Specifying Software Component Frameworks using UML and OCL

H. Conrad Cunningham
Yi Liu

Software Architecture Research Group
Dept. of Computer & Information Science
University of Mississippi
Project

Title: Acxiom Laboratory for Software Architecture and Component Engineering (ALSACE)

Sponsor: Acxiom Corporation

Type: Curriculum development

Begin Date: January 2002

Personnel:
- Dr. H. Conrad Cunningham (PI)
- Ms. Yi Liu (PhD student)
- Student TBD
Project Goals

- Develop software components course
  - systematic, technology-independent methods
  - Java 2 Enterprise Edition (J2EE) for practical exercises
- Design and implement example software systems
  - simple course registration system?
  - online voting system?
- Lay foundation for further software architecture research and course development
Course Approach

• Emphasize abstraction
• Evolve sequence of system specification models
  – build on object-oriented analysis and design techniques
  – construct system of components
• Employ design patterns and design by contract
• Use standard notations
  – Unified Modeling Language (UML)
  – Object Constraint Language (OCL)
• Apply J2EE implementation and deployment technologies
Software Component

A self-contained software construct with
• conformance to a component standard
• clearly specified functionality
• well-defined runtime interfaces that hide implementation details
• capability for independent deployment
• support for composition and replacement
Reference Books


### Architectural Layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Interface</strong></td>
<td>Creates what the user sees. Handles UI logic.</td>
</tr>
<tr>
<td><strong>User Dialog</strong></td>
<td>Dialog logic. Transient state corresponds to the dialog. Can sometimes be used with multiple UIs.</td>
</tr>
<tr>
<td><strong>System Services</strong></td>
<td>Operations are new transactions. Can be used with a variety of user dialogs or batch. Components correspond to business systems. No dialog or client-related state.</td>
</tr>
<tr>
<td><strong>Business Services</strong></td>
<td>Components correspond to stable business types to groups. Operations can be combined with others in a transaction. Usually have associated databases.</td>
</tr>
</tbody>
</table>
Development Workflow

1. Business requirements
2. Business concept models
3. Use case models
4. Technical constrains
5. Component specs & architectures
6. Components
7. Use case models
8. Applications
9. Provisioning
10. Specification
11. Requirements
12. Assembly
13. Test
14. Deployment
15. Applications
Requirements Definition Phase

• Domain (business concept) model
  – discover classes for real-world entities and related concepts
  – show with UML class diagrams

• Use case model
  – capture requirements by detailing user interactions
  – show with UML use case diagrams and scenarios
Course Registration System

- Students registering for classes
  - select classes initially
  - modify or display class schedule later
- Instructors
  - display teaching schedule and class rosters
- Billing system
  - generates tuition bills
- System administrators
  - maintain student and instructor lists
  - manage course information
Domain Model

- Person
  - Term: 1
  - Instructor: 0..1
  - Student: 0..1
- Term
  - Section: *
  - Student Schedule: 1
  - Instructor Schedule: *
- Student Schedule
  - Bill: *
- Instructor Schedule
  - Course: *
Scenarios

Name: Make a Schedule
Initiator: Student
Goal: Make a new student schedule

Main success scenario:
1. Student logs in the system
2. Student asks to make a new schedule for a term
3. Student selects section
4. Repeat 3 until the student finishes selecting courses
5. Submit the schedule
6. System notifies the billing system
7. Billing system generates tuition bill

Extensions:
1. Login failed
   a) Fail
2. Section is full.
   a) Fail
3.1 Schedule sections for same time
   a) Fail
3.2 Schedule sections with overlapping times
   a) Fail
Specification Phase (1)

1. Develop Business Type Model
2. Identify Business Interfaces
3. Identify System Interfaces & Ops
4. Create Initial Comp Specs & Architecture
5. Discover Business Operations
6. Refine Interfaces & Ops
7. Refine Component Specs & Architecture
8. Define Interfaces Information Models
9. Specify Operation Pre/Postconditions
10. Specify Component-Interface Constraints

Diagram: Business Concept Type Model → Use Case Model → Component Identification → Component Interaction → Component Specification
Component Identification Stage

- **Specify system interfaces**
  - give a system interface for each use case

- **Define business type model**
  - remove out-of-scope entities from domain model
  - specify business rules
  - identify core business types
  - create business interface for each core type
  - allocate responsibilities for other types and associations

- **Identify initial component architecture**
  - assign one component for each business interface
System Interfaces

<<interface type>>
IMakeASchedule

- getClassInfo()
- makeSchedule()

<<interface type>>
IGetInstructorInfo

- getInstructorSchedule()
- printStudentList()

<<interface type>>
IUpdateASchedule

- getStudentSchedule()
- getClassInfo()
- updateStudentSchedule()

<<interface type>>
IAdministrator

- newStudent()
- getStudentInfo()
- updateStudent()
- getInstructor()
- updateInstructor()
- newInstructor()
- newCourse()
- updateCourse()

