Creation

Jörg Cassens

Contextualized Computing and Ambient Intelligent Systems

Universität Hildesheim
medieninformatik
IMAI – Institut für Mathematik und Angewandte Informatik

1 Tutorial 1

Assignment 3.1: Dourish & Anderson

- Required reading for week 3
  - Dourish, Paul, and Ken Anderson. *Collective information practice: exploring privacy and security as social and cultural phenomena*. Human-computer interaction 21.3 (2006): 319-342.
- The text will be discussed in the tutorial 07.05.2018
- Course readings can be downloaded in the learnweb
- Every text has a wiki-page in the learnweb
  - Use it to describe the text
  - Use it to link the text to the course
- Results of the discussion may also be written up

Descriptive Framework Version 2

Contextualisation

- Contextual Parameter
  - Environment – things, services, people
  - Personal – mental & physical information about user
  - Social – roles & relations
  - Task – what is the user doing
  - Spatio-Temporal – when & where are we
  - Other
- Kind of Contextualisation
  - Awareness – what aspects are taken into account?
  - Sensitivity – what aspects are changed?

Descriptive Framework Version 2

Ambience

- Perception
  - Mediality – media types
  - Codality – semantic representation
  - Modality – human senses
- Reasoning
  - *Context Awareness*
  - *Context Sensitivity*
  - Other
- Action
  - Mediality – media types
  - Codality – semantic representation
  - Modality – human senses

Descriptive Framework Version 2

Interaction

- Implicit vs. Explicit
  - Implicit input – through behaviour
  - Explicit input – through use voice/gesture/keyboard…
  - Implicit output – change of material setting
  - Explicit output – confirmations, texts…
- Weaviness (work in progress)
  - Is the system woven into the background?
- Naturalness (work in progress)
  - Is there a "most natural" way to express something?
- Enabling social interaction (work in progress)

2 Introduction

Prevalent Paradigm
Prevalent computing paradigm designed for personal information management

- desktops and laptops with fixed configurations of mouse, keyboard, and monitor
- dedicated network services with fixed network addresses and locations
  - printers
  - file servers
  - …
- direct manipulation interfaces
  - representation and manipulation of files, documents, and applications

Ambient Paradigm

Different paradigm of computing environment

- heterogeneous set of devices
  - invisible computers embedded in everyday objects such as cars and furniture
  - mobile devices such as smartphones
  - personal devices such as laptops
  - very large devices such as wall-sized displays and tabletop computers situated in the environments and buildings we inhabit
- All have different operating systems, networking interfaces, input capabilities, and displays
  - some are designed for end user interaction
  - other devices, such as sensors, are not used directly by end users

Ambient Interaction

- interaction mode goes beyond the one-to-one model prevalent for PCs
  - many-to-many model where the same person uses multiple devices
  - and several persons may use the same device
- interaction may be implicit, invisible, or through sensing natural interactions such as speech, gesture, or presence
  - wide range of sensors is required, both sensors built into the devices as well as sensors embedded in the environment
- location tracking devices, cameras, and accelerometers can be used to detect who is in a place and deduce what they are doing
  - provide the user with information relevant in a specific location
  - adapt their device to a local environment or the local environment to them
- Networking is often wireless and ad hoc

Focus

- research is concerned with the underlying technologies and infrastructures that enable the creation and deployment of these types of applications
- addresses a wide range of questions such as:
  - hardware for sensor and actuator platforms
  - operating systems for such platforms
  - allow devices to find each other and use the services on each other
  - design systems support for resource impoverished devices
  - distributed infrastructures for seamless mobility
  - modelling and using context
  - reasoning in, with, and for context
  - applications for such settings as smart rooms and hospitals

Creating Systems

- Building systems is essential to the progress of the field as a whole
- Experimentally prototyping systems enables us to
  - experience them, discover what they are like to use,
- and reason about core precepts
  - boundaries of the system
  - its invisibility
  - the role of its users
  - the degree of artificial intelligence endemic to it

3 Topics & Challenges

Resource-Constrained Devices

- wide range of new devices are built and introduced, which often are resource-constrained
- devices such as mobile phones and music players have limited CPU, memory, and network connectivity compared to a standard PC
- embedded platforms such as sensor networks and smart cards are very limited compared to a PC or even a smartphone
- it is important to recognize the constraints of the target devices, and to recognize that hardware platforms are highly heterogeneous and incompatible with respect to
  - hardware specifications
  - operating system
  - input/output capabilities
  - network
  - ...

Resource-aware Computing

- Resource-aware computing is an approach to develop technologies where the application is constantly notified about the consumption of vital resources
- can help the application (or the user) to take a decision based on available resources now and in the future
  - video streaming will be adjusted to available bandwidth and battery level
  - the user may be asked to go to an area with better wireless local area network coverage
- one of the main hardware constraints to consider when building ubicomp systems and applications is power consumption and/or opportunities for energy harvesting

Energy Harvesting

- Power foraging: technologies for harvesting power in the environment
  - for example, kinetic energy from a walking person
- Cyber foraging is a similar research theme where devices look for places to offload resource-intensive tasks
  - for example, google chromecasts initiated from a phone stream directly from the server to the speaker
- major drain on the battery is wireless communication (also typical for mobile or embedded devices)
  - resource-efficient networking protocols that limit power consumption due to transmitting data, while maintaining a high degree of throughput and reliability
  - processing consumes much less power than communication, mobile ad hoc sensor networks seek to do as much in-network processing as possible (aggregating or averaging values from nearby nodes), and filtering before transmitting values

