Execution Architecture
Software Architecture VO/KU (707.023/707.024)

Denis Helic
KMI, TU Graz

Nov 7, 2012
Outline

1. Definition
2. Design
3. Stereotypes
4. Detailed Design
5. Connecting conceptual and execution views
6. Behavior
7. Execution on the Web
Execution View

- Focuses on the system runtime structure
- Hardware elements, subsystems, processes and threads
- Suited for examining quality attributes, most-notably runtime attributes
- E.g. performance, security, usability, ...
- But also e.g. scalability
- Similarly to conceptual architecture comprised of components and connectors
Components in execution architecture

- **Concurrent** components (abstraction created by execution of a software program)
- If the system is a single-computer, single-process, single-thread system then the execution architecture is very simple

**Figure:** The simplest execution architecture
Components in execution architecture

Thus, execution architecture is needed for distributed, concurrent systems.

Nowadays, huge majority of systems comes into this category.

E.g. network-based systems.

E.g. multi-processor systems (multi-core), sometimes abstraction through OS.

E.g. multi-threaded systems - GUI systems belong here as well (event-thread).
Components in execution architecture

- Hardware (we want investigate this in detail) - only boundaries
- Concurrent subsystems - typically has a dedicated computer that runs it
- Processes - an OS process, runs on a single computer and has its own memory space
- Threads - an OS (or Java) thread - executes concurrently with other threads within the memory space of a parent process
- Typically, we have different execution models depending on granularity
Connectors in execution architecture

- Connectors indicate that one component calls another
- The arrow depicts the call direction
- The arrow head points from the calling component to the called component
- Three different types of arrows for three different calling scenarios
Connectors in execution architecture

- **Synchronous communication**: the calling components waits for a response of the called component
- **Asynchronous communication**: the calling component does not wait for a response
- **Callback**: The calling component receives a response asynchronously by setting a means by which the called component can send response later
Connectors in execution architecture

(a) Synchronous call
(b) Asynchronous call
(c) Callback

Figure: Execution connectors from Software Architecture Primer
**Execution architecture: Example**

*Figure: Example of execution architecture from Software Architecture Primer*
## Conceptual vs. Execution arch.

<table>
<thead>
<tr>
<th>Element</th>
<th>Conceptual</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
<td>Domain-level responsibilities</td>
<td>Unit of concurrent activity</td>
</tr>
<tr>
<td>Connectors</td>
<td>Information flow</td>
<td>Invocation</td>
</tr>
<tr>
<td>Views</td>
<td>Single</td>
<td>Multiple</td>
</tr>
</tbody>
</table>
Conceptual vs. Execution arch.

**Figure:** Conceptual vs. execution from Software Architecture Primer
Here we design a number of different models
Some of them will include physical components, i.e. hardware
Each model is a model at a specific level of granularity
Less details: concurrent subsystems, processes
More details: threads
Concurrent subsystems model

- Top-level execution model
- To get an overview of the running system
- Subsystems can be quite complex and have many processes and/or threads
- However, a concurrent subsystem is not something that is clearly defined
- In some cases it is clear
Concurrent subsystems model

**Figure:** Client-server execution architecture: subsystems
Concurrent subsystems model

- A large number of similar processes should be treated as a single unit
- E.g. the indexing system of a search engine
- A lot of processes there but logically they belong to the same unit
- Crawler, parsers, analyzers, index updating, ...
Concurrent subsystems model

- A process that has a high degree of internal concurrency (threads) should be treated as a concurrent subsystem.
- E.g. a server is typically a single process but might create threads to handle client requests.
- Existing systems are best treated as concurrent subsystems.
- E.g. a file server.
Concurrent subsystems model

- A concurrent subsystem is always long-lived
- Created when the systems is started
- Closed when the system is shutdown
- Operates throughout the system lifetime
Restricting a concurrency model to processes depicts the execution structure.

Basically, you examine each concurrent subsystem for processes.

You do not go into details on external systems.

