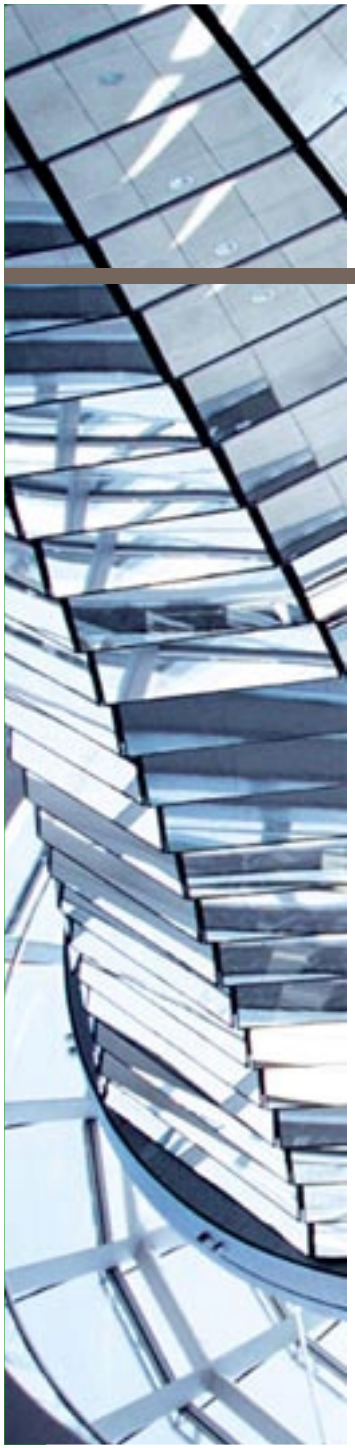


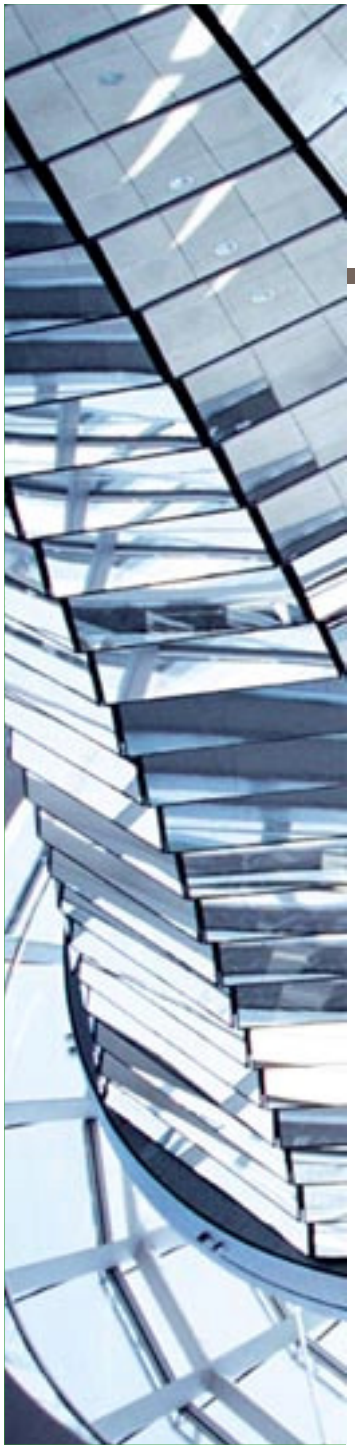
6. Client Architecture
Reference Architectures and Patterns

Winter Semester 2008 / 2009
Prof. Dr. Bernhard Humm
Darmstadt University of Applied Sciences
Department of Computer Science

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

The lecture in the context of the entire course

1. Introduction
2. A reference architecture for business information systems
3. Application kernel
4. Persistence and transaction
5. Authorization
6. Client architecture
7. Exception handling
8. Other reference architectures: SOA, BI, systems integration, …