Volatile Execution Environments

- service discovery: technologies and standards that enable devices to
  - discover each other
  - set up communication links
  - use each others' services
- when a portable device enters a smart room, it may want to discover and use nearby resources such as public displays and printers

  - Several service discovery technologies are in daily use
  - Jini, UPnP, DLNA, Bonjour/multicast DNS (mDNS), and the Bluetooth discovery protocol
- Nevertheless, several challenges to existing approaches still exist, including
  - lack of support for multicast discovery beyond local area networks
  - lack of support beyond one-to-one device/service pairing
  - rather cumbersome methods for pairing devices

Distribution

- systems often distributed; they entail interaction between different devices
  - mobile, embedded, or server-based
- these devices have different networking capabilities
- Spontaneous
  - devices continuously connect and disconnect
  - create and destroy communications links
- from a communication perspective, these devices may leave the room (or run out of battery) at any time
- therefore, communication between the mobile devices and the services in the smart room needs to gracefully handle such disconnection

Change in Communication Structure

- Another type of volatility arises due to changes in the underlying communication structure
  - topology, bandwidth, routing, and host naming
- in an ad hoc sensor network, the network topology and routing scheme is often determined by
  - nodes available at a given time, the physical proximity of the nodes in the network, their current workload, and battery status
- devices entering the room do not know the network name or addresses of the local services
- services discovery would entail obtaining some network route to the service
- unlike most traditional distributed systems, the connectivity changes in ambient systems are common rather than exceptional, and often of a more basic nature

Heterogeneous Execution Environments

- applications often involve a wide range of hardware, network technology, operating systems, input/output capabilities, resources, sensors, etc.
- in contrast to the traditional use of the term application, which typically refers to software that resides on one to three physical nodes, a ubiquitous application typically spans several devices, which need to interact closely and in concert in order to make up the application
  - a Smart Room is an application that relies on several devices, services, communication links, software components, and end user applications, which needs to work in complete concert
- handling heterogeneity is not only a matter of being able to compile, build, and deploy an application on different target platforms—such as building a desktop application to run on different versions of Windows, Mac OS, and Linux

Varying Specifications

- to a large degree a matter of continuously – at runtime – being able to handle heterogeneous execution environments
- different parts of the application run on devices with highly varying specifications
- when a user enters the smart room and wants to access the public display and print a document, this may involve a wide range of heterogeneous devices, each with their specific hardware, operating systems, networks interfaces, etc.
  - the user may be carrying a smartphone running iOS
  - he may be detected by a location tracking system based on infrared sensors running the TinyOS
  - his Windows-laptop may use mDNS for device discovery
  - the public display may be running Linux using the X protocol for sharing its display with nearby devices

Standards

- challenge of heterogeneity partly arises because a standard technology stack including hardware, operating system, etc., has yet to mature
  - partly true, and existing or new technology platforms may gradually be able to handle the requirements in a more homogeneous and consistent manner
- On the other hand, there will always be a need to use different kinds of technologies
  - from small, embedded sensors, to large public display and mobile hand- held devices
- heterogeneous hardware devices are a fundamental part of ambient applications
- the corresponding operating systems and software stacks need to be specifically optimized to this hardware
  - small sensor nodes need a software stack optimized for their limited resources
  - large display similarly needs a software stack suited for sophisticated graphics and advanced input technologies

Fluctuating Usage Environments

- Contemporary computing is primarily targeted at information management in the broadest sense
  - Users use PCs for information management locally or on servers
  - they engage in a one-to-one relationship with the PC
  - the physical use context is fairly stable and is often tied to a horizontal surface such as an office desk or the dining table at home
  - Even the use of smartphones is fairly stable although they are used in different and often less-than-ideal environments
  - the number and complexity of peripherals are limited and include well-known devices such as printers, external hard drives, cameras, and servers

Ambient Complexity

- ambient systems live in a far more complicated and fluctuating usage environment
- Users have not one but several personal devices, such as laptops, mobile phones, watches, etc
- The same device may be used by several users, such as the public display in the smart room or a smart blood pressure monitor in the patient ward of a hospital
  - a need to support this many-to-many configuration between users and devices
- compared to the desktop, the physical (work) setting exhibits a larger degree of alternation between many different places and physical surroundings
  - Mobile devices mean that work can be carried around and done in different places

  - computers embedded into, for example, furniture that is constantly used by different people
- doing a task is no longer tied to one device such as the PC, but is now distributed across several heterogeneous devices as explained above
- This means that users need technology that helps them stay focused on a task without having to deal with the complexity of setting up devices, pairing them, moving data, ensuring connectivity, etc

Invisible Computing

- Handling and/or achieving invisibility is a core challenge
  - For example, monitoring human behaviour at home and providing smart home control of the heating systems
- In many of these cases, the computers are invisible to the users in a double sense
  - the computers are embedded into buildings, furniture, medical devices, etc., and are as such physically invisible to the human eye
  - Second, the computers operate in the periphery of the users' attention and are hence mentally invisible (imperceptible).
- From a systems perspective, obtaining and handling invisible computing is a fundamental change from traditional computing

Invisibility as Fundamental Change

- traditional systems rely heavily on having the users' attention;
  - users either use a computer or they don't
- This means, for example:
  - the system software can rely on sending notifications and error messages to users, and expect them to react
  - ask for input in the contingency where the system needs feedback in order to decide on further actions
  - ask the user to install hardware and/or software components
  - can ask the user to restart the device
- Moving toward invisible computing, these assumptions completely break down
- Mitigation strategies include autonomic computing, contingency management and graceful degradation

