Systematic Development of Complex Web-based User Interfaces

Rikard Boström

Anders Ellvin

Introduction

- Web apps failing in achieving basic SE-stds
- Increasing complexity causing concern about quality
- The UI largely affects the effectiveness and maintainability
- Article addresses systematic development of Web-UIs
- A SE-based approach

Presentation overview

- Modelling User Interface Requirements
  - Functional requirements
  - Complementing requirements.
- DIWA
- DAWID

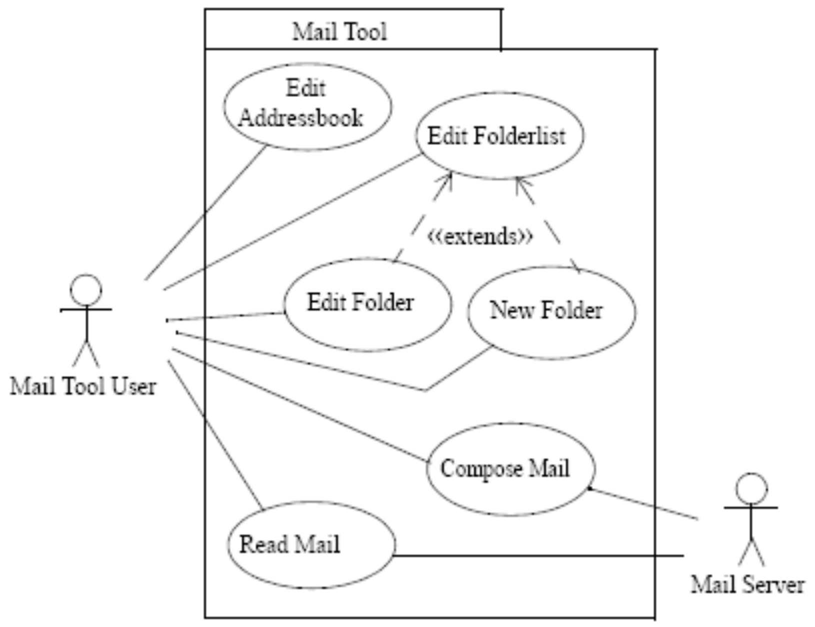
Modelling User Interface Requirements

- Functional requirements
- Fundamental characteristics of UI like static structure, dynamic behavior etc

