

Model-driven rapid prototyping with Umlle

Andrew Forward, Omar Badreddin, Timothy C.
Lethbridge*, and Julian Solano

School of Electrical Engineering and Computer
Science, University of Ottawa,

Presenter: Maryam Davari

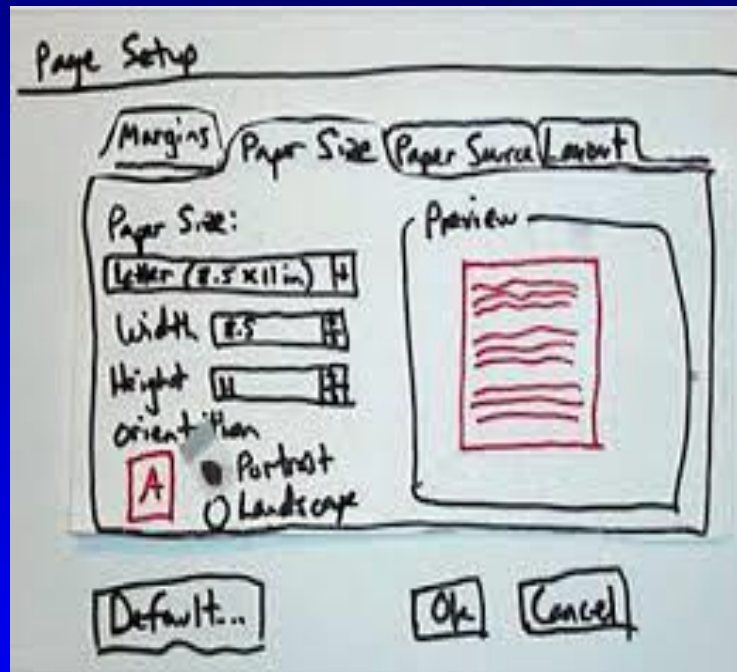
Outline

- Prototype
- Objective of the paper
- Model-driven software development (MDSD)
- MDSD challenges for prototyping
- Umple
- UmpleOnline
- Class Diagram in Umple
- The Umple user-interface prototype generator

Outline (cont.)

- The Umple modeling and prototyping approach
- State Machine in Umple
- Mixin in Umple
- Umple's architecture
- Related work for generation of prototypes and user interfaces
- Conclusion

Prototype



Prototype (cont.)

- Has an indispensable role in software industry.
- Creates a manifestation in the simplest form.
- Filters qualities without distorting the understanding.

Objective of the paper

- How to prototype the essential aspects.
- focus on
 - application-domain data (UML class diagram).
 - behavior (state machine).
- Filters details such as algorithms and mechanisms.

Model-driven software development

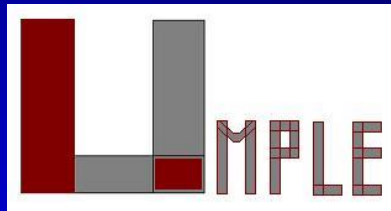
- Is an alternative to round-trip engineering.
- Uses languages such as UML.
- Generates executable system.
- Is inherently possible to generate prototypes from models.

MDSD challenges for prototyping

- Models are not intended for modeling the user interface.
- Combining generated codes from models with other codes is not easy.
- There is a need to generate tangible artifact.
- Software modelers do not understand the consequences of modeling decision.
- Limitations in approaches to prototype generation from UML models.

Umple

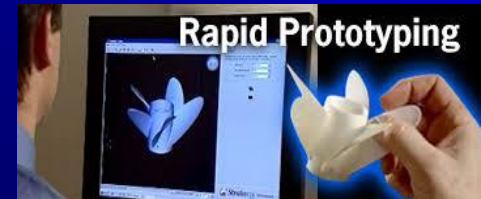
- Stands for “UML Programming Language”.



=



+



- Can be embedded in several programming languages.