Autonomic Computing

- Autonomic computing aims to develop computer systems capable of self-management
- Goal: overcome complexity of computing systems management
- Autonomic computing refers to adapting to unpredictable changes while hiding intrinsic complexity for the users
- An autonomic system makes decisions on its own, using high-level policies
- Whereas autonomic computing is, to a large degree, conceived with server architectures in mind, multiagent systems research seeks to create software agents that work on behalf of users while migrating around in the network onto different devices
- Agents are designed to ensure that lower-level systems issues are shielded from the user, thereby maintaining invisibility of the technology

Contingency Management

- Research on contingency management seeks to prevent users from being involved in attending to errors and failures
- Traditional exception handling assumes that failures are exceptional
- contingency management views failures as a natural contingent part of running a system
- For ambient systems, techniques for proactive management of failures and resource limitations need to be put into place
- For example, off-loading an agent before a mobile device runs out of battery, and ensure proactive download of resources before leaving network coverage

Graceful Degradation

- Graceful degradation addresses how the system responds to changes
  - in particular, failures in the environment
- Most existing technology assume the availability of certain resources such as Internet connectivity and specific servers to be present permanently
- in situations where these resources are not available, the entire system may stop working
- Real life demands systems that can cope with the lack of resources
- systems should be able to adapt gracefully to these changes, preserving as much functionality as possible using the resources that are available

Security & Privacy

- Security and privacy is challenging to all computing
- For ambient systems, security and privacy challenges are increased due to the volatile, spontaneous, heterogeneous, and invisible nature
  - particularly imperceptible monitoring

Security & Privacy: Trust

Trust
First, trust is often lowered in volatile systems because the principals whose components interact spontaneously may have no a priori knowledge of each other and may not have a trusted third party

- a new device that enters a hospital cannot be trusted to be used for displaying or storing sensitive medical data, and making the necessary configuration may be an administrative overhead that would prevent any sort of spontaneous use
- Hence, using the patient's mobile phone may be difficult to set up

Security & Privacy: Assumptions

Assumptions
Second, conventional security protocols tend to make assumptions about devices and connectivity that may not hold

- portable devices more easily stolen and tampered with
- resource-constrained embedded devices may not have sufficient computing resources for asymmetric public key cryptography
- software does often not get updated after initial release
  - "fire and forget"-strategy of vendors for cheap hardware
  - incompetence
- Many security protocols cannot rely on continuous online access to a server, which makes it hard to issue and revoke certificates

Security & Privacy: Context

Context
Third, the nature of ambient systems creates the need for a new type of security based on location and context; service authentication and authorization may be based on context and not the user

- people entering a cafe may be allowed to use the café's printer
- if a device wants to use the café's printer, it needs to be verified that this device indeed is inside the cafe
- it does not matter who uses the printer, the cafe cares only about where the user is
- Vice versa, the customer only cares that he connects to the printer in the café

Security & Privacy: User Data

Sensors
Fourth, new privacy challenges emerge

- By introducing sensor technology, ambient systems may gather extensive data on users including information on
  - location, activity, social interaction, speech, video, and biological data
- if these systems are invisible in the environment, people may not even notice that data are being collected
- hence, designing appropriate privacy protection mechanisms is central
- key challenge is to manage that users provide numerous identifiers to the environment while moving around and using services
  - networking IDs such as MAC, Bluetooth, and IP addresses
  - (user-) names
  - IDs of tags such as RFID tags
  - payment IDs such as credit card numbers

Security & Privacy: Fluctuations

Fluctuations
Fifth, the fluctuating usage scenarios also set up new challenges for security

- numerous devices, users continuously create new associations
- if all or some of these associations need to be secured, device and user authentication happens very often
- Existing user authentication mechanisms are, to a large degree, designed for few (1–2) and long-lived (hours) associations between a user and a device or service
  - a user logs into a PC and uses it for the whole workday
- in ambient scenarios, users may enter a smart room and use tens of devices and services in a short period (minutes)
  - traditional user authentication e.g. using user names and passwords not feasible
  - Moreover, if the devices are embedded or invisible, it may be difficult and awkward

4 Creating Systems

Why?

- Prototyping future systems to explore ubiquity in practice
- Empirical exploration of user reactions
- Gathering datasets to tackle computational problems
- Creating experiences for public engagement or performance

- Creating research test beds to agglomerate activity and stimulate further research
- Explore a hypothesis more naturalistically
- Test the limits of computational technologies
- Addressing the perceived needs of a problem domain or pressing societal issue

Low-Fidelity Prototyping

- graphical storyboards of proposed interactions.
- simple scenarios that can be discussed
- paper prototypes
- models of devices

Anything that can add richness to the discussion of the system with potential users

Medium-Fidelity Prototyping

- Video prototypes
  - can communicate the concepts in the system quite effectively
  - act as a useful reference for explaining the system later on
  - rapid prototypes of user interfaces using prototyping toolkits can afford a more realistic synthesis of the intended user experience
- Wizard of Oz
  - prototypes of parts of the system
  - not implemented parts are simulated by human
  - behaviour of the system to be emulated and thus experienced by others

High-Fidelity Prototyping

Partially working systems

- Horizontal prototype
  - all the intended functionality, but only at the top level
  - Example: initiate a shopping spree, but cannot order
  - Good for testing high level goals and action plans
- Vertical prototype
  - only one or two tasks are implemented in detail
  - Example: shop til you drop, but cannot see shipping information
  - Good when only few tasks are seen as particularly complex or important
- Chauffered prototype
  - Considerable functionality, but little or no error detection
  - How: A well trained assistant accepts and executes requests on behalf of the actual test user
  - Orthogonal to vertical and horizontal

