# SFWR ENG 3A04: Software Design II

Dr. Ridha Khedri

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Term 1, 2008–2009

**Acknowledgments**: Material based on *Software Architecture Design* by Tao et al. (Chapter 2)

# Outline of Part I

# Outline of Part II

# Part I

## Review of Previous Lecture

# Part II

## Today's Lecture

- The architect must be aware of

- The architect must be aware of
  - the alternatives in proposing design solutions

- The architect must be aware of
  - the alternatives in proposing design solutions
  - which alternatives that are more suitable to capture the functional and non-functional requirements

- The architect must be aware of
  - the alternatives in proposing design solutions
  - which alternatives that are more suitable to capture the functional and non-functional requirements
- This awareness comes from understanding the software architecture design space with all its dimensions

- A software architecture can be given from several perspectives

# Software Architecture Design Space
## Introduction

- A software architecture can be given from several perspectives
  - Software code units (Elements are source, binary code files, software modules, or software component deployment units)

- A software architecture can be given from several perspectives
  - Software code units (Elements are source, binary code files, software modules, or software component deployment units)
  - Project's runtime structure (Elements are threads, processes, sessions, transactions, objects, or software component instances)

- A software architecture can be given from several perspectives
  - Software code units (Elements are source, binary code files, software modules, or software component deployment units)
  - Project's runtime structure (Elements are threads, processes, sessions, transactions, objects, or software component instances)
  - Allocation structure (Project management structure)

- A software architecture can be given from several perspectives
  - Software code units (Elements are source, binary code files, software modules, or software component deployment units)
  - Project's runtime structure (Elements are threads, processes, sessions, transactions, objects, or software component instances)
  - Allocation structure (Project management structure)
- Each type of structure uses different connector types and different performance attributes than the others

- A software project is implemented in multiple source files

## Software Architecture Design Space Software Code Structure

- A software project is implemented in multiple source files
  - Executable files, library files, binary software component modules (usually in form of DLL, JavaBeans and Enterprise JavaBeans), deployment descriptors and other resource files

- A software project is implemented in multiple source files
    - Executable files, library files, binary software component modules (usually in form of DLL, JavaBeans and Enterprise JavaBeans), deployment descriptors and other resource files
- At software development time, the main software elements are source code modules or files

- A software project is implemented in multiple source files
    - Executable files, library files, binary software component modules (usually in form of DLL, JavaBeans and Enterprise JavaBeans), deployment descriptors and other resource files
- At software development time, the main software elements are source code modules or files
- Each of these software modules will be assigned

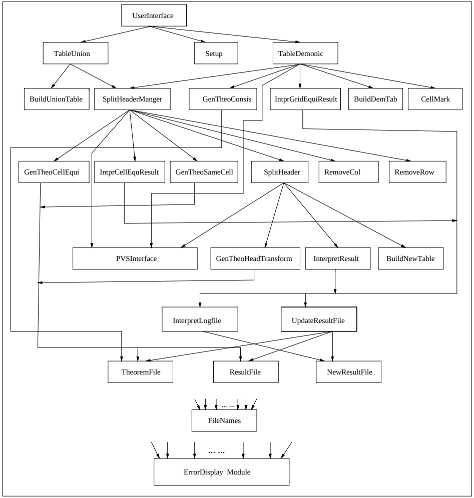
- A software project is implemented in multiple source files
  - Executable files, library files, binary software component modules (usually in form of DLL, JavaBeans and Enterprise JavaBeans), deployment descriptors and other resource files
- At software development time, the main software elements are source code modules or files
- Each of these software modules will be assigned
  - Functional and non-functional attributes

- A software project is implemented in multiple source files
  - Executable files, library files, binary software component modules (usually in form of DLL, JavaBeans and Enterprise JavaBeans), deployment descriptors and other resource files
- At software development time, the main software elements are source code modules or files
- Each of these software modules will be assigned
  - Functional and non-functional attributes
  - Public APIs (Application Programming Interface): defined for each module to separate the interfaces and implementations of a module

- A software project is implemented in multiple source files
  - Executable files, library files, binary software component modules (usually in form of DLL, JavaBeans and Enterprise JavaBeans), deployment descriptors and other resource files
- At software development time, the main software elements are source code modules or files
- Each of these software modules will be assigned
  - Functional and non-functional attributes
  - Public APIs (Application Programming Interface): defined for each module to separate the interfaces and implementations of a module

