Developing User-Centric Adaptive Systems*

Martin Wirsing
In cooperation with Andreas Schroeder

Institut für Informatik
Ludwig-Maximilians-Universität München

*Partially supported by REFLECT (FET-IST Project Nr. 215893)
Outline

- User-Centric Adaptive Systems and the Bio-cybernetic Feedback Loop
- Case Study: The Intelligent Co-Driver
- Challenges in Physiological Computing
- Framework Support
- Development Aspects
- Concluding Remarks
Feedback Loops

- Controllers (e.g. robots) interact with their environment using a feedback loop
- Through iterations of
  - Sensing the environment
  - Analysing the sensor results
  - Actuating the reaction

Examples:
- Thermostat
- Cruise control in a car
- Robot swarm, ...
Computers and gadgets are often perceived as stubborn and inflexible
Humans adapt to the computer (!)
On reason: communication asymmetry between users and computing infrastructures

[Google]
Examples

- Google Glass
- Smart phones use context information (location, time, ...)
- iPhone 5s measures user activities
- Apple watch and
- Angel Sensor bracelet give bio-feedback

„Modern“ Human Computer Interaction
User-Centric Adaptive Systems

- **Goals of user-centric adaptive systems**
  - Assist us while being “invisible”
  - Adapt to the user’s changing context, state and demands
  - Influence the user’s environment to improve her/his well-being

- **Context: Location, activity, etc.**
  - **State**
    - Physical (e.g. movement, temperature)
    - Cognitive (e.g. high mental workload)
    - Emotional (e.g. anger, annoyance, happiness)
  - **Demands**
    - Relax when stressed during work
    - Avoid distraction while handling difficult driving situation
The Bio-cybernetic Feedback Loop

- User behaviour influences system adaptation
- System adaptations influence user behaviour
- Feedback loop involves both computing infrastructure and users as well as context awareness

Context information
Time and location
Appointments and tasks
User preferences

User input
Physiological data
Emotional, Cognitive, Physical state
Physiological Computing

- Physiological Computing offers monitoring channel delivering continuous data in the absence of direct user interaction.
- Use real-time psychophysiology to represent internal user state
- User state can be e.g. cognition, motivation, emotion, physical comfort
- Uses state as the basis for real-time system adaptation

Case Study: The Intelligent Co-Driver

Developing User-Centric Adaptive Systems

2008 – 2011
The Intelligent Co-Driver

Footage: Callum Egan
Napier University Edinburgh
Four Bio-Cybernetic Loops (I)

- **Cognitive Loop**
  - Determines cognitive load
  - from frontal theta; heart rate (HR) and heart rate variability (HRV)
  - characterised by decrease of HRV and slight increase in HR
  - Suppresses phone calls and reduces music volume on cognitive overload

- **Emotional Loop**
  - Determines emotional state
  - from emotional valence a. arousal
  - deduced from skin conductance (SC) and skin temperature (ST)
  - Selects songs based on current and target emotional state

Four Bio-Cybernetic Loops (II)

- **Driving Comfort Loop**
  - Sitting comfort
    - determined from seat pressure distribution and car vibrations
    - driver posture changes and car vibrations indicate sow sitting comfort
  - Actuates stabilizing seat cushions based on currently experienced comfort

- **Driving Loop**
  - Determines driving performance from lateral and longitudinal accelerations
  - Gives gradual feedback on driving behaviour
Challenges
Input Data Challenges

- Sampling rates vary
  - ~ 30 Hz for Skin temperature and skin conductance
  - ~ 10 kHz for EEG and ECG
  - High sampling rates require efficient pre-processing and down sampling in C++ and Assembly

- Physical signal noise
  - Movement artefacts (e.g. through breathing or sensor displacement)
  - Epochs of missing data through sensor contact loss or signal jamming
  - Environmental interference (e.g. environmental temperature)
  - General signal imprecision and noise

- Interpersonal variations and relative nature of input data and
  - Skin conductance depends on measured skin trajectory
  - Skin conductance response varies from person to person
  - Heart rate features some interpersonal variation, but is largely comparable
Data Abstraction Challenges

- Feature extraction is not enough
  - No simple mapping from physiological features to psychological state
  - Features are linked to multiple psychological constructs or environmental effects
  - Psychological effects manifest through multiple features

- Examples
  - Decrease in skin temperature can indicate increase in emotional valence or decrease in environmental temperature
  - Increase in skin conductance level can indicate increase in emotional arousal or increased physical activity
  - Even increase in emotional arousal may be due to the physiological system, or due to a mere thought

[Diagram of Feature to Construct Mapping]
Data Abstraction Example: Emotion

\[ A = \frac{1}{\sum w_{ai}} \sum_i (c_{ai} \cdot s_i) \]

\[ V = \frac{1}{\sum w_{vi}} \sum_i (c_{vi} \cdot s_i) \]

