Design and Implementation of an Internet based Calendar System

Winter semester 2000/2001

Thomas Jungmann &
Reinhold Klapsing

University of Essen

Information systems and software engineering

Univ. Prof. Dr. Rony G. Flatscher

Introduction
Overview of this presentation

- Introduction
- Requirements analysis
- System architecture
- Database design
- System usage (‚walkthrough‘)
- Conclusion and future work

Introduction
Scope of work

- Design and implementation of an internet based calendar system
- Access to calendar with simple web browser
- No groupware-functionality like free/busy-time planning
- No interoperation with other calendar systems

Introduction
Standards

- IETF Working Group „Calendaring and Scheduling“ (calsched)
- Main work:
  - RFC 2445 Internet Calendaring and Scheduling Core Object Specifications (iCalendar)
    - Specifies the objects and data types (MIME-Type text/calendar)
  - RFC 2446 iCalendar Transport-Independent Interoperability Protocol (iTIP)
    - Interoperation of calendar systems using iCalendar Objects
  - Several other RFCs and Internet drafts as well, but all concerning interoperation between calendar systems

Introduction
Definition of terms (1/2)

- Calendar
  - A collection of calendar events associated with a specific user
- Calendar Event
  - An entry in a calendar that represents an event for a specific user
- Calendar User
  - An entity that uses a calendaring system

Introduction
Definition of terms (2/2)

- Calendar User Agent
  - The client application that a Calendar User utilizes to access and manipulate a calendar (the web browser)
- Calendar Service
  - The collection of programs that receive and interpret the Calendar Users commands and also generate and format the output for the user
- Calendar Store
  - The database that stores the calendars

Requirements analysis Overview

- 2 steps of requirements analysis
  - Step 1: Technical considerations must not dominate users needs
    => no technical terms/solutions in mind, only **Users View** in ‚plain english‘
  - Step 2: search for appropriate technical solution for these needs from the **Software Designers** point of view and refinement of needs

Requirements analysis

Users view

- Easy access to the calendar from everywhere, no special software needed (e.g. Internet Café Scenario)

Programmers view

- Calendar User Agent must be standard web browser, communication over HTTP, HTML and CGI only
- Web server must support CGI as well (‚Apache‘ will be used, because it is available on many platforms)

Requirements analysis

Users view

- System must be able to work with multiple users

Programmers view

- Probably large amount of data (incl. meta-data for admin. purposes) => use of a powerful database recommended
- MySQL will be used (relieable, available for many platforms, ANSI SQL 92 Standard used)
- Session management needed to distinguish users (HTTP is stateless)

Requirements analysis

Users view

- Calendar data is private, need for confidentiality => User must authenticate before use

