CSE 601: Distributed Systems

Toukir Ahammed

Architectural Styles

Architectural Styles

Important styles of architecture for distributed systems:

- Layered architectures
- Object-based architectures
- Data-centered architectures
- Event-based architectures

Architectural Styles

Resource-based Architectures

Resource-based Architectures

- View a distributed system as a huge collection of resources that are individually managed by components
- Resources may be *added* or *removed* by (remote) applications, and likewise can be *retrieved* or *modified*
- This approach has now been widely adopted for the Web and is known as **Representational State Transfer (REST)**
- REST is an architectural style based on transferring representations of resources from a server to a client

Resource-based Architectures

There are four key characteristics of RESTful architectures:

1. Resources are identified through a single naming scheme
2. All services offer the same interface, consisting of at most four operations
3. Messages sent to or from a service are fully self-described
4. After executing an operation at a service, that component forgets everything about the caller (stateless execution)

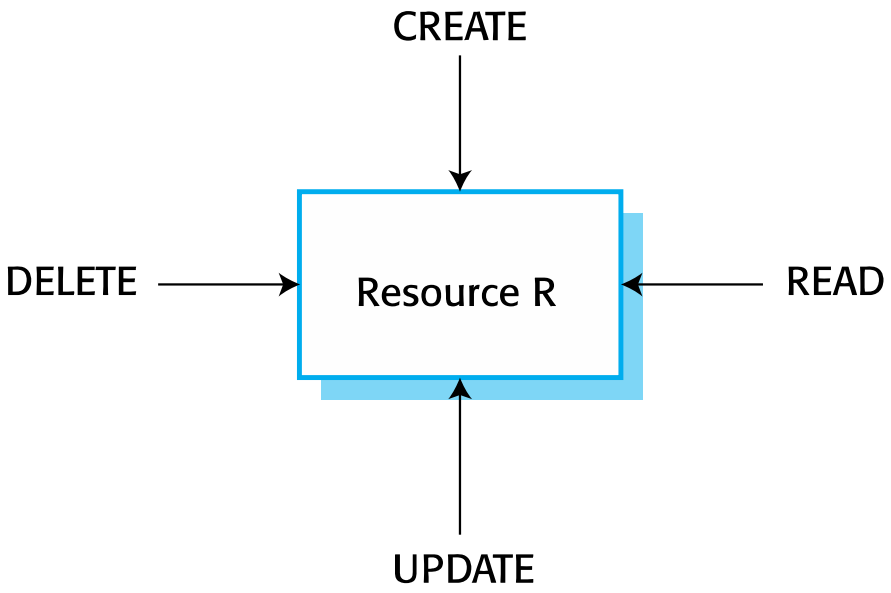
Resource-based Architectures

Basic Operations

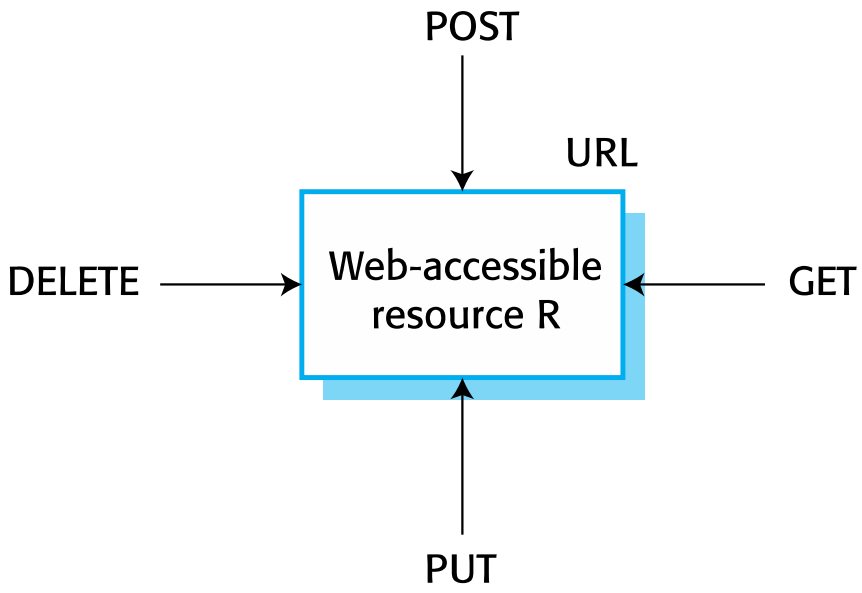
Operation	Description
PUT	Create a new resource
GET	Retrieve the state of a resource in some representation
DELETE	Delete a resource
POST	Modify a resource by transferring a new state

Resource-based Architectures

Basic Operations



(a) General resource actions



(b) Web resources

Resource-based Architectures

Example: Amazon's Simple Storage Service (Amazon S3)

- *Objects* (i.e., files) are placed into *buckets* (i.e., directories)
- *Buckets* cannot be placed into buckets
- Operations on *ObjectName* in bucket *BucketName* require the following identifier: https://s3.amazonaws.com/BucketName/ObjectName

Typical operations

All operations are carried out by sending HTTP requests:

- Create a bucket/object: POST, along with the URI
- Listing objects: GET on a bucket name
- Reading an object: GET on a full URI

Architectural Styles

Event based architectures

Event based Architectures

- As systems continue to grow and processes can more easily join or leave, it becomes important to have an architecture in which dependencies between processes become as loose as possible
- An architecture is needed in which there is a strong separation between *processing* and *coordination*
- *Coordination* can be discussed from two dimensions: temporal and referential.

Event based Architectures

- **Temporal coupling** means that processes that are communicating will both have to be up and running.
- **Referential coupling** generally appears in the form of explicit referencing in communication.
  - For example, a process can communicate only if it knows the name or identifier of the other processes it wants to exchange information with.

Event based Architectures

	Temporally coupled	**Temporally decoupled**
Referentially coupled	Direct	Mailbox
Referentially decoupled	Event-based	Shared data space

Event based Architectures

- **Direct coordination**: when processes are temporally and referentially coupled, coordination takes place directly.
  - In real life, talking over cell phones (and assuming that a cell phone has only one owner), is an example of direct communication.
- **Mailbox coordination**: when processes are temporally decoupled, but referentially coupled
  - there is no need for two communicating processes to be active at the same time to let communication take place.
  - Instead, communication takes place by putting messages in a (possibly shared) mailbox

Event based Architectures

- **Event-based coordination**: The combination of referentially decoupled and temporally coupled systems
  - processes do not know each other explicitly
  - a process **publish** a notification describing the occurrence of an event
  - processes may **subscribe** to a specific kind of notification
  - it is generally required that the subscriber is up-and-running at the time the
  - notification was published

Event based Architectures

- **Shared data space**: the combination of referentially and temporally decoupled processes
  - The key idea is that processes communicate entirely through tuples, which are structured data records consisting of several fields, very similar to a row in a database table
  - Processes can put any type of tuple into the shared data space.
  - To retrieve a tuple, a process provides a search pattern that is matched against the tuples. Any tuple that matches is returned
  - A process subscribes to certain tuples by providing a search pattern; when a process inserts a tuple into the data space, matching subscribers are notified

Publish-subscribe architectures