COMP519 Web Programming

Lecture 1: Overview of COMP519

Handouts

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Contents

COMP519 Web Programming

- Module co-ordinator: Dr Ullrich Hustadt
  U.Hustadt@liverpool.ac.uk

University home > Computer Science > People > Ullrich Hustadt

Ullrich Hustadt

About

Position

Reader

Address

Department of Computer Science
University of Liverpool
Liverpool L69 3BX
United Kingdom

Contact Details

E-mail: U.Hustadt@liverpool.ac.uk
Phone: +44 (0)151 79-54231
Mobile: +44 (0)7970 24 7431
Fax: +44 (0)151 79-54235

Qualifications

PhD in Computer Science, Universität des Saarlandes, November 1999.
Thesis: Resolution-Based Decision Procedures for Subclasses of First-Order Logic.
MSc in Computer Science, University of Dortmund, March 1991.
Thesis: Unifikation höherer Stufe: Techniken und Anwendungen.

Research

- Research interests
- Papers and Publications
- Project involvement
- Software

Professional Activities

- Scientific initiatives
- Scientific meetings

Teaching

- COMP284
- COMP519
- Supervision of BSc Projects
- Supervision of MSc Projects

Administration

- Duties

Learning Outcomes

By the end of this module, a student should

1. be able to use a range of technologies and programming languages available to organisations and businesses and be able to choose an appropriate architecture for a web application
2. be able to develop reasonably sophisticated client-side web applications using one or more suitable technologies and to make informed and critical decisions in that context
3. be able to develop reasonably sophisticated server-side web applications using one or more suitable technologies and to make informed and critical decisions in that context

Learning Outcomes in a Nutshell

By the end of this module, a student should

- be able to develop web applications

We will cover the following languages

- HTML/CSS
- JavaScript
- PHP

We rely on knowledge of

- programming in general
  $\rightsquigarrow$ acquired via COMP517
- databases (creation, querying, transactions)
  $\rightsquigarrow$ acquired via COMP518

Delivery

- On-campus
  3 hours of lectures per week
  2 hours of practicals per week
  for 10 weeks

  See your personal timetable for details

- Online
  Lectures notes
  Pre-recorded lectures
  Practical worksheets
  available on Canvas

- Study guide
  to indicate how you should proceed through these

Delivery

On Canvas the Study Guide is under 'Modules'

- Week 1
  - Lecture 1: Overview of COMP519 (PDF)
    - Lab Intro 1: Preparation for COMP519 Exercises, Labs, and Assignments
    - Lab Intro 2: Using the Departmental Linux Systems
    - Lab Intro 3: Access Control, Synchronisation and Remote Access
  - Lecture 2: HTML (Part 1): HTTP, HTML5, HTML Elements, HTML Characters (PDF)
  - Lecture 2: HTML (Part 1): HTTP, HTML5, HTML Elements, HTML Characters (Video)
  - Lecture 2 Reading
    - Exercises 1: HTML (1)
  - Lecture 3: HTML (Part 2): Structure, Headings, Lists, Paragraphs, Div, Span, Hyperlinks (PDF)
  - Lecture 3: HTML (Part 2): Structure, Headings, Lists, Paragraphs, Div, Span, Hyperlinks (Video)
  - Lecture 3 Reading

Delivery

- In the practical sessions you work through worksheets with exercises
- A demonstrator will be there to help you with the completion of the exercises
- There are no model solutions for the exercises
  $\rightsquigarrow$ you learn by doing, not by reading
- Completion of exercises will be tracked
  but does not contribute to the module mark
- The exercises prepare you for the assignments

Recommended Books

1. J. Niederst Robbins: Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics (5th ed). O'Reilly, 2018.
2. R. Nixon: Learning PHP, MySQL & JavaScript: with jQuery, CSS & HTML5. O'Reilly, 2018.
3. A. Beautieu: Learning SQL (2nd ed). O'Reilly, 2009.
4. N. C. Zakas: Professional Javascript for Web Developers (2nd ed). Wiley, 2009.

`http://readinglists.liverpool.ac.uk/modules/comp519.html`

Assessment

- Assessment:

  Four programming assignments each worth 25% of the module mark (64 hours, one working day per week)

  1. HTML/CSS
  2. JavaScript
  3. PHP
  4. REST (PHP)

  First three already available on Canvas

- Assignments are like exams

  ⇝ you can ask what something in an assignment means

  ⇝ you cannot ask how to solve an assignment

  ⇝ you cannot ask whether a solution is correct

  ⇝ you cannot help each other

Assessment

- Assignments have equal weight but are not equally difficult

	A1	A2	A3	A4
2017-18	78.9	73.6	46.0	
2018-19	70.7	63.5	59.5	
2019-20	74.2	73.2	59.9	52.8
2020-21	70.2	72.1	44.9	57.2
2021-22	74.8	67.8	51.9	49.8

- Significant downward deviations between years are caused by
  (a) collusion resulting in 0 marks for the students involved
  (b) very low marks due to solutions not being set up correctly on our web server