UmpleOnline

Umple Online Draw on the right, write (Umple) model code on the left. Generate Java, PHP or Ruby code from your models. Visit [Umple Home Page](#) for help. [Download Eclipse Plugin](#)

[Bookmarkable URL](#)

```
class Bank{
  isA FinancialInstitution;
  1 -- * Branch;
}

class FinancialInstrument{
}

class ReusableFinancialInstrument{
  isA FinancialInstrument;
  number;
  PIN;
  * -> 1 Currency;
}

class CreditCard {
  isA ReusableFinancialInstrument;
  creditLimit;
  name;
}

class DebitCard {
  isA ReusableFinancialInstrument;
}

class Cheque {
  isA FinancialInstrument;
  amount;
  Date date;
  sequenceNumber;
}
```

LOAD & SAVE

TOOLS

DRAW

- Class
- Association
- Generalization
- Delete
- Undo
- Redo

SYNCHRONIZE

- Sync Diagram

GENERATE

Java Code

Generate Code

```
classDiagram
    class FinancialInstitution {
        name : String
    }
    class Bank {
    }
    class Branch {
        name : String
        address : String
    }
    class CreditCard {
    }
    class ReusableFinancialInstrument {
    }
    class DebitCard {
    }
    class BankAccount {
        accountNumber : String
        balance : String
        overdraftOrCreditLimit : Float
    }
    class Cheque {
        amount : String
        date : Date
        sequenceNumber : String
    }

    FinancialInstitution <|-- Bank
    FinancialInstitution <|-- CreditCard
    FinancialInstitution <|-- ReusableFinancialInstrument
    FinancialInstitution <|-- DebitCard
    FinancialInstitution <|-- BankAccount
    FinancialInstitution <|-- Cheque
    Bank "1" -- "*" Branch
    BankAccount "1" -- "*" Cheque
```

The UmpleOnline web application showing a UML mode being edited textually and graphically

Umlple (cont.)

- Supports either textually and graphically.
- Textual editors supports:
 - Syntax-Driven editing
 - Searching
 - Auto indentation
 - Mixins
- Graphical Models Support:
 - Better communication & collaboration.
 - Relationships are clearer.

Umple (cont.)

- Provides a tool to edit UML class diagram in the graphical form.
- Provides a set of textual notations for UML modeling abstractions such as classes, associations, states, transitions.
- Allows end users to quickly create class diagram, state machine and Mixins.

Class Diagram in Umple

Umple Online Draw on the right, write (Umple) model code on the left. Generate Java, C++, PHP or Ruby code. Visit [the User Manual](#) or [the Umple Home Page](#) for help. [Download Umple](#) [Report an Issue](#)

Line= [Create Bookmarkable URL](#)

```
1 class Student {
2 }
3
4
5 class CourseSection {
6 }
7
8 class Registration {
9
10 String grade; // an attribute: get and set
    methods are generated
11
12 * -- 1 Student; // an association: A
    registration has 1 Student
13
14 * -- 1 CourseSection; // another
    association
15 }
16
```

SAVE & RESET

TOOLS

LOAD

Class Diagrams
Select Example

Choose from Dropbox

DRAW

Class

```
classDiagram
    class Student
    class CourseSection
    class Registration {
        grade : String
    }
    Student "1" -- "*" Registration
    CourseSection "1" -- "*" Registration
```

A class diagram corresponding to Umple code.

Example of Uml

```
1 namespace education;
2
3 class School {
4     String name;
5     String address;
6     String description;
7
8     0..1 -- * Person student;
9     key {name}
10 }
11
12
13 namespace human;
14 class Person {
15
16     String name;
17     Integer idNumber;
18     key {idNumber}
19     String firstLetterOfName() { //
20         return left(getName(),1);
21     }
22 }
23 }
```

SAVE & RESET

TOOLS

LOAD

Class Diagrams
Select Example

DRAW

- Class
- Association
- Generalization
- Delete

```
classDiagram
    class School {
        name : String
        address : String
        description : String
    }
    class Person {
        name : String
        idNumber : Integer
    }
    School "0..1" -- "*" Person : student
```

Two classes in two different namespaces with corresponding diagrams

The Umple user-interface prototype generator

- At any point in the modeling or development process, users can quickly generate a prototype.
- Users can create instance of classes and links of associations.
- Semantic rules are respected at run-time.
- User can follow links.
- Users can view and update object's attributes.
- Users can change predefined theme.

The Umple modeling and prototyping approach (cont.)