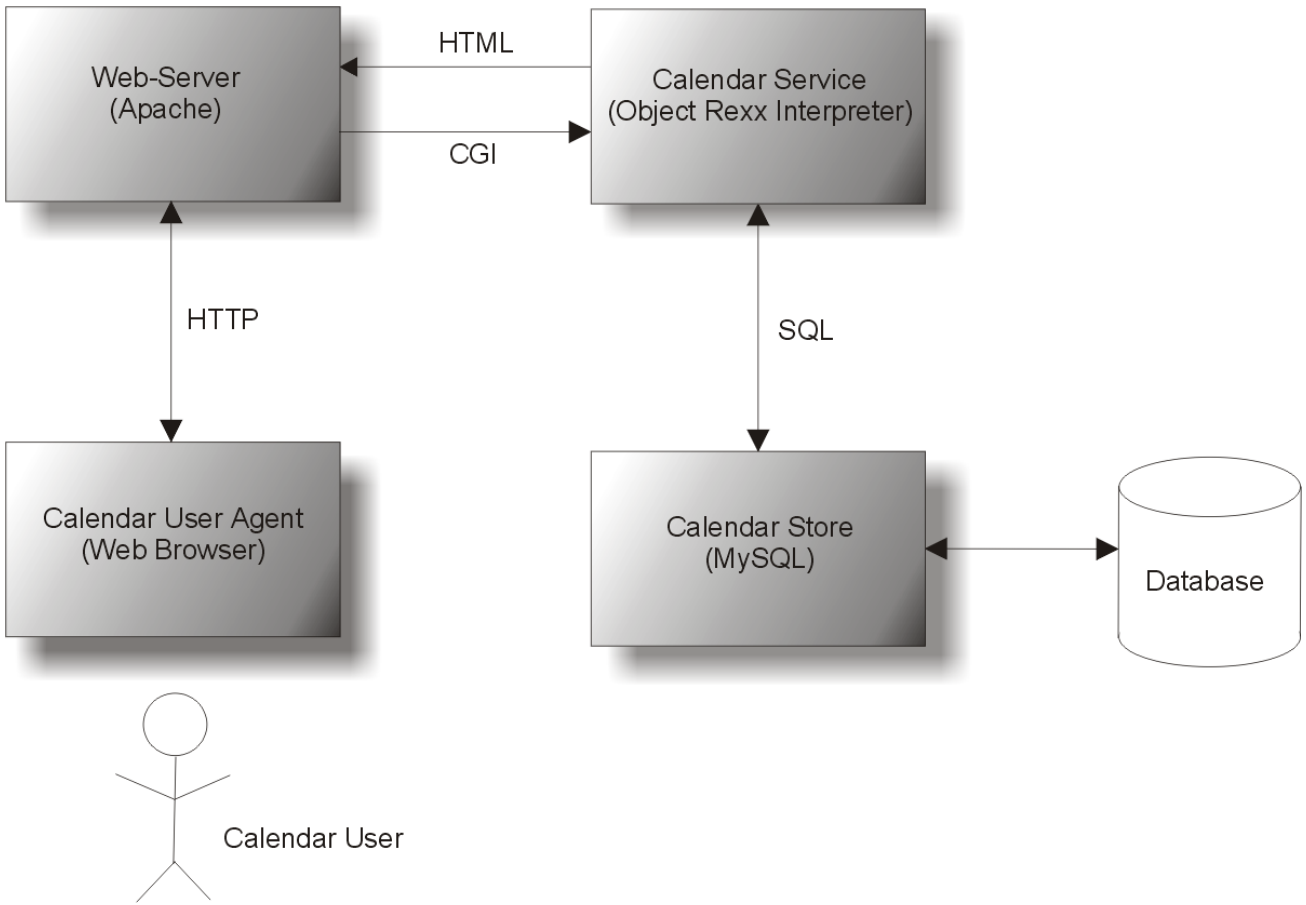
Programmers view

- Password check during logon
- Sessions must timeout after certain time

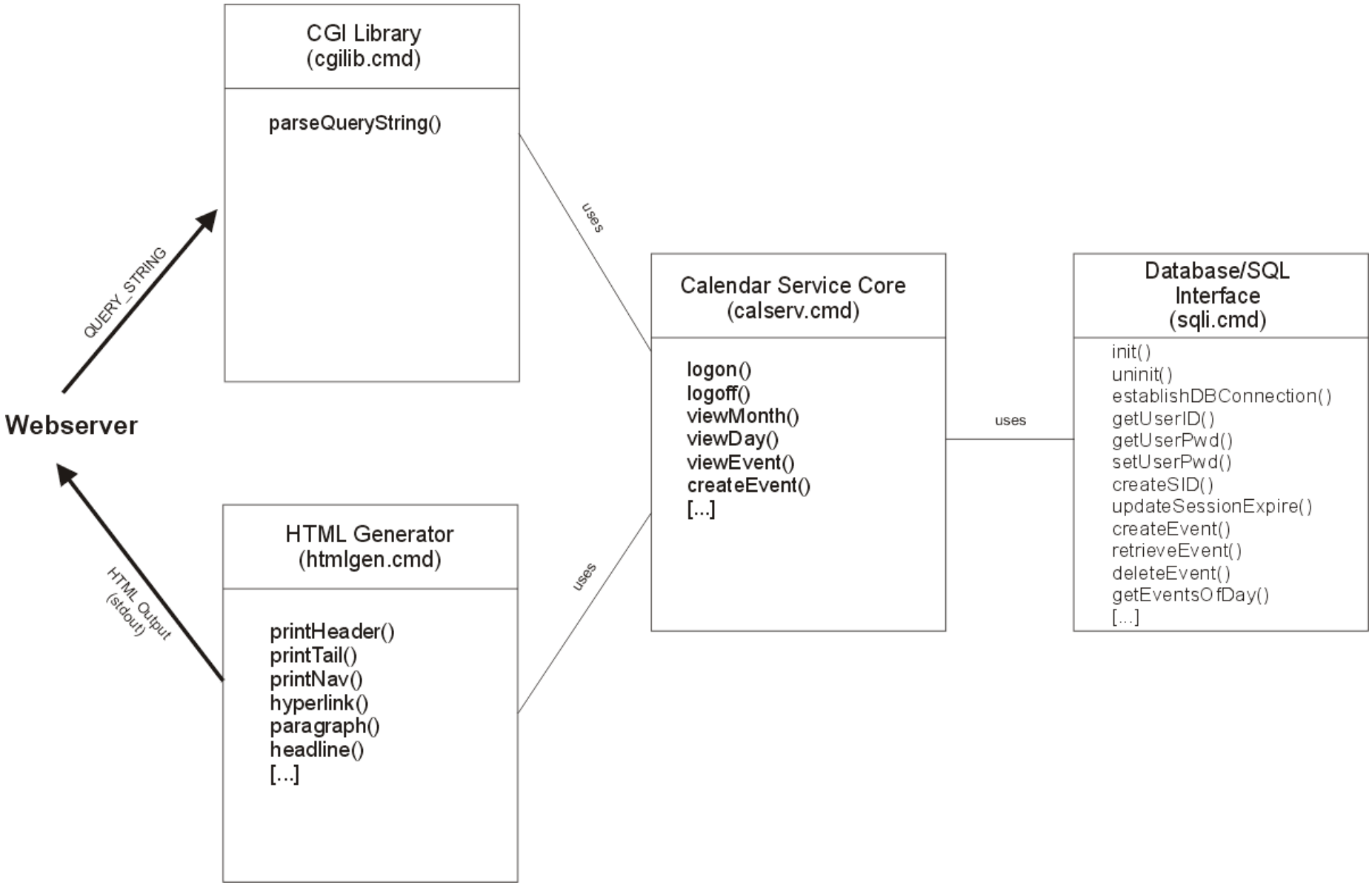
Requirements analysis

- Other (technical) requirements
  - Programming language Object Rexx
    - Scripting language with powerful string parsing functions preferable (because of HTML/CGI)
    - Available on many platforms
    - Interface to many databases available (Rexx/SQL)
- Session management
  - Can be achieved by
    - HTML Hidden Form fields
    - CGI PATH_INFO Mechanism
    - RFC 2109, HTTP State Management Mechanism: „Cookies“

System architecture



System architecture: Diagramm of classes



System architecture
Basic operation principle

- Calendar Service Core is the main module and started by the web server on receiving a CGI-Request
- No direct communication with other components (web server, database)

➡ Performed by interface classes

System architecture Interfaces

- After main script start => creation of interface objects:

```
initVars:   PROCEDURE EXPOSE db html cgi.
    db     = .sqlInterface~NEW
    html   = .HTMLgen~NEW
    parser = .cgiParse~NEW
    cgi.   = parser~parseQueryString()
    return
```

System architecture Interfaces

- Self-Initializing through constructor:

```
::METHOD INIT        /* Constructor */
        if rxFuncQuery("SQLLoadFuncs") then do
            call rxFuncAdd "SQLLoadFuncs","rexxsql","SQLLoadFuncs"
            call SQLLoadFuncs
        end
        self~establishDBConnection()
```

- Advantages of separate interfaces:
  - easy adaption in changing environments
  - easy reuse of code in similar applications

System architecture Interfaces

- SQL/DB-Interface Class
  - Colletion of methods for database access
  - Most methods prepare textstring with SQL-Command, send it to the database and return the results
  - Example:

```
::METHOD getUserName
      parse arg uID
      cmd = "select uniqueName from user where uID='"||uID||"';"
      res = SQLCommand(SQLout,cmd)
      if SQLout.uniqueName.0 then     /* SQLout.uniqueName.0 = Resultcount */
          return SQLout.uniqueName.1
      else
          return -1               /* userID not found */
```