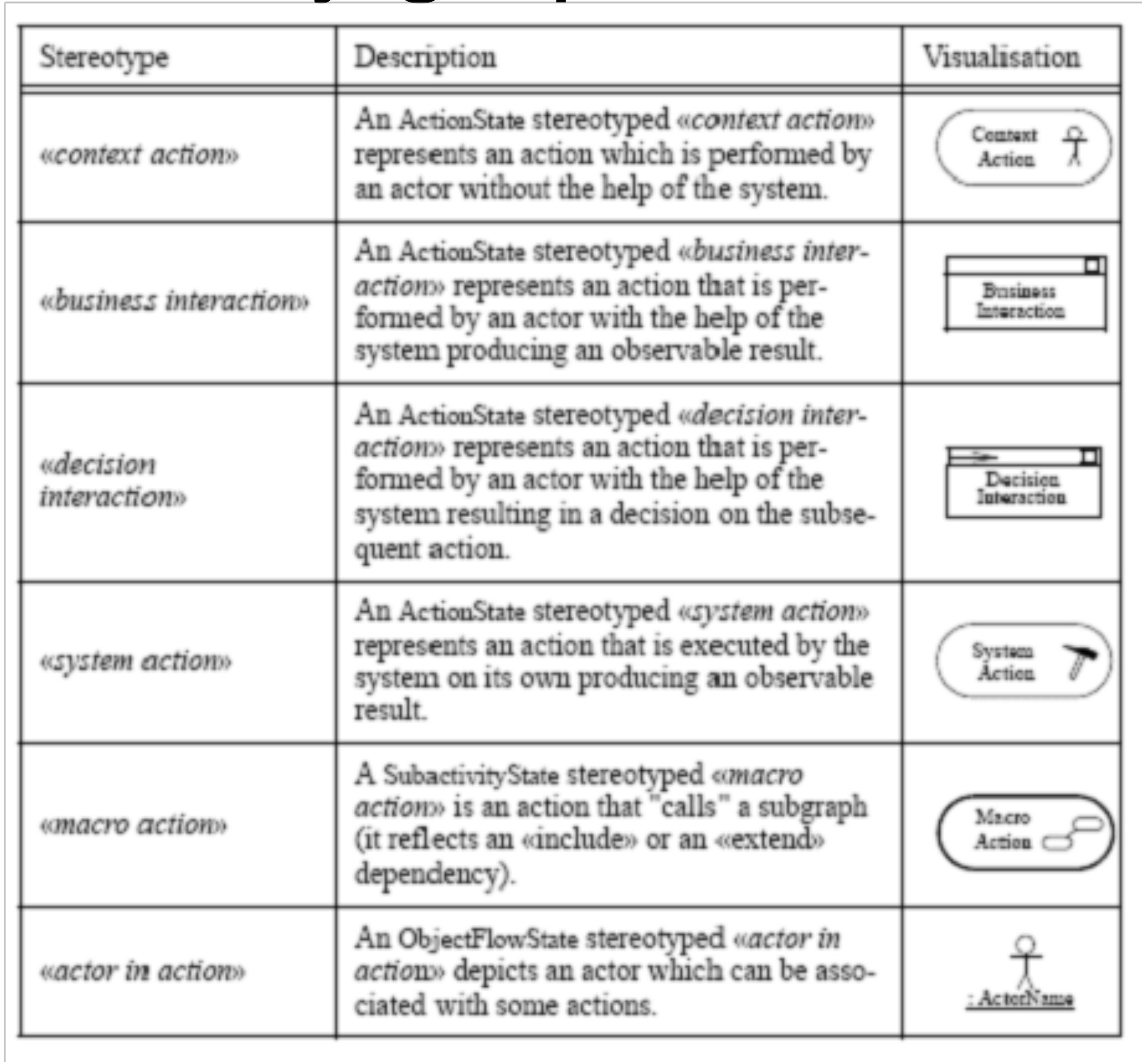
Functional requirements

- Actors
- Use Cases
- Activity Graphs
- Domain class model

Use Case diagram

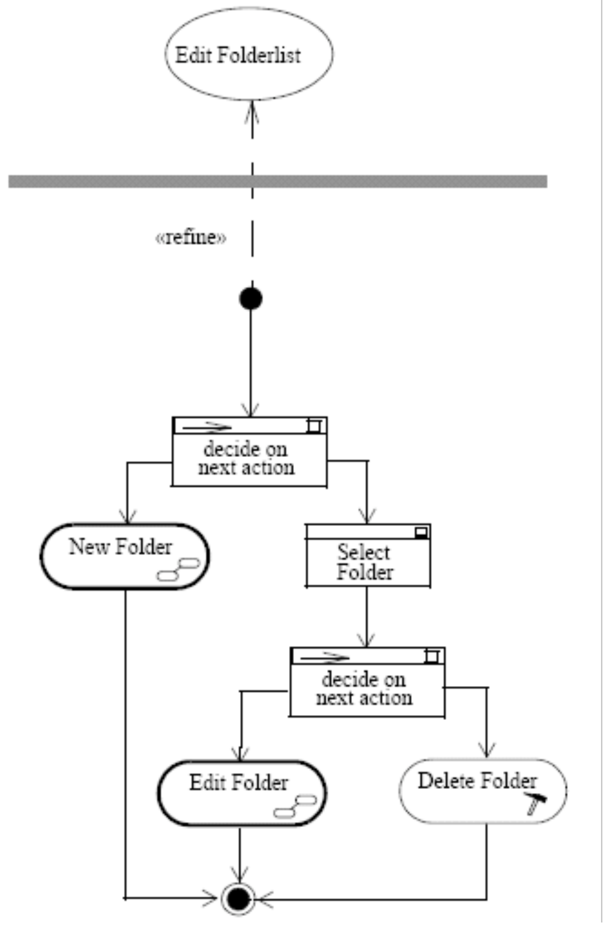


Activity graph elements

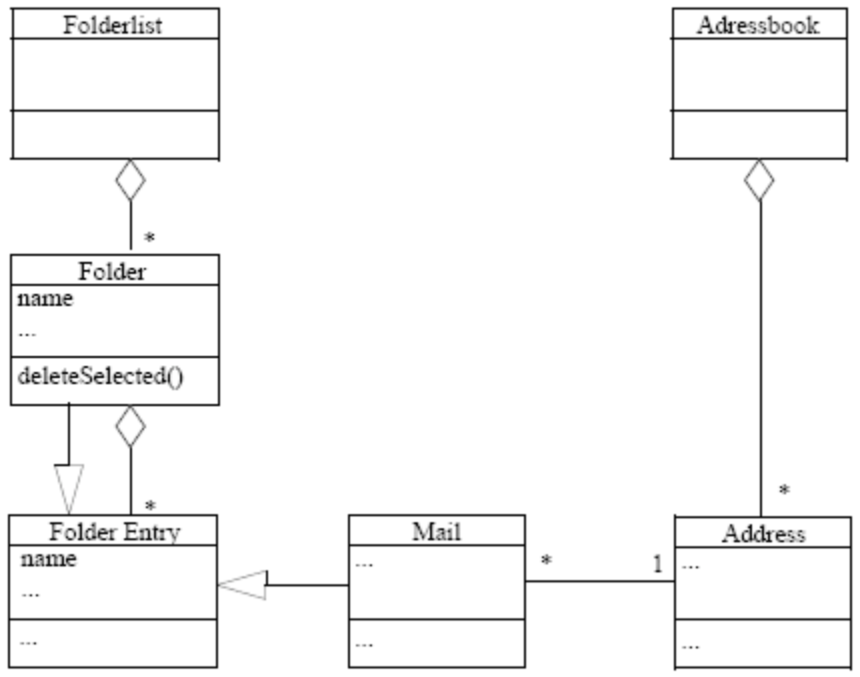


Stereotype	Description	Visualisation
«context action»	An ActionState stereotyped *«context action»* represents an action which is performed by an actor without the help of the system.	Context Action
«business interaction»	An ActionState stereotyped *«business inter-action»* represents an action that is per-formed by an actor with the help of the system producing an observable result.	Business Interaction
«decision interaction»	An ActionState stereotyped *«decision inter-action»* represents an action that is per-formed by an actor with the help of the system resulting in a decision on the subse-quent action.	Decision Interaction
«system action»	An ActionState stereotyped *«system action»* represents an action that is executed by the system on its own producing an observable result.	System Action
«macro action»	A SubactivityState stereotyped *«macro action»* is an action that "calls" a subgraph (it reflects an «include» or an «extend» dependency).	Macro Action
«actor in action»	An ObjectFlowState stereotyped *«actor in action»* depicts an actor which can be asso-ciated with some actions.	:ActorName

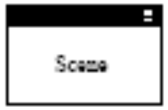
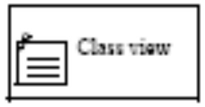
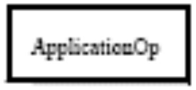
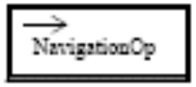
Activity Graph Example