Agenda

➔ **Clients**

Reference architecture

GUI library

Dialog

Dialog frame

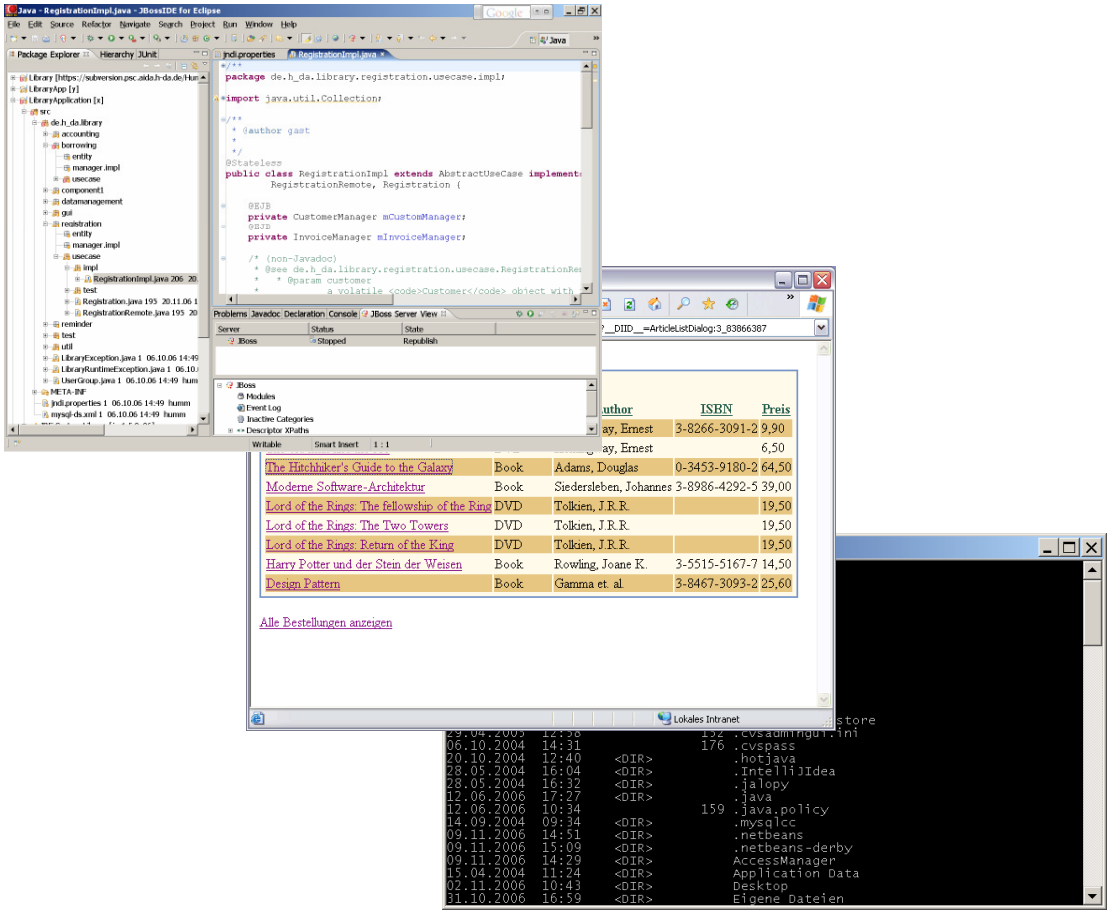
Application kernel access

Sequences

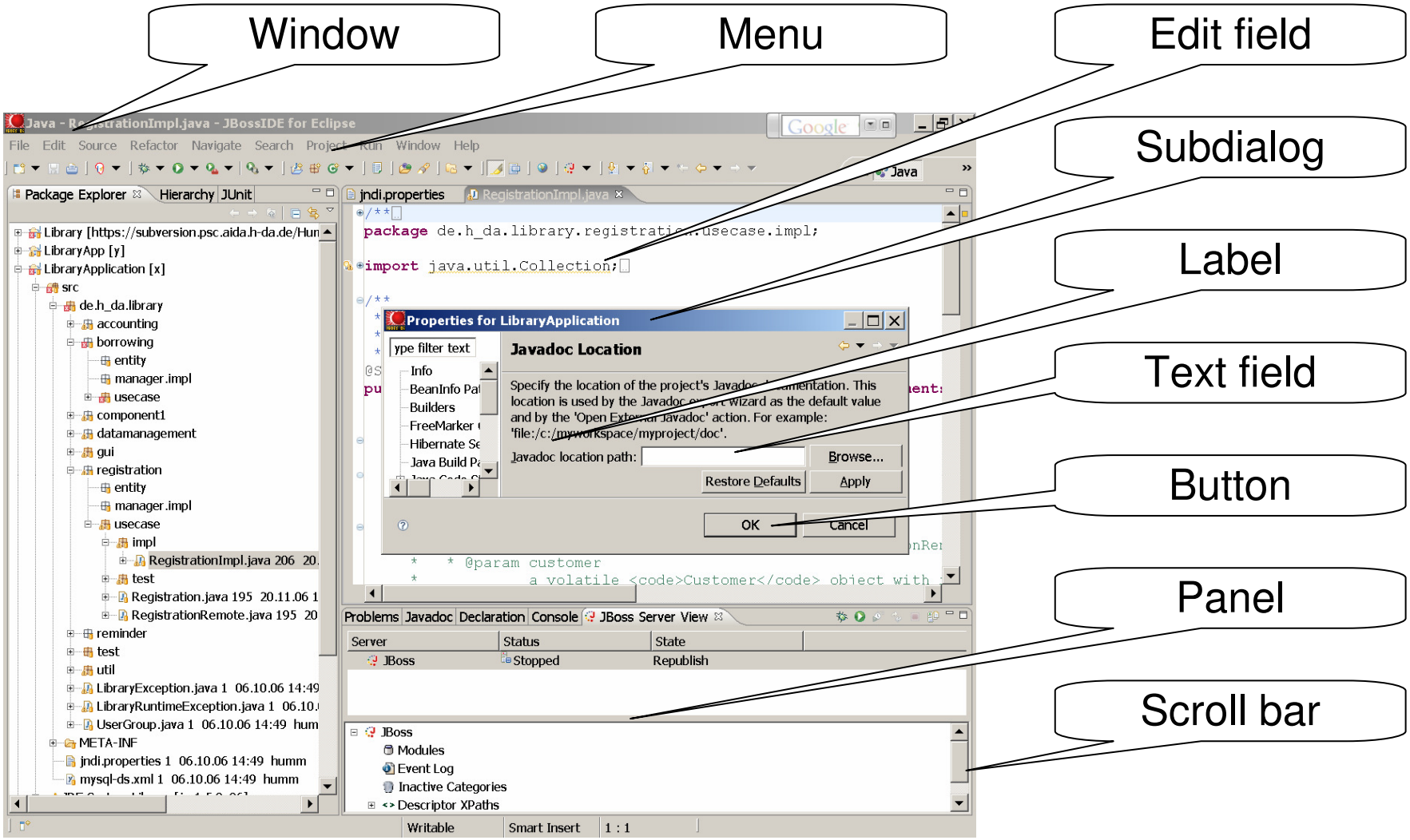
Literature

Client: all types of user interfaces

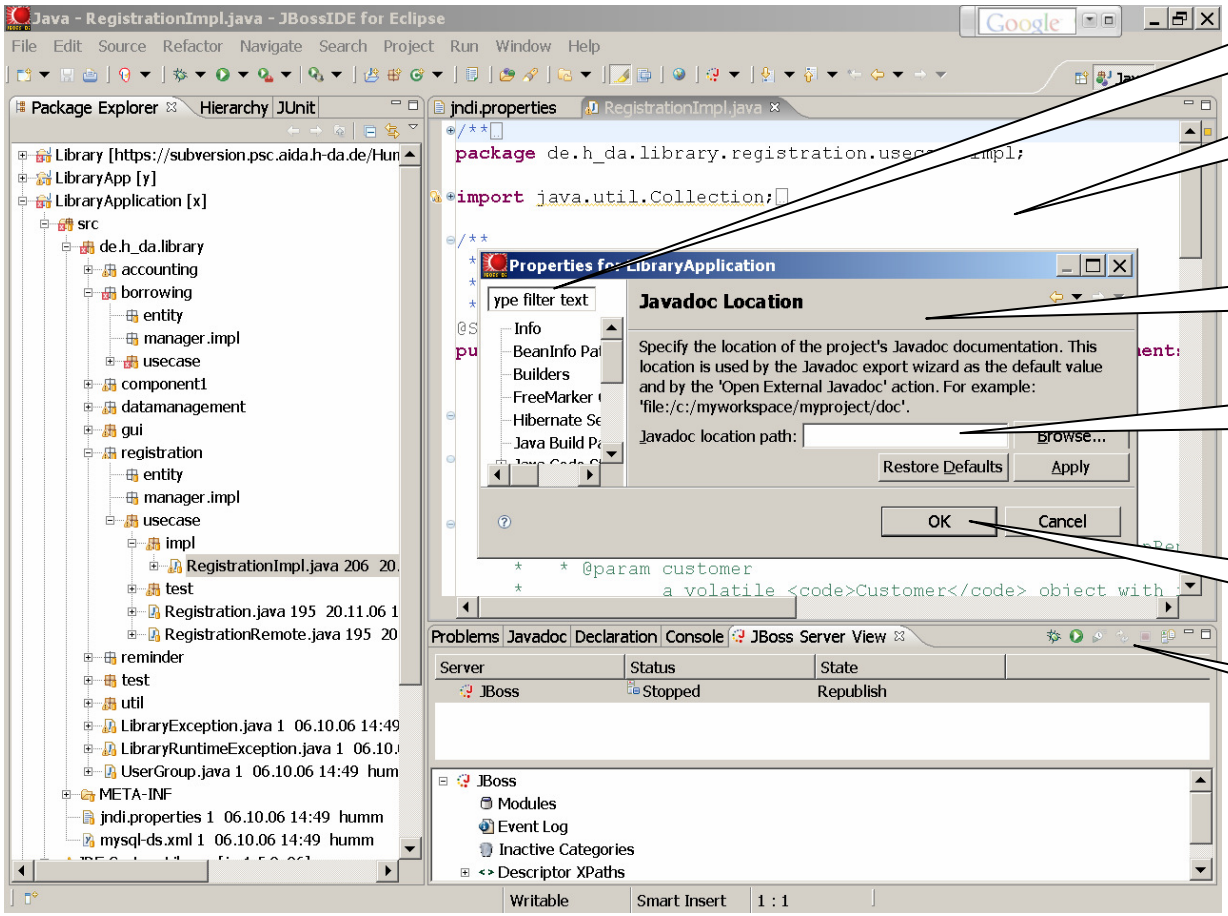
- Graphical (GUI) and textual (TUI)
- Web-based (Internet portal) and native (e.g., Java Swing)
- Distributed (client / server – thin client) and local (fat client)