Green-lighting

- In general, the more labour-intensive options are only really worth investing in
  - if the project itself is a significantly larger undertaking or
  - there is additional value to having the prototypes or associated media
- One of the cheapest and lightest weight mechanisms is simply to present the proposed system to someone else to gain informal feedback
- If they find the idea entirely preposterous or can see obvious significant flaws, it is certainly worth revisiting your scenario
- Don't propose projects to members of your project team (only)

Empathic Systems

- From a systems design perspective, it is far from clear what the interfaces and internals should be, necessitating an experimental approach
- Weiser [1991] espoused embedded virtuality and calm computing
  - computational devices are effectively invisible to its users
  - interfaces to such systems were through entirely natural, sensor-driven, and tactile interactions
- such systems almost empathically support the user in their daily tasks, requiring a high degree of knowledge about the user's desires and intents
- Necessary to explore the boundary between "the system" and "the user" to find the balance points for computational tractability and effective user support

Semantic Rubicon

- key challenge: consider the "barrier" between the physical world and virtual (computational) world
- interfaces often distributed, may have many forms of input and output involving several devices, and often incorporate subtle, oblique forms of interaction involving hidden or ambient sensors and displays
- Kindberg and Fox [2002] divide between responsibilities of system and user as "semantic rubicon"
  - "demarcates responsibility for decision making between the system and the user."
- More specifically, in terms of system design, crossing the semantic rubicon implies defining
  - the knowledge the system can have of the physical world and of user(s) behaviour
  - the knowledge the user has of the system and how they might influence it
  - the mechanisms and permissible interactions for one to influence the other

Computational Knowledge

- One needs to decide
  - what knowledge a system will need about the real world to function
  - how it will get into the system
  - how to represent it
  - how this state will be maintained
  - what to do if it is incorrect
- Unless this knowledge is easy to sense, or trivial to reason with, one you must also decide
  - what the implications are if the knowledge is imperfect or
  - conclusions are erroneously reached

Implications

- significant difference in implication if the outcome of misconstruing the user's situation while laying still is to call the emergency services rather than dim the lighting
- designing when to involve the user with decisions
  - or in the context of the semantic rubicon, when the decision of the system becomes the decision of the user

  well be crucial to the acceptability
- especially in safety-critical, sensitive or deployed settings

Computational Knowledge about Physical World

Key questions you should ask yourself are

1. What can be reliably sensed?
2. What can be reliably known?
3. What can be reliably inferred?

The degree to which you can answer these questions for the intended function of your system will help determine the feasible scope, or set some of the research challenges.

Sensing Problems

- limits to what your system can know about the physical world and the people who inhabit it
  - sensors have innate properties due to construction and underlying physics
  - may not be optimally placed or sufficiently densely deployed
  - high-level sensors, such as location systems, have complex behaviours governed by properties of the environment
- activity you wish to observe may simply be challenging to detect
  - due to its subtlety, or
  - difficult to isolate from other activities, noise, or
  - the concurrent activities of other people
- question of reliability of what is sensed in the presence of partial or total sensor failure

Presenting Information to the User

- challenge of designing systems that exhibit "tolerance for ignorance" [Friday et al., 2005]
- consider the scope and boundaries to your system
- "seamful design": be aware of the seams or the boundaries and inaccuracies of the system
- strategies for presenting information to the user:
  - Pessimistic: Only show information that is known to be correct
  - Optimistic: Show everything as if it were correct
  - Cautious: Explicitly present uncertainty
  - Opportunistic: Exploit uncertainty

Traditional & Cautious Approaches

- pessimistic and optimistic approaches are "more traditional"
- common to present a location of a user on a map as a dot
- Adjusting the size or representation of such a dot to reflect the confidence in location would enable the user to develop a greater trust and understanding of the system.
  - cautious approach is widely adopted on a typical mobile phone: the "bars of signal strength"
- Cautious or even opportunistic designs may offer systems that are amenable to user comprehension or even appropriation

Opportunistic Approaches

- What can and cannot be sensed or its underlying seams may even be an opportunity for design
- Explicitly define, what is
  - sensible,
  - sensable,
  - and desirable
- categorize uses of devices
- spot opportunities for other types of interaction with their devices that they had not originally foreseen

Users' Roles I

- corollary of semantic rubicon and seamfulness of your system: carefully plan the role the user will play in the system's operation
- Ambient systems often differ significantly in the degree of understanding and "intelligence" they are intended to show toward the users' goals and desires
- spectrum of design choices as to when to involve the user in sensing or understanding the physical world and in decision making or instigating actions
  - at one end of the spectrum, we might consider a scenario where the system fully "understands" the user's wants, and takes actions preemptively in anticipation of these
  - At the other extreme, perhaps more cautiously, we might design assuming no action is taken by the system without user assent, or where the user provides sensory inpuz

Users' Roles II

- A compromise position might involve
  - partially automating to support the user's perceived needs
  - but offering the ability for the user to intervene to cancel or override
- A further approach might be adaptive
  - using machine learning that starts by involving the user in decisions
  - learns from this, moving toward automation of common or consistently detected tasks
  - crucially, can move back to learning mode again if unreliable or undesirable

Checkpoints

To help consider where on this scale parts of your system might lie, consider:

- The frequency or inconvenience of potential user involvement
- The severity or undesirability of the consequences if the system gets it wrong
- The reliability of detecting the appropriate moment and appropriate action
- The acceptability to the user of automating the behaviour

Mental Models

Key Question
What do you intend for the user to understand or perceive of the system in operation?

