect has at least one element
- includes (object) : return true if object is an element of the collection
- union (collection) : return a collection that holds both collections.
- intersection (collection) : return a collection that holds all elements in both collections.

Set vs Bag (1)

- A set is a collection in which each element is unique. A bag is a collection in which elements may be duplicated.
- In OCL, navigation through just one association with multiplicity greater than 1 generates a set; and navigation through more than one such association generates a bag.

Set vs Bag (2)

Does the following OCL expression specify the invariant that attribute numberOfConstomers in class ProgramParnter holds the number of customers who participate in one or more loyalty programs offered by this program partner?

```
context ProgramPartner
inv: numberOfParticipants:
        numberOfCustomers
           = programs.participants -> size ()
```


OrderedSets vs. Sequences

- An OrderedSet is a set in which the elements are ordered; a sequence is a bag in which the elements are ordered.
- Navigation through a single association marked {ordered} generates an OrderedSet; navigation through more than one such association generates a sequence.

```
context LoyalProgram
inv: firstLevel:
        levels -> first ().name = 'Silver'
```

UTA

Pre-/Post-condition

- Preconditions and postconditions are constraints that specify the applicability and effect of an operation without stating an algorithm or implementation.

```
context LoyalAccount::isEmpty () : Boolean
pre:  -- none
post: result = (points = 0)
```


UTA

Previous values in postcondition

- A postcondition expression can refer to values at the start of the operation or upon completion of the operation.

```
context Customer::birthdayHappens ()
post: age = age@pre + 1
```

- It is also possible to refer to the pre-value of a query operation.

```
context Service::upgradePointsEarner (amount: Integer)
post: calcPoints () = calcPoints@pre() + amount
```

UTA

Inheritance (1)

- In the R&L example, the program partners want to limit the number of bonus points they give away; they set a maximum of 10000 points to be earned using services of one partner.

```
context ProgramPartner
inv: totalPoints:
        deliveredServices.transactions.points -> sum() < 10000
```


UTA

Inheritance (2)

- The predefined operation oclIsTypeOf can be used to determine if an element of a collection is an instance of a class.

```
context ProgramPartner
inv: totalPointsEarning:
        deliveredServices.transactions
          -> select ( oclIsTypeOf (Earning) ).points -> sum () < 10000
```

Let expression

- The let expression enables to define a "local" variable.

```
context CustomerCard
inv: let correctDate : Boolean =
                self.validFrom.isBefore (Date::now) and
                self.goodThru.isAfter (Date::now)
     in
       if valid then
          correctDate = true
       else
           correctDate = false
```


Comments

- Comments are necessary to help understanding.
  - A line comment is indicated by two minus signs: --
  - Comments that span more than one line can be enclosed between /* and */.

UTA

Introduction to OCL

- Model Driven Architecture
- Overview of OCL
- OCL By Example
- Summary

UTA

Summary

- MDA is the ultimate paradigm for software development.
- OCL is a declarative, strongly-typed query and constraint language.
- The combination of OCL and UML allows precise models to be built, which is a crucial step towards MDA.
- Additional info. that can be added by OCL include (1) initial values; (2) query operations; (3) invariants; and (4) pre-/post-conditions.