Static aspects of a client



Dynamic aspects of a client



Dialog control / workflow

Displaying data

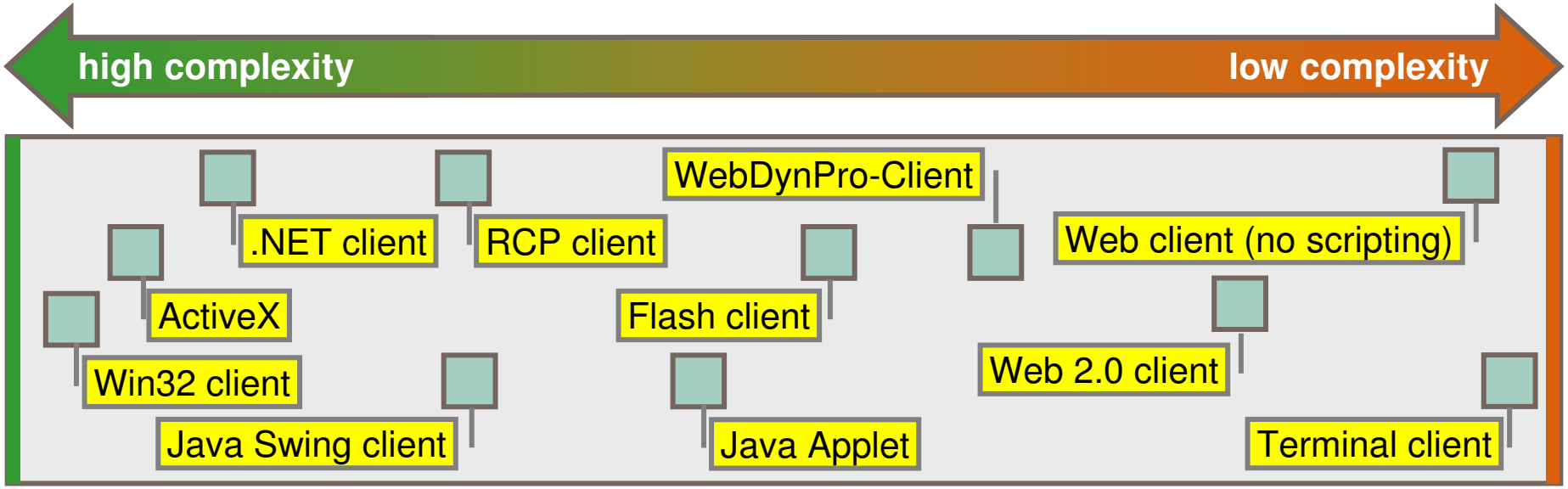
multi-language support

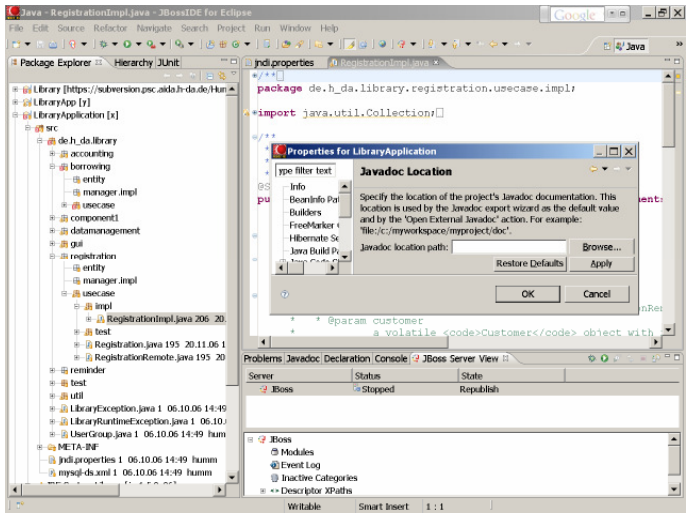
Validating user input

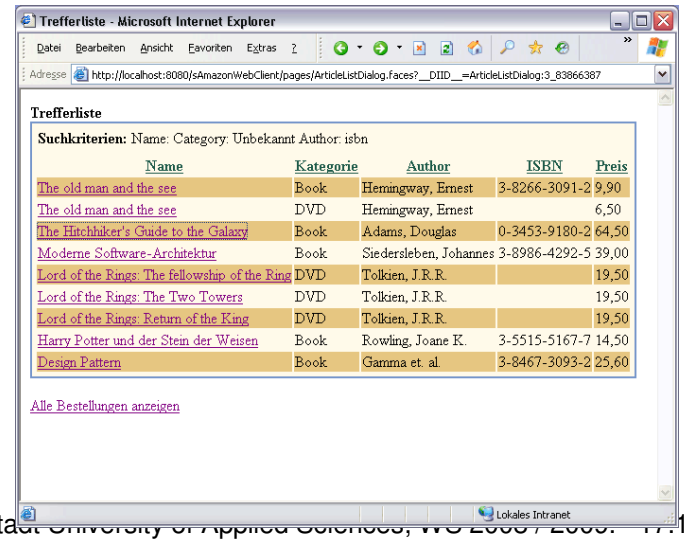
Performing actions

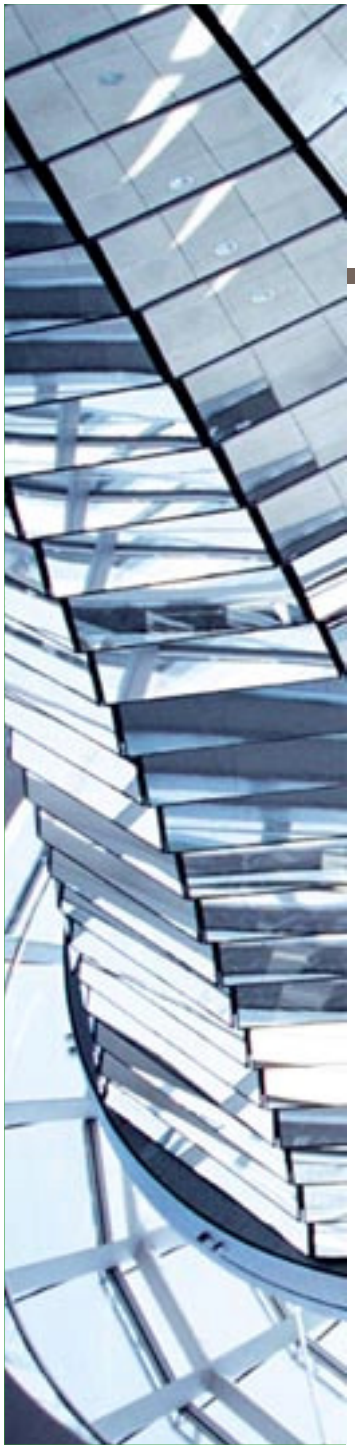
State handling

Client technologies









Agenda

Clients

➔ **Reference architecture**

GUI library

Dialog

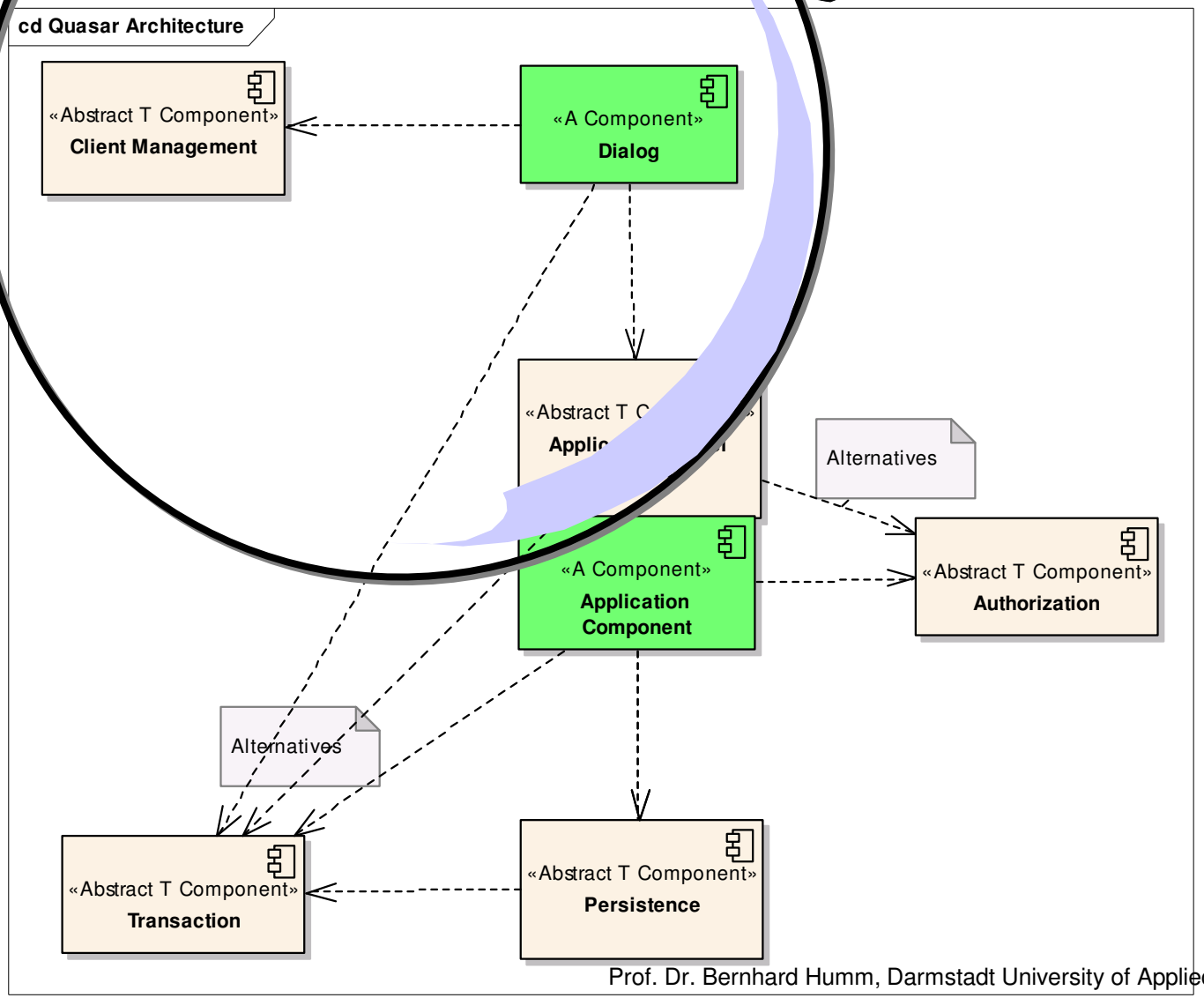
Dialog frame

Application kernel access

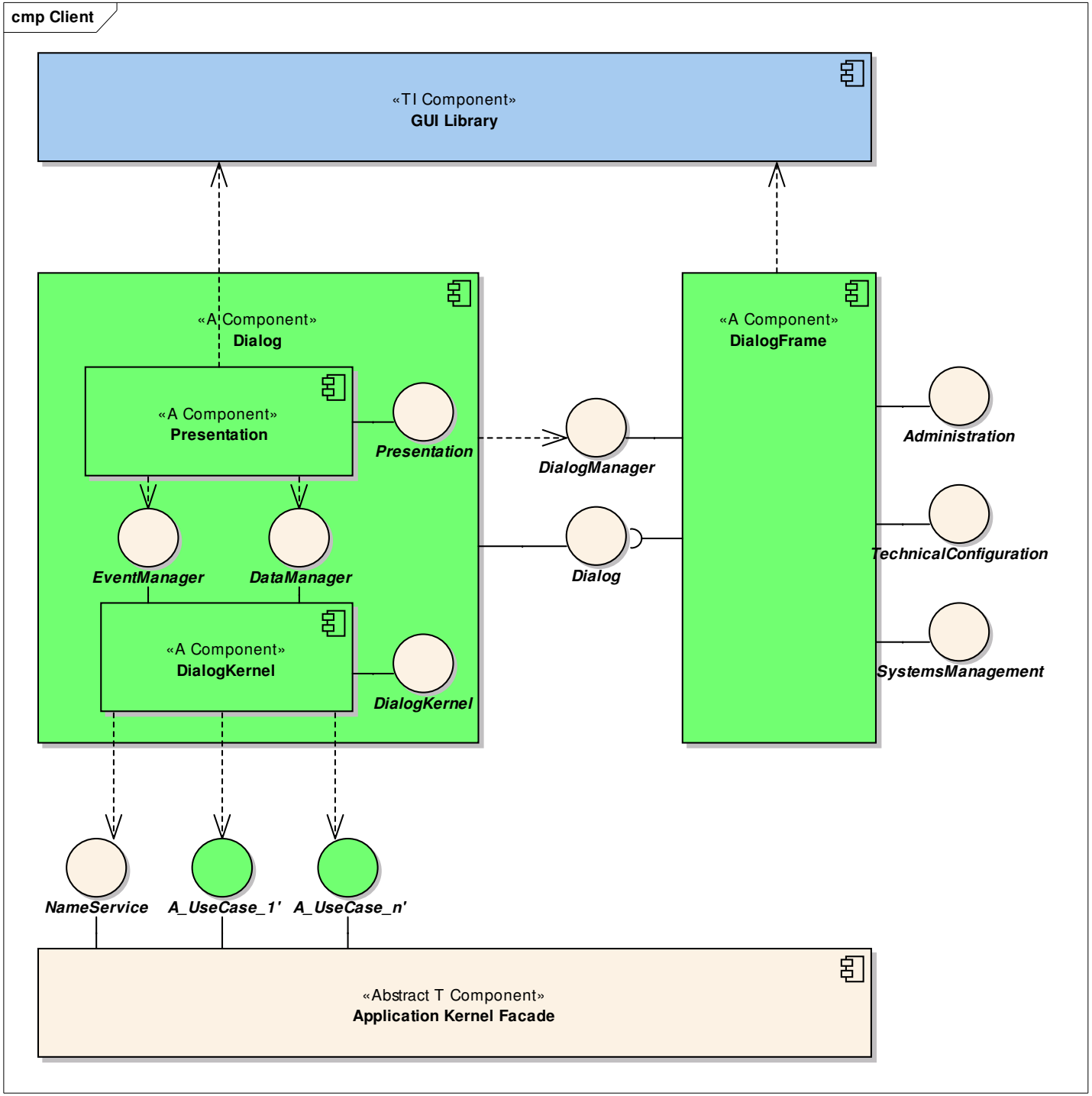
Sequences

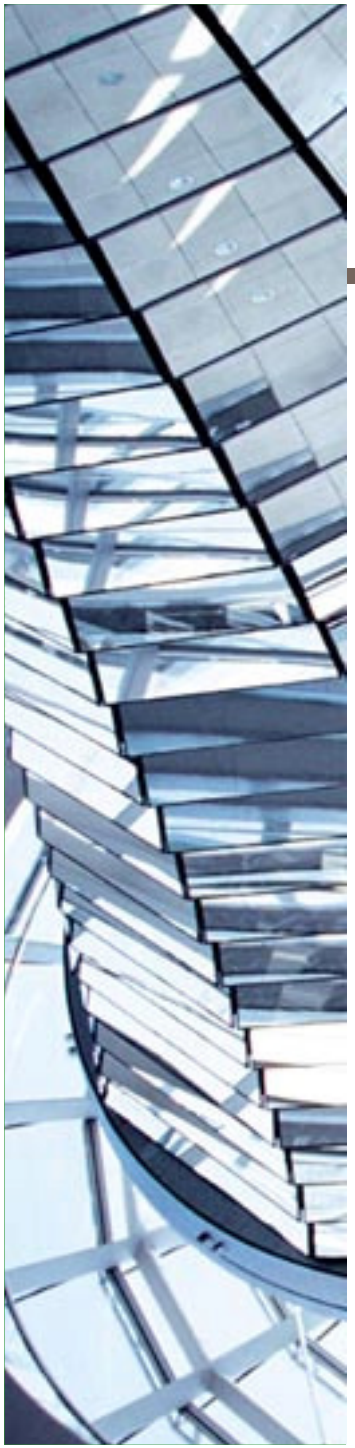
Literature

Reference architecture for business information systems Quasar (Quality Software Architecture)