System architecture Interfaces

- CGI-Parse-Interface
  - Has only one public method `CGIParse()`
  - Reads CGI-String from environment
  - Reverses the URL-Encoding
  - Splits variable pairs and assigns them to a Rexx-Stem variable (cgi.), which is then returned to the main script

System architecture Interfaces

- HTML-Generator Class:
  - Writes output of the calendar service as HTML to the standard output
  - Generates Header
  - Simplifies the use of CGI-Forms, hyperlinks and other formatting structures
  - All output is written by this class – no other component writes to the standard output

System architecture Interfaces

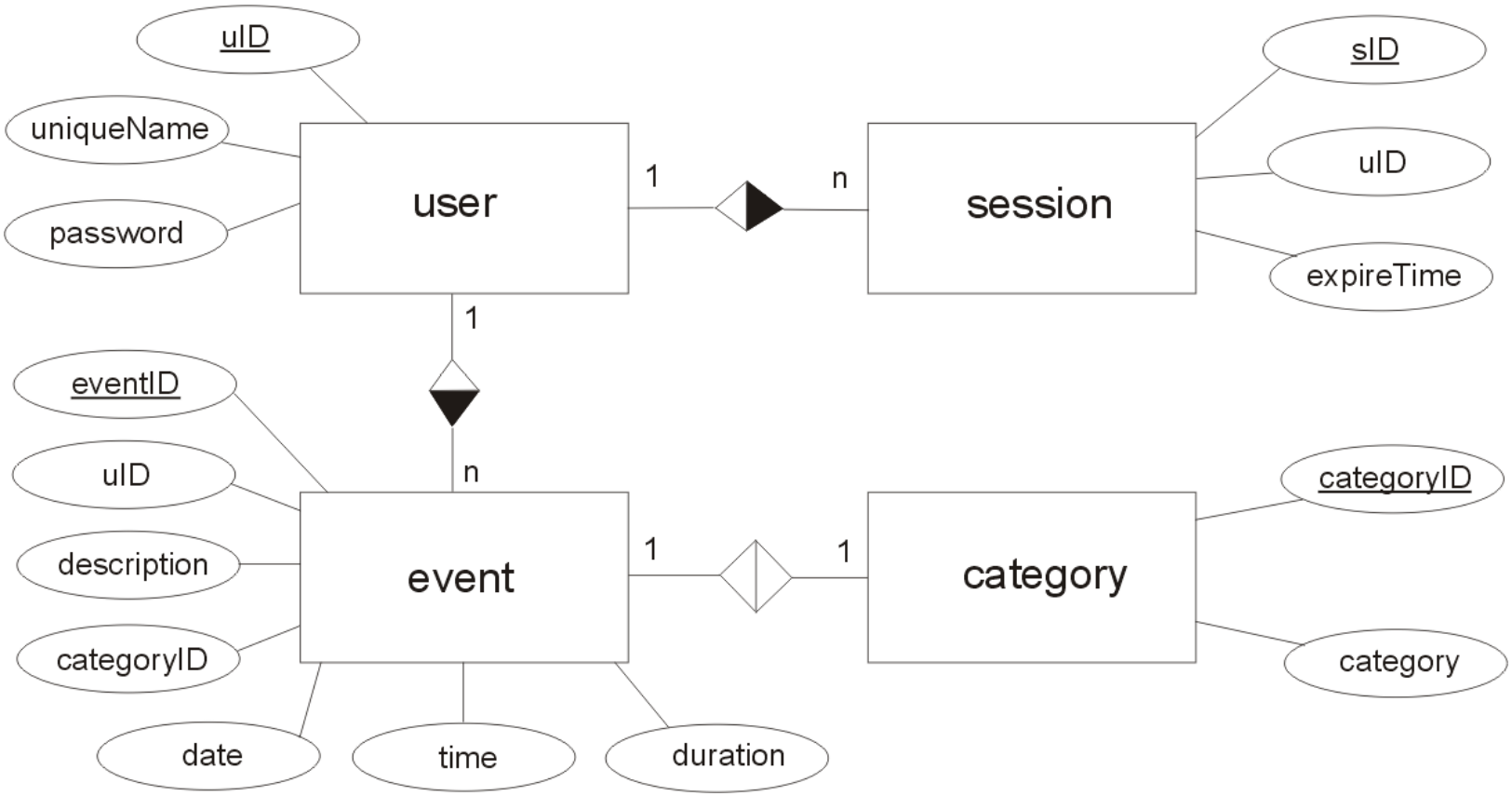
Example from the HTML-Generator Class:

```
::METHOD printHeader
    parse arg title
    sq = d2c(39)
    say "content-type:text/html"
    say
    say "<HTML>"
    say "<HEAD><TITLE>"||title||"</TITLE></HEAD>"
    say "<BODY BGCOLOR='#D0D0D0' TEXT='#006600'>"
```

