




Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

	
IBM Software Group	
Mastering Object-Oriented Analysis and Design with UML 2.0 Module 12: Subsystem Design	
	
	

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

Objectives: Subsystem Design

- ♦ Describe the purpose of Subsystem Design and where in the lifecycle it is performed
- ♦ Define the behaviors specified in the subsystem's interfaces in terms of collaborations of contained classes
- ♦ Document the internal structure of the subsystem
- ♦ Determine the dependencies upon elements external to the subsystem

2



Subsystem Design is where you flesh out the detailed collaborations of classes that are needed to implement the responsibilities documented in the subsystem interfaces. In order to support these collaborations, additional relationships between subsystems may need to be defined.

Mastering OOAD w/UML 2.0 – Instructor Notes

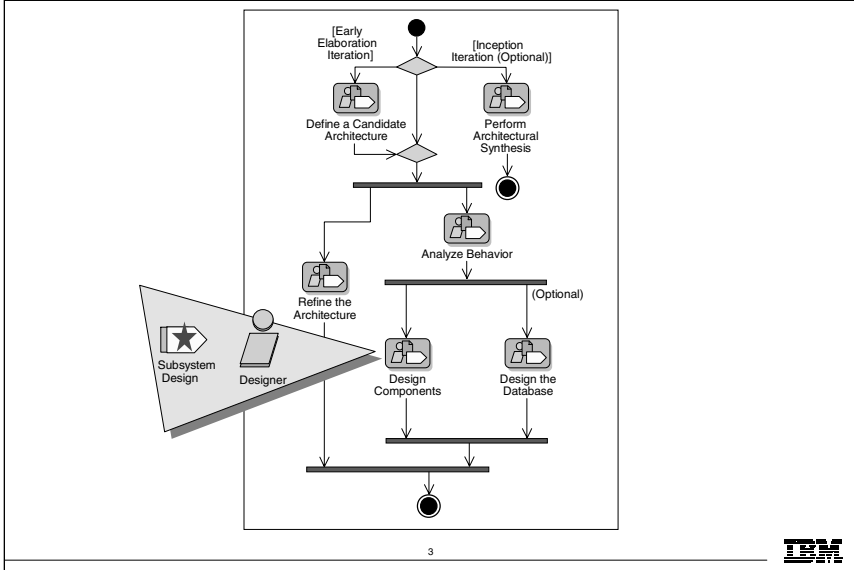
Instructor Notes:

Subsystem Design is where you design the inside of the *black box* that was identified during Identify Design Elements.

As in the saying “Everything I’ve ever known I learned in kindergarten,” everything you need to know in Subsystem Design you learned in Use-Case Analysis and Use-Case Design — identifying abstractions and allocating responsibilities to those abstractions, as well as incorporating any necessary mechanisms.

You might want to note that Subsystem Design can be applied to packages. In that case, you would concentrate on fleshing out the details of the public class operations. As was mentioned earlier, in essence, the public classes are a package’s interface.

Subsystem Design in Context



As you may recall, the above diagram illustrates the workflow that you are using in this course. It is a tailored version of the Analysis and Design core workflow of the Rational Unified Process.

At this point, you have defined the design classes, subsystems, their interfaces, and their dependencies. Some of the things that you have identified up to this point are components or subsystems: “containers” of complex behavior that, for simplicity, you treat as a “black box.” At some point, you need to flesh out the details of the internal interactions. This means that you need to determine what classes exist in the subsystem and how they collaborate to support the responsibilities documented in the subsystem interfaces. You do this in **Subsystem Design**.

In **Subsystem Design**, you look at the responsibilities of a subsystem in detail. You define and refine the classes that are needed to implement the responsibilities, and refine subsystem dependencies, as needed. The internal interactions are expressed as collaborations of classes and possibly other components or subsystems. The focus is on the subsystem.

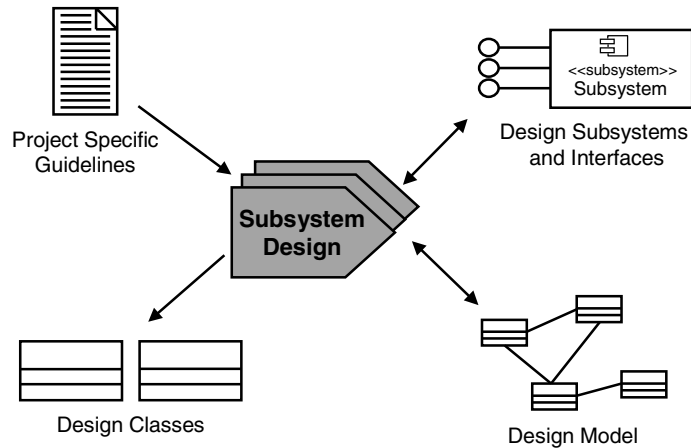
The activity is iterative and recursive, but eventually feeds into Class Design.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

In Subsystem Design, the concentration is on developing the “interface realizations.”

Subsystem Design Overview



4



Subsystem Design is performed once per Design Subsystem.

Purpose:

- To define the behaviors specified in the subsystem's interfaces in terms of collaborations of contained design elements and external subsystems/interfaces
- To document the internal structure of the subsystem
- To define realizations between the subsystem's interfaces and contained classes
- To determine the dependencies upon other subsystems

Input Artifacts:

- Design Subsystems and Interfaces
- Project Specific Guidelines
- Design Model

Resulting Artifacts:

- Design Subsystems and Interfaces
- Design classes
- Design Model

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

Subsystems and interfaces were first introduced in the Concepts of Object Orientation module. They were identified and modeled in Identify Design Elements. The concepts are repeated here for review purposes.

In the UML, any classifier (for example, class, subsystem, component) can have interface(s); however, to keep things simple, in this course we will concentrate on interfaces for subsystems.

Emphasize the following:

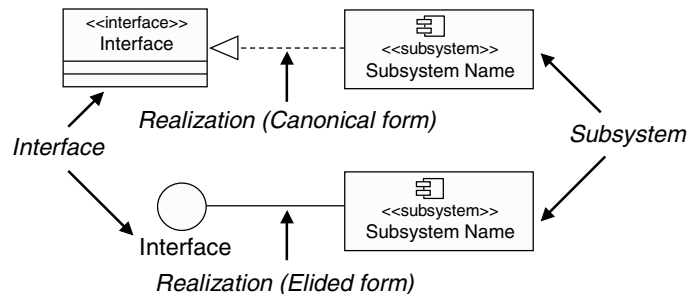
- Canonical vs. elided notation.
- Multiple interfaces are possible per subsystem.
- Interface can be realized by multiple subsystems.
- Interface is not a part of the subsystem, it is external.

Remember that interfaces and abstract classes/generalization are not the same concept. An interface is a full-fledged citizen that may be “realized” (excuse the overloaded term) using abstract classes. The realization of an interface though, is not the same as inheriting from an abstract base class.

Review: Subsystems and Interfaces

A Subsystem:

- ♦ Is a “cross between” a package and a class
- ♦ Realizes one or more interfaces that define its behavior



A subsystem is a model element that has the semantics of a package, such that it can contain other model elements, and the semantics of a class, such that it has behavior. A subsystem realizes one or more interfaces, which define the behavior it can perform.

A subsystem may be represented as a UML package (a tabbed folder) with the «subsystem» stereotype.

An interface is a model element that defines a set of behaviors (a set of operations) offered by a classifier model element (specifically, a class, subsystem or component). A classifier may realize one or more interfaces. An interface may be realized by one or more classifiers.

Interfaces are not abstract classes, as abstract classes allow you to provide default behavior for some or all of their methods. Interfaces provide no default behavior.

