Supporting the Evolution of Service Oriented Web Applications using Design Patterns

Manolis Tzagarakis, Computer Technology Institute, Greece

Michalis Vaitis, University of the Aegean, Greece

Nikos Karousos, University of Patras, Greece

Outline

- Web Applications
- **S**ervice **O**riented **A**rchitectures (SOA)
- Issues in building SOA based Web Applications
- Design Patterns for integrating and evolving services in Web Applications
- Conclusions

Web Applications

- a way of delivering content and services using internet technology
  - HTTP, XML, HTML, Web Services
- Multidisciplinary
  - requires a diverse range of technologies and involves diverse concerns
- One class of Web Applications deals with the interaction of Web based and information systems.

Web Applications

- Still developed in a rather ad hoc manner causing many problems
  - not what the user wanted
  - not **maintainable** or **scalable and can't evolve**
  - short "useful life"
  - lack of performance and security

"Web systems that are kept running via continual stream of Patches or upgrades developed without systematic approaches" (Dart, 2001)

Web Applications

- The problem gets even more complicated, if a Web Applications need to be build on a **Service Oriented Architecture (SOA).**
- Traditional **client-server architectures** have already been introduced in Web Application.
  - Methodologies/Methods and tools to build uponsuch architectures are already present
- Yet, these approaches **are inappropriate for Service Oriented Architectures.**

Web Applications

- The **SOA Context**: **Why SOA?**
- A way of delivering functionality using services: a form of computations that are autonomous accomplishing a specific task
  - **Reusable** across Web Applications
  - Can be **queried and negotiated**
  - Are **discoverable**
  - Are **composable**
    - Creating **complex services** from **simpler ones**
  - Fine-grained apporach to providing functionality

Service Oriented Architecture

- How does SOA **differ** from traditional CS?
  - This **impacts** the development Web apps

Service Oriented Architecture

Client-Server	Service Oriented
Early Binding	**Late** Binding
Domestic (evolves smoothly and planned)	**Feral** (evolve abrupt and uncontrolled)
Location **dependent**	Location **independent**
Single interface	**Multiple Interfaces**
Development oriented	**Integration** oriented
Tightly coupled	**Loosely** coupled
Monolithic	Composable
Stable	Unstable

Different context requires different approaches for developing Web Applications

Case study: Providing Hypermedia Services to Web Applications

- Development based on Callimachus
  - A CB-OHS
  - Service oriented
- Provides a number of **generic hypermedia services** (in the form of components)
  - **Taxonomic component**, used to provide taxonomic services
    - E.g. such as in directory services, e.g. openCategory, getCategoryPath, getChildren