- To grow comfortable with it, adopt it, and potentially appropriate it, the user must be able to form a mental model of cause and effect or a plausible rationale for its behaviour
- Mental models on the user side can only be influenced by induction

Always Runtime

- ambient systems are composed of distributed, potentially disjoint, and partially connected elements (sensors, mobile devices, people, etc.)
- "partially connected" here reflects that these elements will often not be reliably or continuously connected to each other
- the system is the product of spontaneous exchanges of information when elements come together
- interaction patterns and duration will vary with the design and ambition of any given system, but it is important to consider a key precept:
  - once deployed, all changes happen at runtime
- typically no simultaneous access to all the elements to (for example) upgrade them or restart them

Runtime Implications

1. Systems requiring a carefully contrived startup order are likely to fail
2. If the availability of elements may be sporadic, your system should be able to gracefully handle disconnection and reconnection or rebinding to alternate services
3. Assume that individual components may fail or be temporarily isolated and design your system accordingly so that state can be recovered
4. Decide proactively how to handle data when an element is disconnected
   - are the data kept (e.g., buffered) until reconnection, and if so, how much will you buffer before discarding
   - What strategy will you choose to decide which data to keep or discard (oldest, freshest, resample, etc.)?
5. Consider including version information in protocols used in systems designed to run longitudinally to at least identify version mismatches

Transient Connections

- network connections (or the lack or failure thereof) can have profound effects on the performance of systems and, crucially, the end user experience
- effects on unsuspecting software throughout a device's software stack can be serious
  - network names stop being resolved
  - closed connections can lead to software exceptions that stop portions of the code from executing
  - input/output system calls can block leading to stuck or frozen user interfaces
- considering what will happen if elements in the system that you are assuming to be always available fail, will help you identify and ideally mitigate for these potential problems

Network Fails

- when networks fail, it is common for data to be buffered and dropped at many levels in the protocol stack
- when data are often sensor traces informing the system of important events relating to interactions in the world, this buffering can introduce an array of associated problems
- For example, old (buffered) data can be misleading if not timestamped and handled accordingly
- Finally, your fresh data over a multiplexed interface will be behind the buffered data
- If fresh data are due to user interaction, then the system will appear very unresponsive until the buffer is drained
- very common for a frustrated user to try to interact multiple times or in many ways in the face of inexplicable delays in unresponsive interfaces, thereby exacerbating the problem

State of the World

- transient connections and component failures have an impact on how consistent the state of your overall system will be
- important to design in strategies for recovering from both of these cases
- parts of your system may be replicable or sufficiently available to use well-known techniques to mask such failures and achieve some degree of fault tolerance
- however, software components are often intimately linked to specialist or personal hardware, or may be placed in unique locations, which makes traditional techniques involving redundant replication or fail-over inappropriate

Alternatives

- Optimistic replication of state, which allows partitioned elements to continue to function while disconnected and then reconcile the journal of changes made offline upon reconnection
- Converging on eventually consistent state
- Use of persistent stores or journals to allow recovery of state (locally or remotely)
- Externalizing state (e.g., to a middleware platform, such as a Tuple Space), so that most components are lightweight and can recover
- Use of peer caches to replicate state for later repair
- Epidemic propagation of state using "gossip"-style protocols

Is It Working?

- Debugging ambient systems is extremely challenging
- elements are often distributed and may not be available or remotely accessible for debugging
- in many cases, embedded elements may not have much, if any, user interface
- common requirement is to monitor the system's output to check status messages or to be able to perform tests by injecting commands to emulate interactions and test components

Debugging Strategies

- Use of conventional mechanisms such as log files and network packet tracing to passively monitor running components
- Including status protocol messages that can be intercepted (often as periodic heartbeat messages)
- Adding status displays including use of hardware such as LED blink sequences, audible and visual feedback
- Including diagnostic interfaces such as embedded web servers that can be interrogated
- Enabling remote access to components such as remote shells, etc.
- Externalizing of state or communications by using a middleware such as a publish-subscribe event channel, Tuple Space, Message Oriented Middleware, etc.

5 Implementing Systems

"Off-the-Shelf" Components

- design of your ubicomp system is just the first step in realizing it
- design is often refined as implementation choices are made and their limits tested
- pragmatic choices have to be made as to define what you will build and what you will appropriate to construct your system
- seek third party components from hardware and software vendors or free and open source solution

Caveat

- Do not underestimate how much time can be spent in attempting to integrate disparate pieces of hardware and software
- Software perhaps successful or designed for one domain will not necessarily confer similar benefits to your domain
- Chosen software or hardware may place constraints on what you can build, or offer far more functionality than you require
- proprietary hardware or software may imply the need to work around features and limitations that are outside of your control
- Versatile toolkits, for all their tempting power and flexibility, may introduce unneeded functionality and unwanted software bloat