Interfaces may be represented as classes with the “interface” stereotype, or may be represented as “lollipops.”

Realization is a semantic relationship between two classifiers. One classifier serves as the contract that the other classifier agrees to carry out.

The realization relationship may be modeled as a dashed line with a hollow arrowhead pointing at the contract classifier (canonical form), or when combined with an interface, as a “lollipop” (elided form).

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

Discuss some of the benefits of doing this well: portability, reuse, insulation from change, and so on.

There should not be any public classes within a subsystem; the only thing a subsystem should expose to the outside world is its interfaces.

These guidelines will be applied throughout this module as we describe how to perform the detailed design of a subsystem. This slide is meant to provide the student with an “overall vision” for the subsystem design.

Subsystem Guidelines

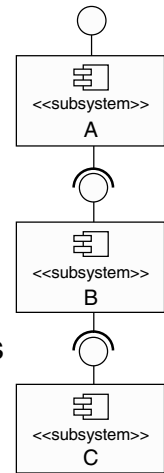
♦ Goals

- Loose coupling
- Portability, plug-and-play compatibility
- Insulation from change
- Independent evolution

♦ Strong Suggestions

- Do not expose details, only interfaces
- Depend only on other interfaces

Key is abstraction and encapsulation



6



Each subsystem should be as independent as possible from other parts of the system. It should be possible to evolve different parts of the system independently from other parts. This minimizes the impact of changes and eases maintenance efforts.

It should be possible to replace any part of the system with a new part, provided the new part supports the same interfaces. In order to ensure that subsystems are replaceable elements in the model, the following conditions are necessary:

- A subsystem should not expose any of its contents. No element contained by a subsystem should have “public” visibility. No element outside the subsystem should depend on the existence of a particular element inside the subsystem.
- A subsystem should depend only on the interfaces of other model elements, so that it is not directly dependent on any specific model elements outside the subsystem. The exceptions are cases where a number of subsystems share a set of class definitions in common, in which case those subsystems “import” the contents of the packages that contain the common classes. This should be done only with packages in lower layers in the architecture, and only to ensure that common definitions of classes that must pass between subsystems are consistently defined.
- All dependencies on a subsystem should be dependencies on the subsystem interfaces. Clients of a subsystem are dependent on the subsystem interface(s), not on elements within the subsystem. In that way, the subsystem can be replaced by any other subsystem that realizes the same interfaces.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

The process is similar to that defined in Use-Case Analysis, but instead of use cases we are working with interface operations. In Subsystem Design, we concentrate on developing “interface realizations” rather than “use-case realizations.”

Ask the students where the subsystem responsibilities are documented.

Answer: In the interfaces that the subsystem realizes.

Subsystem Design Steps

- ◆ Distribute subsystem behavior to subsystem elements



- ◆ Document subsystem elements



- ◆ Describe subsystem dependencies



- ◆ Checkpoints



7



This slide shows the major steps involved in the **Subsystem Design** activity.

We first must take the responsibilities allocated to the subsystems and further allocate those responsibilities to the subsystem elements.

Once the subsystem elements have been identified, the internal structure of the subsystems (a.k.a. subsystem element relationships) must be documented.

Once you know how the subsystem will implement its responsibilities, you need to document the interfaces upon which the subsystem is dependent.

Finally, we will discuss the kinds of things you should look for when reviewing the results of **Subsystem Design**.

Mastering OOAD w/UML 2.0 – Instructor Notes

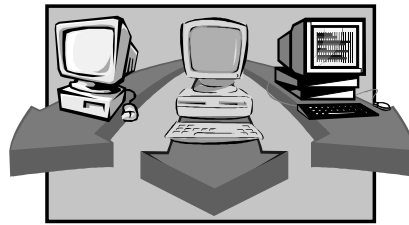
Instructor Notes:

In Subsystem Design, you define subsystem elements to implement subsystem responsibilities.

Note: The design mechanisms were mapped to the design elements in the Identify Design Mechanisms module.

Subsystem Design Steps

- ☆ ♦ **Distribute subsystem behavior to subsystem elements**
- ♦ **Document subsystem elements**
- ♦ **Describe subsystem dependencies**
- ♦ **Checkpoints**



8



You first must take the responsibilities allocated to the subsystems and further allocate those responsibilities to the subsystem elements.

The purpose of this step is to:

- Specify the internal behaviors of the subsystem
- Identify new classes or subsystems needed to satisfy subsystem behavioral requirements.

In designing the internals of the subsystem interactions, you will need to take into consideration the use-case details; the architectural framework, including defined subsystems, their interfaces and dependencies; and the design and implementation mechanisms selected for the system.

Up to this point, you have created interaction diagrams in terms of design elements (that is, design classes and subsystems). In this activity, you will describe the "local" interactions within a subsystem to clarify its internal design.

Mastering OOAD w/UML 2.0 – Instructor Notes

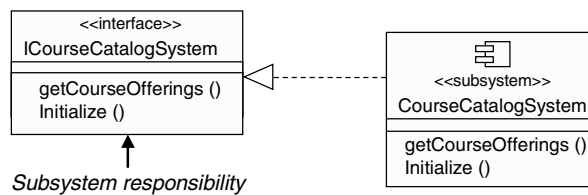
Instructor Notes:

Essentially, an `<<interface realization>>` stereotyped use case is created inside the `<<subsystem>>` package with the same name as the interface it models. Thus, there is one `<<interface realization>>` use case (for example, UML collaboration) for each interface the subsystem realizes. The interface realization contains the static and dynamic models that describe how the subsystem realizes the interfaces (for example, a class diagram and interaction diagrams — at least one interaction diagram per interface operation). As with use-case realizations, the interface realization does not contain any classes; they “live” in the `<<subsystem>>` package or in other design element packages.

This fits well with the large-scale “systems-of-systems” development idea where you would have subsystem use cases and their corresponding realizations; it also keeps the diagrams neat.

Subsystem Responsibilities

- ◆ Subsystem responsibilities defined by interface operations
 - Model interface realizations
- ◆ Interface may be realized by
 - Internal class behavior
 - Subsystem behavior



The external behaviors of the subsystem are defined by the interfaces it realizes. When a subsystem realizes an interface, it makes a commitment to support each and every operation defined by the interface.

The operation may be in turn realized by:

- An operation on a class contained by the subsystem; this operation may require collaboration with other classes or subsystems.
- An operation on an interface realized by a contained subsystem.

The collaborations of model elements within the subsystem should be documented using sequence diagrams that show how the subsystem behavior is realized. Each operation on an interface realized by the subsystem should have one or more documenting sequence diagrams. This diagram is owned by the subsystem, and is used to design the *internal* behavior of that subsystem.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

You may find additional classes or subsystems during this activity. If so, the added classes and relationships must be consistent with the established architecture.

Sometimes you see repeated patterns of interaction in these early interface realizations that give birth to new subsystems and some revisions in the interface realizations. It is important for the architect to look within and across subsystems for common behaviors (common collaborations) within the subsystems, pulling it out where possible. This is a reuse scavenging activity that falls under the Identify Design Elements umbrella.

It is important to describe the collaborations to support the design and implementation mechanisms defined in Identify Design Mechanisms.

Distributing Subsystem Responsibilities

- ♦ Identify new, or reuse existing, design elements (for example, classes and/or subsystems)
- ♦ Allocate subsystem responsibilities to design elements
- ♦ Incorporate applicable mechanisms (for example, persistence, distribution)
- ♦ Document design element collaborations in “interface realizations”
 - One or more interaction diagrams per interface operation
 - Class diagram(s) containing the required design element relationships
- ♦ Revisit “*Identify Design Elements*”
 - Adjust subsystem boundaries and dependencies, as needed

10



For each interface operation, identify the classes (or, where the required behavior is complex, a contained subsystem) within the current subsystem that are needed to perform the operation. Create new classes/subsystems where existing classes/subsystems cannot provide the required behavior (but try to reuse first).