Typically, such models will be only slightly more detailed than concurrent subsystem models.
Process model

Figure: Client-server execution architecture: processes
Execution stereotypes

- Similarly to conceptual components execution components can belong to stereotypes
- Again we will use three different stereotypes
- Each stereotype has a particular clearly defined semantics
- In execution architecture this semantics describes the kind of concurrent activity
Execution stereotypes

- User-initiated: the component performs action because of user input
- This components are always user-interfaces
- In typical case such components exhibit a certain amount of internal concurrency
- E.g. an event thread that listens to user-input events
- Enhances responsiveness of user-interface, and in general usability
Execution stereotypes

- **Active**: the component generates activity internally
- E.g. components loop continuously or wake up periodically
- Typical for real-time portions of the system
- E.g. a crawler in a search engine might be an active component
- Whenever there is a new page it generates activity and invokes parser, analyzer, indexer, ...
Execution stereotypes

- Services: the component waits for requests of other components and generates responses for such requests.
- Typically performs a complex task and has clearly defined protocol for communication with other components.
- E.g. database, web, file servers.
- In most cases services are concurrent subsystems that exhibit a large amount of internal concurrency.
Execution stereotypes

Figure: Execution stereotypes from Software Architecture Primer
We start first with the big picture: concurrent systems
Our system is a Web application
What concurrent subsystems do we have?
Obviously: a Web browser and a Web server
The (part of) application logic is on the server side
We need a Web server which can run applications
Thus, the Web server is actually a Web application server
Additionally, we have an external system - another concurrent subsystem
Sample Execution Architecture

Figure: Concurrent subsystem execution architecture
Sample Execution Architecture

Figure: Concurrent subsystem execution architecture
Sample Execution Architecture

Figure: Concurrent subsystem execution architecture
Detailed execution model

- Includes processes and threads
- Threads do not have their own memory space nor they have their own copy of the code in memory
- The code is loaded only once by their parent process
- The memory is typically shared by the threads
- We need to take care about thread synchronization
Quality attributes

- Many quality attributes are addressed by the execution architecture.
- E.g. usability in GUIs is addressed by a special event thread.
- E.g. a highly reliable component needs a separate execution component.
- E.g. security typically requires a separate execution component.
Example 1: GUI Event Thread

- We have a multimedia player, e.g. it executes an animation but the GUI needs to be responsive
- Which threads do we have?
- How do they communicate?
Example 2: Web server cache with dynamic content

- E.g. a wiki system where users edit content
- In cache you have all documents + valid/invalid flag
- If valid serve from cache
- If invalid: reload in cache, set valid flag
- How is caching executed?
Example 3: Web server cache with a database server

- E.g. a content management system with content stored in a database
- What happens when a field is updated in the database?
- Note, database server is a separate process
- Which threads do we have on the Web server side?
Sources of concurrency

- Forks in a use-case map require concurrent activities
- Components that perform significant amounts of computation are best modeled as concurrent activities (usability, performance)
- Network components are also concurrent activities (usability, performance)
Sample Detailed Execution Architecture

Figure: Detailed execution architecture
Sample Detailed Execution Architecture

**Figure:** Detailed execution architecture
Sample Detailed Execution Architecture

Figure: Detailed execution architecture
Connecting conceptual and execution views

Binding execution and conceptual models

- We need to decide where conceptual components reside in the execution architecture.
- Which of them might be in a browser and which of them in logic component on the server.
- No rules for this!
- Depends on the quality attributes we need to satisfy.
- E.g. if performance is needed - if large number of users move some conceptual components to the client.
Binding execution and conceptual models

**Figure:** Binding conceptual and execution architecture
Connecting conceptual and execution views

Binding execution and conceptual models

![Diagram showing the binding of conceptual and execution models](image)

**Figure**: Binding conceptual and execution architecture
Connecting conceptual and execution views

Binding execution and conceptual models

Figure: Binding conceptual and execution architecture
Execution behavior

- We will use-case maps to model behavior
- Actually we need only to verify that execution architecture supports the desired behavior
- We can use the same use-case maps from the conceptual architecture
Execution behavior

Figure: Execution behavior
Execution behavior

Figure: Execution behavior
Concurrency on the Web

- We had concurrency on the server side
- With introduction of AJAX it is possible to have concurrency in a browser
- You can communicate asynchronously with the server
- If you decide to do so then you have to think about the updating strategy
Concurrency on the Web

- How do we communicate asynchronously HTTP
- There is no native HTTP support for asynchronous communication
- You have to simulate this
- Typically by polling
- HTML5 introduces WebSockets API
Concurrency on the Web

Figure: Asynchronous communication server/client