- A software project is implemented in multiple source files
  - Executable files, library files, binary software component modules (usually in form of DLL, JavaBeans and Enterprise JavaBeans), deployment descriptors and other resource files
- At software development time, the main software elements are source code modules or files
- Each of these software modules will be assigned
  - Functional and non-functional attributes
  - Public APIs (Application Programming Interface): defined for each module to separate the interfaces and implementations of a module

# Software Architecture Design Space Software Code Structure




We might think of many types of dependency relations, such as:

- Component/Module $A$ contains Component/Module $B$

We might think of many types of dependency relations, such as:

- Component/Module $A$ contains Component/Module $B$
- Component/Module $A$ follows Component/Module $B$

We might think of many types of dependency relations, such as:

- Component/Module *A* contains Component/Module *B*
- Component/Module *A* follows Component/Module *B*
- Component/Module *A* delivers data to Component/Module *B*

We might think of many types of dependency relations, such as:

- Component/Module $A$ contains Component/Module $B$
- Component/Module $A$ follows Component/Module $B$
- Component/Module $A$ delivers data to Component/Module $B$
- Component/Module $A$ uses Component/Module $B$

- The type of dependencies we are interested in are those that determine the complexity of the relations between components

- The type of dependencies we are interested in are those that determine the complexity of the relations between components
- The amount of knowledge that components/modules have of each other should be kept to a minimum

- The type of dependencies we are interested in are those that determine the complexity of the relations between components

- The amount of knowledge that components/modules have of each other should be kept to a minimum

- In proper design, the information flow between components is restricted to flow that comes through method calls

- The type of dependencies we are interested in are those that determine the complexity of the relations between components
- The amount of knowledge that components/modules have of each other should be kept to a minimum
- In proper design, the information flow between components is restricted to flow that comes through method calls
- The graph depicting the "uses" relation is therefore often called a call graph

Connectors in structure graphs can have attributes including the following:

- Direction: If module A invokes a method (in general sense) of module B during execution, then there is a unidirectional connector from module A to module B

Connectors in structure graphs can have attributes including the following:

- Direction: If module A invokes a method (in general sense) of module B during execution, then there is a unidirectional connector from module A to module B
- Synchronization: A method invocation can be synchronous or asynchronous

Connectors in structure graphs can have attributes including the following:

- Direction: If module A invokes a method (in general sense) of module B during execution, then there is a unidirectional connector from module A to module B
- Synchronization: A method invocation can be synchronous or asynchronous
  - Asynchronous operation = a process operates independently of other processes

Connectors in structure graphs can have attributes including the following:

- Direction: If module A invokes a method (in general sense) of module B during execution, then there is a unidirectional connector from module A to module B
- Synchronization: A method invocation can be synchronous or asynchronous
  - Asynchronous operation = a process operates independently of other processes
  - Synchronous operation = a process runs only as a result of some other process being completed or handing off operation

Connectors in structure graphs can have attributes including the following:

- Direction: If module A invokes a method (in general sense) of module B during execution, then there is a unidirectional connector from module A to module B
- Synchronization: A method invocation can be synchronous or asynchronous
  - Asynchronous operation = a process operates independently of other processes
  - Synchronous operation = a process runs only as a result of some other process being completed or handing off operation
- Sequence: Some connectors must be used in a particular sequence (label the connector with a sequence ID and a sequence number)

- At runtime a project can be threads, processes, functional units, and data units

- At runtime a project can be threads, processes, functional units, and data units
- These elements may run on the same computer or on multiple computers across a network

# Software Architecture Design Space   Software Runtime Structure

- At runtime a project can be threads, processes, functional units, and data units
- These elements may run on the same computer or on multiple computers across a network
- An element in a code structure can implement or support multiple runtime elements

*In a client-server application, the same client module may run on many client computers*

- At runtime a project can be threads, processes, functional units, and data units
- These elements may run on the same computer or on multiple computers across a network
- An element in a code structure can implement or support multiple runtime elements

  *In a client-server application, the same client module may run on many client computers*