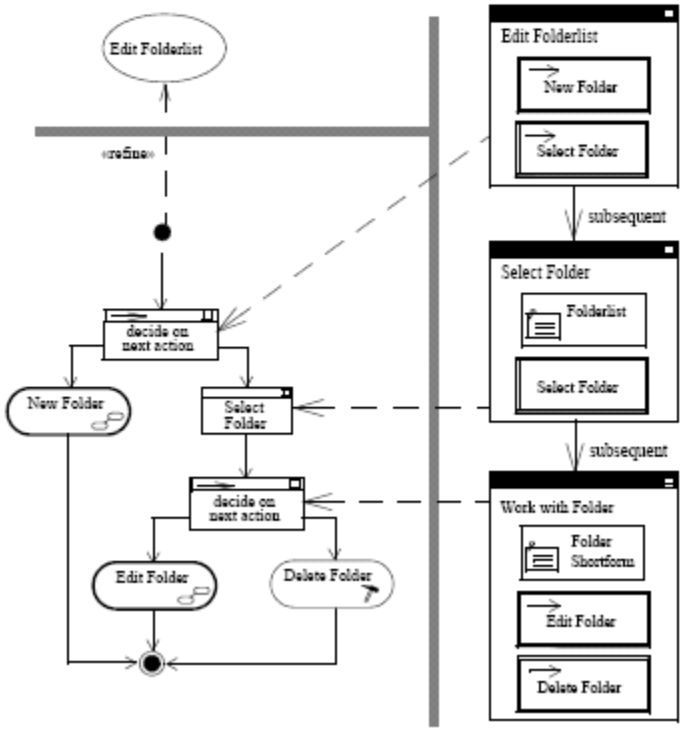
Domain Class Model



User Interface Elements

Stereotype	Description	Visualisation
«scene»	A class stereotyped *«scene»* represents an abstraction of (a part of) a screen.	
«class view»	A class stereotyped *«class view»* is an abstract presentation of instances and relations of a domain class.	
«application operation»	A scene operation stereotyped *«application operation»* is activated by the user during a *«business interaction»*.	
«navigation operation»	A scene operation stereotyped *«navigation operation»* is activated by the user during a *«decision interaction»*.	

Example



Edit Folderlist
«refine»
decide on next action
New Folder
Select Folder
decide on next action
Edit Folder
Delete Folder
Edit Folderlist
New Folder
Select Folder
subsequent
Select Folder
Folderlist
Select Folder
subsequent
Work with Folder
Folder Shortform
Edit Folder
Delete Folder

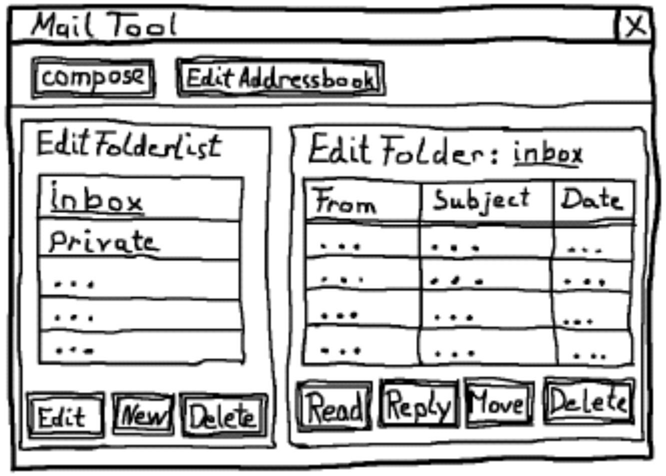
UI Structure and Navigation

- UI:s are decomposed into building blocks (windows, web pages)
- Windows and web pages can in turn be divided into e.g. panels or table cells
- Building blocks are composed from scenes in a three step process