<<interface type>>
ILogin

- getID()
- getPassword()
- access()
Core Type

• Has business identifier, usually independent of other identifiers

• Exists independently, no mandatory relationships except with categorizing types
Business Type Model with Core Types

```
<<core>>
Person
| name : String |
| id : String   |
| dept : String |
| username : String |
| password : String |
| role[3] : String |

<<interface>>
IPersonMgt

<<type>>
Instructor
| name : String |
| ID : String   |
| dept : String |
| email : String |

<<interface>>
ITermMgt

<<type>>
Student Schedule
| schedule : scheduleDetails |

<<core>>
Student
| name : String |
| ID : String   |
| dept : String |
| email : String |
| major[4] : String |

<<data type>>
ScheduleDetails
| scheduleRef : String |
| id : String |
| section : sectionDetails |

<<type>>
Student
| 0..1 |

<<type>>
Term
| termNo : int |
| term : String |

<<core>>
Term
| termNo : int |
| term : String |

<<type>>
Instructor Schedule
| instruSchedule : scheduleDetails |

<<type>>
Section
| sectionNo : integer |
| termCode : String |
| courseNo : String |
| year : integer |
| classroom : String |
| instructorID : String |

<<interface>>
ICourseMgt

<<type>>
Section
| sectionNo : integer |
| termCode : String |
| courseNo : String |
| year : integer |
| classroom : String |
| instructorID : String |

<<interface>>
ICourseMgt

<<data type>>
ScheduleDetails
| scheduleRef : String |
| id : String |
| section : sectionDetails |
```

9-Nov-2001
Initial Component Architecture

Registration System

Billing System

PersonMgt

TermMgt

CourseMgt
Specification Phase (2)
Component Interaction Stage

• Develop interaction model for each system interface operation by walking thru scenarios
• Discover business interface operations and their signatures
• Define any needed component object architecture constraints
• Refine and factor the interfaces
# System Interface Operation Signatures

<table>
<thead>
<tr>
<th>Interface Type</th>
<th>Operation</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMakeASchedule</td>
<td>getCourseInfo</td>
<td>(termInfo: TermDetails, sectionInfo: SectionDetails, course: CourseDetails, course_avail: Boolean)</td>
</tr>
<tr>
<td></td>
<td>makeStudentSchedule</td>
<td>(sectionInfo: SectionDetails, studentInfo: StudentDetails, schedule: ScheduleDetails)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface Type</th>
<th>Operation</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>IUpdateStudentSchedule</td>
<td>getCourseInfo</td>
<td>(termInfo: TermDetails, sectionInfo: SectionDetails, course: CourseDetails, course_avail: Boolean)</td>
</tr>
<tr>
<td></td>
<td>getStudentSchedule</td>
<td>(studentID: String, schedule: ScheduleDetails)</td>
</tr>
<tr>
<td></td>
<td>updateSchedule</td>
<td>(ID: String, sectionInfo: SectionDetails, operation: &quot;add&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface Type</th>
<th>Operation</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGetInstructorInfo</td>
<td>getInstructorSchedule</td>
<td>(instructorID: String, instructorInfo: InstructorDetails, scheduleInfo: ScheduleDetails)</td>
</tr>
<tr>
<td></td>
<td>printStudentInfo</td>
<td>(section: SectionDetails)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface Type</th>
<th>Operation</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILogin</td>
<td>login</td>
<td>(username: PersonDetails, password: PersonDetails, role: PersonDetails)</td>
</tr>
</tbody>
</table>

# Business Operation Signatures

<<interface type>>

**IPersonMgt**

- addNewStudent (in student : StudentDetails) : Boolean
- updateStudentInfo (in studentInfo : StudentDetails, out student : StudentDetails) : Boolean
- getStudentDetails (in ID : studentID) : studentDetails
- makeStudentschedule (in sectionInfo : SectionDetails, in studentInfo : StudentDetails, out schedule : ScheduleDetails) : Boolean
- updateStudentSchedule (in ID : String, in sectionInfo : SectionDetails, in operation : "add"or"drop", out schedule : ScheduleDetails) : Boolean
- addNewInstructor (in instructor : InstructorDetails) : Boolean
- updateInstructorInfo (in instructorDetails : InstructorDetails, out instructor : InstructorDetails) : Boolean
- getInstructorDetails (in ID : InstructorID) : instructorDetails
- makeInstructorschedule (in sectionInfo : SectionDetails, in instructorInfo : InstructorDetails, out schedule : ScheduleDetails) : Boolean
- updateInstructorSchedule (in ID : String, in sectionInfo : SectionDetails, in operation : "add"or"drop", out schedule : ScheduleDetails) : Boolean