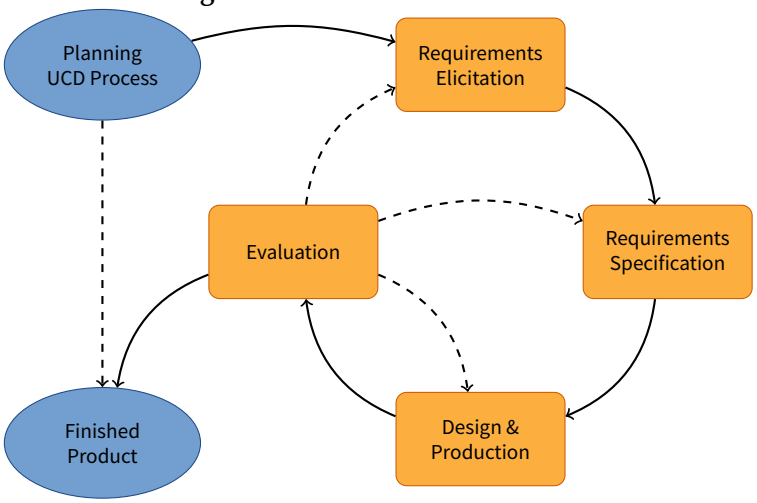
Deploying Systems

- One of the most valuable lessons to take from looking at successful systems is the need to mature the system through actual use
  - "eating your own dog food"
  - deploying and using the system initially yourself
  - ideally also with other users, who are not developers
- gain early feedback and highlight usability and interaction issues that may otherwise get missed until decisions are too well entrenched to be easily reversed
- agile development process where simple prototypes are put out early and refined during the development cycle

UCD & Participatory Design

- Deploying systems for people to use is always a costly process
- Designing a system that meets peoples' expectations, and indeed, helping set those expectations requires great care and expertise
- The key is identifying the stakeholders and involving them in discussions from an early stage
  - User-Centred Design Processes
  - Participatory Design Processes
- many issues due to the real world an d organizational settings that can catch the unwary developer by surprise

User Centred Design



Participatory Design

☞ Public hearing in urban planning (cc-by-sa Kaihsu Tai)

Real-World Issues

- The need to comply with health and safety or disabilities legislation, which can constrain the citing of equipment and place certain usability requirements for disabled users
- To be sensitive to data protection legislation, which may impact what data you can store, whether users have the right to opt-in, opt-out, or declare (e.g., with notices) that the system is in operation
- Environmental factors (including weather, pollution, etc.) can have a devastating effect on equipment that is not adequately protected
- Privacy and organizational sensitivity
  - potentially open vulnerabilities (perceived or actual) to expose private information or interfere with existing systems or processes
  - particularly true for organizations managing sensitive data or in high-pressure situations, such as healthcare and emergency services

Ongoing Costs

- With any system, there is an ongoing cost in supporting it
- proportional to the length of the deployment
- Robust engineering and clever design can help mitigate this cost, but it is a research challenge in itself to drive this to zero and make the system self-maintaining
- To keep down the impact of remote maintenance and support, you should ensure that it is possible to remotely monitor it, ideally as the user perceives it
- Remote access via the network is also important for resolving problems, especially if the system is inaccessible or far removed from the project team
- If the system is physically inaccessible, then this is likely to cause problems going forward
- Particularly in unsupervised deployments, there is always a chance of unexpected or accidental intervention
- Equipment that is installed and left in working order can sometimes find itself unplugged unexpectedly

Expect the Unexpected

- the unexpected is the hardest thing to prepare for
- Volatility is unfortunately endemic to the real world and hence to ambient systems
- How will your system react to
  - Presence or use by unknown users
  - Unrecognized devices
  - Changes to the wireless environment
  - Devices being power cycled
  - Batteries failing
  - You not being there
  - "Improper" use

Issues

- Hardware
  - Cost, Security, Environment, Power, Network, Space, Safety issues
- Software
  - Deployment and updates, Debugging, Security, Integration, Performance and scalability, Fault tolerance, Heterogeneity
- User Setting
  - Usability, Learning, Politics, Privacy, Adaptation, Trust, Support

Evaluation

- Simulation
  - In particular object-oriented simulations
  - Agents with particular goals, believes, intentions interact via simulated sensors with the real software
  - Data and/or modelling necessary
- Proof-of-concepts
  - field studies as done by Marc Weiser at PARC
  - Rudimentary and/or incomplete (see prototypes)
- Implementing and Evaluating Applications
  - Large-scale implementations
  - Long running systems
  - significant amount of users
  - Field-study

Proof of Concept

- A PoC is a rudimentary and/or incomplete realization of a certain technical concept or design to prove that it can actually be realized and built, while also to some degree demonstrating its feasibility in a real implementation
- Not a theoretical (mathematical) proof of anything; it is merely a proof that the technical idea can actually be designed, implemented, and run
- Creating PoCs is the most prevalent evaluation strategy in ambient systems
  - Weiser's tabs, pads and boards
- A PoC is a somewhat weak evaluation strategy
- It basically shows only that the technical concept or idea can be implemented and realized
- Actually, however, a PoC tells us very little about how well this technical solution meets the overall goals and motivation of the research

End-User Applications

- A stronger evaluation approach is to build end user applications using ambient systems component and infrastructures, and then put these applications into subsequent evaluation
- These applications can then be evaluated by end users in either a simulated environment or in a real-world deployment

End-User Applications: Strength

- Using underlying systems technologies such as components, toolkits, or middleware infrastructures to build real applications, demonstrates that the systems components are indeed useful for building systems
- The act of building these applications helps the systems researcher to judge whether their building blocks actually help the application developer meet his or her application goals
- Once the application is built and put into use, it provides a test bed for the underlying systems components and helps you answer more non-functional questions, such as: How well does the system scale, per form, and handle errors?

