

## 48433 Software Architecture

### Tutorial 4 – Execution architecture

In this tutorial, you are learning how to think of software architecture from an execution architecture perspective, particularly in terms of processes and concurrency, and how to represent this in a process view diagram and use-case maps. You are also practising how to link these representations back to runtime quality attributes, such as performance, reliability and security.

#### 1. Process view

Construct a process view by using the following procedure, derived from the textbook.

1. Identify concurrent components by considering the following things.  
Don't worry at this stage if you mix up processes and concurrent subsystems – your diagram can have both.
  - Off-the-shelf components.
    - Existing (“legacy”) systems
    - Commercial off-the-shelf subsystems, eg. web servers, application frameworks, databases
    - Re-usable in-house components.
  - Runtime quality attributes. Components with different quality needs should not be in the same subsystem. Here are some triggers for creating concurrent activities:
    - Reliability/criticality: Isolate these components in their own subsystem
    - Performance: especially real-time requirements
    - Security: Isolation of components; security-specific components
    - Usability: May separate presentation components to improve user experience. May need to support diverse user populations. Some presentation components may be graphic or data intensive.
  - Other factors to consider:
    - Development and organisational constraints: Create a concurrent component for ease of development.
  - Indicators of concurrent activity
    - Forks in a use-case map: Examine the set of use-case maps you produced for the conceptual architecture.
    - Components that perform significant amounts of work.
    - Components that have to wait for “the real world” – hardware systems, networks, user input.
    - Components that are known to require separate hardware.
  - Lastly, opposing forces. These are reasons for *not* creating concurrent components.
    - Performance liabilities. Context-switching between concurrent activities and communication between concurrent subsystems incur overhead.
    - Complex interaction between concurrent activities. Can result in hard to maintain and less reliable systems.
2. Identify connections between concurrent components.
3. Add stereotypes (user-initiated, active, service), where appropriate.

## 2. Visualising behaviour with use-case maps

Using the same set of events that you worked with in the previous tutorial for exploring the behaviour of the conceptual architecture, draw use-case maps to explore the execution architecture in terms of its quality attributes.

1. Take two or three events and construct use-case maps for those events on the same execution architecture diagram. Make sure you
  - a. Label each event
  - b. Consider using the AND, OR, SEQ forks, to model the flow of activity across components.
2. Reflect on the set of use-case maps you have just produced and identify issues of concurrency
  - a. What is the nature of each activity triggered by a specific event? (eg. user or real-time)
  - b. Do the traces expose any components that are problematic in terms of quality attributes?
  - c. Do you need to add, split or replicate components?
3. Re-factor the architecture to improve its quality attributes.
4. Supplement the diagrams with written explanations of what each use-case map demonstrates, and your refactoring decisions.
5. Repeat this procedure for the remaining events. You may add other significant events.
6. Walk through one of your use-case maps on the board to the whole class. The rest of the class is encouraged to question and comment on your artefacts

### For next time

Explore the non-runtime quality attributes, MeTRiCS by extending your application of use-case maps to lifecycle events.