COMPONENT DIAGRAMS

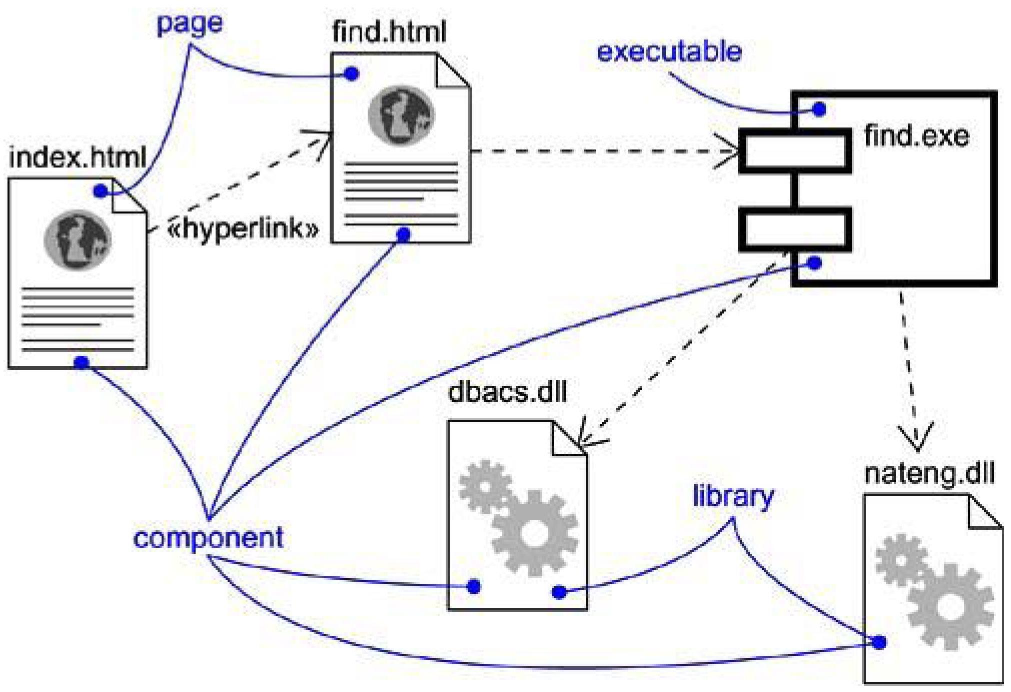

Figure 1: Component Diagram

Component Diagrams

- Component diagrams are used in modeling the physical aspects of object-oriented systems.
- A component diagram shows the organization and dependencies among a set of components.
- Component diagrams are used to model the static implementation view of a system.
- Component diagrams are essentially class diagrams that focus on a system's components.
- Graphically, a Component diagram is a collection of vertices and arcs.
- Component diagrams are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering.

- Component diagrams commonly contain Components, Interfaces and Dependency, generalization, association, and realization relationships. It may also contain notes and constraints.

<u>Common Uses</u>

component diagrams are used in one of four ways

- To model source code
- To model executable releases
- To model physical databases
- To model adaptable systems

Modeling Source Code

To model a system’s source code,

- Either by forward or reverse engineering, identify the set of source code files of interest and model them as components stereotyped as files.
- For larger systems, use packages to show groups of source code files.
- Consider exposing a tagged value indicating such information as the version number of the source code file, its author, and the date it was last changed. Use tools to manage the value of this tag.
- Model the compilation dependencies among these files using dependencies. Again, use tools to help generate and manage these dependencies.

Figure 2 shows five source code files.