Creation of new classes and subsystems should force reconsideration of subsystem content and boundary. Be careful to avoid having effectively the same class in two different subsystems. Existence of such a class implies that the subsystem boundaries may not be well-drawn. Periodically revisit Identify Design Elements to re-balance subsystem responsibilities.

The collaborations of model elements within the subsystem should be documented using interaction diagrams that show how the subsystem behavior is realized. Each operation on an interface realized by the subsystem should have one or more documenting interaction diagrams. This “internal” interaction diagram shows exactly what classes provide the interface, what needs to happen internally to provide the subsystem’s functionality, and which classes send messages out from the subsystem. These diagrams are owned by the subsystem, and are used to design the internal behavior of the subsystem. The diagrams are essential for subsystems with complex internal designs. They also enable the subsystem behavior to be easily understood, rendering it reusable across contexts.

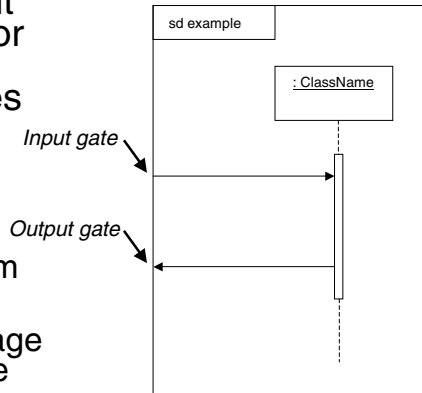
These internal interaction diagrams should incorporate any applicable mechanisms initially identified in Identify Design Mechanisms (for example, persistence, distribution, and so on.)

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

What Are Gates?

- ♦ A connection point in an interaction for a message that comes into or goes outside the interaction.
 - A point on the boundary of the sequence diagram
 - The name of the connected message is the name of the gate



11

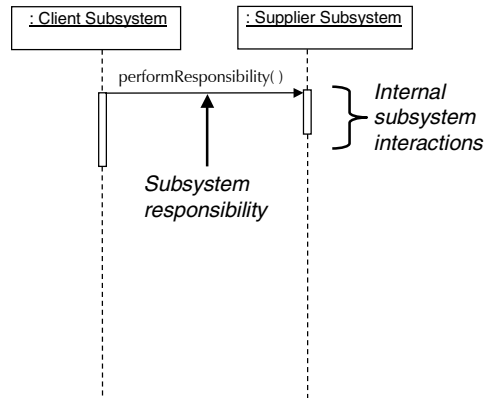


A gate is a parameter that represents a message that crosses the boundary of an interaction or interaction fragment. Messages within the interaction can be connected to the gate. If the interaction is referenced within another interaction, messages may be connected to its gates. When the interaction is executed, messages connected through gates will be delivered properly.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

Subsystem Interaction Diagrams



Black box view of subsystems

12



All elements on a diagram should represent the same level of abstraction.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

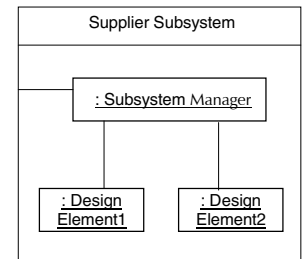
You may find additional classes or subsystems during this activity. If so, the added classes and relationships must be consistent with the established architecture.

Sometimes you see repeated patterns of interaction in these early interface realizations that give birth to new subsystems and some revisions in the interface realizations. It is important for the architect to look within and across subsystems for common behaviors (common collaborations) within the subsystems, pulling it out where possible. This is a reuse scavenging activity that falls under the Identify Design Elements umbrella.

It is important to describe the collaborations to support the design and implementation mechanisms defined in Identify Design Mechanisms.

Internal Structure of Supplier Subsystem

- ◆ Subsystem Manager coordinates the internal behavior of the subsystem.
- ◆ The complete subsystem behavior is distributed amongst the internal Design Element classes.



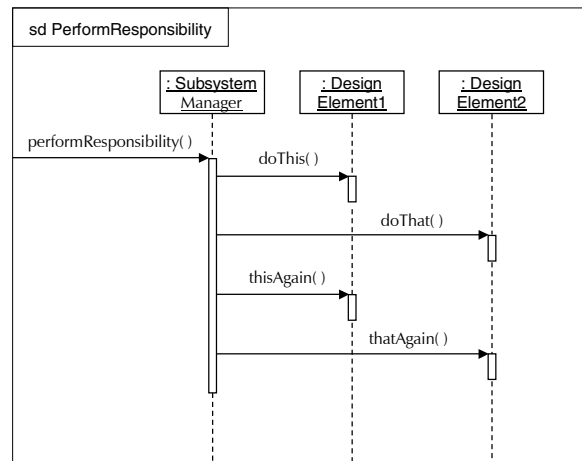
13



Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

Modeling Convention: Internal Subsystem Interaction



White box view of Supplier Subsystem

14



This slide describes the modeling conventions we will be using to model the internal subsystem element interactions.

As discussed earlier, there should be at least one interaction diagram per interface operation to illustrate how the operations offered by the interfaces of the subsystem are performed by model elements contained in the subsystem.

These interaction diagrams should start with a message coming through a gate, the message should be mapped to/associated with the interface operation that is being modeled in the interaction diagram.

The remainder of the diagram should model how the <<subsystem>> class delegates responsibility for performing the invoked operation to the other subsystem elements.

It is recommended that you name the interaction diagram using the operation name. This naming convention simplifies future tracing of interface behaviors to the classes that implement the interface operations.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

This is the same diagram that appeared in the Use-Case Design module.

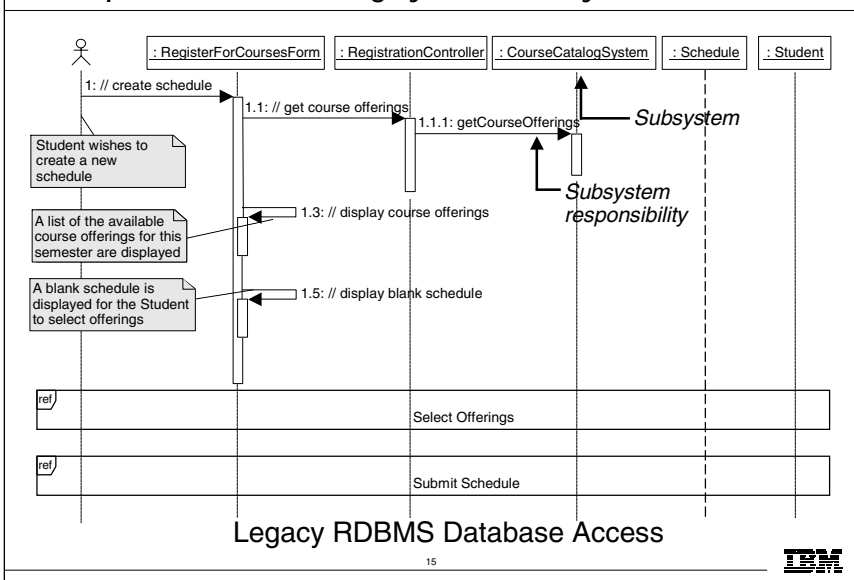
Emphasize that we will be incorporating legacy RDBMS persistency, specifically, retrieving the course offerings for a particular semester from the legacy Course Catalog System.

On the presented slide, the subsystem is shown in white, but this does not show up in the black-and-white manuals.