System architecture
Main Program

```
call initVars
html~printHeader('The Web Calendar')
select
    when cgi.action = 'logon'       then call logon         --check username and password
                                                            --and create session
    when cgi.action = 'newuser'     then call newuser       --create a new user account
    when cgi.action = 'logout'      then call logout        --invalidate session
    when cgi.action = 'overview'    then call overview      --display overview of this month
    when cgi.action = 'goto'        then call gotoMonth     --navigate to specific month
    when cgi.action = 'gotoform'    then call gotoForm      --show HTML-Form for gotoMonth
    when cgi.action = 'viewday'     then call viewDay       --display all events of day
    when cgi.action = 'view'        then call viewEvent     --display event details
    when cgi.action = 'create'      then call createEvent   --add event to database
    when cgi.action = 'newentry'    then call newentryForm  --show HTML-Form for createEvent
    when cgi.action = 'Edit'        then call editEventForm --show HTML-Form for updateEvent
    when cgi.action = 'update'      then call updateEvent   --accept modifications for event
    when cgi.action = 'Delete'      then call deleteEvent   --delete event permanently
    otherwise call abort 'Unknown CGI-Action'
end
html~printTail
DROP db             --drop references to interfaces
DROP html
exit 0
```

Database design
Entity relationship model



Database design
Entities

- Attributes of entity ‚User‘:
  - Name (system wide unique)
  - Password
  - User ID (an alias for the name, both are unique, but a number is easier to handle)

Database design
Entities

- Attributes of entity ‚Event‘:
  - Event ID
  - User ID, a reference to the owner of that event
  - Description, the actual event description
  - Time of event (date and time)
  - Duration
  - Category

Database design
Entities

- Attributes of entity ‚Session‘:
  - Session ID, unique identifier
  - User ID, reference to user who initiated the session
  - Expire time, date and time of session expire
- Attributes of entity ‚Category‘:
  - Category ID
  - Category Name, plain text description of category (Birthday, Meeting, Call etc.)

Database design
Relationship between entities

- 3 binary relations between the four entities
  - Event <> Category: One event belongs to exactly one category, but many events to the same category => connectivity ‚one to many‘
  - User <> Event: Each event belongs to one user, one user can have many events => ‚one to many‘
  - User <> Session: Each session belongs to one user, but how many sessions can a user have? More than one, because it is possible that an old session was not ended properly => ‚one to many‘

Database design
Table definitions

```
mysql> describe user;
+------------+----------+------+-----+---------+-------+
| Field      | Type     | Null | Key | Default | Extra |
+------------+----------+------+-----+---------+-------+
| uID        | int(11)  |      | PRI | 0       |       |
| uniqueName | char(25) |      | UNI |         |       |
| password   | char(25) | YES  |     | NULL    |       |
+------------+----------+------+-----+---------+-------+
```