End-User Applications: Weaknesses

- Essentially, if not carefully designed, the application and the subsequent evaluation may tell us little, if anything, about the systems aspect of the whole application
- It is important to be absolutely clear about what your test application will tell you about your systems components, infrastructure, or toolkit
- It is easy to get distracted by the demands of building useful and usable application in themselves and lose sight of the real purpose of the exercise, which may purely be to understand the pros and cons of the systems part
- Moreover, whether or not the evaluation of an application turns out to be extremely successful may have very little to do with the systems properties of the application
- For example, an application may fail simply because of poor usability, or because it is so novel that users have a hard time actually using it

Released Systems

- The strongest evaluation of ambient systems components is to release them for third party use, for example, as open source
- In this manner, the system research is used and evaluated by other than its original designers, and the degree to which the system components helps the application programmers to achieve their goals directly reflects the qualities of the system components
- One may even argue that there is a direct correlation between the number of application developers and researchers using the system in their work, and the value and merits of the workReleasing and maintaining systems software does, however, require a substantial and continuing effort

Released Systems: Weaknesses

- Releasing systems building blocks such as hardware platforms, operating systems, toolkits, infrastructures, middleware, and programming APIs entail a number of things such as a stable and well-tested code base, technical, and API documentation; tuto- rials helping programmer to get started; example code and applications; and setting up licensing policies.
- And once the system has been released for third party use, issues of bug reporting and fixing, support, general maintenance, and new system releases need to be considered.

6 Tutorial 2

Feedback I

- Time/Importance allocation papers vs. lecture
  - Originally, papers meant to support lectures
  - Discussions more fruitful & longer than anticipated – thank you!
  - Current goal is about a mix 1/3 papers, 1/3 lectures, 1/3 tutorial/discussions?
- Form of paper discussions
  - Following the tips on academic reading ("What to look for")
  - More guidelines needed?
- Content of paper discussions
  - I do make sure "my points" are covered...
  - ...but I understand this is difficult to see
  - First step: discussion summaries in Learnweb
  - Would it be useful to have summary slides available?

Feedback II

- More pictures
  - It's complicated, but will try
  - Not easy to anticipate where they will be needed/useful
- Missing roadmap & summary – where are we?
- Missing context of papers
  - Recognized as problem of course evolution
  - Considering an orientation session after project week
- Length of assigned reading
  - Following the tips on academic reading ("How to read")
- Unreadable hand writing
  - I know, ask
- Upload slides before lecture
  - No, but you have had the current set for a while
- What is relevant for the exam
  - What are you afraid of?

Assignment 3.2: Dourish

- Required reading for week 4
  - Dourish, Paul. "What we talk about when we talk about context." Personal and ubiquitous computing 8, no. 1 (2004): 19-30.
- The text will be discussed in the tutorial 14.05.2018
- Course readings can be downloaded in the learnweb
- Every text has a wiki-page in the learnweb
  - Use it to describe the text
  - Use it to link the text to the course
- Results of the discussion may also be written up

Assignment 3.3: Picard

- Required reading for week 5
  - Tom Geller: "How Do You Feel? Your Computer Knows." Communications of the ACM Vol. 57(1), pp. 24-26. Jan. 2014
  - Rosalind W. Picard: "Affective Computing". MIT Technical Reports – TR 321. Nov. 1995
- The text will be discussed in the tutorial 28.05.2018
- Course readings can be downloaded in the learnweb
- Every text has a wiki-page in the learnweb
  - Use it to describe the text
  - Use it to link the text to the course
- Results of the discussion may also be written up

Assignment 3.3: Picard

- Form groups of 3-6
- Discuss what consequences the work of Picard may have for ambient and contextualised systems
- Possible areas:
  - What is the relation between emotion and context?
    - Emotion as context parameter
    - Emotion as additional parameter
  - Eliciting emotions in ambient systems
    - Benefits
    - Scenarios of use
  - Ambient systems showing emotions
    - Benefits
    - Scenarios of use
  - Other areas
- Discuss your idea in the course

Descriptive Framework Version 3

Contextualisation

- Contextual Parameter
  - Environment – things, services, people
  - Personal – mental & physical information about user
  - Social – roles & relations
  - Task – what is the user doing
  - Spatio-Temporal – when & where are we
  - Other
- Kind of Contextualisation
  - Awareness – what aspects are taken into account?
  - Sensitivity – what aspects are changed?

Descriptive Framework Version 3

Ambience

- Perception
  - Mediality – media types
  - Codality – semantic representation
  - Modality – human senses
- Reasoning
  - *Context Awareness*
  - *Context Sensitivity*
  - Other
- Action
  - Mediality – media types
  - Codality – semantic representation
  - Modality – human senses

Descriptive Framework Version 3

Interaction

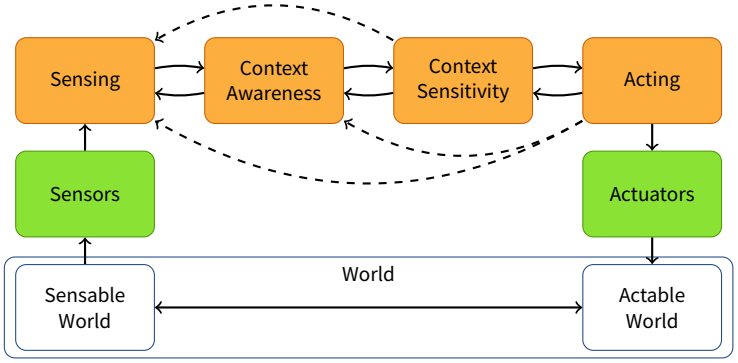
- Implicit vs. Explicit
  - Implicit input – through behaviour not primarily aimed at interacting with the computerised system (walking through a door, using a whiteboard…)
  - Explicit input – primarily aimed at interacting with the computerised system (voice or gesture commands…)
  - Explicit output – designed to get the users' attention (voice output…)
  - Implicit output – change of material setting where the users' attention is not the primary goal (opening doors…)
- Emotion
  - Does the system sense emotions?
  - Does the system show emotions?