- Several code structure elements may implement or support a single runtime element

- At runtime a project can be threads, processes, functional units, and data units
- These elements may run on the same computer or on multiple computers across a network
- An element in a code structure can implement or support multiple runtime elements

  *In a client-server application, the same client module may run on many client computers*

- Several code structure elements may implement or support a single runtime element

- At runtime a project can be threads, processes, functional units, and data units
- These elements may run on the same computer or on multiple computers across a network
- An element in a code structure can implement or support multiple runtime elements

  *In a client-server application, the same client module may run on many client computers*

- Several code structure elements may implement or support a single runtime element

  *Many threads run multiple methods from different classes that may be packaged in different code units*

- At runtime a project can be threads, processes, functional units, and data units
- These elements may run on the same computer or on multiple computers across a network
- An element in a code structure can implement or support multiple runtime elements

  *In a client-server application, the same client module may run on many client computers*

- Several code structure elements may implement or support a single runtime element

  *Many threads run multiple methods from different classes that may be packaged in different code units*

The connectors at this level inherit attributes from their source-code structure counterparts:

- Multiplicity: One element can be connected to multiple other elements if it needs to invoke methods of multiple elements at runtime

The connectors at this level inherit attributes from their source-code structure counterparts:

- Multiplicity: One element can be connected to multiple other elements if it needs to invoke methods of multiple elements at runtime
- Distance and connection media: Two connected elements may communicate

The connectors at this level inherit attributes from their source-code structure counterparts:

- Multiplicity: One element can be connected to multiple other elements if it needs to invoke methods of multiple elements at runtime
- Distance and connection media: Two connected elements may communicate
  - in the same thread

The connectors at this level inherit attributes from their source-code structure counterparts:

- Multiplicity: One element can be connected to multiple other elements if it needs to invoke methods of multiple elements at runtime
- Distance and connection media: Two connected elements may communicate
  - in the same thread
  - in the same process

The connectors at this level inherit attributes from their source-code structure counterparts:

- Multiplicity: One element can be connected to multiple other elements if it needs to invoke methods of multiple elements at runtime
- Distance and connection media: Two connected elements may communicate
  - in the same thread
  - in the same process
  - on the same computer

The connectors at this level inherit attributes from their source-code structure counterparts:

- Multiplicity: One element can be connected to multiple other elements if it needs to invoke methods of multiple elements at runtime
- Distance and connection media: Two connected elements may communicate
  - in the same thread
  - in the same process
  - on the same computer
  - or on different computers across a network

The connectors at this level inherit attributes from their source-code structure counterparts:

- Multiplicity: One element can be connected to multiple other elements if it needs to invoke methods of multiple elements at runtime
- Distance and connection media: Two connected elements may communicate
  - in the same thread
  - in the same process
  - on the same computer
  - or on different computers across a network
  - the communication media may vary from copper/optical cable or wireless based LAN to the Internet, etc.

Connectors (continued)

- Universal invokable: Allows ANY external software system to invoke the method at the connector's target (allows heterogeneous enterprise information systems)

Connectors (continued)

- Universal invokable: Allows ANY external software system to invoke the method at the connector's target (allows heterogeneous enterprise information systems)
- Self-descriptive: Allows external software systems to invoke its target method without the pre-installation of any software specific for the method

  *Bluetooth devices from different companies may be able to discover each other and exchange information*

- Some software architectures are best implemented by a particular software management structure

# Software Architecture Design Space Software Management Structure

- Some software architectures are best implemented by a particular software management structure
- Software management structures are also often used for project resource allocation

- Some software architectures are best implemented by a particular software management structure
- Software management structures are also often used for project resource allocation
- Software runtime structures represent the technical essence of software architectures (the other types of structures are derived from it)

- Some software architectures are best implemented by a particular software management structure
- Software management structures are also often used for project resource allocation
- Software runtime structures represent the technical essence of software architectures (the other types of structures are derived from it)
- 

*See previous Call Graph*

At runtime,

- Each software element has its well-defined functions

At runtime,

- Each software element has its well-defined functions
- Elements are connected into a dependency graph through connectors

At runtime,

- Each software element has its well-defined functions
- Elements are connected into a dependency graph through connectors
- The elements of a software architecture are usually refined through multiple transformation steps based on the project requirement specification