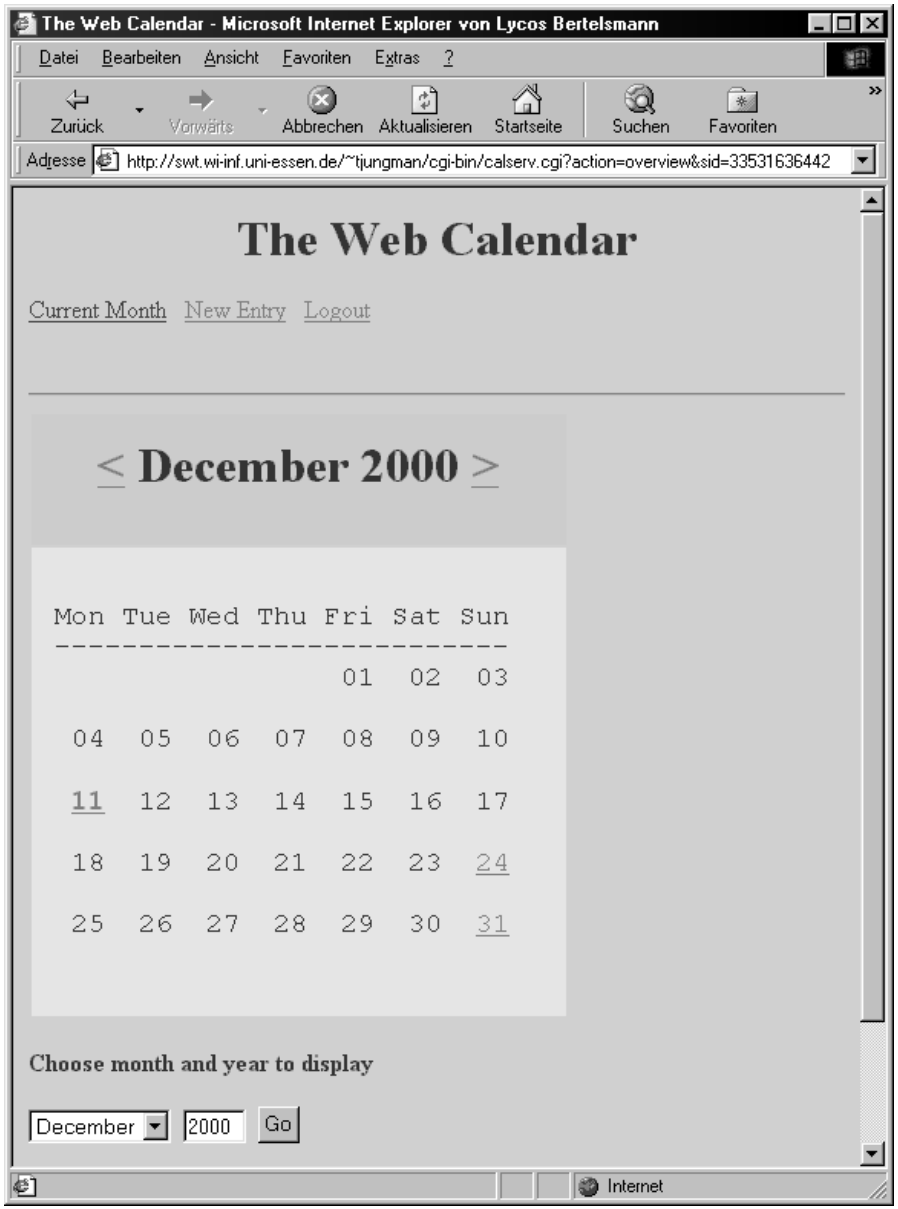
```
mysql> describe event;
+-------------+-----------+------+-----+------------+----------------+
| Field       | Type      | Null | Key | Default    | Extra          |
+-------------+-----------+------+-----+------------+----------------+
| eventID     | int(11)   |      | PRI | 0          | auto_increment |
| uID         | char(25)  |      |     |            |                |
| categoryID  | int(11)   |      |     | NULL       |                |
| description | char(100) | YES  |     | NULL       |                |
| eventDate   | date      |      |     | 0000-00-00 |                |
| eventTime   | time      | YES  |     | NULL       |                |
| duration    | time      | YES  |     | NULL       |                |
+-------------+-----------+------+-----+------------+----------------+
```