Client Architecture





Agenda

Clients

Reference architecture

➔ **GUI library**

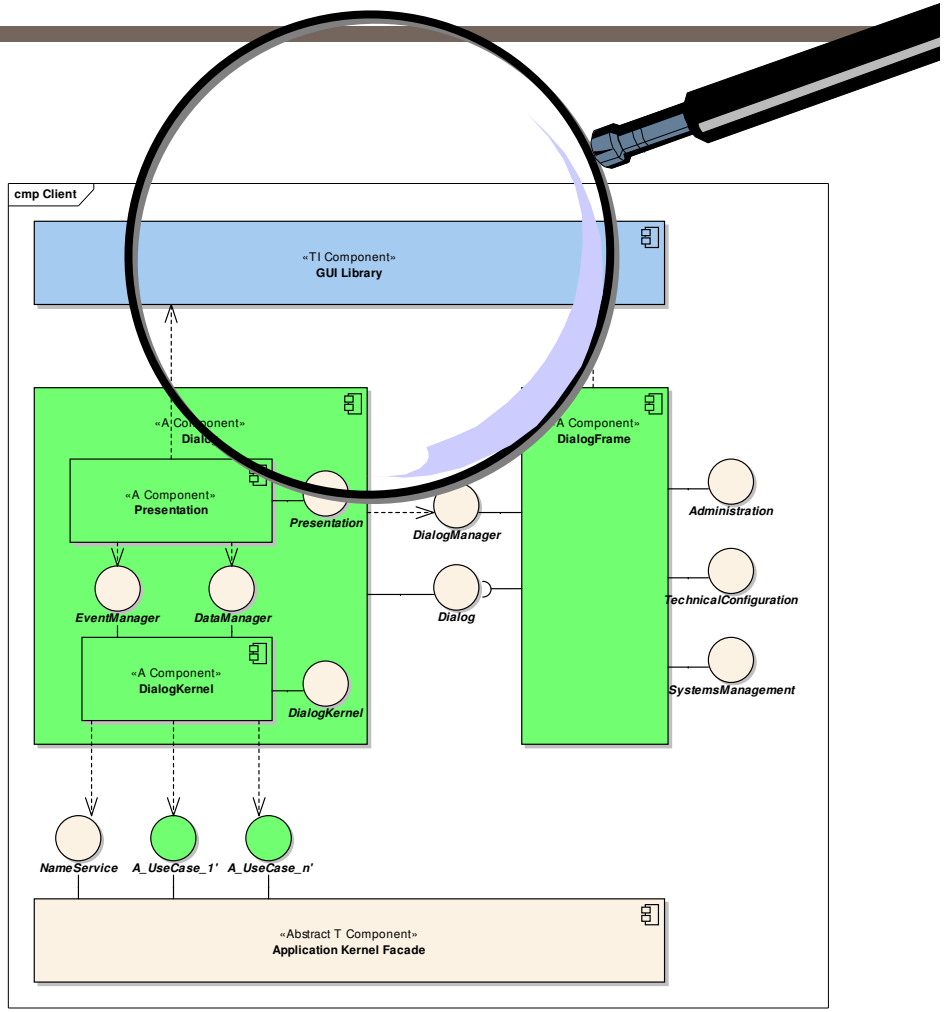
Dialog

Dialog frame

Application kernel access

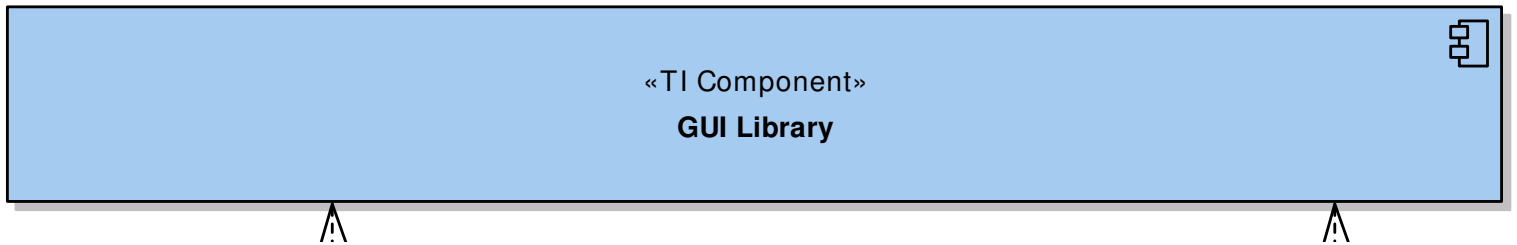
Sequences

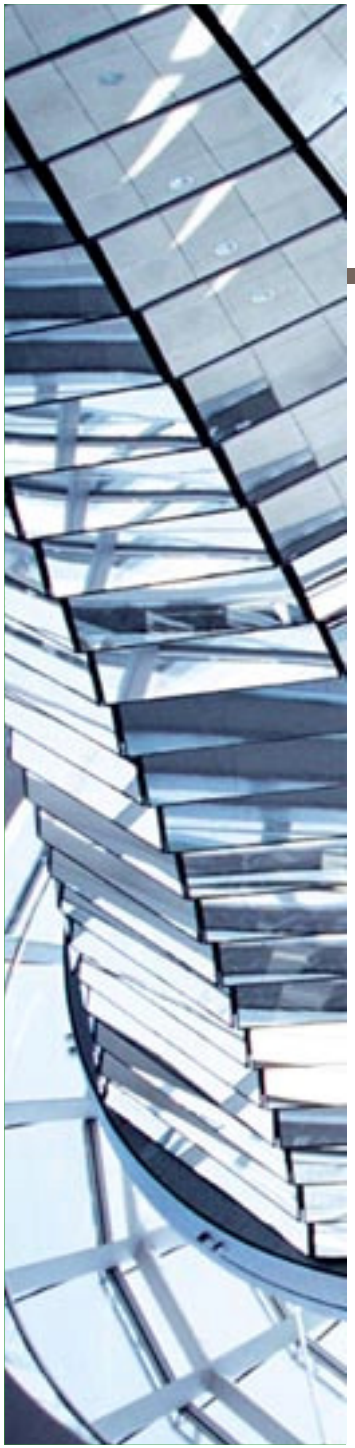
Literature



GUI Library

- Platform-specific library for GUI widgets and GUI events
- Examples:
  - Java Swing
  - Java AWT
  - JSF
  - .NET
  - Active X