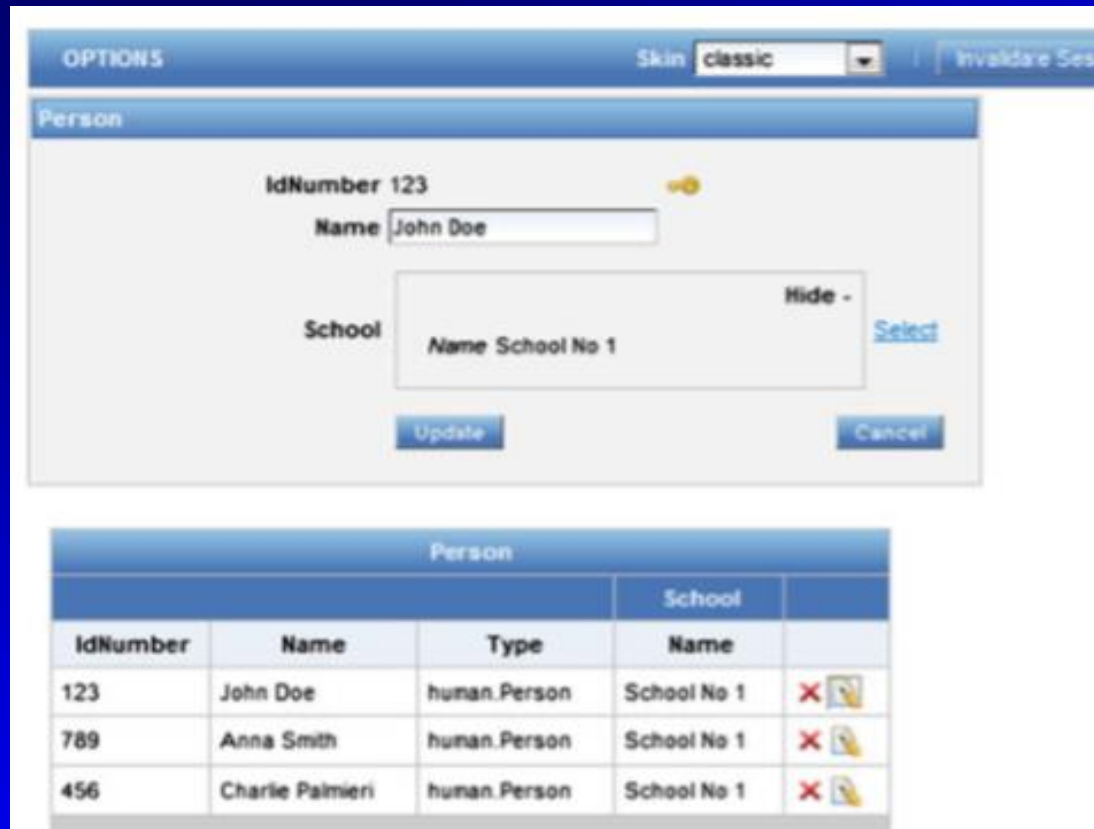
The screenshot displays the Umple prototyping interface for the 'Person' class. At the top, there is a header bar with 'OPTIONS' on the left, a 'Skin' dropdown menu set to 'classic', and an 'Invalidate Session' button on the right. Below the header is a form titled 'Person' with three input fields: 'IdNumber' (with a key icon), 'Name', and 'School' (with a 'Select' link). At the bottom of the form are 'Add' and 'Cancel' buttons.

Below the form is a table titled 'Person' showing existing instances. The table has five columns: 'IdNumber', 'Name', 'Type', 'School', and an action column. The 'School' column is currently empty for all rows. The action column contains a red 'X' and a yellow key icon for each row.

| Person | | | | |
|----------|------------------|--------------|--------|-----|
| | | | School | |
| IdNumber | Name | Type | Name | |
| 123 | John Doe | human.Person | | ✗ 🔑 |
| 789 | Anna Smith | human.Person | | ✗ 🔑 |
| 456 | Charlie Palmieri | human.Person | | ✗ 🔑 |

Prototype interface for creating, editing and deleting instances of the Person class

The Umple modeling and prototyping approach (cont.)



The image shows a web application interface. At the top, there is a header bar with the text "OPTIONS" on the left, "Skin classic" in a dropdown menu in the center, and "Invalidate Session" on the right. Below the header is a form titled "Person". The form contains the following fields and controls:

- IdNumber**: 123
- Name**: John Doe (text input field)
- School**: A dropdown menu showing "Name School No 1" and a "Hide -" button.
- Buttons**: "Update" and "Cancel" buttons at the bottom of the form.

Below the form is a table titled "Person" with the following data:

| Person | | | School | |
|----------|------------------|--------------|-------------|----------|
| IdNumber | Name | Type | Name | |
| 123 | John Doe | human.Person | School No 1 | X [icon] |
| 789 | Anna Smith | human.Person | School No 1 | X [icon] |
| 456 | Charlie Palmieri | human.Person | School No 1 | X [icon] |

An instance of Person with a linked school.