Database design
Table definitions

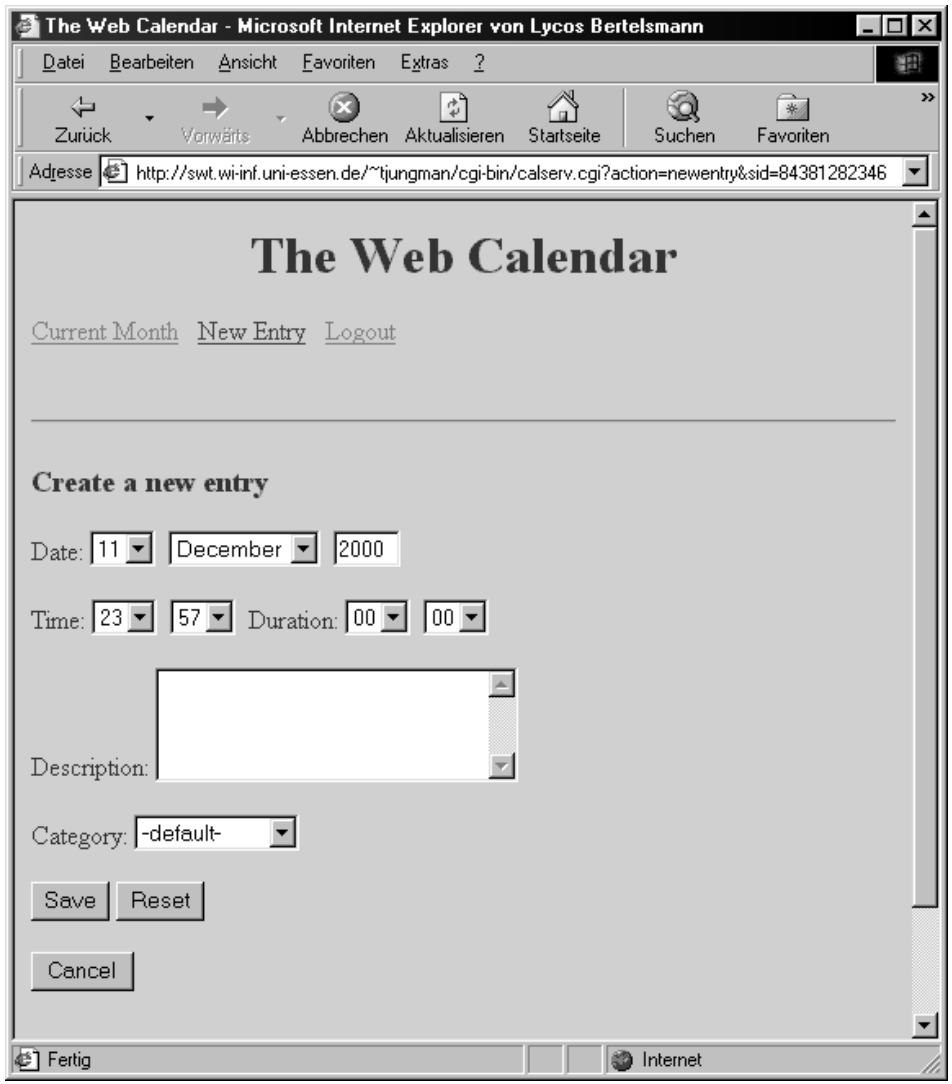
```
mysql> describe category;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| categoryID | int(11)  |      | PRI | 0       | auto_increment |
| category   | char(26) | YES  |     | NULL    |                |
+------------+----------+------+-----+---------+----------------+

mysql> describe session;
+------------+----------+------+-----+---------------------+-------+
| Field      | Type     | Null | Key | Default             | Extra |
+------------+----------+------+-----+---------------------+-------+
| sID        | char(25) |      | PRI |                     |       |
| uID        | char(25) |      |     |                     |       |
| expireTime | datetime |      |     | 0000-00-00 00:00:00 |       |
+------------+----------+------+-----+---------------------+-------+
```