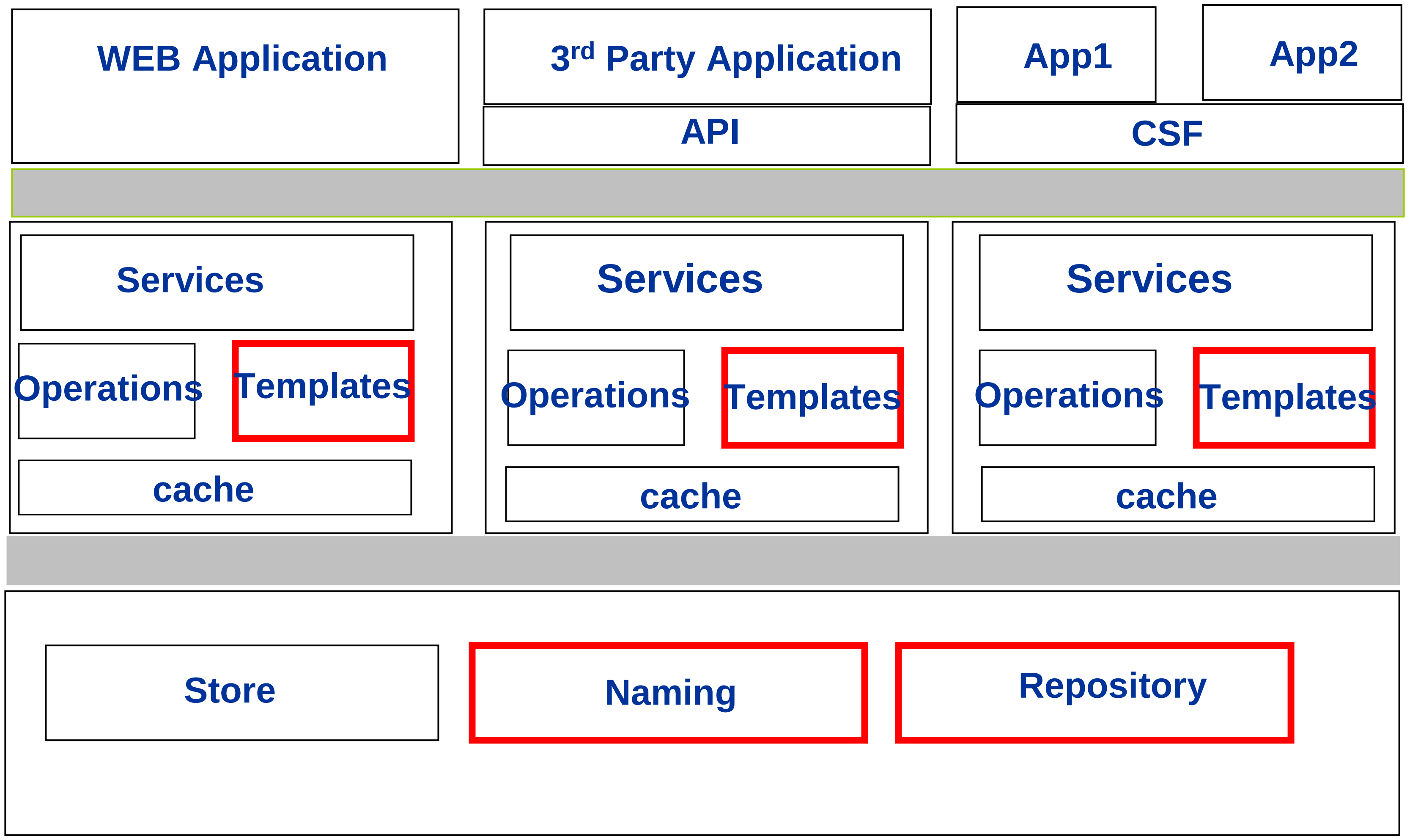
Hypermedia Services

- Provides a number of hypermedia services (in the form of components)
  - **Navigational component**, provides navigational services
    - Create/Traverse link
    - Create/open node and anchor
- Does **not store content/data**
  - Handled by the content/data services