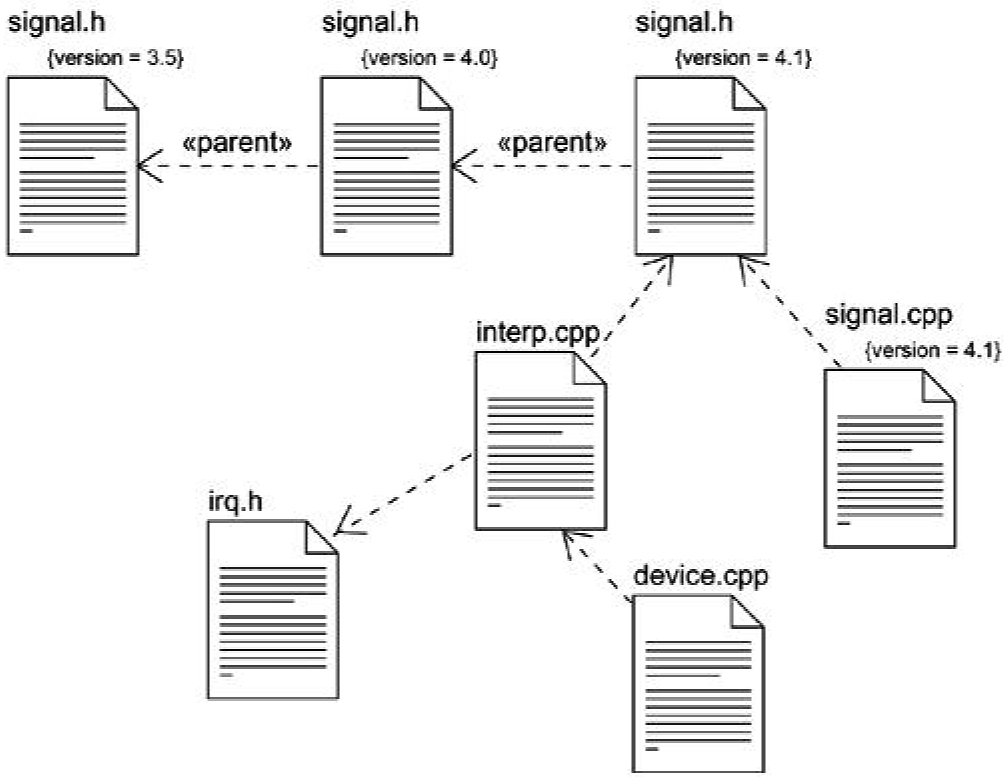

Figure 2: Modeling Source Code

Modeling an Executable Release

To model an executable release,

- Identify the set of components you'd like to model. Typically, this will involve some or all the components that live on one node, or the distribution of these sets of components across all the nodes in the system.
- Consider the stereotype of each component in this set. For most systems, you'll find a small number of different kinds of components (such as executables, libraries, tables, files, and documents). You can use the UML's extensibility mechanisms to provide visual cues(clues) for these stereotypes.
- For each component in this set, consider its relationship to its neighbors. Most often, this will involve interfaces that are exported (realized) by certain components and then imported (used) by others. If you want to expose the seams in your system, model these interfaces explicitly. If you want your model at a higher level of abstraction, elide these relationships by showing only dependencies among the components.

Figure 3 models part of the executable release for an autonomous robot.

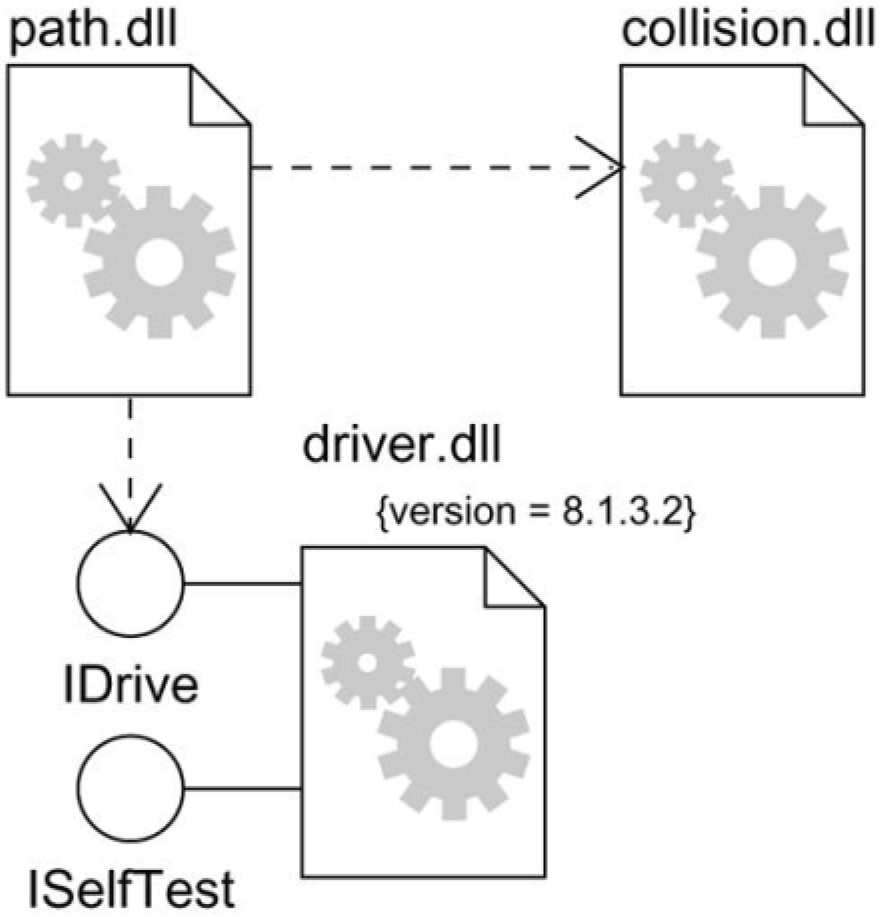

Figure 3: Modeling an Executable Release

Modeling a Physical Database

To model a physical database,

- Identify the classes in your model that represent your logical database schema.
- Select a strategy for mapping these classes to tables. You will also want to consider the physical distribution of your databases. Your mapping strategy will be affected by the location in which you want your data to live on your deployed system.
- To visualize, specify, construct, and document your mapping, create a component diagram that contains components stereotyped as tables.
- Where possible, use tools to help you transform your logical design into a physical design.