[ANN, ENS, MAR, ARO, VEL, SAD, MIS, REL]

\[ [A. Schroeder, MW: Developing Physiological Computing Systems, Software Engineering 2012,] \]
Creating Working Feedback Loops

- Leveraging physiological information to produce adaptive response
- Adaptive response can be of three different types
  - **Bio-feedback** – inform the user about inferred physiological state. Examples: bio-feedback visualization, audio warnings for anger management
  - **Operational** – use information in the direct operation of the system. Example: determine next song to play based on current emotional state
  - **Reflective** – adapt the system behaviour based on the current user state. Example: adjust game difficulty based on game experience intensity to create sinuous waves of game intensity
- Creation of appropriate and coherent feedback loops involves
  - Prototyping, trial-and-error
  - Psychological experiments and user studies
Emotional Loop Example

- Concept: guiding the user into the desired mood
- Simplified target mood selection: relaxed, neutral, energetic
- Physiological input: SCL and ST to determine current emotional state
- Changes in mood effected by played songs are recorded and used to produce KDE-based estimations of effects for song selection
Framework Support
REFLECT Framework

- Component-oriented framework based on Java and OSGi
- Configuration and reconfiguration access through Java APIs and internal DSLs
- Achieves self-awareness and adaptivity through (re-) configuration
Component-orientation for Adaptation

- In the context of physiological computing, components allow quick creation and testing of system variants
  - Encapsulation of variants behind common interfaces
  - Leveraging configuration control to switch between variants
- Run-time reconfigurations can be used
  - To modify the system behaviour in reflective feedback loops (i.e. based on user state)
  - For hot code update – additionally needs means to add/remove code
  - Tracking volatile devices (e.g. sensors and actuators)
- Reconfiguration can involve
  - Changing component parameters, sending commands to components
  - Adding and removing components and connectors
  - Changing the operational state of components (e.g. from active to inactive)
REFLECT Three Layer Architecture

Layered architecture hierarchically organizes the feedback loop:

Application layer
- Application
- Control Logic

Reflective layer
- Analyzer
- Coordinator

Tangible layer
- Sensor
- Actuator

DataContextService
DataStore
Component Manager
Distribution Manager
Emotional Loop Component Structure

Application layer
- SongRating
  - Valence Energy
- SongChooser
  - Valence Energy
  - ui
  - player
- SongEvaluator
  - evaluator

Reflective layer
- MoodMapper
  - Valence Energy
  - features

Tangible layer
- NexusProxy
  - SCL_raw
  - ST_raw
- SkinFeatures
  - SCL_raw
  - ST_raw
  - features
  - ui
- EmotionUI
  - ui
- MusicPlayer
  - player

DataContextService
DataStore
Component Manager
Distribution Manager
Data Processing

- Central storage for physiological data simplifies access and externalizes management
- Mocking of system parts becomes feasible through feeding recorded data into the central storage
Data Window

Producer → put → Consumer

defined wrt. time

last avg slice

auto-purging
Data Window benefits

- Data window structure useful in physiological feature extraction
  - Auto-purging, time-bound data buffers
  - Provide a simple and natural programming interface for feature extraction (e.g. statistical features)
  - Allow for creation of live slices from relative to the start or the end of the underlying data window
  - Slices are again data windows
- Needed to leverage sensors and actuators deployed in the field
- Minimizing network load requires local pre-processing of sensor data
- Local central storage allow for selective replication of data
  - Specification of update rate, and local buffer size allows to create local views of remote data in node-local stores
  - Central storage prepares the analysis for later distribution: analysis steps are decoupled through the central storage.
- Commands also need to be distributed. We found useful to provide
  - Point-to-point message-based communication
  - Event-based (selective) broadcast mechanism
Development Aspects
Agile Development Practices

- Knowledge in physiological computing is not settled
- Experimentation is integral part of process, and requires executable prototypes
- Quick feedback on feasibility is needed
- Iterations allow to focus on quality aspects valued by the customer
### November Sprint

<table>
<thead>
<tr>
<th>Ticket</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>#110</td>
<td>Change songlist refresh so that it's not polling-based</td>
</tr>
<tr>
<td>#115</td>
<td>Remove Playing state in !SongChooser</td>
</tr>
<tr>
<td>#116</td>
<td>Affective Music Player state/rationale UI</td>
</tr>
<tr>
<td>#117</td>
<td>Affective Music Player sensor data UI</td>
</tr>
<tr>
<td>#118</td>
<td>Prepare AMP for distribution</td>
</tr>
<tr>
<td>#119</td>
<td>Fix current song progress bar</td>
</tr>
<tr>
<td>#120</td>
<td>Align UI with state / listener-based communication</td>
</tr>
<tr>
<td>#121</td>
<td>Smooth dim down of sound volume</td>
</tr>
<tr>
<td>#122</td>
<td>Fix song length display</td>
</tr>
<tr>
<td>#123</td>
<td>Create bundle, container and empty UI for 2nd UI</td>
</tr>
<tr>
<td>#124</td>
<td>2nd UI displayed tab should correspond to 1st UI</td>
</tr>
<tr>
<td>#127</td>
<td>Show current user emotional state</td>
</tr>
<tr>
<td>#128</td>
<td>Show next songs ratings</td>
</tr>
</tbody>
</table>