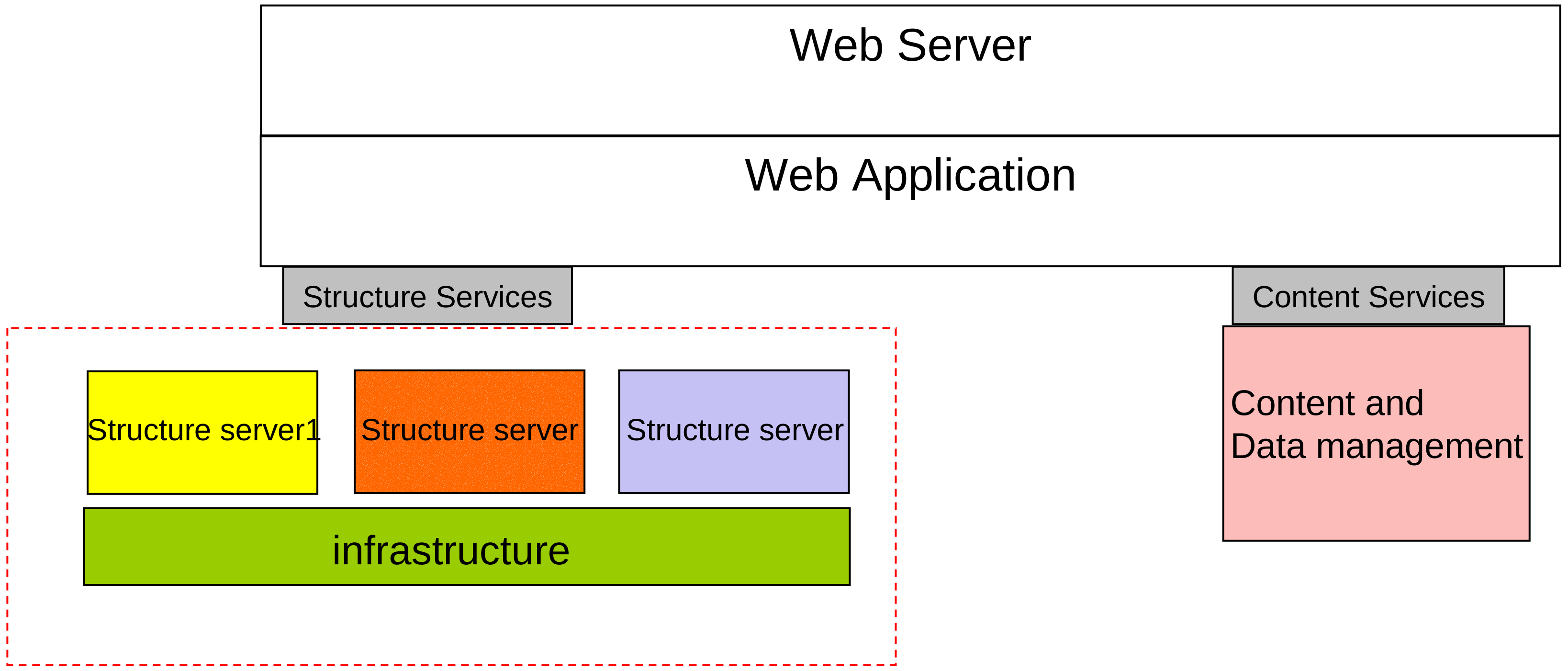
Hypermedia Services

- Services are **available through servers** (structure servers)
  - TCP/IP daemons listening on port
  - Support one or more protocols (XML based)
  - Can relocate,
- Clients
  - Can be third party applications (e.g. MS Word)
  - Custom applications
  - Integrated with **Web Applications**

Callimachus Architecture: Providing hypermedia services



Hypermedia Services in Web Applications



Example message from to structure server

```
POST /executeOperation HTTP/1.1
Content-Type: NavProtocol v1.2
Content-Length: 540
User-Agent: Callimachus MS-Office plugin v2.4

<?xml version=”1.0”?>
<!DOCTYPE np.xml>
<NavProt version=1.2>
    <NPMessageHeader>
          <Host>150.140.18.219</ Host >
          <Agent>Callimachus MS-Office plugin v2.1</Agent >
          <SessionID>0x562AAA2222</SessionID>
          <Operarion>OpenNode</Operation>
          <Request Time>2/3/2003 11:08:52</ Request Time>
     </NPMessageHeader>
     <NPMessageBody>
           <NPOpenNodeRequest>
                      <Node>
                         <NodeName>TestNode</NodeName>
                      </Node >
           </NPOpenNodeRequest>
      </NPMessageBody>
</NavProt>
```

Web applications using hypermedia services

- Mediate between user/browser and services
- At the Web Application layer, **hypermedia services** and **content services** are invoked
  - E.g. **openCategory/getPathOfCategory** for directory services
  - XML containing a list of **id’s or content references** is retrieved and resolved
- At the Web Application layer, structure and content are **merged and/or transformed** into the appropriate format
  - HTML, XML etc.

Development methodology

- Callimachus supports rapid prototyping approach of service provision
  - **Evolutionary** prototyping (**not throw-away**)
  - Short release cycles
  - Many releases
- The services are then **integrated** with the web application
- **Constant evolution** of services

Issues encountered

- Constant evolution of services (cont.)
  - Methods **change**
    - New are added, existing are removed to meet the requirements of the Web Application
  - Components **evolve**
    - Support for new protocols
      - E.g. as indicated by web developer or the Web Application Layer
- How can a **systematic approach** to such kinds of evolution be achieved ?
  - ensuring rapid prototyping

Design Patterns

- Design patterns are used support the **smooth evolution** of hypermedia services and their integration into Web Applications
  - In particular
    - support changes at hypermedia services layer due to new Web App needs
    - Support changes at the Web Application layer due to changes in backend services
- Types of patterns with respect to services
  - At the **hypermedia service level**
  - At the **Web Application level**