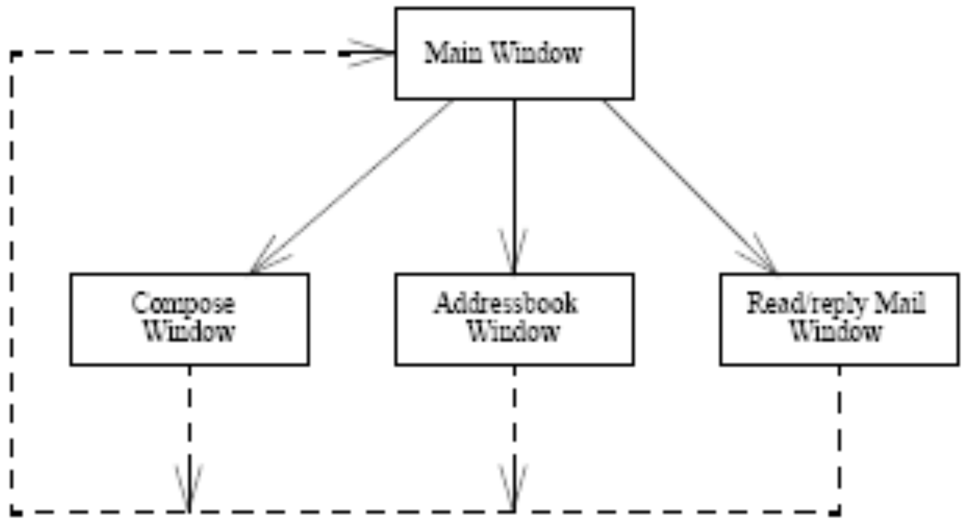
Three step process

- Merge related ”atomic scenes” to ”superscenes” in an iterative process
- Compose superscenes to windows (internal window structure)
- Arrange windows according to a hierarchical structure (external windows structure)

Possible draft of the internal structure of the main window



External window structure



DIWA

- Framework for the development of high quality GUIs
- Based on the fundamental Software Engineering (SE) principles

SE-principles

- **Hierarchical structure** – decomposition into smaller components
- **Homogenous components** – same structure and treatment for all components

SE-principles

- **Separation of concern** – Responsibilities of a component cohesive and separated from other components
- **Acyclic communication** – Acyclic use dependencies

The DIWA-approach

- Provides a logical separation of the UI from the functional core of the application
- Decomposition of the UI into user interface objects (UIOs)
- Both simple and complex (composite) components are treaded as UIOs

The User Interface Object (UIO)

- Encapsulates 3 associated parts
  - Dialog behavior (Dialog Control)
  - Screen layout (Presentation)
  - Accessing the functional core (Application Interface)
- The 3 parts complies with the SE-principles separation of concern and acyclic communication

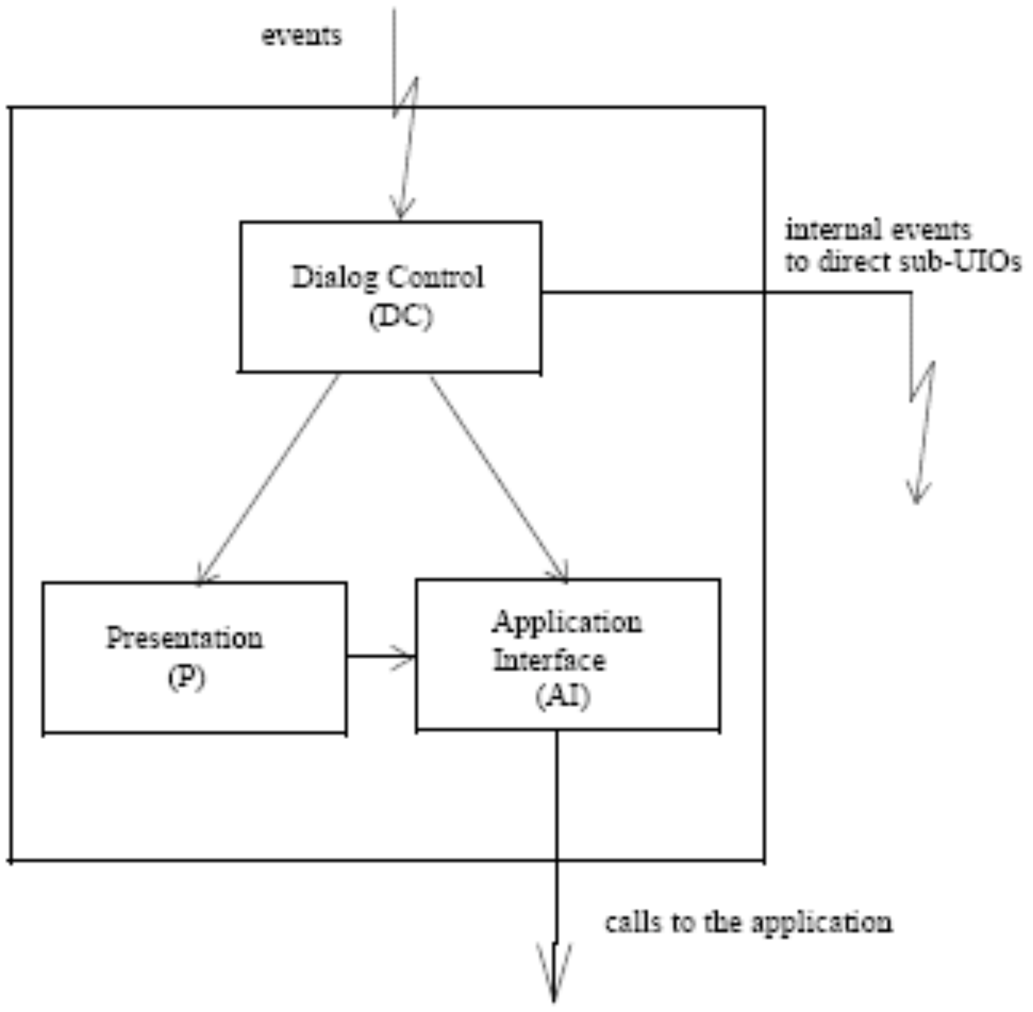
Dialog Control (DC)