### December Sprint

<table>
<thead>
<tr>
<th>Ticket</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>#131</td>
<td>Define design: 2nd UI, Affective Music Player</td>
</tr>
<tr>
<td>#134</td>
<td>Define design: 3rd UI, Affective Music Player</td>
</tr>
<tr>
<td>#167</td>
<td>List with next songs not correct</td>
</tr>
<tr>
<td>#184</td>
<td>Move song importer to details view, create staged song rating</td>
</tr>
<tr>
<td>#186</td>
<td>Adjust Emotion Details View to the design</td>
</tr>
<tr>
<td>#187</td>
<td>Adjust valence and arousal bars</td>
</tr>
<tr>
<td>#189</td>
<td>Readjust sizes and position of panels</td>
</tr>
<tr>
<td>#190</td>
<td>Adjust input display to the design</td>
</tr>
<tr>
<td>#125</td>
<td>Scroll song with drag-n-drop on progress bar and song list (NTH)</td>
</tr>
</tbody>
</table>

### Sprint goals

**November Sprint**
- Provide template for other loops
- Distributed deployment

**December Sprint**
- Define and implement UI for secondary and third displays
Prototype UI development
Developing User-Centric Adaptive Systems

Prototypes timeline

<table>
<thead>
<tr>
<th>Year 1</th>
<th>Year 2</th>
<th>Year 3</th>
<th>Year 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literature review</td>
<td>Experiment designs</td>
<td>Controlled experiment w. offline data interpretation</td>
<td>Feedback</td>
</tr>
<tr>
<td>In-Car test</td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
</tr>
<tr>
<td>Controlled exp. w. full prototype</td>
<td>Desk experiment w. full prototype</td>
<td>In-Car psych. experiment</td>
<td></td>
</tr>
</tbody>
</table>

P4
Verification and Validation

- Aspects requiring validation
  - Security and privacy
  - Safety
  - Functionality

- Verification of functionality
  - Inference of psychological state from physiological data
  - Correct operation of the bio-cybernetic loop

- Empirical validation
  - Existing psychological studies are performed in controlled environments with medical equipment – transfer needs to be tested
  - Working bio-cybernetic loop concepts need to be established
  - Existing studies provide crucial guidance, but are not enough
Emotional Loop Validation

- Experiment with 10 participants in desk work environments
- Effects on mood are visible
  - Positive mood induction was consistent
  - Negative mood was lower than expected
Verification of Reconfigurations

- Difficult to predict the possible evolutions of a system comprising reconfigurations
- Formal approach
  - Formally specify the behaviour of components
  - Formally specify the reaction of the configuration to external inputs and component behaviours
  - Verify that the composition of behaviours operates as desired
Verification approach

- Extend metric interval linear-temporal logic (MITL) with connectivity predicate to capture changes in port connectivities

\[ P \cdot p \sim Q \cdot p \]

- Create a contract-based framework for the verification of component-based systems
  - Run semantics \[ \llbracket \phi \rrbracket = \{ \rho | \rho \models \phi \} \]
  - Behaviour specification via contracts \((A, G)\)
- Assume-Guarantee contract framework (based on set of runs)
  - Contract refinement \((A, G) \succeq (A', G')\)
  - Contract implementation \(M \models_\Sigma (A, G)\)
  - Parallel composition of contracts \(C \parallel^A D\)
Results

- Semantic-level results
  - Abstraction preserves contract satisfaction
  - Composition preserves contract satisfaction

- Syntactic-level results
  - Soundness and completeness results

Theorem 3 (Completeness of REMITL). For any composite component signature $\Sigma$ the proof system $\Pi_{\text{REMITL}}^\Sigma$ is a sound and complete proof system for REMITL.

Theorem 4 (Soundness of REMITL). The inference rules in Fig. 4 are sound: $\Gamma \vdash_{\Sigma} \phi$ implies $\Gamma \models_{\Sigma} \phi$ for every finite set of $\Sigma$-formulas $\Gamma$ and every $\Sigma$-formula $\phi$. 
Concluding Remarks

- Challenges in physiological feedback loops
  - Handling input data
  - Creating correct abstractions
  - Designing working bio-cybernetic loops

- Component-based framework and methodological support
  - Components and agile development
  - Testing, verification and own empirical evaluations

- Ethics
  - Not to forget: potential drastic impacts on privacy, autonomy of the user, integrity of self