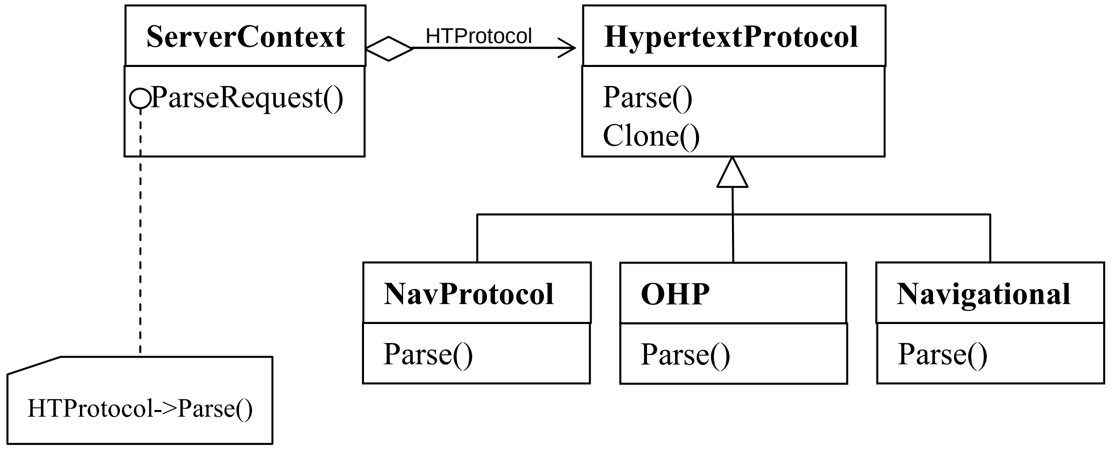
Design Patterns at the hypermedia service level

- **Problem:** different web applications might invoke services from the same component using different protocols.
- How can the same hypermedia services be available through different protocols without rewriting these services?
- Two issues here
  - Parsing requests for services
    - Different protocols need different parsing algorihtms
  - Invoke the appropriate method based on the specified operation in the protocol

Design Patterns at the hypermedia service level

- **Protocol Parser**
  - **Decouples** parsing request from **invoking** the method/operation
  - Based on the Strategy and prototype patterns (GoF)
  - Allows the parsing algorithm to vary according the incoming request
    - The appropriate algorithm **can be determined** at runtime.

Pattern structure



Benefit: New protocols can easily be plugged in and are available at runtime, without the need for recompilation.

Selection of the appropriate protocol is done **based on the HTTP Content-Type Header value**

Creation and registration of protocol implementations

```
class hypertextProtocolFactory {
  private:
      hash_map<const char *, hypertextProtocol*> htProtocolLibrary;
  protected:
  public:
      hypertextProtocolFactory (){
         registerProtocol( “NavProtocol”, new navProtocol() );
         registerProtocol( “OHP”, new OHP() );
         registerProtocol( “Navigational”, new Navigational() );
      }
     ~hypertextProtocolFactory(){};

    //Registers a new protocol handler
    int registerProtocol(char *pName, hypertextProtocol *ht);
    // Searches the library and
    // returns the appropriate protocol handler. Calls the Clone
    // method of protocol handler objects
    hypertextProtocol *getProtocol (char *pName);
} // hypertextProtocolFactory
```