The requirements documents for the Course Registration System do not really include any specific requirements for the Course Catalog System, beyond the fact that it is a read-only legacy system with an RDBMS database. Thus, for the rest of the Subsystem Design module, we will concentrate on the application of the RDBMS persistency mechanism, and not on any of the other subsystem design details.

The Subsystem and Subsystem Responsibility are presented in white text on the slide. This is to emphasize these two objects. This does not show up in the black and white books.

Example: CourseCatalogSystem Subsystem in Context



The above sequence diagram is the same as was shown in the Use-Case Design module. It demonstrates how interactions are modeled between design elements, where one of the elements is a subsystem. In the Use-Case Design module, we did not flesh out the internals of the CourseCatalog subsystem. That is the purpose of this activity, **Subsystem Design**.

The sequence diagram sets the context of what will be performed in **Subsystem Design**. It puts requirements on the subsystem, and is the primary input specification to the task of creating local interactions for the subsystem.

In the above example we see the operations that the CourseCatalog subsystem must support. This shows the simple way that a client (RegistrationController here) deals with the task of requesting the course offerings from the legacy course catalog system. The `ICourseCatalogSystem::getCourseOfferings()` documentation specifies the following: "Retrieve the course offerings available for the specified semester." Thus, the retrieval of the course offerings from the legacy database is the responsibility of the CourseCatalog subsystem. It is now, in **Subsystem Design**, that we will describe exactly *how* this is done, using the defined RDBMS persistency mechanism.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

If you do not want to cover all of the persistency interaction diagrams, you can use the rationale that the key scenarios we are focusing on for this iteration do not require all database CRUD behaviours.

If you are teaching an introductory audience, or If you skipped the introduction of the RDBMS mechanism in the Identify Design Mechanisms module, then you should skip these next few slides on incorporating the RDBMS persistency mechanism.

Note: On the presented slide, the italicized text is also blue, but this does not show up in the black-and-white manuals.

Incorporating the Architectural Mechanisms: Persistency

♦ Analysis-Class-to-Architectural-Mechanism Map from Use-Case Analysis

Analysis Class	Analysis Mechanism(s)	
Student	Persistency, Security	OODBMS Persistency
Schedule	Persistency, Security	
CourseOffering	<i>Persistency, Legacy Interface</i>	RDBMS Persistency
Course	<i>Persistency, Legacy Interface</i>	
RegistrationController	Distribution	

OODBMS Persistency was discussed in Use-Case Design

16



During Use-Case Analysis, applicable mechanisms for each identified analysis class were documented. This information, along with the information on what analysis classes became what design elements allows the applicable mechanisms to identify a design element.

In this example, you have been concentrating on course registration. Thus, the above table contains only the classes for the Register for Courses Use-Case Realization that have analysis mechanisms assigned to them.

In this section, you will incorporate the legacy RDBMS persistency mechanism because access to the legacy systems has been encapsulated within a subsystem (the CourseCatalog subsystem). The legacy interface mechanism distinguishes the type of persistency. Remember, legacy data is stored in an RDBMS.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

The italicized text reflects the RDBMS incorporation design decisions made for the current design.

On the presented slide, the italicized text is also blue, but this does not show up in the black-and-white manuals.

The check mark indicates what steps have already been completed (in Identify Design Mechanisms).

Review: Incorporating JDBC: Steps

1. Provide access to the class libraries needed to implement JDBC

✓ ▪ *Provide java.sql package*

2. Create the necessary DBClasses

▪ One DBClass per persistent class

▪ Course Offering persistent class => DBCourseOffering



✓ = **Done**

17



This slide summarizes the steps that can be used to implement the RDBMS Persistency mechanism (JDBC) described in this module. The italicized text describes the architectural decisions made with regard to JDBC for our Course Registration example.

These steps were first introduced in the Identify Design Mechanisms module. They are repeated here for convenience (with some additions). The check marks indicate what steps have already been completed.

It is here, in **Subsystem Design**, where you actually incorporate this mechanism. Now you will define the actual DBClasses (and their dependency on the java.sql package) and develop the detailed interaction diagrams.

- The java.sql package contains the design elements that support the RDBMS persistency mechanism. For our example, the CourseCatalogSystem subsystem will depend on it since that is where the created DBCourseOffering class will be placed (see below). Thus, a dependency will need to be added from CourseCatalogSystem to java.sql.
- There is one DBClass per persistent class. For our example, there is a persistent class, CourseOffering. Thus, a DBCourseOffering class will be created.

See the next slide for the rest of the steps.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

The italicized text reflects the RDBMS incorporation design decisions made for the current design.

On the presented slide, the italicized text is also blue, but this does not show up in the black-and-white manuals.

Review: Incorporating JDBC: Steps (continued)

3. Incorporate DBClasses into the design
 - Allocate to package/layer
 - *DBCOURSEOFFERING placed in CourseCatalogSystem subsystem*
 - Add relationships from persistency clients
 - *Persistency clients are the CourseCatalogSystem subsystem clients*
4. Create/Update interaction diagrams that describe:
 - Database initialization
 - Persistent class access: Create, Read, Update, Delete

18



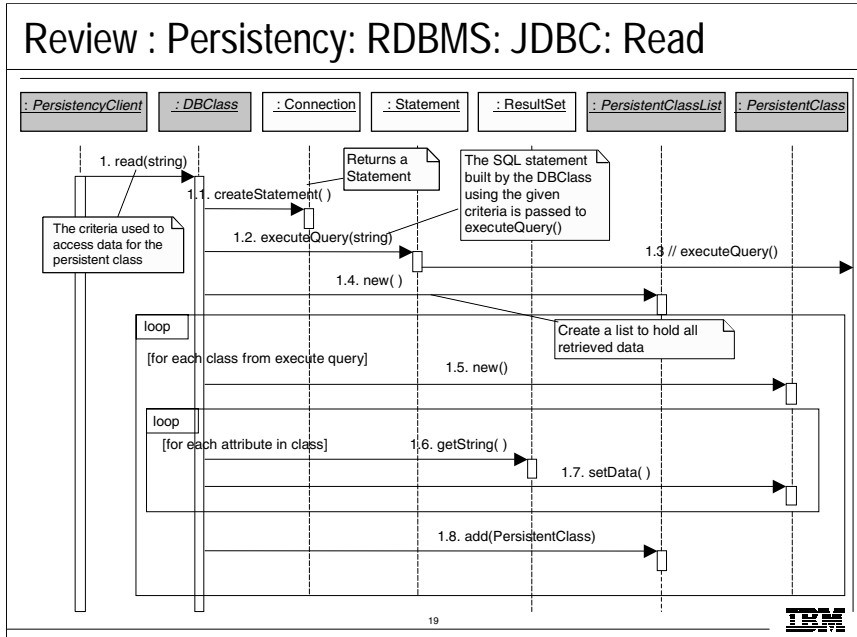
This slide continues the summary of the steps that use the RDBMS Persistency mechanism (JDBC). The italicized text describes the architectural decisions made with regard to JDBC for our Course Registration example.

- Once created, the DBClasses must be incorporated into the existing design. They must be allocated to a package/layer. For our example, the *DBCOURSEOFFERING* class will be placed in the *COURSECATALOGSYSTEM* subsystem.
- Once the DBClasses have been allocated to packages/layers, the relationships to the DBClasses from all classes requiring persistence support (that is, the persistency clients) will need to be added. For our example, the persistency clients are the clients of the *COURSECATALOGSYSTEM* subsystem. These dependencies have already been established (see earlier context diagram).
- The interaction diagrams provide a means to verify that all required database functionality is supported by the design elements. The sample interaction diagrams provided for the persistence architectural mechanisms during Identify Design Mechanisms should serve as a starting point for the specific interaction diagrams defined in detailed design.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

This is a review slide from the *Identify Design Mechanisms* module.