Figure 4 shows a set of database tables drawn from an information system for a school.

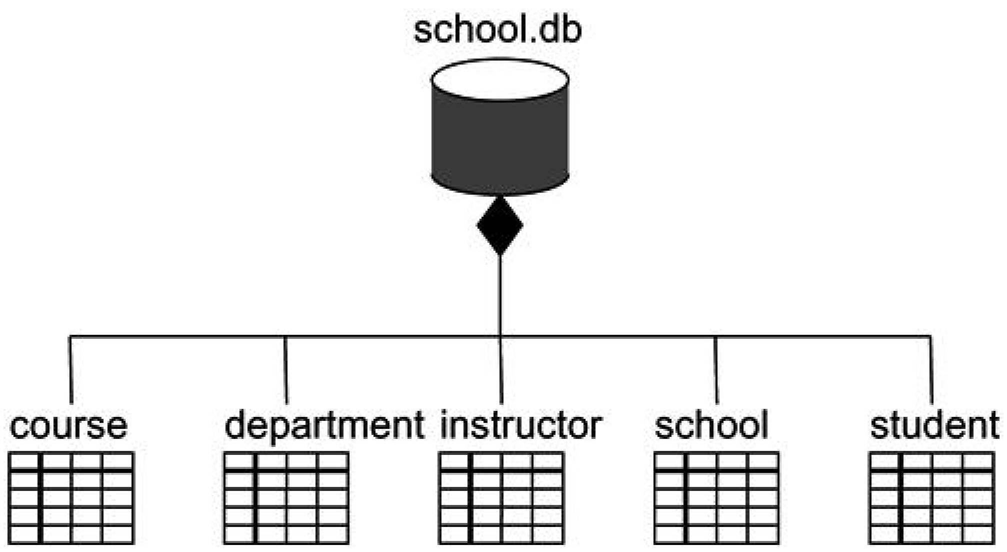

Figure 4: Modeling a Physical Database

Modeling Adaptable Systems

To model an adaptable system,

- Consider the physical distribution of the components that may migrate from node to node. You can specify the location of a component instance by marking it with a location tagged value, which you can then render in a component diagram (although, technically speaking, a diagram that contains only instances is an object diagram).
- If you want to model the actions that cause a component to migrate, create a corresponding interaction diagram that contains component instances. You can illustrate a change of location by drawing the same instance more than once, but with different values for its location tagged value.

Figure 5 models the replication of the database from figure 4.

Source : http://praveenthomasln.wordpress.com/2012/04/07/component-diagrams-s8-cs/

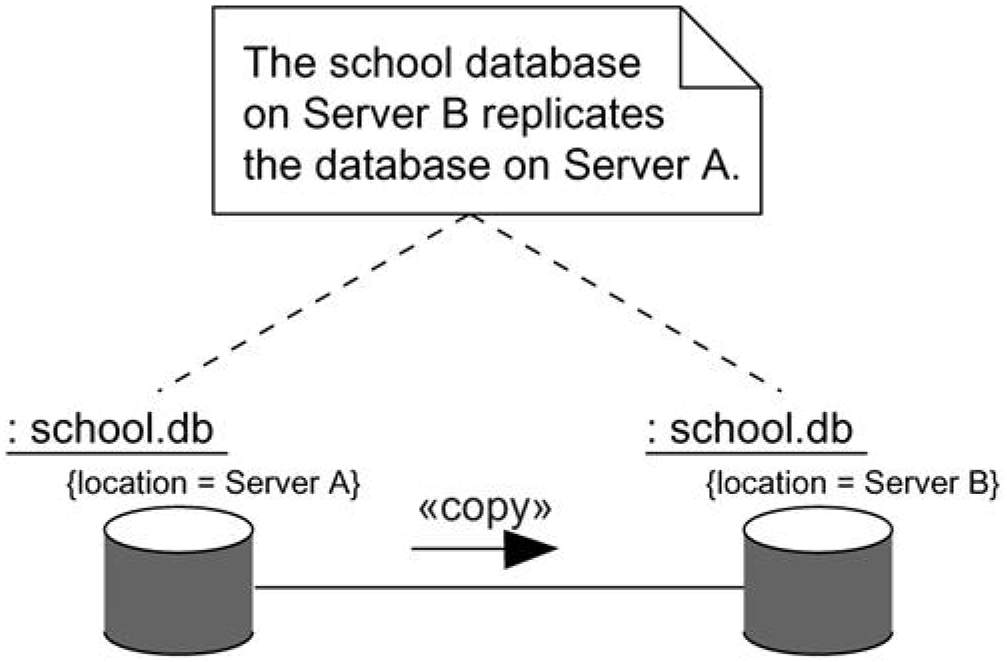

Figure 5: Modeling Adaptable Systems