- Serves as the interface of the UIO
- Retrieves events and performs the appropriate actions
  - Send the event to the Presentation
  - Calls an application function via the Application Interface
  - Passing it to a subsequent UIO

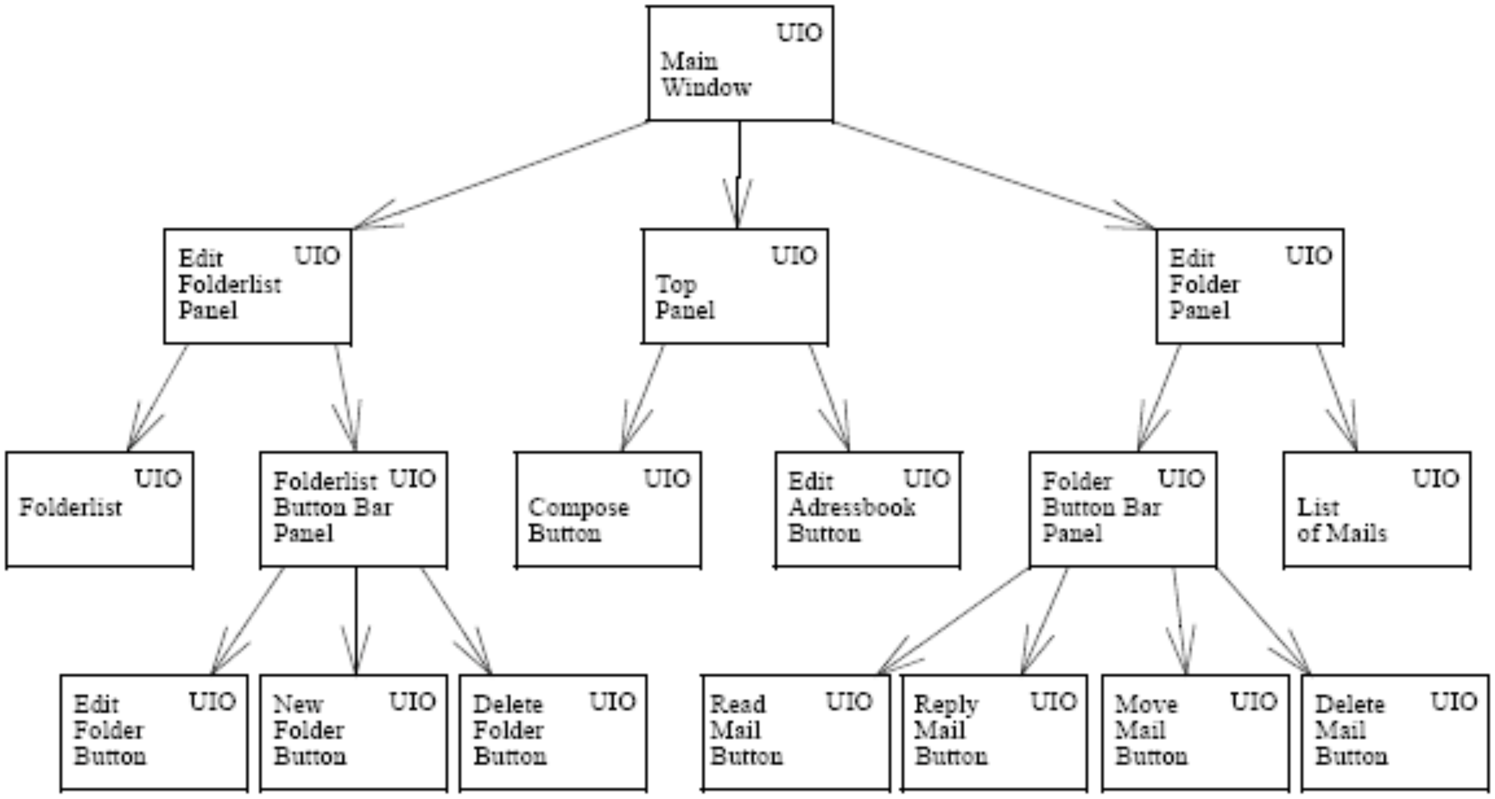
Presentation (P) and Application Interface (AI)