Agenda

Clients

Reference architecture

GUI library

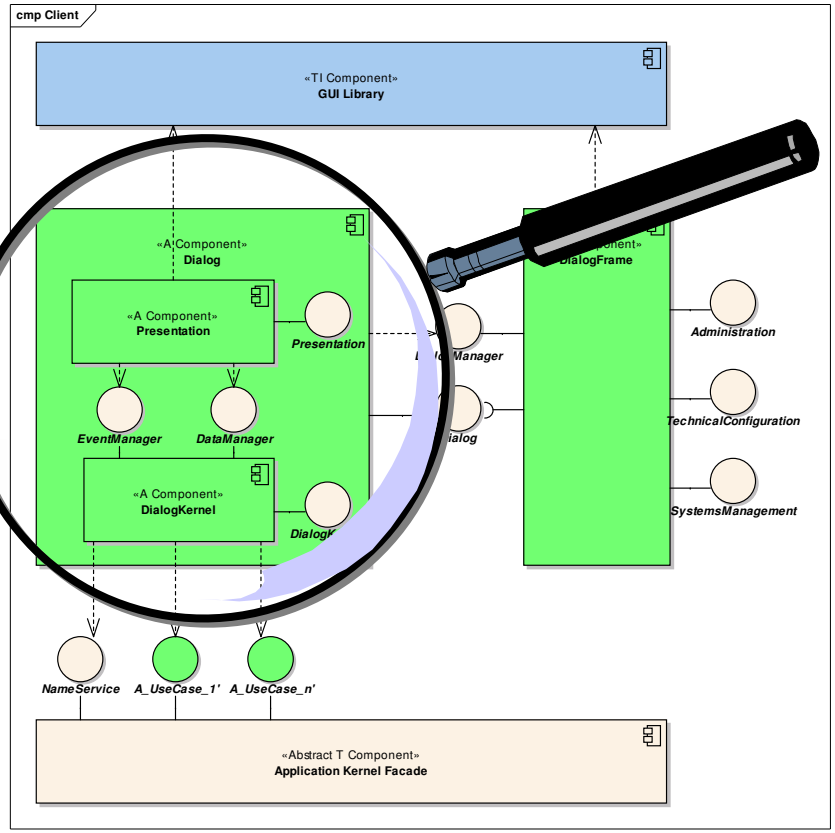
➔ **Dialog**

Dialog frame

Application kernel access

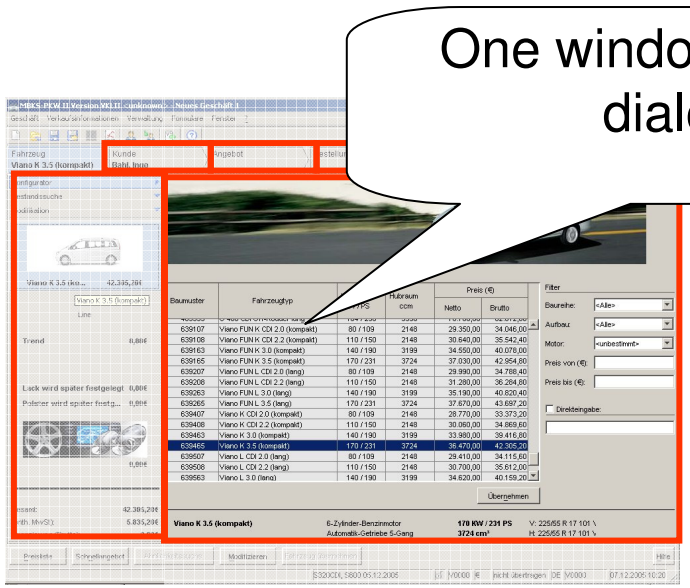
Sequences

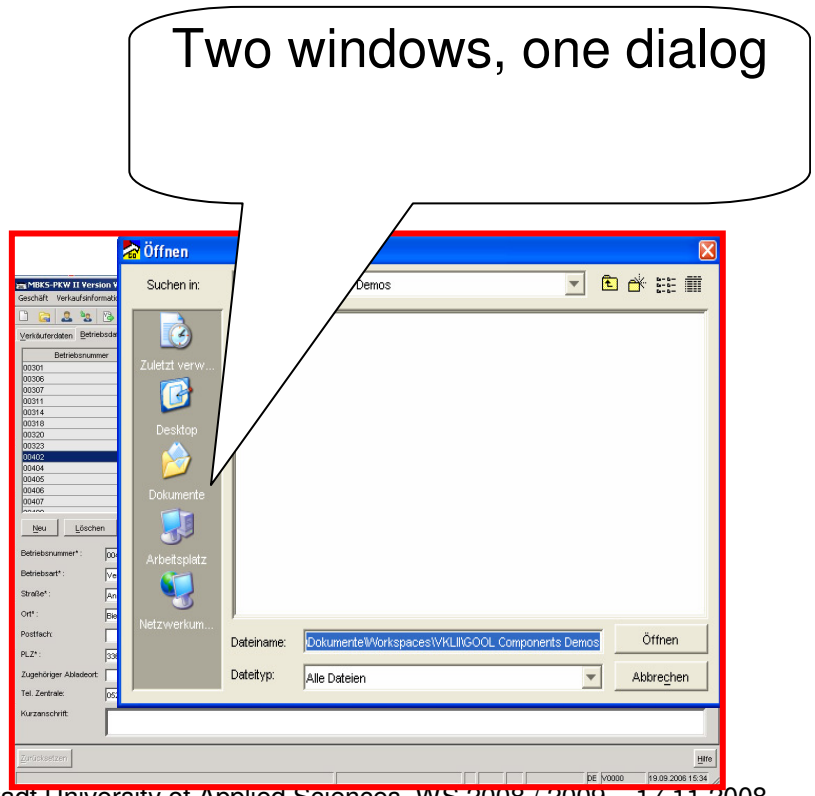
Literature



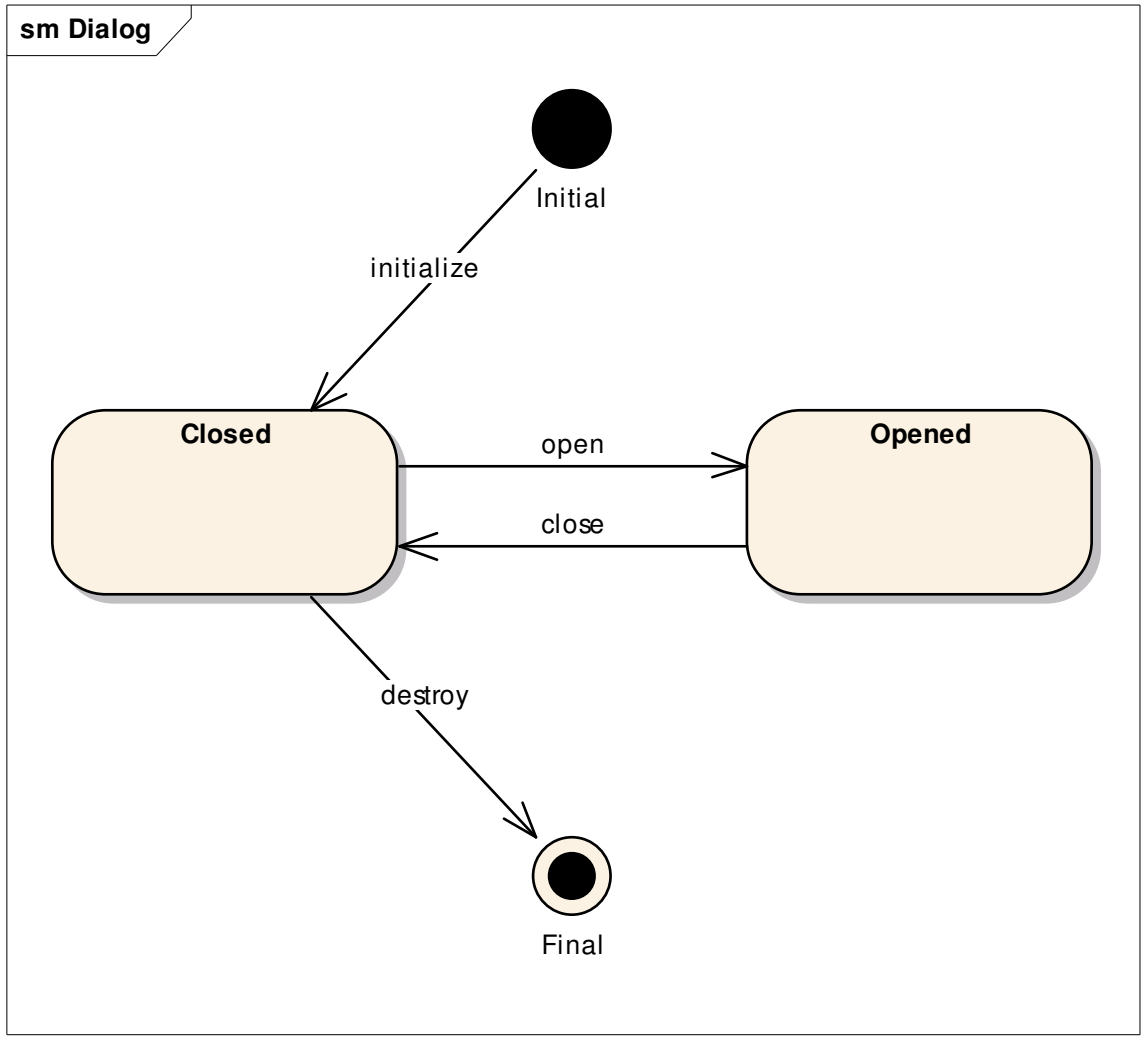
Dialog

- A dialog is a sensible (from the user‘s point of view) unit of UI elements including their functionality
- A dialog supports one or more use cases
- Dialogs may have subdialogs





Dialog life cycle: State transition diagram



Dialog states:

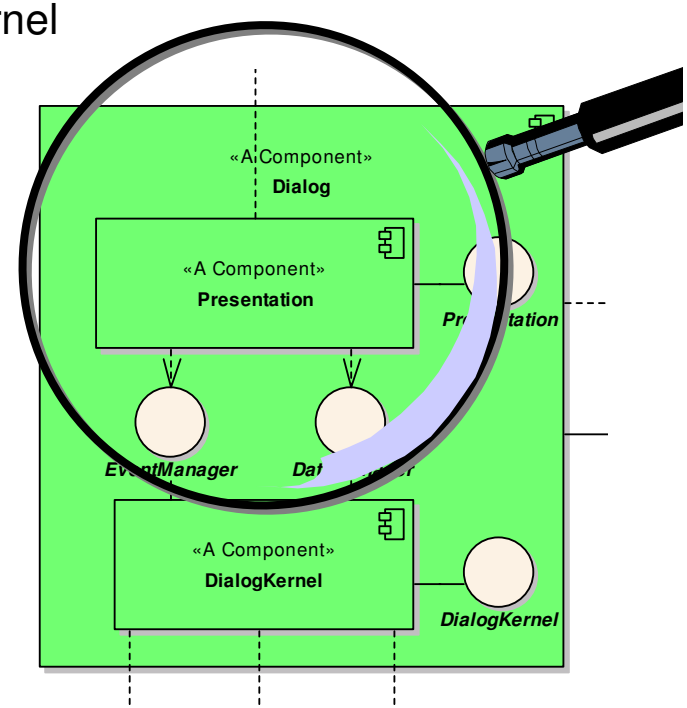
- Opened: the visual representation of the dialog is shown (maybe overlaid by another window)
- Closed: the visual representation of the dialog is hidden

Interface Dialog

Method Summary	
void	**close**() [command] The dialog is being closed.
void	**destroy**() [command] The dialog is destroyed.
DialogState	**getState**() [basicQuery] The current DialogState (opened or closed) of the dialog is returned.
VisualRepresentation	**getVisualRepresentation**() [basicQuery] The visual representation of the dialog is returned
void	**initialize**(DialogManager dialogManager) [command] The dialog is initialized.
void	**open**() [command] The dialog is opened.

Presentation

- Displays data of the dialog kernel to the user (dialog kernel → presentation)
- Allows to get data from the user to be stored in the dialog kernel (presentation → dialog kernel)
- Accepts user events and passes them to the dialog kernel (presentation → dialog kernel)

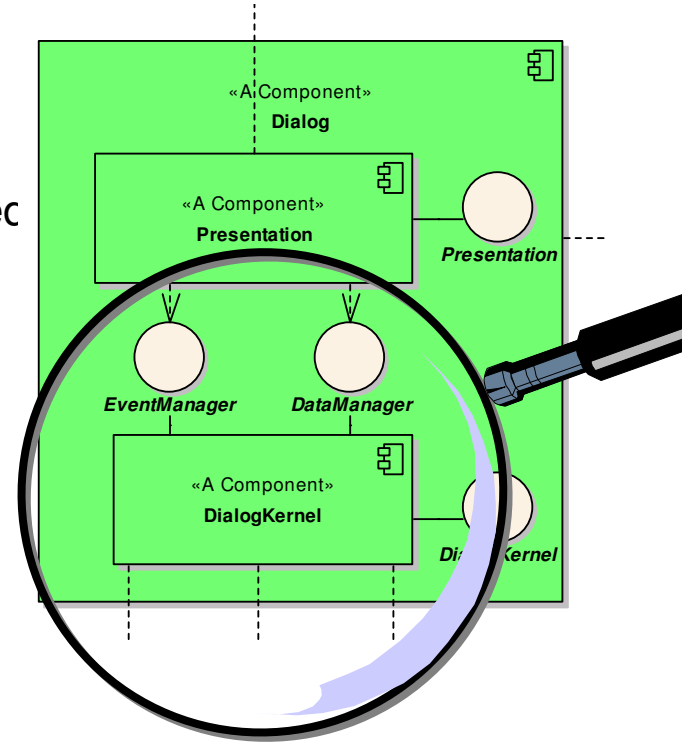


Interface Presentation

Method Summary	
void	**destroy**() [command] The presentation including its visual representation is destroyed and cannot be used within the dialog any more.
VisualRepresentation	**getVisualRepresentation**() [basicQuery] The VisualRepresentation is returned
void	**initialize**(DataManager dataManager, EventManager eventManager) [command] The presentation is initialized with its resources and can be used by the dialog.
void	**update**() [command] The presentation ist called to update itself (observer pattern) e.g., due to changed data in the dialog kernel

Dialog Kernel

- Manages the dialog‘s application data, e.g.,
  - Customer data entered by user
  - Search result obtained from the application kernel
- Manages the dialog state, e.g.,
  - Customer account overdrawn
  - No user right to perform a certain action (→ button disablec
- Performs actions, e.g.
  - Invoke use case in application kernel
  - Update presentation



Interface DialogKernel

Method Summary	
void	**destroy**() [command] The dialog kernel is destroyed.
void	**initialize**() [command] The dialog kernel is initialized.

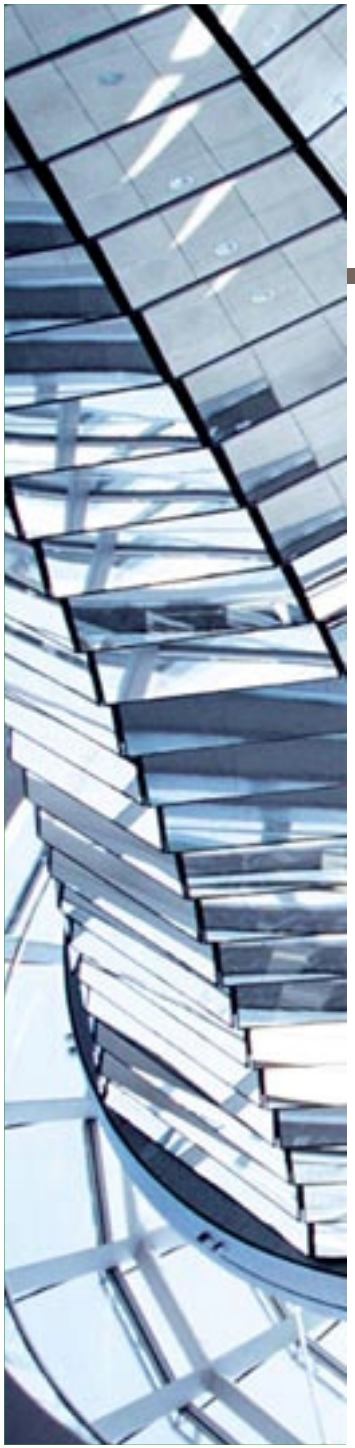
Interface EventManager

Method Summary

void	**trigger**(Event event) [command] An Event is triggered at this EventManager.

Interface DataManager (Pool)

Method Summary	
java.util.Set<java.lang.Object>	**executeQuery**(Query query, java.util.List<java.lang.Object> arguments) [basicQuery] Executes query on this pool and returns the result set
java.lang.Object	**fetch**(UID uid) [basicQuery] Returns the object stored in the Pool under the unique identifier uid.
UID	**getUID**(java.lang.Object object) [basicQuery] Returns the unique identifier of object if object is stored in the pool, error objectNotFound otherwise.
UID	**insert**(java.lang.Object object) [command] Inserts object to this pool and returns the unique identifier
void	**remove**(java.lang.Object object) [command] Removes object from this pool