System usage Screenshots

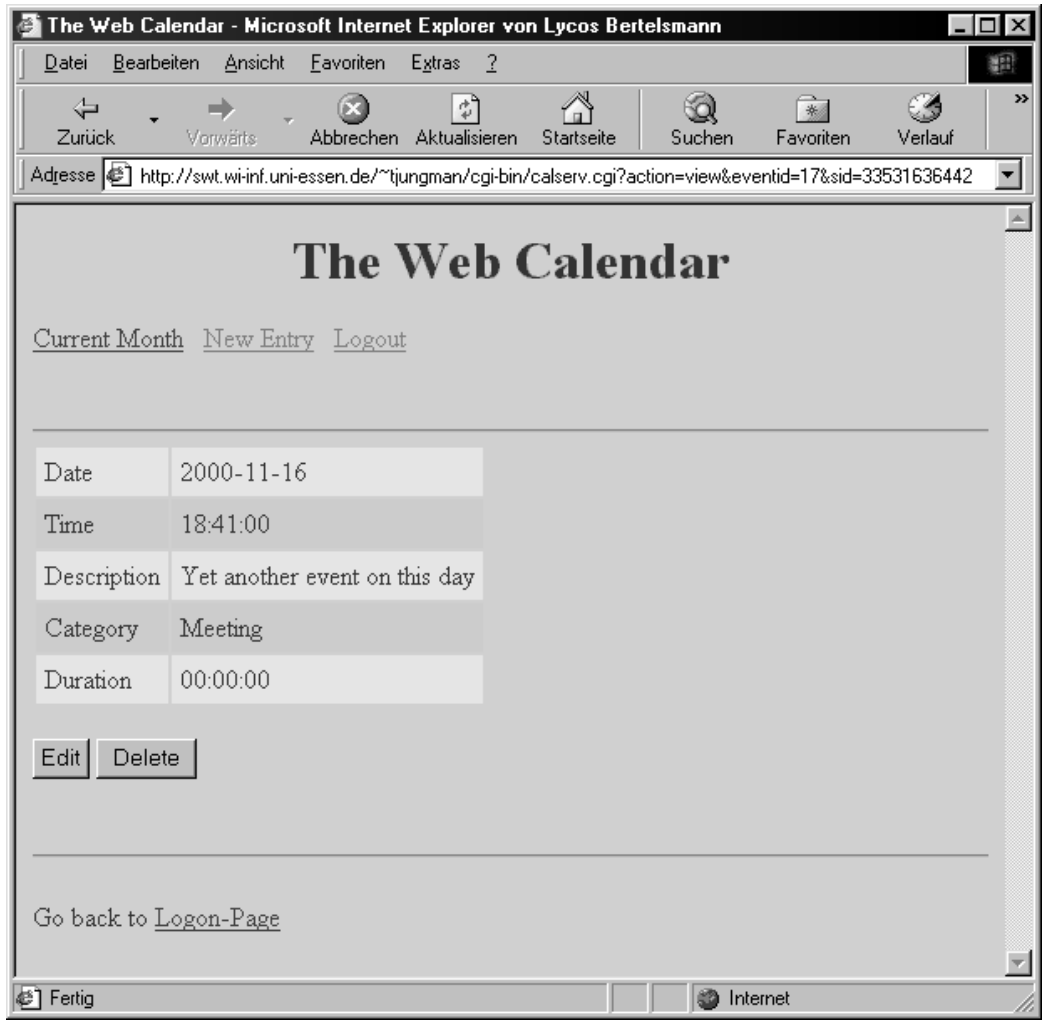
System usage Screenshots



System usage Screenshots



System usage Screenshots



Conclusion and future work

- Interoperation
  - Calendar system is standalone, no interoperations with other systems, no interoperation between users (free/busy schedule)
  - IETF has already released standards for data types and protocols for interoperation
- Conceptual improvements
  - Interface model has not been implemented strictly
  - Exchanging HTML for WML is not easy to do, as there is HTML specific code mixed into the core script

Conclusion and future work

- Object orientation
  - System was written in Object Rexx, but little concepts of the object oriented paradigm were actually used. To much procedural thinking
  - ‚Real‘ OO-Design also possible, e.g. Events have methods to create, alter or delete themselves, User objects have methods to check their passwords, etc.
- Security
  - Based only on passwords and session timeout.
  - Unencrypted, so sniffing attacks possible
  - Even worse: CGI-GET-Method used for data transmission => cache-logfiles store all information
  - Improvement: use of POST-Method
  - Even better: use of Secure Socket Layer SSL

Sum up

- **INTERNET CALENDAR SYSTEM:**
  - Can be used from everywhere, even with a WAP-capable cellular phone
  - All components are freely available (MySQL only for non-comercial use)
  - Distributed system: Web server and Database server can be placed on different machines
  - Easy to use intuitive user interface
  - Year 2000 compliant ;)