- Each software element may have different synchronization and performance constraints

    *Some elements are reentrant (it can be safely executed concurrently) objects or software components, while some are not reentrant and there should not be more than one thread executing in it at any time*

- Each software element may have different synchronization and performance constraints

  *Some elements are reentrant (it can be safely executed concurrently) objects or software components, while some are not reentrant and there should not be more than one thread executing in it at any time*

  *As in the case of a server element, an element could be invoked only by a limited number of other elements at execution time, or it could be invoked by unlimited number of other elements (Performance issue rise)*

Basic guidelines for mapping runtime elements of a software architecture to their implementations:

- If an element is reentrant, it can be implemented by any thread or process

Basic guidelines for mapping runtime elements of a software architecture to their implementations:

- If an element is reentrant, it can be implemented by any thread or process
  - Reentrant elements are usually more efficient (avoid many synchronization + support shared thread/process pools)

Basic guidelines for mapping runtime elements of a software architecture to their implementations:

- If an element is reentrant, it can be implemented by any thread or process
  - Reentrant elements are usually more efficient (avoid many synchronization + support shared thread/process pools)
  - Business logics may not allow some elements to be reentrant

Basic guidelines (continued)

- If an element is not reentrant and multiple threads or processes may need to communicate with it, it must be run on separate threads or processes so they can be thread-safe

Basic guidelines (continued)

- If an element is not reentrant and multiple threads or processes may need to communicate with it, it must be run on separate threads or processes so they can be thread-safe
- If an element has high multiplicity and its performance is important to the global system performance, use an application server for its implementation

Basic guidelines (continued)

- If an element is not reentrant and multiple threads or processes may need to communicate with it, it must be run on separate threads or processes so they can be thread-safe
- If an element has high multiplicity and its performance is important to the global system performance, use an application server for its implementation
  - It can take advantage of thread and resource pooling, data caching, and dynamic element life cycle management for conserving resources

Basic guidelines (continued)

- If there are heavy computations in the elements for deployment at a particular location, consider using a cluster of processors

Basic guidelines (continued)

- If there are heavy computations in the elements for deployment at a particular location, consider using a cluster of processors
    - The objective in determining the size of the cluster is

Basic guidelines (continued)

- If there are heavy computations in the elements for deployment at a particular location, consider using a cluster of processors
    - The objective in determining the size of the cluster is
        - to balance the computation load

Basic guidelines (continued)

- If there are heavy computations in the elements for deployment at a particular location, consider using a cluster of processors
    - The objective in determining the size of the cluster is
        - to balance the computation load
        - to minimize the total communication traffic

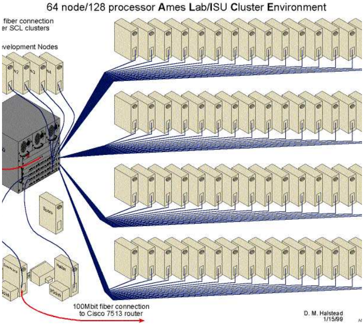



Figure: A cluster of 64 dual-processor Pentium Pros

# Software Architecture Design Space Software Elements

Basic guidelines (continued)

- If an element is assigned well-defined complex functions + similar off-the-shelf software exist + its performance is not critical, then use off-the-shelf solution (more cost-effective, BUT you are responsible for its performance)

Basic guidelines (continued)

- If an element is assigned well-defined complex functions + similar off-the-shelf software exist + its performance is not critical, then use off-the-shelf solution (more cost-effective, BUT you are responsible for its performance)
- A complex element can be expanded into a sub-system with its own elements and connectors

Basic guidelines (continued)

- If an element is assigned well-defined complex functions + similar off-the-shelf software exist + its performance is not critical, then use off-the-shelf solution (more cost-effective, BUT you are responsible for its performance)

- A complex element can be expanded into a sub-system with its own elements and connectors

  - A well-defined interface should be used (encapsulate the sub-system's design and implementation details)

Basic guidelines (continued)

- A complex element can be transformed into a sequence of layered elements

Basic guidelines (continued)

- A complex element can be transformed into a sequence of layered elements
  - Each layer provides a virtual machine or interface to its immediate upper layer element