Agenda

Clients

Reference architecture

GUI library

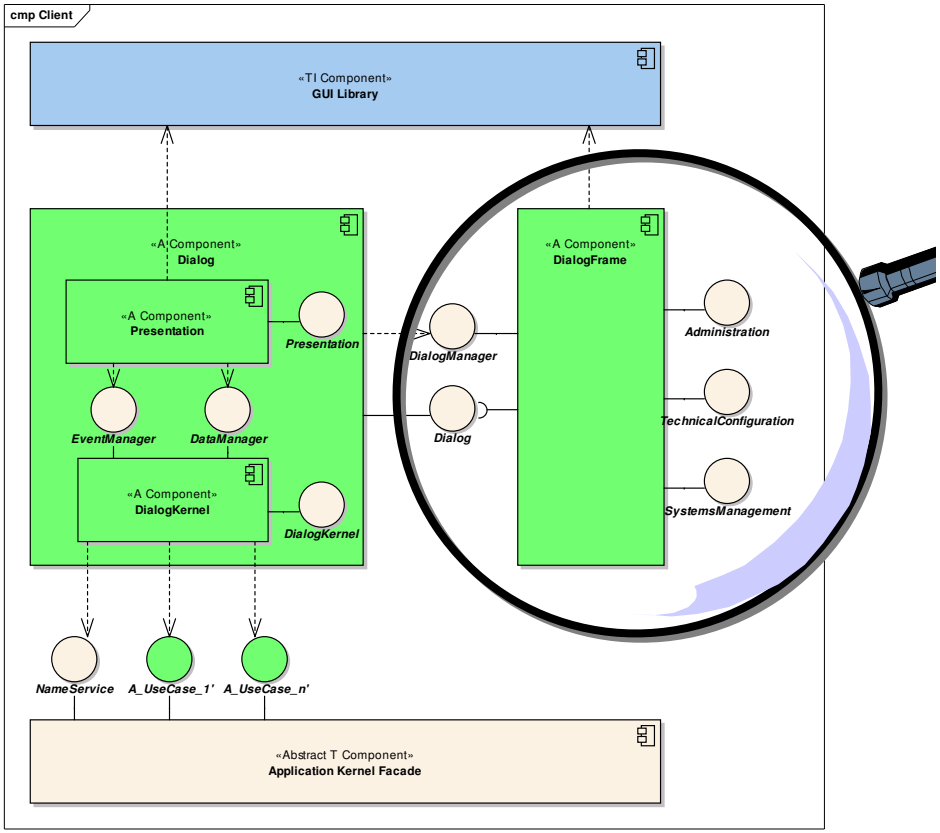
Dialog

➔ **Dialog frame**

Application kernel access

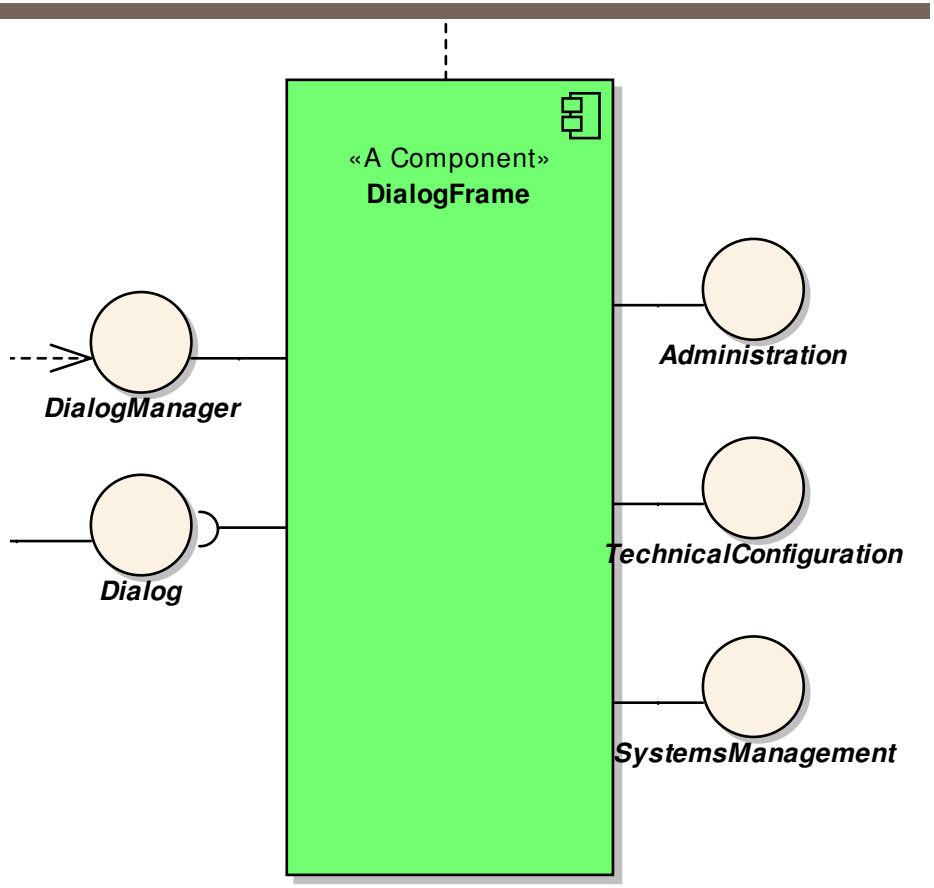
Sequences

Literature



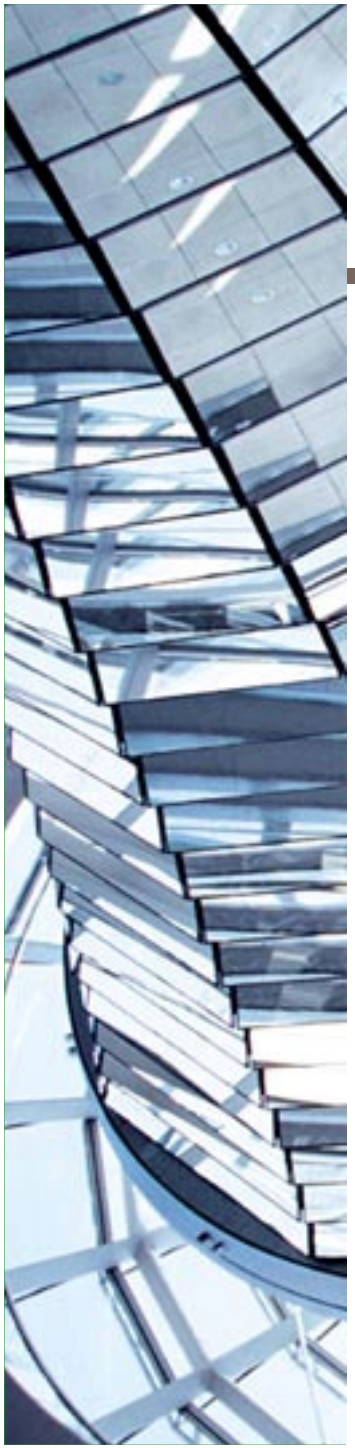
Dialog Frame

- Encompasses the client application, i.e., the main method to be executed on the client
- Manages user sessions, possibly including authentication and authorization
- Performs dialog control per user session, e.g., search dialog followed by search result dialog
- Manages the life cycle of dialogs, i.e., opening and closing dialogs
- Facilitates communication between the dialogs



Interface DialogManager

Method Summary	
Dialog	**makeDialog**(DialogType dialogType) [command] An instance of a Dialog, denoted by the DialogType, is provided.
void	**releaseDialog**(Dialog dialog) [command] An instance of a Dialog is released.