State Machine in Uml

- Uml supports the specification of state machine.
- Adheres to UML semantic except one condition.
- Provides compact representation of behavior, and is ideal for rapid prototyping.
- Is optional feature of Uml.

State Machine in Uml

```
1 class trafficLightSystem
2 {
3   carTraffic{
4
5     Red{
6       entry / {goingRed()}
7       after(redTimer)[!emergency] -> Yellow;
8
9       emergencyNotice -> AllRed;
10    }
11   }
12 }
13 }
14 }
15 }
16 }
17 }
18 }
19 }
20 }
21 }
22 }
23 }
```

SAVE & RESET

TOOLS

LOAD

Class Diagrams

Select Example

Choose from Dropbox

DRAW

Class

Association

OPTIONS

sm trafficLightSystem carTraffic

```
graph TD
  Start(( )) --> Red
  Red -- "after(redTimer)[!emergency]" --> Yellow
  Red -- "emergencyNotice" --> AllRed
```

sm trafficLightSystem pedestrainTraffic

```
graph TD
  Start(( )) --> DontWalk
  DontWalk -- "goingRed[!emergency]" --> Walk
  DontWalk -- "emergencyNotic" --> DonotWalk
```

Example of state machine

Mixin in Umpire

- Enables to inject developed codes into a set of classes.
- Useful for prototype development.
- Enhances reusability of components.
- Can be applied to state machines.

Mixin in Umlple (cont.)

```
1 statemachine
2 coreTrafficController{
3   Red {
4     after(redTimer) -> Green;
5   }
6   Green {
7     after(greenTimer) -> Yellow;
8   }
9   Yellow {
10    after(yellowTimer) -> Red;
11  }
12  //*****//
13  class TrafficLightController{
14
15  HWat as coreTrafficController {
16
17  }
18  }
19
20
```

SAVE & RESET

TOOLS

LOAD

Class Diagrams

Select Example

Choose from Dropbox

DRAW

- Class
- Association
- Generalization

```
stateDiagram-v2
    [*] --> Red
    Red --> Green : after(redTimer)
    Green --> Yellow : after(greenTimer)
    Yellow --> Red : after(yellowTimer)
```

An example of Mixin in state machine

Mixin in Umlle (cont.)

```
1 statemachine
2 coreTrafficController{
3   Red {
4     after(redTimer) -> Green;
5   }
6   Green {
7     after(greenTimer) -> Yellow;
8   }
9   Yellow {
10    after(yellowTimer) -> Red;
11  }
12  //*****//
13  class TrafficLightController{
14
15  FrFy as coreTrafficController {
16
17  Red {
18    + midnightHour ->
19    FlashingRed; }
20
21  FlashingRed {
22    morningHour -> Red; }
23  }
```

SAVE & RESET

TOOLS

LOAD

State Machines

Select Example

Choose from Dropbox

DRAW

- Class
- Association
- Generalization
- Delete
- Undo

```
graph TD
  Start(( )) --> Red
  Red -- "after(redTimer)" --> Green
  Green -- "after(greenTimer)" --> Yellow
  Yellow -- "after(yellowTimer)" --> Red
  Red -- "midnightHour" --> FlashingRed
  FlashingRed -- "morningHour" --> Red
```

Adding additional states that are not part of the basic state machine

Mixin in Umlle (cont.)

```
1 statemachine coreTrafficController{
2   Red {
3     after(redTimer) -> Green;
4   }
5   Green {
6     after(greenTimer) -> Yellow;
7   }
8   Yellow {
9     after(yellowTimer) -> Red;
10  }
11 }
12 //*****//
13
14 class TrafficLightController{
15
16   HWay as coreTrafficController {
17
18     Green{
19       - after (greenTimer) ->Yellow;
20       after (greenTimer) -> Red;}
21   }
22 }
23
```