<<interface type>>

**ITermMgt**

- getTermInfo (out term : TermDetails)
- newSection (in termC : termNo, in course : CourseDetails, in sectionNo, in classroom, out sectionInfo : SectionDetails) : Boolean
- updateSection (in sectionInfo : SectionDetails, out sectionInfo : SectionDetails) : Boolean

<<interface type>>

**ICourseMgt**

- getCourseInfo (out courseInfo : CourseDetails)
- newCourse (in course : CourseDetails) : Boolean
- updateCourse (in courseCode : CourseNo, out courseInfo : CourseDetails) : Boolean
Constraints on Component Objects

<<comp spec>>
Registration System

1{frozen} 1{frozen} 1{frozen}

<<interface type>>
IPersonMgt
ICourseMgt
ITermMgt
Specification Phase (3)
Component Specification Stage

• Refine each interface specification
  – define interface information model (subset of business type model)
  – specify component invariants using OCL
  – specify operation preconditions and postconditions using OCL

• Add implementation constraints to components
• Define component interaction constraints
Interface Information Model

```
addNewStudent (in student : StudentDetails) : Boolean
updateStudentInfo (in student : StudentDetails, out student : StudentDetails) : Boolean
getStudentDetails (in ID : studentID) : studentDetails
makeStudentschedule (in sectionInfo : SectionDetails, in studentInfo : StudentDetails, out schedule : ScheduleDetails) : Boolean
updateStudentSchedule (in ID : String, in sectionInfo : SectionDetails, in operation : "add"or"drop") : Boolean
addNewInstructor (in instructor : InstructorDetails) : Boolean
updateInstructorInfo (in instructor : InstructorDetails, out instructor : InstructorDetails) : Boolean
getInstructorDetails (in ID : InstructorID) : instructorDetails
makeInstructorschedule (in sectionInfo : SectionDetails, in instructorInfo : InstructorDetails, out schedule : ScheduleDetails) : Boolean
updateInstructorSchedule (in ID : String, in sectionInfo : SectionDetails, in operation : "add"or"drop") : Boolean
```

**Student**
- name : String
- id : String
- dept : String
- email : String

**Person**
- name : String
- id : String
- dept : String
- email : String
- username : String
- password : String
- role[3] : String

**Term**
- termNo : int
- term : String

**Course**
- courseNo : String
- courseName : String
- courseID : int

**Section**
- sectionNo : int
- termNo : int
- courseID : int
- classroom : String
- year : int

**ScheduleDetails**
- scheduleRef : String
- id : String
- section : sectionDetails

**Student Schedule**
- schedule : scheduleDetails

**Instructor Schedule**
- instruSchedule : scheduleDetails

**Instructor**
- name : String
- id : String
- dept : String
- email : String
OCL Operation Specification

IPersonMgt::makeStudentSchedule(in sectioninfo:sectionDetails, in studentinfo:studentDetails, out schedule:scheduleDetails):Boolean

Pre:
----------------- section and student information are valid
Course ->exists(c|c.id = sectioninfo.courseId) and
Term->exists(t|t.termNo = sectioninfo.termNo) and
Section -> exists (se|se.sectionNo = sectioninfo.sectionNo) and
Person->exists(z|id = studentinfo.studentID) and
Student ->exists(y|id = studentinfo.studentID)

Post:
Result implies
StudentSchedule@pre->forall(x|x.scheduleRef <> schedule.scheduleRef)
and
let s = ( StudentSchedule – StudentSchedule@pre)->
asSequence->first in
s.schedule.scheduleRef = schedule.scheduleRef and
s.schedule.id = schedule.id and
schedule.id = studentInfo.studentID and
s.schedule.section = schedule.section and
schedule.section = sectioninfo.section
Component Specification Diagram

<<comp spec>>
Registration System

<<comp spec>>
Billing System

<<comp spec>>
PersonMgt

<<comp spec>>
TermMgt

<<comp spec>>
CourseMgt

IMakeASchedule
IGetInstructorInfo
IAdministrator
ILogin
IUpdateSchedule
IAdministrator
ILogin
IUpdateSchedule
IGetInstructorInfo
IMakeASchedule
Component Specification Constraints

[[<<comp spec>>
  Registration System
  1{frozen}
  <<interface type>>
  IPersonMgt
  1{frozen}
  <<interface type>>
  ICourseMgt
  1{frozen}
  <<interface type>>
  ITermMgt]]
Development Workflow Revisited

- Requirements
  - Business requirements
  - Business concept models
  - Use Case Models

- Specification
  - Technical constrains
  - Component specs & architectures

- Provisioning
  - Components

- Assembly
  - Use case Models
  - Applications

- Test

- Deployment
Summary

• Develop a software components course
  – emphasizing rigorous, systematic methods
  – using standard notations UML and OCL
  – applying J2EE technologies

• Reinforce course with example systems
  – illustrating faithful application of the methods
  – supporting extension and modification by students
  – providing means for investigation of development methods