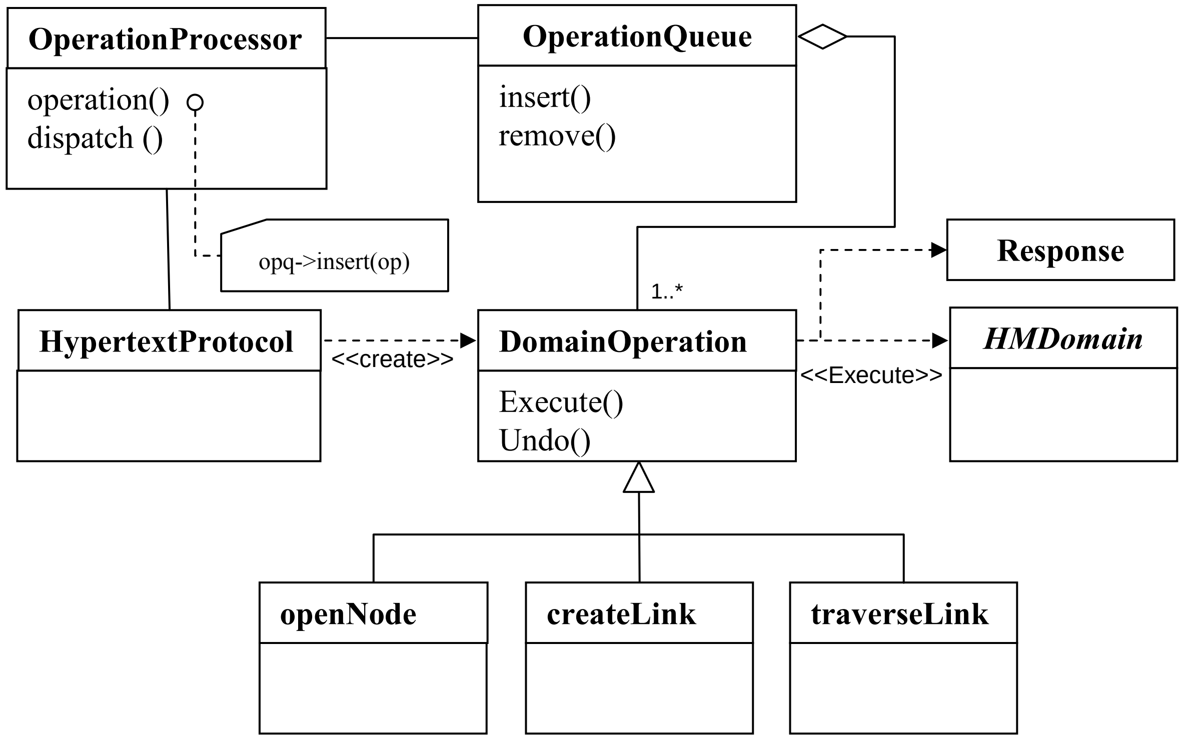
Design Patterns at the hypermedia service level

- **Problem:** a web application might require new services to be available at the hypermedia level.
- How can new services easily and systematically be developed and made available?
  - E.g. service that moves a “subtree” from one category to another
  - In a **plug and play** fashion

Design Patterns at the hypermedia service level

- **Service execution framework**
  - **Decouples** service **invocation** from service **execution**
  - Based on the active object and command processor patterns (GoF)
  - Allows to be determined if an invocation can be executed.
  - Allows also advanced functionalities such as
    - Queuing, logging, scheduling of service invocations is possible

Pattern structure



Service invocations are modeled as separate objects
These objects can be queues, logged and scheduled
e.g. priority scheduling

Design Patterns at the Web Application level

- Deal mainly with **invoking the hypermedia** (and content services)
- Main purpose is to facilitate service invocation
  - Provide single access point for requests that originate from users and/or other apps
  - Offer generic templates to **re-occurring invocation schemes**

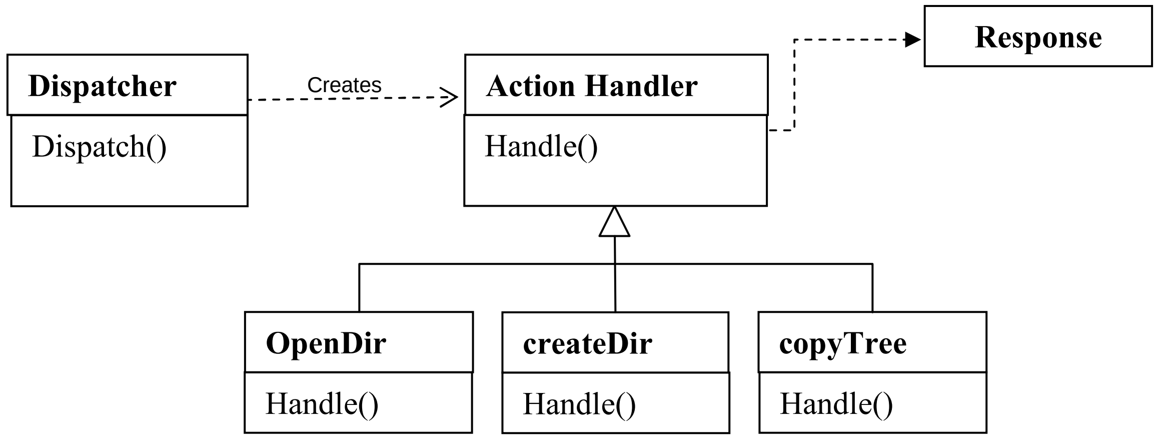
Design Patterns at the hypermedia service level