Basic guidelines (continued)

- A complex element can be transformed into a sequence of layered elements
  - Each layer provides a virtual machine or interface to its immediate upper layer element
  - Each layered element hides away some low-level system details from the upper layers

Basic guidelines (continued)

- A complex element can be transformed into a sequence of tiered elements

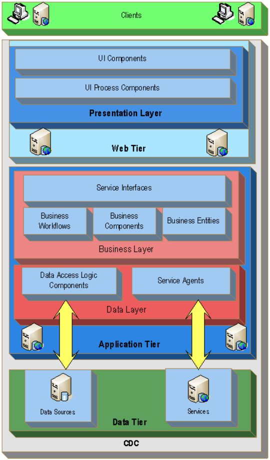
Basic guidelines (continued)

- A complex element can be transformed into a sequence of tiered elements
  - The business logic can be achieved by processing data with a sequence of discrete processing stages

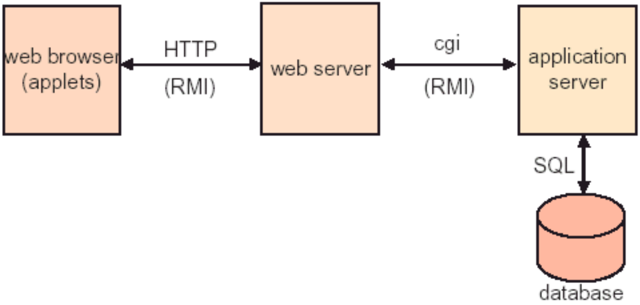
Basic guidelines (continued)

- A complex element can be transformed into a sequence of tiered elements
  - The business logic can be achieved by processing data with a sequence of discrete processing stages
  - The processing stages can be implemented by tiered elements with well-defined interfaces and balanced workloads

## Application of many Layers and Physical Tiers



Example of a multitier architecture



- In the most abstract form, a connector just indicates the necessity during system execution time for one of the elements to send a message to another element and potentially get some return message

- In the most abstract form, a connector just indicates the necessity during system execution time for one of the elements to send a message to another element and potentially get some return message

- Refinement of the software architecture:

- In the most abstract form, a connector just indicates the necessity during system execution time for one of the elements to send a message to another element and potentially get some return message
- Refinement of the software architecture:
  - if the two elements are mapped to the single process, the connector could be mapped to a local method invocation

- In the most abstract form, a connector just indicates the necessity during system execution time for one of the elements to send a message to another element and potentially get some return message

- Refinement of the software architecture:
  - if the two elements are mapped to the single process, the connector could be mapped to a local method invocation
  - If the two elements are mapped to two different processes on the same computer, then the connector could be mapped to a local message queue or an operating system pipe

- In the most abstract form, a connector just indicates the necessity during system execution time for one of the elements to send a message to another element and potentially get some return message
- Refinement of the software architecture:
  - if the two elements are mapped to the single process, the connector could be mapped to a local method invocation
  - If the two elements are mapped to two different processes on the same computer, then the connector could be mapped to a local message queue or an operating system pipe
  - If the two elements are mapped to two different computers, then remote method invocation or Web service invocation could be used

- Software connectors can be classified based on many types of connector attributes

- Software connectors can be classified based on many types of connector attributes
    - synchronization mode

- Software connectors can be classified based on many types of connector attributes
  - synchronization mode
  - initiator

- Software connectors can be classified based on many types of connector attributes
  - synchronization mode
  - initiator
  - implementation type

- Software connectors can be classified based on many types of connector attributes
  - synchronization mode
  - initiator
  - implementation type
  - active time

- Software connectors can be classified based on many types of connector attributes
  - synchronization mode
  - initiator
  - implementation type
  - active time
  - span

- Software connectors can be classified based on many types of connector attributes
  - synchronization mode
  - initiator
  - implementation type
  - active time
  - span
  - fan-out

- Software connectors can be classified based on many types of connector attributes
  - synchronization mode
  - initiator
  - implementation type
  - active time
  - span
  - fan-out
  - information carrier

- Software connectors can be classified based on many types of connector attributes
  - synchronization mode
  - initiator
  - implementation type
  - active time
  - span
  - fan-out
  - information carrier
  - environment

- Software connectors can be classified based on many types of connector attributes
    - synchronization mode
    - initiator
    - implementation type
    - active time
    - span
    - fan-out
    - information carrier
    - environment
    - etc.