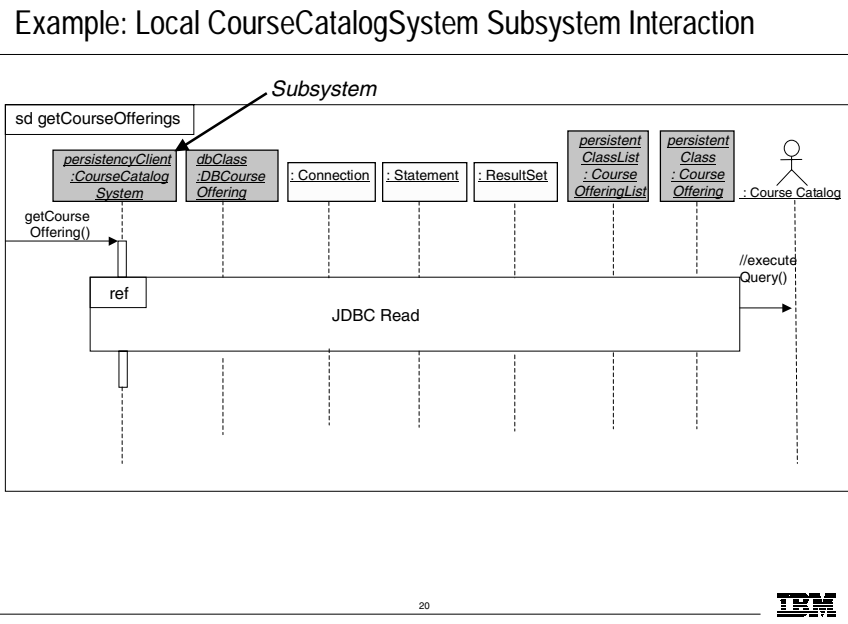
To read a persistent class, the persistency client asks the DBClass to read. The DBClass creates a new Statement using the Connection class `createStatement()` operation. The Statement is executed, and the data is returned in a ResultSet object. The DBClass then creates a new instance of the PersistentClass and populates it with the retrieved data. The data is returned in a collection object, an instance of the PersistentClassList class.

Note: The string passed to `executeQuery()` is *not* the exact same string as the one passed into the `read()`. The DBClass builds the SQL query to retrieve the persistent data from the database, using the criteria passed into the `read()`. This is because we do not want the client of the DBClass to have the knowledge of the internals of the database to create a valid query. This knowledge is encapsulated within DBClass.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

This is a review slide from the *Identify Design Mechanisms* module.



To read a persistent class, the persistency client asks the DBClass to read. The DBClass creates a new Statement using the Connection class *createStatement()* operation. The Statement is executed, and the data is returned in a ResultSet object. The DBClass then creates a new instance of the PersistentClass and populates it with the retrieved data. The data is returned in a collection object, an instance of the PersistentClassList class.

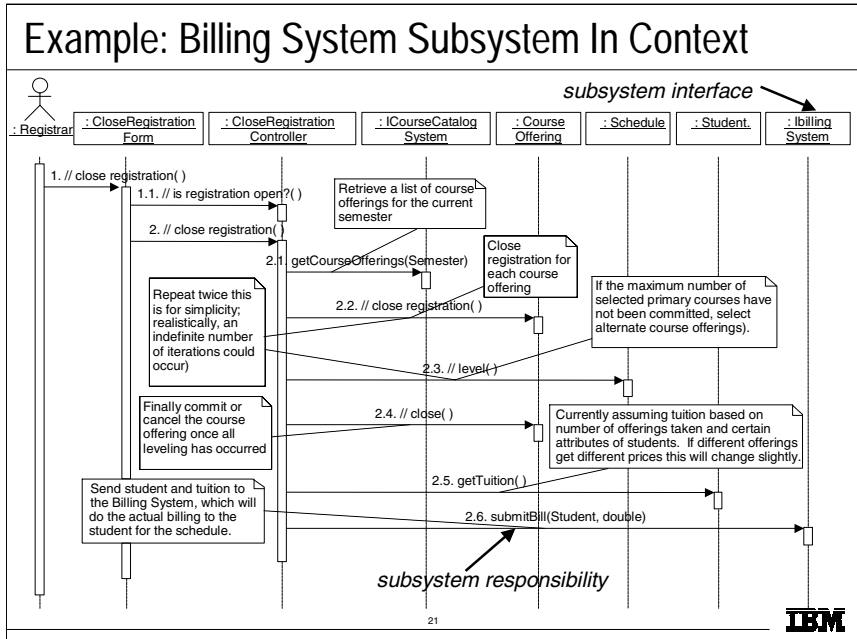
Note: The string passed to *executeQuery()* is *not* the exact same string as the one passed into the *read()*. The DBClass builds the SQL query to retrieve the persistent data from the database, using the criteria passed into the *read()*. This is because we do not want the client of the DBClass to have the knowledge of the internals of the database to create a valid query. This knowledge is encapsulated within DBClass.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

On the presented slide, the subsystem interface is shown in white, but this does not show up in the black-and-white manuals.

The requirements do not include any specific requirements for the Billing System. Thus, for the purposes of the Subsystem Design module, we will concentrate on the basics of building a transaction to the Billing System and communicating with the Billing System through a specific boundary class.



In this example, we will demonstrate the design of a subsystem that does not require the incorporation of an architectural mechanism.

The above sequence diagram is a portion of the Close Registration use-case realization sequence diagram. The internals of the Billing System subsystem have not been designed yet. That is the purpose of this activity, **Subsystem Design**.

This diagram sets the context of what will be performed in **Subsystem Design**. It puts requirements on the subsystem, and is the primary input specification for the task of creating local interactions for the subsystem.

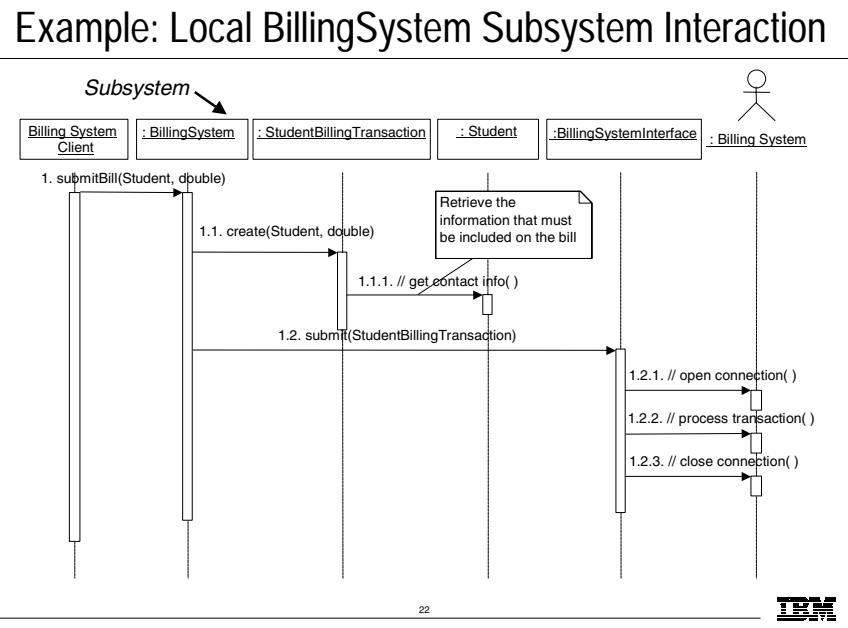
In the above example for the BillingSystem subsystem, we see the operations the subsystem must support. This shows the simple way that some client (CloseRegistrationController here) deals with the task of submitting a student bill to the legacy Billing System. The IBillingsystem:submitBill() operation documentation specifies the following: "Billing information must be converted into a format understood by the external Billing System, and then submitted to the external Billing System." Thus, the actual generation and submission of the bill is the responsibility of the Billing System subsystem. In **Subsystem Design**, we will describe exactly *how* this is done.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

Message Parameters: In most cases, we want to direct the student to use the parameter name (theTuition), rather than the type (double) when describing the parameters of a message. Parameters are more meaningful in most cases. If you look at the example `submitBill(Student, double)`, this would be much more descriptive displayed as `submitBill(aStudent, theTuition)`.