- **Problem**: new code to handle new user requests may be defined in different modules making the code unstructured thus unmaintainable
- How can a **systematic approach** to new user request handlers established?

Design Patterns at the hypermedia service level

- **Action Dispatcher Pattern**
- Select the appropriate action by **dispatching centrally all** incoming requests.
- Adding new action handlers can systematically tackled
- Based on Factory and Dispatcher pattern

Pattern Structure

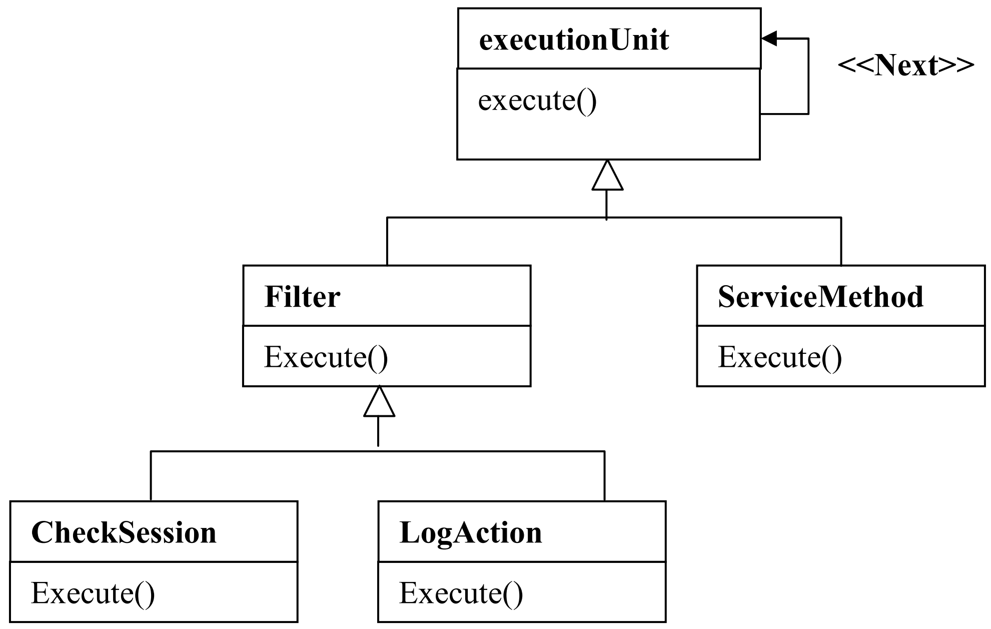


Selection of the specific action is done via an creational pattern (e.g. factory method)

Design Patterns at the hypermedia service level

- **Problem**: during the handling of a user request, a number of operations and/or services need to be invoked sequentially.
  - E.g. prior to request execution, validate the user request through filters (preprocessing)
    - **Filter1 -> Filter2 ->Filter 3**
  - E.g. a number of services need to be invoked sequentially. If one invocation or execution fails, the entire action should fail.

Pattern Structure



Can be configured declaratively (e.g. in a configuration file)
Can be determined at run time.

Conclusions

- Adoption of a Service oriented architecture inherently introduces the problem of evolution of **Web Applications** and the **services** they depend on.
- We have shown how evolution issues are addressed in Web Applications based on Callimachus
- Such evolution issues are addressed using design patterns

Conclusions

- Design patterns facilitate a **smooth and controlled evolution** of web applications and services
- Many more patterns can be identified and introduced to **address more evolution concerns**.