Agenda

Clients

Reference architecture

GUI library

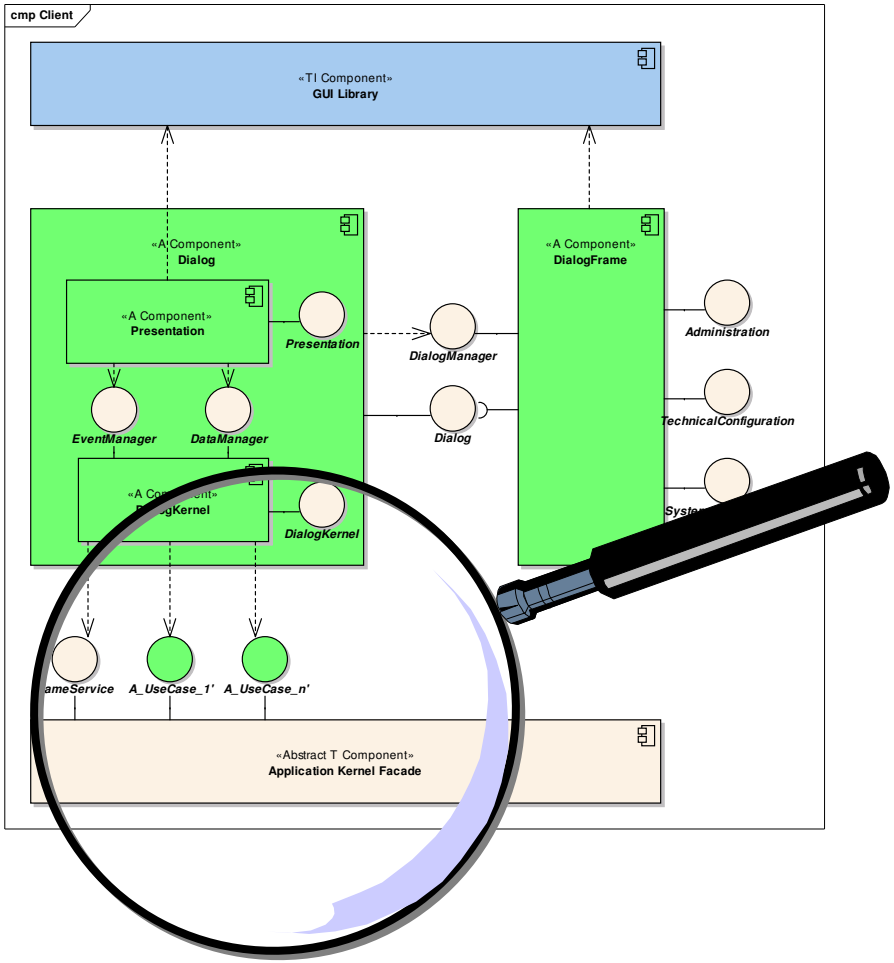
Dialog

Dialog frame

➔ **Application kernel access**

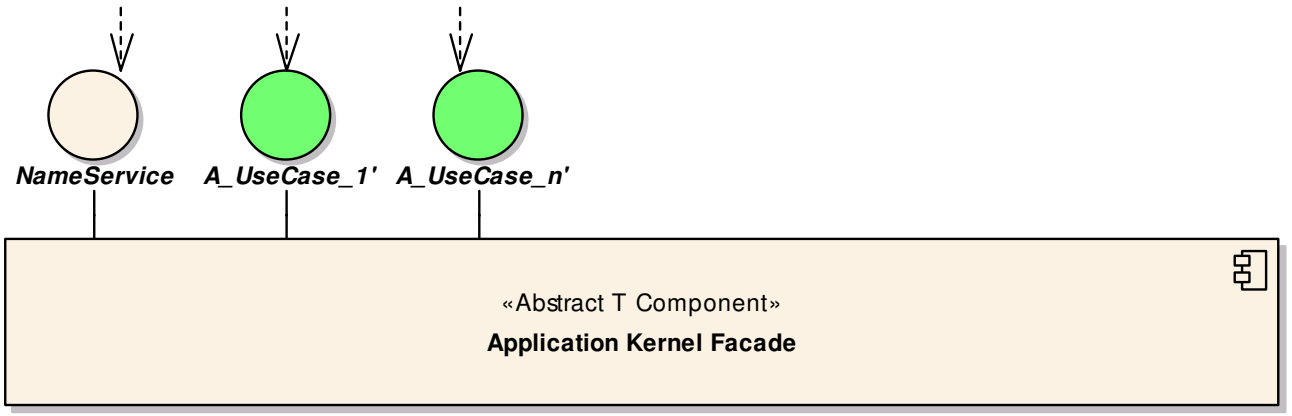
Sequences

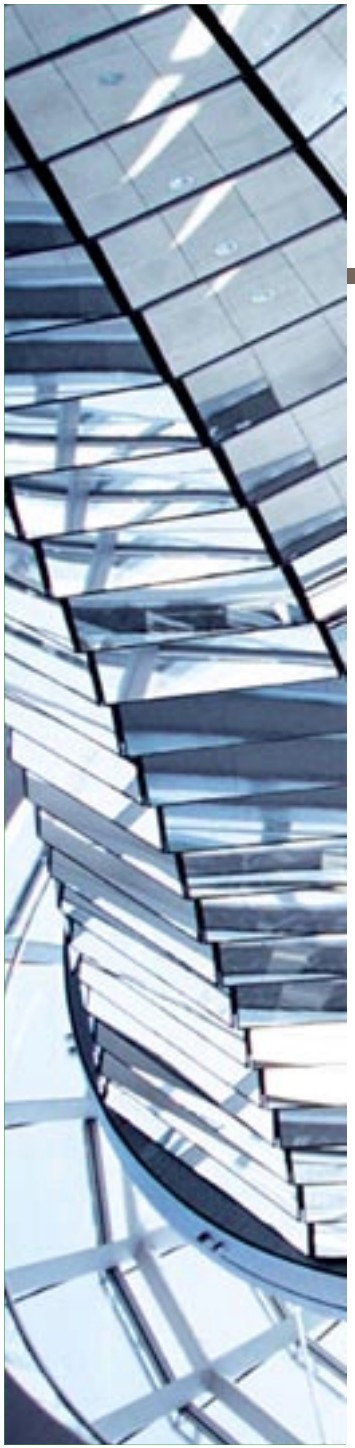
Literature



Application Kernel Access

- The client communicates with the application kernel via the application kernel facade
- Provides location transparency (client / server communication)
- Provides a name service to access use cases
- Provides use case interfaces
- Provides additional services like authorization and transaction