- Software connectors can be classified based on many types of connector attributes
  - synchronization mode
  - initiator
  - implementation type
  - active time
  - span
  - fan-out
  - information carrier
  - environment
  - etc.
- Synchronization mode perspective:

- Software connectors can be classified based on many types of connector attributes
  - synchronization mode
  - initiator
  - implementation type
  - active time
  - span
  - fan-out
  - information carrier
  - environment
  - etc.
- Synchronization mode perspective:
  - Blocking connectors

- Software connectors can be classified based on many types of connector attributes
  - synchronization mode
  - initiator
  - implementation type
  - active time
  - span
  - fan-out
  - information carrier
  - environment
  - etc.
- Synchronization mode perspective:
  - Blocking connectors
  - Non-blocking connectors

An initiator is an incident element of a connector that can make a request to its partner

- Connector's initiator perspective:

An initiator is an incident element of a connector that can make a request to its partner

- Connector's initiator perspective:
  - One-initiator connectors

An initiator is an incident element of a connector that can make a request to its partner

- Connector's initiator perspective:
  - One-initiator connectors
  - Two-initiator connectors

An initiator is an incident element of a connector that can make a request to its partner

- Connector's initiator perspective:
  - One-initiator connectors
  - Two-initiator connectors
- For a system to support callback between its two sub-systems, the two sub-systems must be connected by a two-initiator connector

- Connector information carrier perspective:

- Connector information carrier perspective:
  - Variable (Two threads in a same process)

- Connector information carrier perspective:
  - Variable (Two threads in a same process)
  - Environment Resource (register, pipes, files or local message queues)

- Connector information carrier perspective:
  - Variable (Two threads in a same process)
  - Environment Resource (register, pipes, files or local message queues)
  - Method invocation and Message

- Connector implementation type perspective:

SFWR ENG 3A04: Software Design II

Dr. R. Khedri

Introduction

Software Code Structure

Software Runtime Structure

Software Management Structure

Software Elements

Software Connectors

Iterative Refinement of an Architecture

- Connector implementation type perspective:
  - Signature-based (method call: object identifier, method name, arguments) can implement one type of operation

- Connector implementation type perspective:
  - Signature-based (method call: object identifier, method name, arguments) can implement one type of operation
  - Protocol-based can implement multiple operation types (e.g., HTTP protocol)

- Connector implementation type perspective:
  - Signature-based (method call: object identifier, method name, arguments) can implement one type of operation
  - Protocol-based can implement multiple operation types (e.g., HTTP protocol)
- Connective active time perspective:

- Connector implementation type perspective:
  - Signature-based (method call: object identifier, method name, arguments) can implement one type of operation
  - Protocol-based can implement multiple operation types (e.g., HTTP protocol)
- Connective active time perspective:
  - Programmed (a method call will be made at a time specified at programming time)

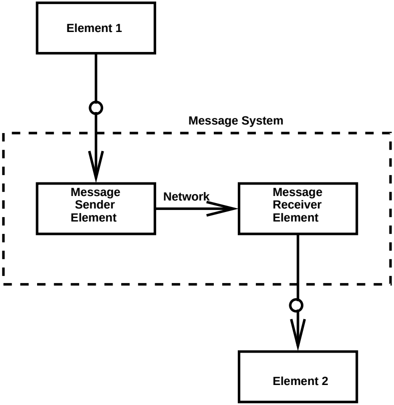
- Connector implementation type perspective:
  - Signature-based (method call: object identifier, method name, arguments) can implement one type of operation
  - Protocol-based can implement multiple operation types (e.g., HTTP protocol)
- Connective active time perspective:
  - Programmed (a method call will be made at a time specified at programming time)
  - Event-driven (reactive systems, GUIs, automata)

- Connector implementation type perspective:
  - Signature-based (method call: object identifier, method name, arguments) can implement one type of operation
  - Protocol-based can implement multiple operation types (e.g., HTTP protocol)
- Connective active time perspective:
  - Programmed (a method call will be made at a time specified at programming time)
  - Event-driven (reactive systems, GUIs, automata)
- Connective Span perspective:

- Connector implementation type perspective:
  - Signature-based (method call: object identifier, method name, arguments) can implement one type of operation
  - Protocol-based can implement multiple operation types (e.g., HTTP protocol)
- Connective active time perspective:
  - Programmed (a method call will be made at a time specified at programming time)
  - Event-driven (reactive systems, GUIs, automata)
- Connective Span perspective:
  - Local (incident elements are located in the same processor)

- Connector implementation type perspective:
  - Signature-based (method call: object identifier, method name, arguments) can implement one type of operation
  - Protocol-based can implement multiple operation types (e.g., HTTP protocol)
- Connective active time perspective:
  - Programmed (a method call will be made at a time specified at programming time)
  - Event-driven (reactive systems, GUIs, automata)
- Connective Span perspective:
  - Local (incident elements are located in the same processor)
  - Network (are normally implemented with the proxy design pattern)

Connector, information carrier, Implementation type, span



# Software Architecture Design Space Software Connectors

- Connector fan-out perspective:

- Connector fan-out perspective:
  - 1-1 (one-to-one)

- Connector fan-out perspective:
  - 1-1 (one-to-one)
  - 1-* (one-to-many): have important impacts on the connector's implementation technology and performance

- Connector fan-out perspective:
  - 1-1 (one-to-one)
  - 1-* (one-to-many): have important impacts on the connector's implementation technology and performance
- Connector environment perspective:

- Connector fan-out perspective:
  - 1-1 (one-to-one)
  - 1-* (one-to-many): have important impacts on the connector's implementation technology and performance
- Connector environment perspective:
  - Homogeneous (same programming language and software framework and run on the same operating system)

- Connector fan-out perspective:
  - 1-1 (one-to-one)
  - 1-* (one-to-many): have important impacts on the connector's implementation technology and performance
- Connector environment perspective:
  - Homogeneous (same programming language and software framework and run on the same operating system)
  - Heterogeneous

- Given a project specification, an abstract high-level software architecture will first be proposed (elements + connectors)

- Given a project specification, an abstract high-level software architecture will first be proposed (elements + connectors)

- The architecture will then go through multiple refinement processes to support particular deployment constraints

 Iterative Refinement of an Architecture

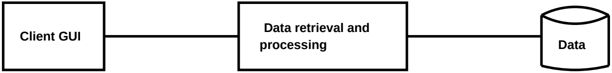
- Given a project specification, an abstract high-level software architecture will first be proposed (elements + connectors)

- The architecture will then go through multiple refinement processes to support particular deployment constraints

- Delay binding of software connectors for more flexible implementation decisions for software connectors

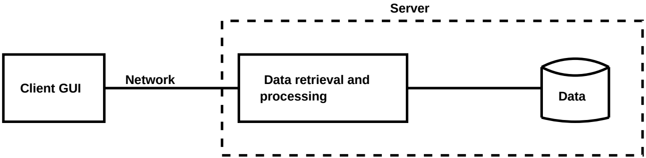
- Given a project specification, an abstract high-level software architecture will first be proposed (elements + connectors)
- The architecture will then go through multiple refinement processes to support particular deployment constraints
- Delay binding of software connectors for more flexible implementation decisions for software connectors
- Seamless integration of multiple software architectural styles in realizing different subsystems or different architectural levels of the same system

Step 1 (standalone):



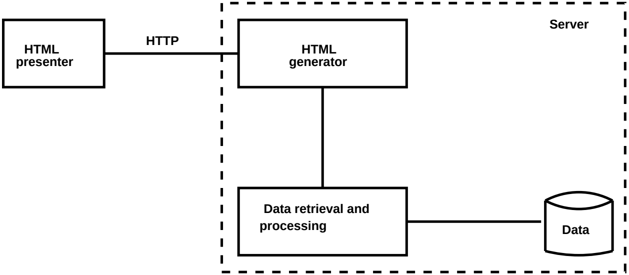

Standalone data presenter

Step 2 (network):




**Networked data presenter**

Step 3 (HTML and HTTP based):




HTML and HTTP based data presenter

Step 4 (Layered Architecture):