- Presentation – Responsible for drawing the UI
- Application Interface – Provides access to the application functions and data

A DIWA UIO


events
Dialog Control
(DC)
internal events
to direct sub-UIOs
Presentation
(P)
Application
Interface
(AI)
calls to the application

Mail Tool Example



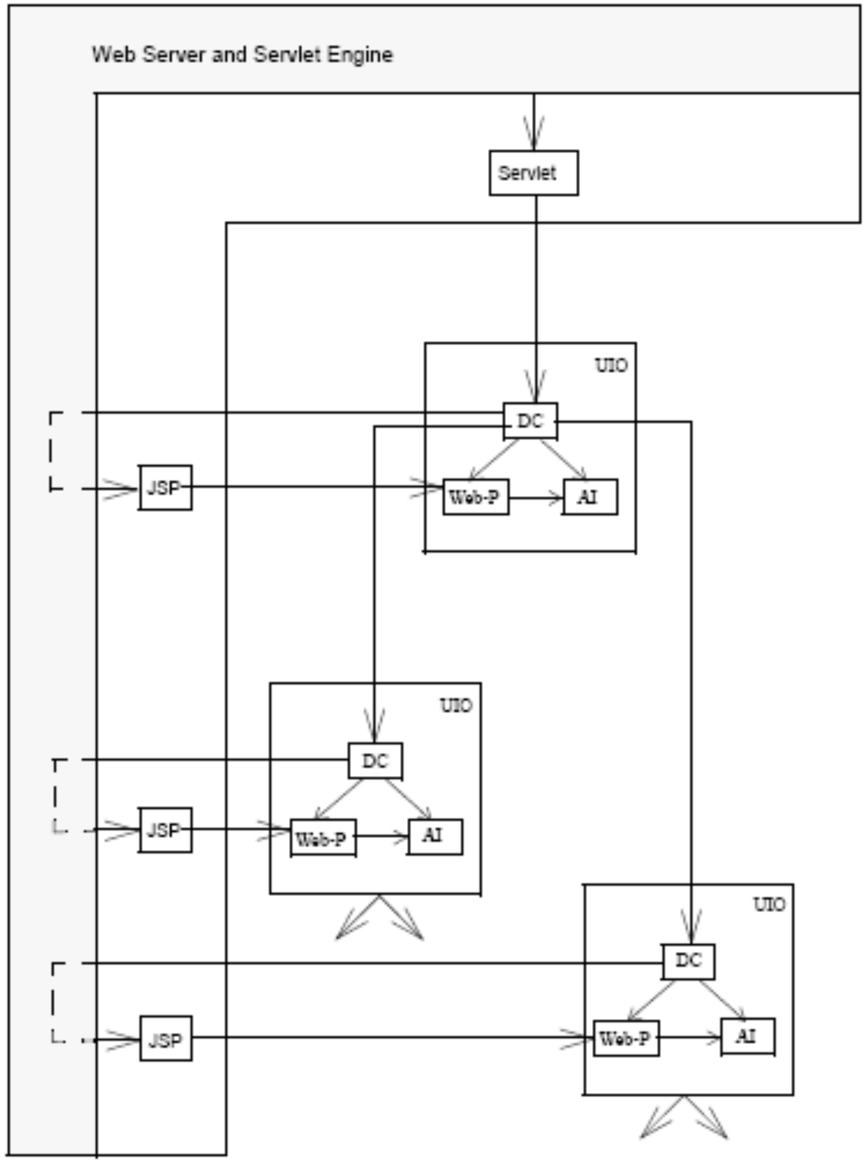
Web-Based User Interfaces

- A Web-UI is divided into a client tier and a server tier
  - Clients browser responsible for displaying Web documents and communication between client and server
  - Server comprises of two parts
    - Web server provides Web documents