Web $\neq$ Internet

Internet

A physical network of networks connecting billions of computers and other devices using common protocols (TCP/IP) for sharing and transmitting information

World Wide Web [Old]

A collection of interlinked multimedia documents
(web pages stored on internet connected devices and accessed using a common protocol (HTTP))

Key distinction:

- The internet is hardware plus protocols while the world wide web is software plus protocols
- The world wide web is an application using the internet to transmit information, just like many others, for example, email, SSH, FTP

History (1)

- 1969: ARPANET (precursor of the Internet)
- 1971: First e-mail transmission
- 1971: File Transfer Protocol (FTP)
- 1972: Vadic VA3400 modem (1,200 bit/s over phone network)
- 1977: RSA public-key cryptography
- 1977-79: EPSS/SERCnet (first UK networks between research institutions)
- 1981: IBM PC 5150
- 1981: Hayes Smartmodem (300 bit/s; computer controlled)
- 1982: TCP/IP standardised
- 1985: FTP on TCP standardised

History (2)

- mid 1980s: Janet (UK network between research institutions with 2 Mbit/s backbone and 64 kbit/s access links)
- 1986: U.S. Robotics HST modem (9600 bit/s)
- late 1980s: TCP/IP networks expand across the world
- 1991: Janet adds IP service
- 1991: Gopher / World Wide Web
- 1991: GSM (second generation cellular network) digital, circuit switched network for full duplex voice telephony
- 1995: First public releases of JavaScript and PHP
- 1997: World Wide Web slowly arrives on mobile phones

History (3)

Current Applications:

- Communication via e-mail, Twitter, etc
- Joint manipulation of concepts and actions: Collaborative editing, Crowd sourcing, Wikis (Wikipedia)
- E-Commerce: Online auctions and markets
- Social media, social networks, virtual learning environments

Web $\neq$ Internet

World Wide Web [New]

An infrastructure that allows to easily develop, deploy, and use distributed systems

Distributed systems

A system in which components located on networked computers communicate and coordinate their actions by passing messages in order to achieve a common goal

Web $\neq$ Internet

World Wide Web [New]

An infrastructure that allows to easily develop, deploy, and use distributed systems

Key points:

- The internet already eased the development of distributed systems by providing an appropriate communication infrastructure for that
- The world wide web further eases the development of distributed systems by providing an appropriate infrastructure for computation
- The world wide web then allows every (authorised) person to instantaneously interact with such systems
- Search engines allow users to easily find distributed systems that are useful to them

Distributed Systems: Fundamental Questions

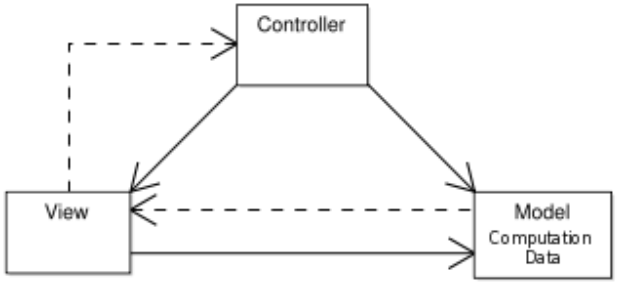
Software developers have to consider a wide, but rather stable, range of questions including:

- Where can or should computations take place?
- Where can or should data be stored?
- How fast can data be transferred/communicated?
- What is the cost of data storage/computations/communication depending on how/where we do it?
- How robustly/securely can data storage/computations/communication be done depending on how/where we do it?
- How much energy is available to support data storage/computations/communication depending on how/where we do it?
- What is the legality of data storage/computations/communications depending on how/where we do it?

The possible answers to each of these questions is also rather stable, but the 'right' answers change

Distributed Systems: Model-View-Controller

We use the Model-View-Controller software design pattern to discuss some of these questions in more detail:

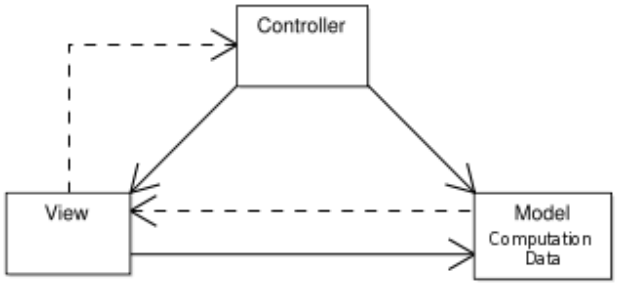


- The model manages the behaviour and data
- The view renders the model into a form suitable for interaction
- The controller receives user input and translates it into instructions for the model

1. Where should the view be rendered?
   - On the user's computer
   - On a central server (farm) possibly shared by a multitude of users

Distributed Systems: Model-View-Controller

We use the Model-View-Controller software design pattern to discuss some of these questions in more detail:

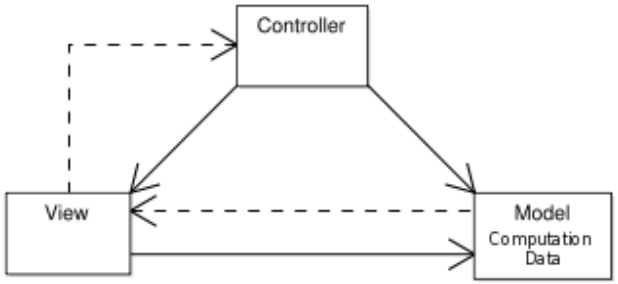


- The model manages the behaviour and data
- The view renders the model into a form suitable for interaction
- The controller receives user input and translates it into instructions for the model

2. Where should the behaviour of the model be computed?
   - Close to the user,
     on a single computer exclusively used by the user
   - Away from the user,
     on a central server (farm) shared by a multitude of users
   - Distributed,
     on several computers owned by a large group of users

Distributed Systems: Model-View-Controller

We use the Model-View-Controller software design pattern to discuss some of these questions in more detail:



- The model manages the behaviour and data
- The view renders the model into a form suitable for interaction
- The controller receives user input and translates it into instructions for the model

3. Where should the data for the model be held?
   - Close to the user,
     on a single computer exclusively used by the user
   - Away from the user,
     on a central server (farm) shared by a multitude of users
   - Distributed,
     on several computers owned by a large group of users

Distributed Systems: Fundamental Questions

- Software developers have to consider a wide, but rather stable, range of questions
- The possible answers to each of these questions is also rather stable
- The 'right' answer to each these questions will depend on
  - the domain in which the question is posed
  - available technology
  - available resources

- The 'right' answer to each of the questions changes over time
- We may go back and forth between the various answers

- The reasons for that are not purely technological, but includes
  - legal factors
  - social factors
  - economic factors

The Pre-PC Era

- 1960ies: Computer terminals start to be used to interact with computers
- 1968: NLS "oN-Line System" (Douglas Engelbart, SRI)

A 'networked' computer system with GUI, off-line mode, 'e-mail', collaborative word processing, hypertext, video conferencing and mouse is demonstrated

(The picture shows one of several terminals connected to a mainframe computer)

Videos of the demo are available at
`http://www.youtube.com/watch?v=JfIgzSoTMOs`

The Pre-PC Era

- 1970ies: Computer terminals continue to dominate
- 1978: DEC VT100

  Intel 8080 processor
  3 kb main memory
  Monochrome graphics

  Like NLS, this is a terminal connected to a mainframe computer via serial lines

Key points:

- The data is stored on the mainframe computer which also computes the behaviour of the model
- The view is computed on the mainframe computer and only displayed on the terminal
- The terminal receives user inputs and relays it to the mainframe computer that translates it into instructions for the model
- This architecture dominated the industry for about 20 years

The PC Era

- 1981: IBM PC 5150
- 1983: Apple Lisa
  First PC with a graphical user interface
- 1985: Microsoft Windows 1.0
- 1987: HyperCard
  Hypermedia system for Mac OS
- 1988: HyperStudio
  HyperCard clone for MS Windows
- 1991: Instant Update
  Collaborative editor for Mac OS
- 1992: CU-SeeMe Video Conferencing

Key points:

- Model, View and Controller are stored and computed locally on the PC
- It took 24 years to catch up with NLS
- This architecture dominated the industry for about 20 years

The Post-PC Era

- 1992: IBM Simon Personal Communicator (First smartphone)
- 1996: Nokia 9000 Communicator
- 2007: Apple iPhone
  Samsung 32-bit RISC ARM
  128MB main memory
  4-16GB flash memory
  'Apps' / Web browser

- 2011: Google Chromebook
  Intel Atom processor
  2GB main memory
  16GB SSD
  Web-based applications

In effect the Chromebook is a 'terminal' connected to Google's servers and others via a wireless network

The Post-PC Era

- 2011: Google Chromebook
  Intel Atom processor
  2GB main memory
  16GB SSD
  Web-based applications

Key points:

- The data is stored on a server farm (the 'cloud') which also computes the behaviour of the model
- The view is either computed on a server farm or on the terminal
- The terminal receives user inputs and either relays those to the server farm or directly translates it into instructions for the model
- This architecture has fought for dominance for 15 years
- Will it dominate the future?

Thin clients, fat clients and cloud clients

- The Google Chromebook gives very similar answers to the fundamental questions as the DEC VT100
  ⇝ the possible answers to the fundamental questions stay the same
- The PC gave very different answers to the fundamental questions
  ⇝ the 'right' answers change with time
- The Google Chromebook is more advanced than the DEC VT100 in (almost) every aspect
  ⇝ we are not going around in circles,
  we always advance technologically

Web Programming versus App Programming

- Web Programming relies on web browsers as means to render user interfaces that are coded in HTML/CSS
- Web Programming relies on HTTP as the main protocol to exchange information within a distributed system
- Web-based apps use a mix of server-side and client-side computing
- Web-based apps can be changed almost instantaneously and on a per-user / per-use basis
- App Programming relies on directly coded 'native' interfaces (Swift/Java)
- App Programming can rely on arbitrary protocols to exchange information within a distributed system
- Programmers have more flexibility and more control when developing 'traditional' apps

It is not obvious which approach is better and in which situation