If there are cases where the parameter name does not add any value, then it would be acceptable to just use the Type.



Designing the internals of a subsystem should yield (local) interaction diagrams like the sequence diagram shown above.

This example “looks inside” the BillingSystem subsystem and shows the collaborations required to implement the `submitBill()` operation of the IBillingSystem interface.

The client object initiating the interaction is abstracted to be an untyped object here. That is because, within the scope of the design of this one subsystem, we do not care who the client is.

The BillingSystem subsystem proxy class actually realizes the IBillingSystem interface. It is the class that delegates the implementation of the interface to the subsystem elements.

The BillingSystem proxy class instance creates a StudentBillingTransaction specific to the external Billing System. This transaction will be in a format that the Billing System can process. The StudentBillingTransaction knows how to create itself using information from the given Student. After creating the StudentBillingTransaction, the BillingSystem proxy class instance submits the transaction to the class instance that actually communicates with the Billing System.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

This is like generating the View of Participating Classes (VOPC) class diagram for the subsystem.

Subsystem Design Steps

- ◆ Distribute subsystem behavior to subsystem elements
- ☆ ◆ Document subsystem elements
- ◆ Describe subsystem dependencies
- ◆ Checkpoints



23



At this point, the responsibilities allocated to the subsystems have been further allocated to subsystem elements, and the collaborations between the subsystem elements have been modeled using interaction diagrams.

Now you must document and model the internal structure of the subsystem. This internal structure is driven by what is required to support the collaborations to implement the subsystem interfaces, as documented in the previous step.

This is where you model the subsystem element relationships.

To document the internal structure of the subsystem, create one or more class diagrams showing the elements contained by the subsystem and their associations with one another. One class diagram should be sufficient, but more can be used to reduce complexity and improve readability.

In addition, a state diagram might be needed to document the possible states the subsystem can assume (interfaces and subsystems are “stateful”).

It is also important to document any order dependencies between subsystem interface operations (for example, op1 must be executed before op2, and so on).

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

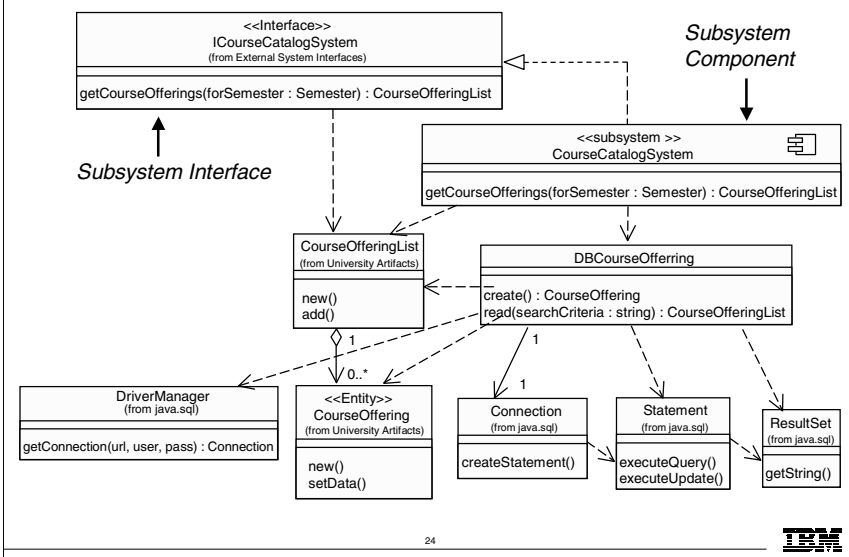
Emphasize the modeling of the interface and subsystem proxy, as well as the incorporation of the JDBC RDBMS Persistency mechanism.

Stress how the subsystem element collaborations drive the relationships defined between them by flipping back to the sequence diagram shown earlier.

If nested subsystems are defined, the class diagrams should show the relationships to the subsystem interfaces. The internals of those subsystems will, in turn, be designed during the Subsystem Design activity for those subsystems.

DBCourseOffering understands the OO-to-DBMS mapping and can interface with the DBMS. This database interface class is used whenever a CourseOffering instance needs to be created, accessed, or deleted. DBCourseOffering “flattens” the object and writes it to the DBMS and reads the object data from the DBMS and builds the object. The advantages of this approach is that the OO model is separable from the DBMS model and the CourseOffering class does not know it is persistent. The CourseOffering class can be reused in an application in which its objects are not persistent.

Example: CourseCatalogSystem Subsystem Elements



This diagram models the subsystem elements and their relationships. These relationships support the required collaborations between the design elements to support the behavior of the subsystem (as documented in the subsystem interfaces). For our purposes, we concentrated on the `getCourseOfferings()` interface operation.

CourseCatalogSystem works with DBCourseOffering to read and write persistent data from the Course Catalog System RDBMS. DBCourseOffering is responsible for accessing the JDBC database using the previously established Connection (see JDBC Initialize interaction diagram). Once a database connection is opened, DBCourseOffering can then create SQL statements that will be sent to the underlying RDBMS and executed using the Statement class. The results of the SQL query is returned in a ResultSet class object.

Note: Elements outside of the subsystem are shown, as well, to provide context. These elements can be identified because their owning package is listed in parentheses below the class name (for example, “from University Artifacts”).

Just as in Use-Case Analysis and Use-Case Design, the subsystem element collaborations modeled previously in the interaction diagrams drive the relationships defined between the participating design elements.

This is exactly the same approach we used to identify analysis class relationships in Use-Case Analysis and to identify design element relationships in Use-Case Design.

Mastering OOAD w/UML 2.0 – Instructor Notes

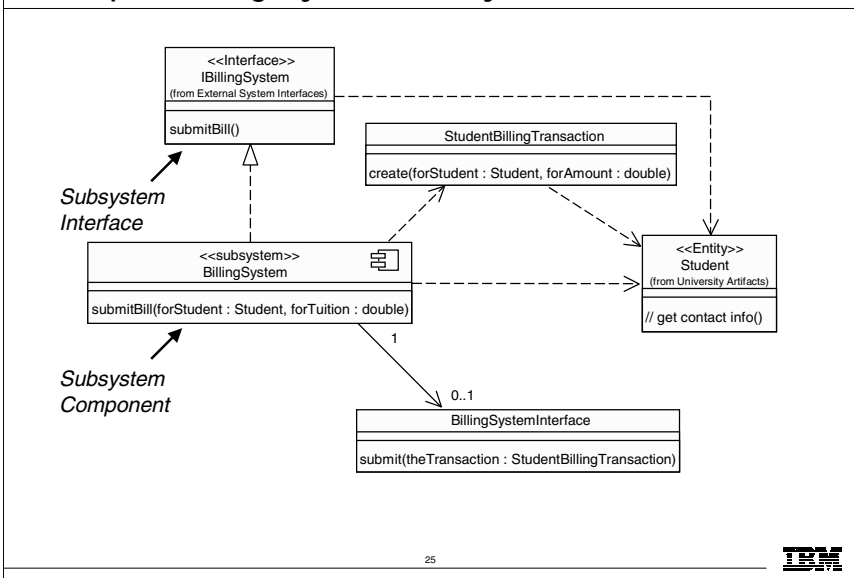
Instructor Notes:

Emphasize the modeling of the interface and subsystem proxy.