    - The Servlet retrieves dynamic content from the application

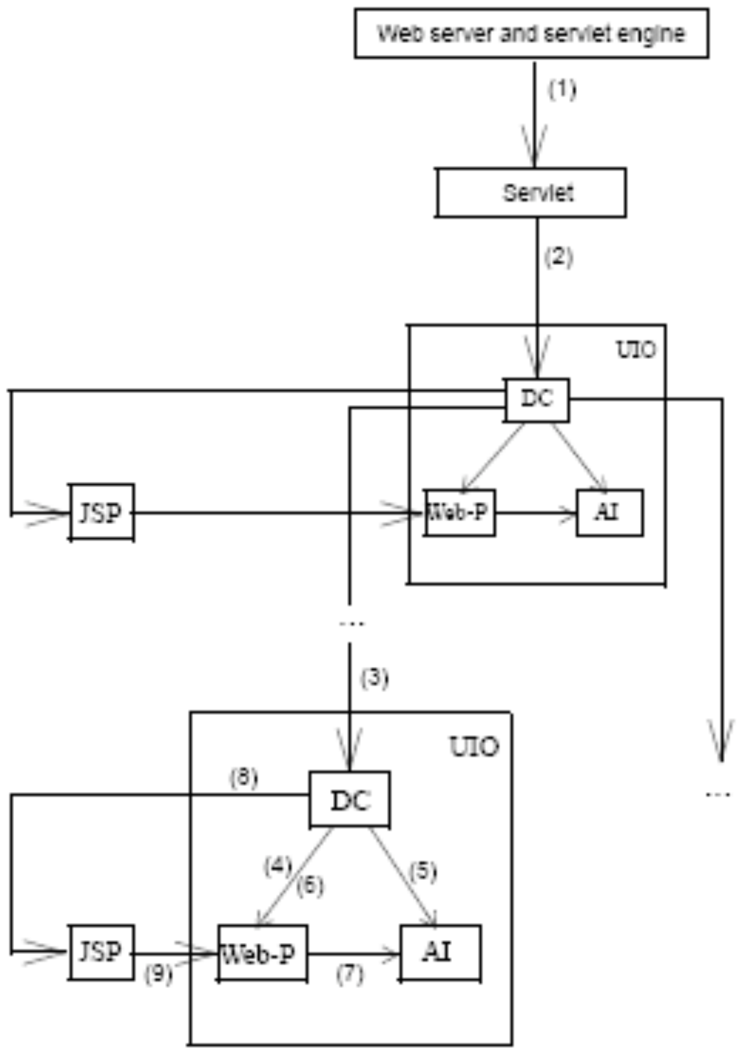
DAWID

- DIWA-based Web User Interface Development
- Refines DIWA for the Web environment
- Uses a Web-UIO much like the DIWA-UIO
- The DAWID-UIO uses a Web-Presentation instead of the original Presentation component

DAWID architecture



Collaboration of the DAWID components



Summary

- A Software Engineering-based approach for developing complex Web-based User Interfaces
- Two steps
  - Gather and model requirements
  - Map requirements into Web-UI software architecture
- The DAWID framework is proposed for the second step