Agenda

Clients

Reference architecture

GUI library

Dialog

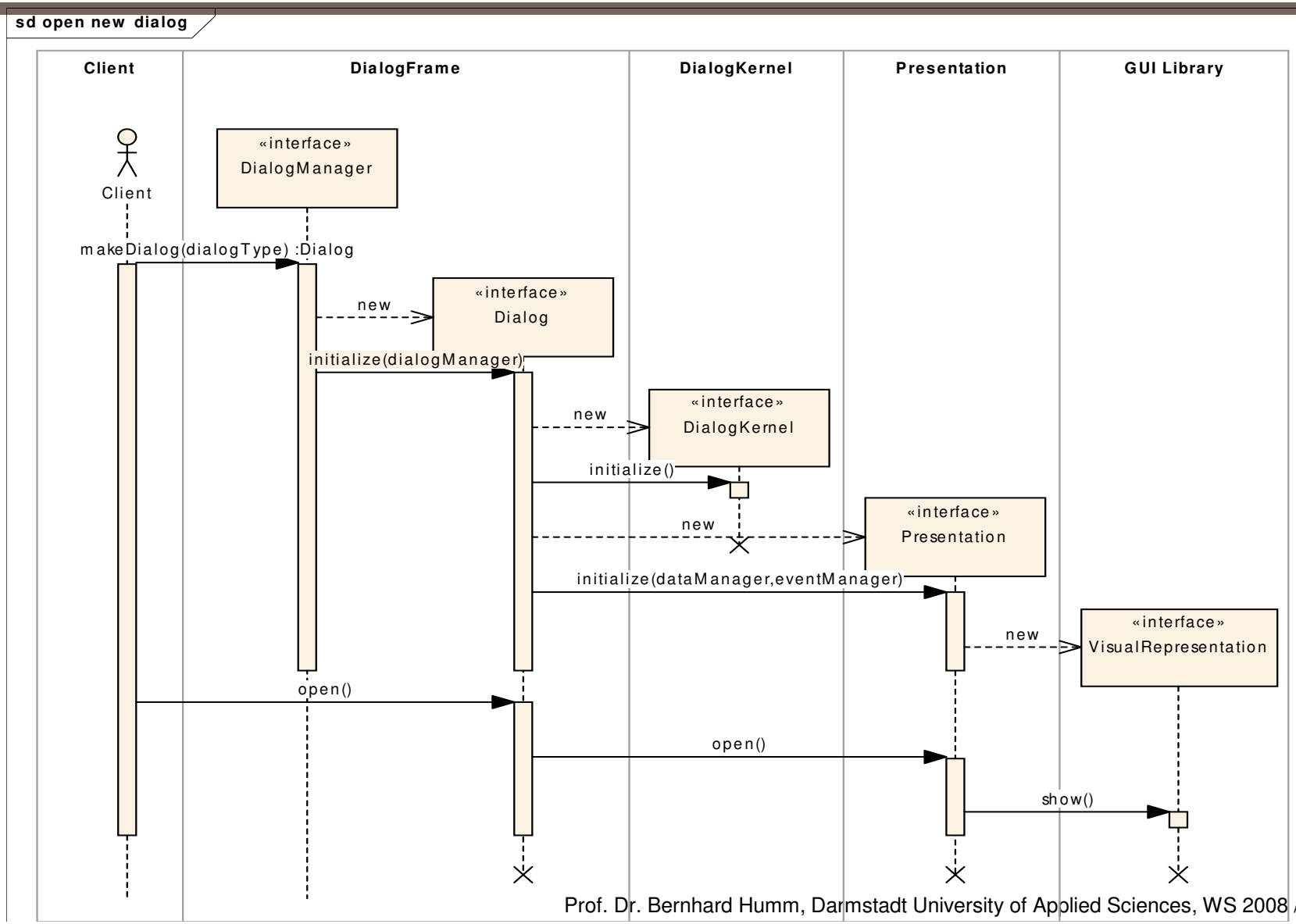
Dialog frame

Application kernel access

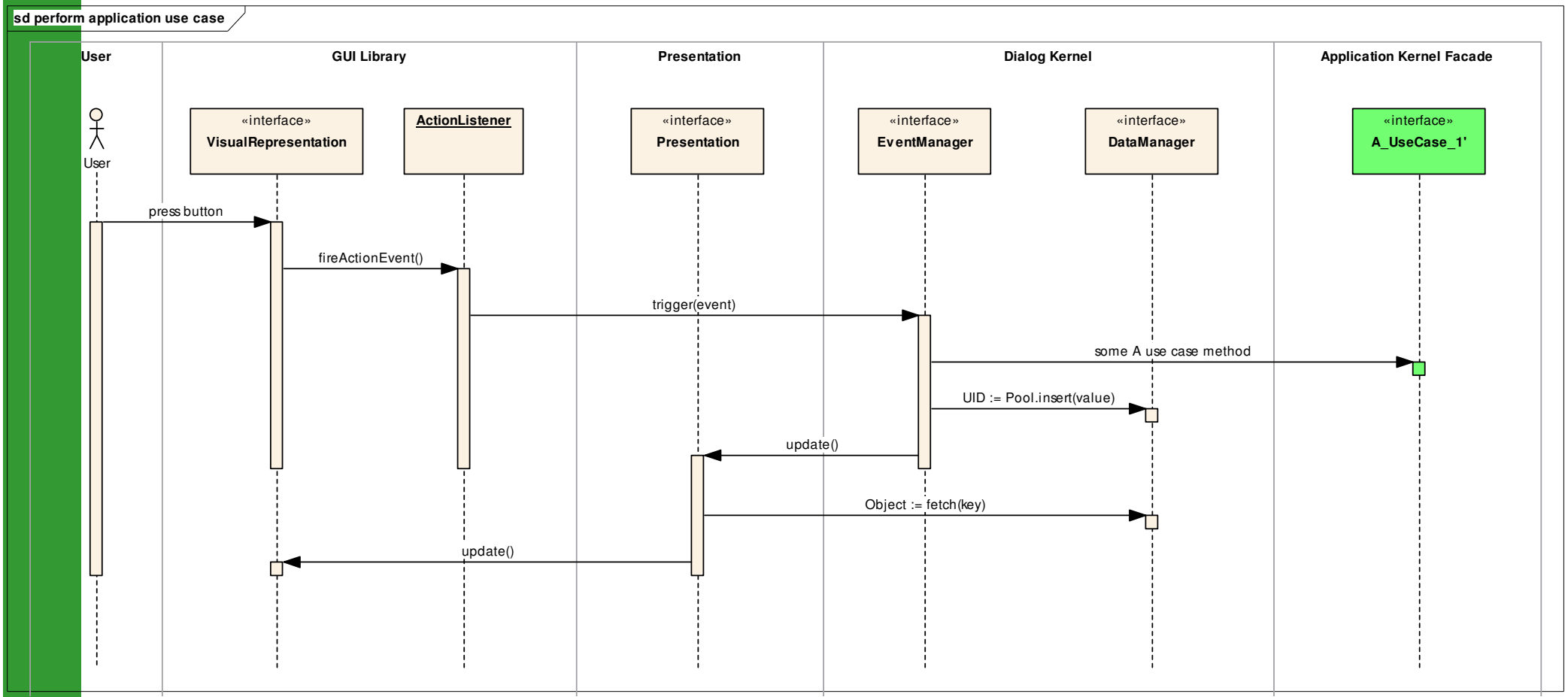
➔ **Sequences**

Literature

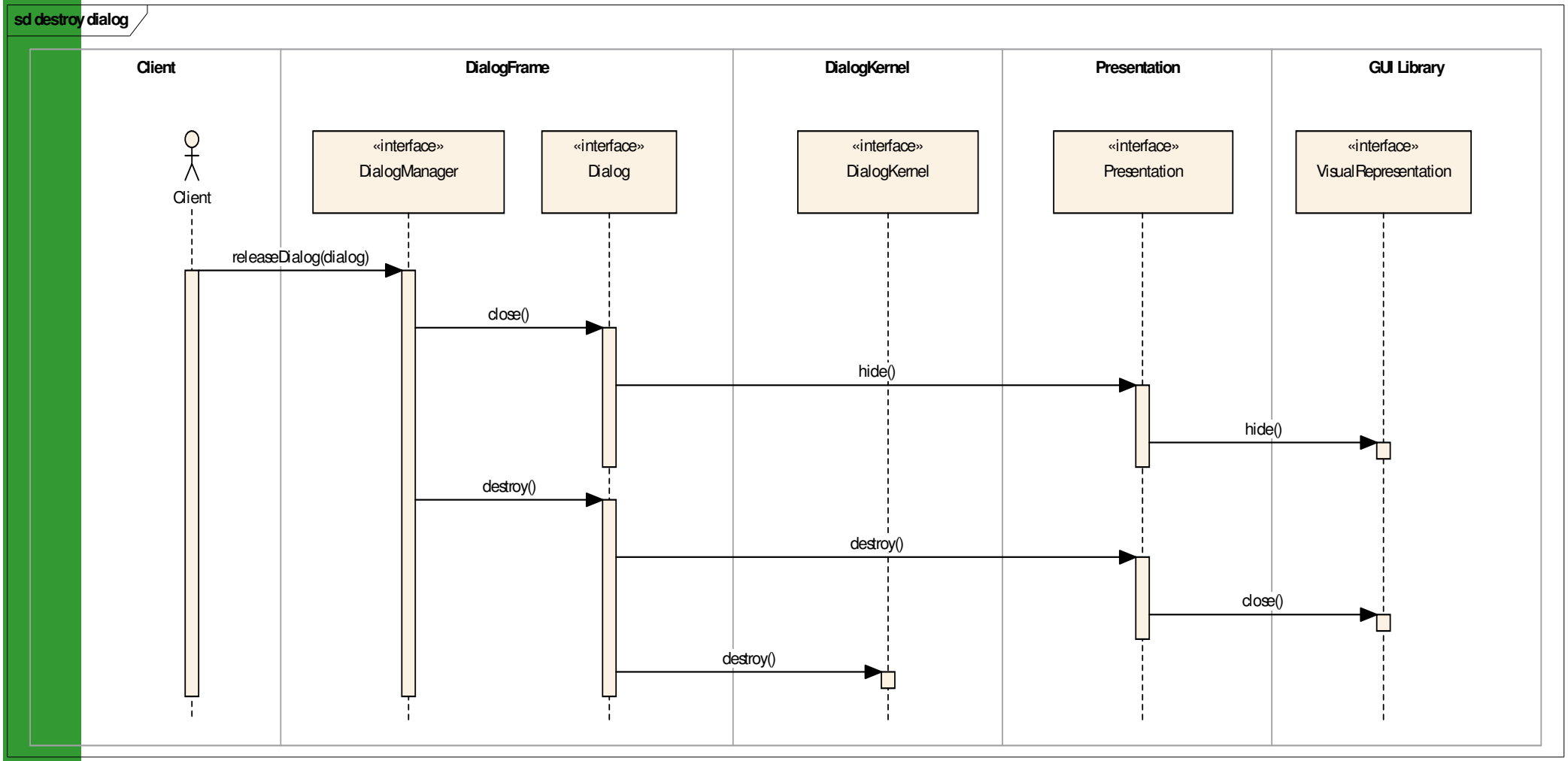
Open new dialog

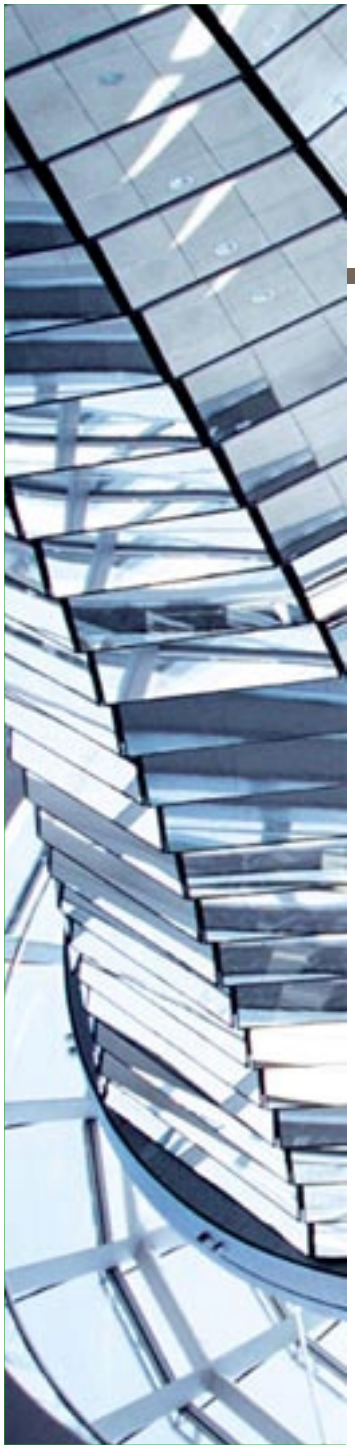


Perform application use case



Destroy dialog





Agenda

Clients

Reference architecture

GUI library

Dialog

Dialog frame

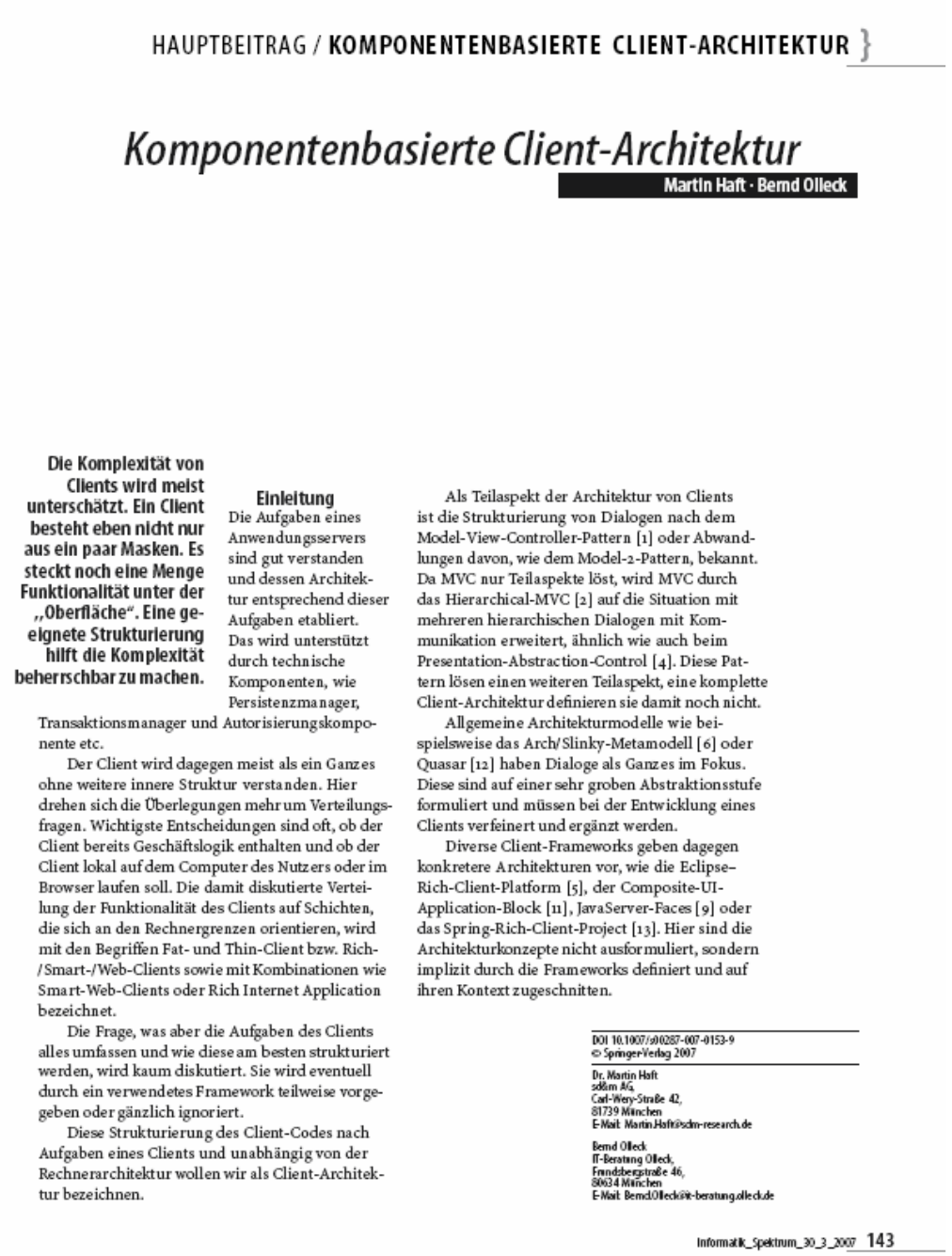
Application kernel access

Sequences

➔ **Literature**

Literature

- Martin Haft, Bernd Olleck: *Komponentenbasierte Client-Architektur*. Informatik Spektrum 30-3-2007, S. 143-158, Springer-Verlag 2007
- (download from SpringerOnline – free from h_da campus)



Komponentenbasierte Client-Architektur

Martin Haft · Bernd Olleck

Die Komplexität von Clients wird meist unterschätzt. Ein Client besteht eben nicht nur aus ein paar Masken. Es steckt noch eine Menge Funktionalität unter der „Oberfläche". Eine geeignete Strukturierung hilft die Komplexität beherrschbar zu machen.

Einleitung

Die Aufgaben eines Anwendungsservers sind gut verstanden und dessen Architektur entsprechend dieser Aufgaben etabliert. Das wird unterstützt durch technische Komponenten, wie Persistenzmanager, Transaktionsmanager und Autorisierungskomponente etc.

Der Client wird dagegen meist als ein Ganzes ohne weitere innere Struktur verstanden. Hier drehen sich die Überlegungen mehr um Verteilungsfragen. Wichtigste Entscheidungen sind oft, ob der Client bereits Geschäftslogik enthalten und ob der Client lokal auf dem Computer des Nutzers oder im Browser laufen soll. Die damit diskutierte Verteilung der Funktionalität des Clients auf Schichten, die sich an den Rechnergrenzen orientieren, wird mit den Begriffen Fat- und Thin-Client bzw. Rich-/Smart-/Web-Clients sowie mit Kombinationen wie Smart-Web-Clients oder Rich Internet Application bezeichnet.

Die Frage, was aber die Aufgaben des Clients alles umfassen und wie diese am besten strukturiert werden, wird kaum diskutiert. Sie wird eventuell durch ein verwendetes Framework teilweise vorgegeben oder gänzlich ignoriert.

Diese Strukturierung des Client-Codes nach Aufgaben eines Clients und unabhängig von der Rechnerarchitektur wollen wir als Client-Architektur bezeichnen.

Als Teilaspekt der Architektur von Clients ist die Strukturierung von Dialogen nach dem Model-View-Controller-Pattern [1] oder Abwandlungen davon, wie dem Model-2-Pattern, bekannt. Da MVC nur Teilaspekte löst, wird MVC durch das Hierarchical-MVC [2] auf die Situation mit mehreren hierarchischen Dialogen mit Kommunikation erweitert, ähnlich wie auch beim Presentation-Abstraction-Control [4]. Diese Pattern lösen einen weiteren Teilaspekt, eine komplette Client-Architektur definieren sie damit noch nicht.

Allgemeine Architekturmodelle wie beispielsweise das Arch/Slinky-Metamodell [6] oder Quasar [12] haben Dialoge als Ganzes im Fokus. Diese sind auf einer sehr groben Abstraktionsstufe formuliert und müssen bei der Entwicklung eines Clients verfeinert und ergänzt werden.

Diverse Client-Frameworks geben dagegen konkretere Architekturen vor, wie die Eclipse–Rich-Client-Platform [5], der Composite-UI-Application-Block [11], JavaServer-Faces [9] oder das Spring-Rich-Client-Project [13]. Hier sind die Architekturkonzepte nicht ausformuliert, sondern implizit durch die Frameworks definiert und auf ihren Kontext zugeschnitten.

DOI 10.1007/s00287-007-0153-9

Dr. Martin Haft
sd&m AG,
Carl-Wery-Straße 42,
81739 München
E-Mail: Martin.Haft@sdm-research.de

Bernd Olleck
IT-Beratung Olleck,
Frundsbergstraße 46,
80634 München
E-Mail: Bernd.Olleck@it-beratung.olleck.de