Stress how the subsystem element collaborations drive the relationships defined between them by flipping back to the sequence diagram shown earlier.

If nested subsystems are defined, the class diagrams should show the relationships to the subsystem interfaces. The internals of those subsystems will, in turn, be designed during the Subsystem Design activity for those subsystems.

Example: Billing System Subsystem Elements



This diagram models the subsystem elements and their relationships. These relationships support the required collaborations between the design elements to support the behavior of the subsystem (as documented in the subsystem interfaces). For our purposes, we concentrated on the `submitBill()` interface operation.

Note: Elements outside of the subsystem are shown, as well, to provide context. These elements can be identified because their owning package is listed in parentheses below the class name (for example, "from University Artifacts").

Just as in Use-Case Analysis and Use-Case Design, the subsystem element collaborations modeled previously in the interaction diagrams drive the relationships defined between the participating design elements.

This is exactly the same approach we used to identify analysis class relationships in Use-Case Analysis and to identify design element relationships in Use-Case Design.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

Subsystem Design Steps

- ◆ Distribute subsystem behavior to subsystem elements
- ◆ Document subsystem elements
- ☆◆ Describe subsystem dependencies
- ◆ Checkpoints



26



At this point, subsystem elements have been defined to implement the subsystem responsibilities, the resulting collaborations between the elements have been modeled using interaction diagrams, and the internal structure of the subsystem (that is, the relationships between subsystem elements) has been modeled using class diagrams.

Now we must document the elements external to the subsystem, upon which the subsystem is dependent. These dependencies may have been introduced when designing the internals of the subsystem as described earlier in this module.

Note: Subsystems might not be able to stand alone; they might need the services of other subsystems. Rather than forcing all the definition work onto the architect (or architecture team), which could become rather bureaucratic and cumbersome, the subsystem designer should feel free to use the services of other subsystems. However, the architect establishes the ground rules for such referencing (through design and layering guidelines), and ultimately must agree with the dependencies.

Mastering OOAD w/UML 2.0 – Instructor Notes

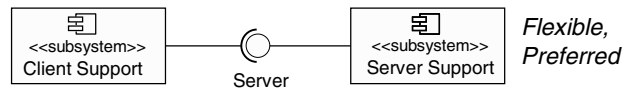
Instructor Notes:

Create a dependency association between the subsystem and each package/subsystem interface the subsystem is dependent upon. As a result, existing subsystem relationships and possibly subsystem interfaces may need to be refined. Using interface dependencies allows flexible frameworks to be designed using replaceable design elements.

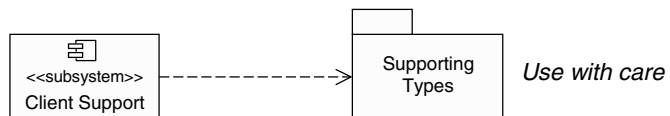
Note: Any changes to subsystem interfaces or to relationships between subsystems must be evaluated from an architecture perspective (see Identify Design Elements).

Subsystem Dependencies: Guidelines

◆ Subsystem dependency on a subsystem



◆ Subsystem dependency on a package



27



When a subsystem element uses some behavior of an element contained by another subsystem or package, a dependency on the external element is needed.

If the element on which the subsystem is dependent is within a subsystem, the dependency should be on the subsystem interface, not on the subsystem itself or on any element in the subsystem. This allows the design elements to be substituted for one another as long as they offer the same behavior. It also gives the designer total freedom in designing the internal behavior of the subsystem, as long as it provides the correct external behavior. If a model element directly references a model element in another subsystem, the designer is no longer free to remove that model element or redistribute the behavior of that model element to other elements. As a result, the system is more brittle.

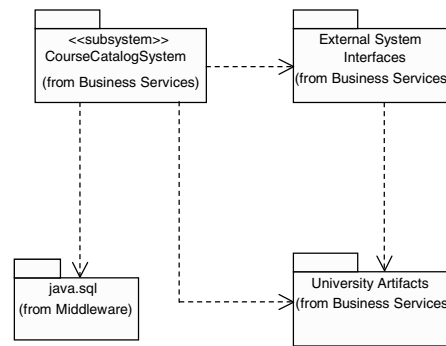
If the element the subsystem element is dependent on is within a package, the dependency should be on the package itself. Ideally, a subsystem should only depend on the interfaces of other model elements for the reasons stated above. The exception is where a number of subsystems share a set of common class definitions, in which case those subsystems “import” the contents of the packages containing the common classes. This should be done only with packages in lower layers in the architecture to ensure that common class definitions are defined consistently. The disadvantage is that the subsystem cannot be reused independent of the depended-on package.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

You may want to demonstrate how these dependencies support the relationships between the enclosed classes by flipping back to the class diagram for the CourseCatalogSystem subsystem.

Example: CourseCatalogSystem Subsystem Dependencies



28



This diagram models the dependencies that the CourseCatalogSystem subsystem has with other design elements. These dependencies support the relationships of the enclosed classes as modeled on the earlier subsystem class diagrams. They are on standard packages that do not have a specific interface. Thus, the CourseCatalogSystem subsystem cannot be reused without the packages it depends on.

The CourseCatalogSystem subsystem is dependent on the java.sql package in order to gain access to the design elements that implement the RDBMS persistency mechanism.

The CourseCatalogSystem subsystem is dependent on the External System Interfaces package for gaining access to the subsystem interface itself (ICourseCatalogSystem). Remember, the subsystem interfaces were not packaged with the subsystems themselves.

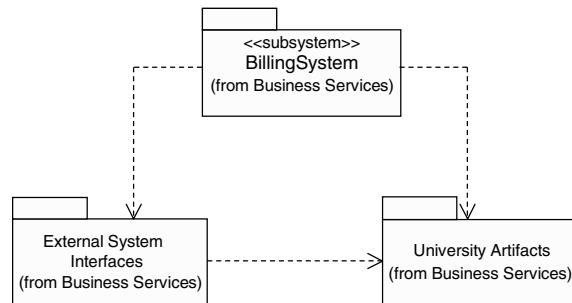
The CourseCatalogSystem subsystem is dependent on the University Artifacts package in order to gain access to the core types of the Course Registration System.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

You may want to demonstrate how these dependencies support the relationships between the enclosed classes by flipping back to the class diagram for the BillingSystem subsystem.

Example: BillingSystem Subsystem Dependencies



29



This diagram models the dependencies that the BillingSystem subsystem has with other design elements. These dependencies support the relationships of the enclosed classes as modeled on the earlier subsystem class diagrams. They are on standard packages that do not have a specific interface. Thus, the BillingSystem subsystem cannot be reused without the packages it depends on.

The BillingSystem subsystem is dependent on the External System Interfaces package in order to gain access to the subsystem interface itself (IBillingSystem). Remember, the subsystem interfaces were not packaged with the subsystems themselves.

The BillingSystem subsystem is dependent on the University Artifacts package in order to gain access to the core types of the Course Registration System.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

Subsystem Design Steps

- ◆ Distribute subsystem behavior to subsystem elements
- ◆ Document subsystem elements
- ◆ Describe subsystem dependencies
- ☆ ◆ Checkpoints

30



Now we will discuss the kinds of things you should look for when reviewing the results of Subsystem Design.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

Checkpoints: Design Subsystems

- ♦ Is a realization association defined for each interface offered by the subsystem?
- ♦ Is a dependency association defined for each interface used by the subsystem?
- ♦ Are you sure that none of the elements within the subsystem have public visibility?
- ♦ Is each operation on an interface realized by the subsystem documented in an interaction diagram? If not, is the operation realized by a single class, so that it is easy to see that there is a simple 1:1 mapping between the class operation and the interface operation?