Descriptive Framework Version 3

Work in Progress

- Weaviness
  - Is the system woven into the background?
  - Is the interaction naturally/culturally sound?
- Social Interaction
  - Does the system enable/enhance social interaction amongst humans?
  - Is the system targetting at supporting individual users?
- Enhance or replace
  - Does the system enhance or replace current solutions?
    - Both "technical" and "non-technical"

Architecture Version 1



General, simplified architecture

Assignment 3.4: Knowledge Base

- Form groups of 3-6
- Discuss in what form the results of the discussion of required readings can be preserved
- Possible technologies:
  - Learnweb Wiki
  - Learnweb Etherpad
  - Learnweb File Sharing
  - Other systems
- Possible responsibilities:
  - Individual
  - Group
  - Dedicated driver
- Discuss your idea in the course

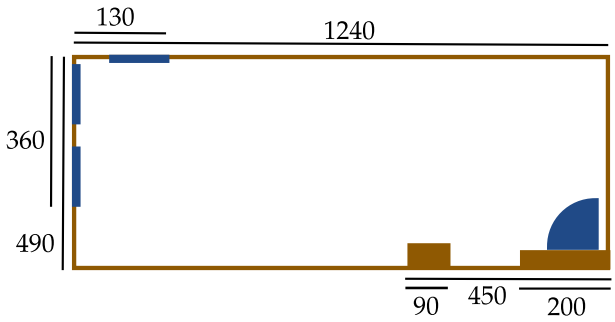
Assignment 3.5: New Lab Room

- Form groups of 3-6
- Develop the outline of a project idea to change A120 into a room you would like to use:
  - Today, traditional computer lab
  - How to change it?
    - Interior decor
    - Furniture
    - Technology
  - Possible technologies:
    - Tab, Pads & Boards
    - Behavioural interfaces
    - Natural language processing
- Pitch your idea in the course

Assignment 3.5: Old Lab Room

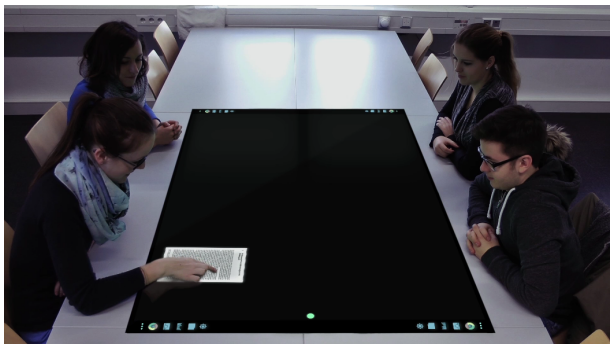
Samelsonplatz, A 120

Assignment 3.5: Old Lab Room Measurements



- 15 computer workstations (9+6)
- 16 group work seats (8+8)

Video 3.1: Universität 2025



☞ Where VR in 2025 (6:45)

Assignment 3.6: Davies & Gellersen; Hansen, Bardram & Soegaard

- Required reading for week 6
  - Davies, N., & Gellersen, H. W. (2002). "Beyond prototypes: Challenges in deploying ubiquitous systems." IEEE Pervasive computing, 1(1), 26-35.
  - Hansen, T. R., Bardram, J. E., & Soegaard, M. (2006). "Moving out of the lab: Deploying pervasive technologies in a hospital." IEEE Pervasive Computing, 5(3), 24-31.
- The text will be discussed in the tutorial 04.06.2018
- Course readings can be downloaded in the learnweb
- Every text has a wiki-page in the learnweb

  - Use it to describe the text
  - Use it to link the text to the course
- Results of the discussion may also be written up

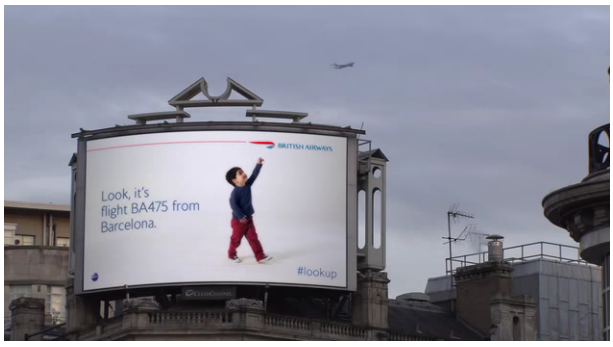
Assignment 3.7: Deficits

- Based on your experience with the previous assignment
  - What were the main obstacles when doing the task?
  - What knowledge or experience do you think is missing?
  - What gaps should be filled with this course?
- Start building your own map of the subject area

Assignment 3.8: Exam (Questions)

- Design your own exam
  - What kind of exam would you expect for this kind of course?
- Design your own exam questions
  - If suitable they might get used in the exam
- Describe and discuss your questions in the learnweb forum

Video 3.2: Look Up



☞ British Airways: lookup in Piccadilly Circus
0:26

The content of this section of the lecture largely follows Bardram and Friday [2010].

References

Literatur

Bardram, J. and Friday, A. (2010). Ubiquitous computing systems. In Krumm, J., editor, *Ubiquitous Computing Fundamentals*, pages 38–94. Boca Raton.

Friday, A., Roman, M., Becker, C., and Al-Muhtadi, J. (2005). Guidelines and open issues in systems support for ubicomp: reflections on ubisys 2003 and 2004. *Personal and Ubiquitous Computing*, 10(1):1–3.

Kindberg, T. and Fox, A. (2002). System software for ubiquitous computing. *IEEE Pervasive Computing*, 1(1):70–81.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, pages 94–104.