SAVE & RESET

TOOLS

LOAD

State Machines

Select Example

Choose from Dropbox

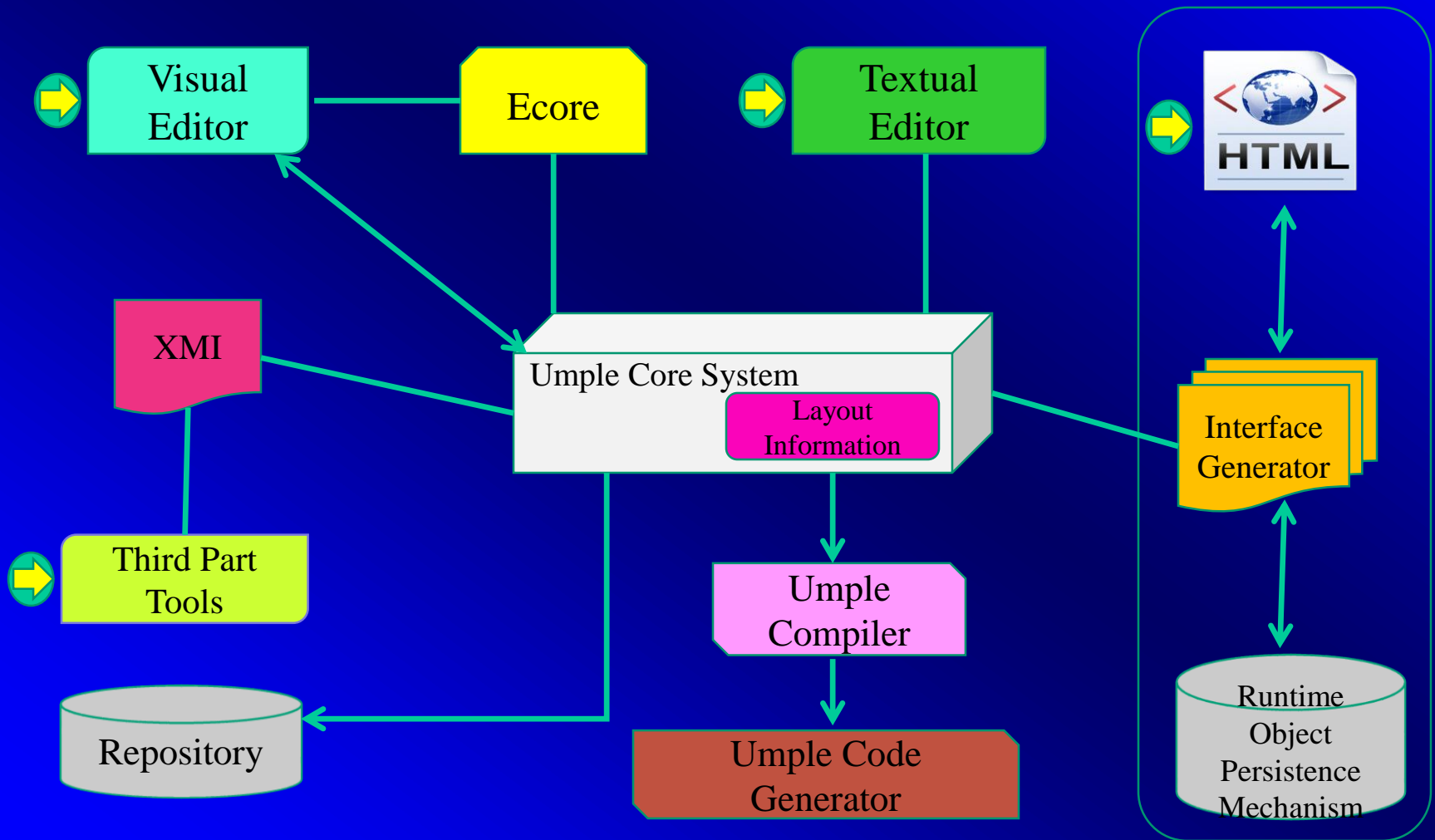
DRAW

- Class
- Association
- Generalization

```
graph TD
    Start(( )) --> Red
    Red -- after(redTimer) --> Green
    Green -- after(greenTimer) --> Yellow
    Yellow -- after(yellowTimer) --> Red
    Null(TrafficLightController_null_null) -- after(greenTimer) --> Red
    Null -- after(greenTimer) --> Yellow
```

Removing or replacing a transition

Umple's architecture



Related work for generation of prototypes and user interfaces

- Generate UI from:
 - State machine models
 - behavior of the system without considering data.
 - Data structure
 - Use case

Conclusion (cont.)

- Umple provides:
 - Accurate reflection of system components and data.
 - Accurate reflection of current system design directions.
 - Quick and cheap generation of a prototype.
 - Maximized potential for reuse and composition.
 - Support for different levels of abstraction.
 - Support for incremental development.

Thank you