31



This checklist includes the key things to look for when assessing the results of **Subsystem Design**.

A designer is responsible for the integrity of the design subsystem, ensuring that:

- The subsystem encapsulates its contents, only exposing contained behavior through interfaces it realizes.
- The operations of the interfaces the subsystem realizes are distributed to contained classes or subsystems.
- The subsystem properly implements its interfaces.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

1. In **Subsystem Design**, you look at the responsibilities of a subsystem in detail. You define and refine the classes that are needed to implement the responsibilities, and refine subsystem dependencies, as needed.
2. Gates are a connection point for a message that comes into or goes outside the interaction. Messages within the interaction can be connected to the gate.
3. If the element on which the subsystem is dependent is within a subsystem, the dependency should be on the subsystem interface, not on the subsystem itself or on any element in the subsystem. This allows the design elements to be substituted for one another as long as they offer the same behavior.

Review: Subsystem Design

- ♦ What is the purpose of Subsystem Design?
- ♦ What are gates?
- ♦ Why should dependencies on a subsystem be on the subsystem interface?



32



Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

Divide the class into teams (or use the previously established teams). Assign one subsystem to each team (or have multiple teams work on the same subsystem).

If the students are a little overwhelmed at this point, you can choose to give them the mechanisms that they are to apply for the specific subsystems (see the Instructor Notes for slide 35 for the list of what mechanisms should be applied to each subsystem).

Exercise: Subsystem Design

- ♦ Given the following:
 - The defined subsystems, their interfaces and their relationships with other design elements (the subsystem context diagrams)
 - Payroll Exercise Solution, Identify Design Elements
 - Patterns of use for the architectural mechanisms
 - Exercise Workbook: Payroll Architecture Handbook, Architectural Mechanisms, Implementation Mechanisms section



33



The goal is to perform the subsystem design of one of the previously identified subsystems. There are not really any specific requirements for these subsystems, so, for the purpose of this exercise, concentrate on incorporating the applicable architectural mechanisms (for example, persistency, security, distribution), and just some basic functionality the subsystem may perform. Do not worry about developing a detailed subsystem design. For such a design, you would need more detailed requirements.

- **Subsystem context class diagrams:** Payroll Exercise Solution, Identify Design Elements,
- **Exercise:** Identify Design Elements, Subsystem Context Diagrams section. Note the operations descriptions.
- **The patterns of use for the architectural mechanisms:** Exercise Workbook: Payroll Architecture Handbook, Architectural Mechanisms, Implementation Mechanisms section.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

Walk the students through the process used to develop the interface realizations.

Emphasize that it's no different from what we did in Use-Case Analysis.

If the students get stuck on what the internals of the subsystems should be, tell them to look at the sample subsystem designs in this module for ideas.

For this exercise, the students should concentrate on the application of the RDBMS persistency mechanism for the Project Management Database System subsystem (the Payroll System is a read-only legacy system with an RDBMS database), the general idea of building a bank transaction for the Bank System subsystem, the general idea of transforming an internal class into a representation suitable for printing, and not on any of the other subsystem design details.

If you skipped the RDBMS persistency mechanism, then do not assign the Project Management Database System subsystem to anyone.

Exercise: Subsystem Design (continued)

- ♦ Identify the following for a particular subsystem(s):
 - The design elements contained within the subsystem and their relationships
 - The applicable architectural mechanisms
 - The interactions needed to implement the subsystem interface operations



34



The process used in **Subsystem Design** is no different from that used in Use-Case Analysis and Use-Case Design, except instead of allocating use-case responsibilities, you are allocating subsystem interface responsibilities. For each subsystem interface operation, you identify design elements that are needed to implement the interface, and then you allocate some responsibilities to it. Next, you develop interaction diagrams to illustrate the necessary collaborations, and class diagrams to model the supporting relationships. You are, in fact, developing "interface realizations" rather than Use-Case Realizations.

There are no explicit requirements for the details of the subsystem (that's what detailed design is all about). In any case, use the documentation for the interface and interface operations to guide you.

Remember, design elements can be classes and/or subsystems.

When developing the interactions, do not forget to incorporate any applicable architectural mechanisms. In this course, we are concentrating on the following architectural mechanisms: persistency, security, distribution, and legacy interface.

The defined design element relationships should support the interactions required to implement the subsystem interface responsibilities/interfaces.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

The exercise solution can be found in the *Payroll Exercise Solution*, Subsystem Design section. See the table of contents for the specific page numbers.

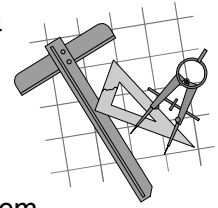
The Payroll Solution contains the design for the following subsystems:

- Bank System (does not require the application of any mechanism).
- PrintService (does not require the application of any mechanism). This has the most interesting design.
- Project Management Database (RDBMS Persistency - Read mechanism).

When reviewing the student subsystem designs, don't concentrate on the details of their subsystem design. Just make sure that the designs are consistent with the interface/subsystem descriptions and that their class and interaction diagrams are consistent.

Exercise: Subsystem Design (continued)

- ♦ Produce the following diagrams for a particular subsystem(s):
 - “Interface realizations”
 - Interaction diagram for each interface operation
 - Class diagram containing the subsystem design elements that realize the interface responsibilities and their relationships
 - Class diagram that shows the subsystem and any dependencies on external package(s) and/or subsystem(s) (subclass dependencies class diagram)



35



There is one interaction diagram per interface operation to ensure that all responsibilities have been allocated to a subsystem element. Naming the diagrams to reflect the operation they model helps with traceability. The interaction diagrams may be communication or sequence diagrams.

As with Use-Case Realizations, for interface realizations there is a class diagram that contains the design elements that realize the interface responsibilities. Just like the VOPC for use-case realizations, for every link on the interaction diagrams, there should be a relationship on the class diagram.

During detailed design, we may have found that the subsystem needs the services of something outside of the subsystem in order to fulfill its responsibilities. In such a case, the subsystem must have a dependency on that element (or the containing package or subsystem). These elements are “suppliers” of the subsystem. Such dependencies are usually monitored and regulated by the architecture team.

The subsystem dependencies class diagram should contain the subsystem and any dependencies on external package(s) and/or subsystem(s). This diagram is meant to show the subsystem dependencies on external design elements.

Remember to use the conventions recommended in the course.

References to sample diagrams within the course that are similar to what should be produced are:

- Subsystem interface operation interaction diagram: 12-21.
- Subsystem class diagram: 12-24 and 12-25.
- Subsystem dependencies class diagram: 12-28 and 12-29.

Mastering OOAD w/UML 2.0 – Instructor Notes

Instructor Notes:

Exercise: Review

♦ Compare your Subsystem Interface Realizations

- ♦ Have all the main and/or subflows for the interface operations been handled?
- ♦ Has all behavior been distributed among the participating design elements?
- ♦ Has behavior been distributed to the right design elements?
- ♦ Are there any messages coming from the interfaces?



36



After completing a model, it is important to step back and review your work. Some helpful questions are:

- Have all the main and/or subflows for the interface operations been handled?
- Has all behavior been distributed among the participating design elements? This includes design classes and interfaces.
- Has behavior been distributed to the right design elements?
- Are there any messages coming from the interface? Remember, messages should not come from an interface because the behavior is realized by the subsystem.