COMP 5047 Pervasive Computing
Week 4: Smartphones and the Android OS

Dr. Rainer Wasinger
School of Information Technologies
1. The paper claims "This research provides a new perspective on the development of complex and intelligent products". Discuss how they demonstrate this. Assess how well they do it.

2. The paper claims "As interactive products also become more intelligent and complex, emotional satisfaction will be an essential requirement for success.". How do they define "emotional satisfaction"?

3. What is meant by "visceral level of interaction"?

4. The paper claims "We explored pet-like characteristics as a convincing model for designing interactions; these characteristics will make future products more usable, emotional and sustainable.". What is the evidence for this? Are you convinced? Why (not)?

5. The reported evaluation is based on "observation and semi-structured interview". Were these described clearly enough that you could repeat them? How well does the paper provide solid evidence for the claim that "Participants showed instinctive intimacy and strong preference for pet-like product".
Part 1: Example Mobile Applications
Example Mobile Applications
Example Mobile Applications

PersonisJ User Modelling across Multiple Mobile Apps (Android)
Part 2: COMP5047 Project Outline
COMP5047 Project - Outline

Assessment of the project work?

Please check the deadlines for each stage of the project on the main class schedule.

All project stages are due at 1pm on the day of the lecture. All work must be lodged on your group trac site - the timestamp must be before that time. For the final project, the trac sites will be locked at that time.

The basic project, for the Pass criteria, is to create a pervasive wall device that tackles the inactivity problem (elaborated in lectures), by enabling people to easily log the details they want to track to help monitor how much time they sit each day.

Criteria for grading prototypes /10

Pass (5+):
Your demonstration of your prototype achieves most of the following:

- Basic prototype with minimal functionality on phone
- AND this enables user to log sitting time
- AND to correct accidental and incorrect entries
- AND a well chosen set of tasks chosen for demo (even if these are not all fully functional)
- AND these are listed on the trac wiki page
- AND you provide the marker with a printout of these

Credit (6.5+):
Demo meets all the following:

- Most Pass criteria are well done
- And presentation is clear and rehearsed and complete within 3 minutes

Distinction (7.5+)
Demo meets all the following:

- ALL Pass criteria solid
- And at least 3 are very well done

Note: Tutor grades all groups in class. For groups those awarded a mark <5 or >8, the lecturer will review the presentation that is on the trac site on the wiki page, linked from the main page and near the top of that page and labelled "Prototypes presentation".
COMP5047 Project – Past Examples

- Sitrix, Sedentary Logger, Sitting Counter, and Walk Record
Past Example - Sitrix

Features:
• Logs sitting time and provides a viewing history.
• Colour coded history as a form of feedback.
Past Example – Sedentary Logger

Features:
- Logs sitting time and provides a viewing history.
- Provides motivational feedback.
- Viewing history is shown per month and per day.
Past Example – Sitting Counter

Features:
- Logs sitting time and provides a viewing history.
- Automatic feature to track sitting time via sensors.
Past Example – Walk Record

Features:
- Manual logging and sensor-based logging for walking.
- Motivational feedback and provision of health information.
PersonisJ: Overview

• PersonisJ (Gerber et al., 2010) is a framework for building mobile, personalised, context-aware applications.
  • It is based on the PersonisAD distributed, scrutable model framework (Assad et al., 2007).

• It supports client-side personalisation, in which:
  • The user-model is stored on the mobile device (i.e. client-side), and
  • The user has control over the model.
  • The model can be accessed even when connectivity to the cloud is restricted or unavailable.
PersonisJ: Overview

- PersonisJ consists of two key components:
  - the PersonisJ Core, and
  - the PersonisJ API
- PersonisJ Core:
  - The PersonisJ Core contains the actual framework.
  - It is an application that is installed on the client Android smartphone.
  - It essentially ‘owns’ the PersonisJ data, which is stored in an SQLite database on the mobile device.
PersonisJ: Overview

- PersonisJ Model Browser:
  - This is an application that provides a simple user interface for browsing and interacting with the data stored in the PersonisJ model.
  - Using this application, one can view the evidences contained inside the PersonisJ model.
  - One can also edit the PersonisJ model from this application.
  - It is a separate Android application to the PersonisJ Core.
PersonisJ models are based on the accretion/resolution (A/R) representation (Kay et al., 2002), in which evidence is accumulated, and reasoning about a model is then achieved within client applications by use of a resolver function.

In PersonisJ, a model is represented as a hierarchy of ‘contexts’ containing ‘components’, and components accreting ‘evidence’.

![Diagram of PersonisJ models hierarchy](image)
PersonisJ evidences represent a piece of information for a particular component. It is essentially a value stored as a string with some metadata indicating how and when it was received.

Evidences can be simple, consisting – for example – of just a single key-value pair, or they can be more complicated, containing a number of key-value pairs (as in the example below).

<table>
<thead>
<tr>
<th>Employment History - Evidence list:</th>
</tr>
</thead>
<tbody>
<tr>
<td>{“resumeld”:“My academic resume”,“employment”:[{“title”:“Summer scholar”,“employer”:“University of Sydney”...},{“title”:“Java programming tutor”,“employer”:“University of Sydney”...}]}...</td>
</tr>
<tr>
<td>{“resumeld”:“My industry resume”,“employment”:[{“title”:“Java programmer”,“employer”:“Example company”...},{“title”:“software engineer”,“employer”:“Example company”...}]}...</td>
</tr>
</tbody>
</table>
PersonisJ: API

• The PersonisJ API provides Object-Oriented access to the PersonisJ model.
• The PersonisJ API is distributed as a jar file. All client applications must include the PersonisJApi.jar and the AndroidCommon.jar files in their build path.
• Key features of the API include:
  – Lookup, ask, and tell operations.
  – Default evidence resolvers and evidence filters.
  – Import and export operations.
PersonisJ: Interacting with the User Model

- **lookup**: Third-party applications gain access to the PersonisJ phone model (assuming they have been granted access on the PersonisJ package whitelist) via the following:

  ```java
  Model model = Model.getPhoneModel(pContext);
  ```

  - Navigating the PersonisJ model hierarchy: The values in PersonisJ are saved to component elements as evidence. The easiest way to access a component is to provide the Model with the path to the component:

  ```java
  Component component = model.getComponentByPath("Applications/SydneyDashboard/Profile/PersonalInformation");
  ```
PersonisJ: Interacting with the User Model

- **ask**: To ask for evidence from PersonisJ, one uses the ask() method, providing also an evidence resolver and optionally also an evidence filter.

  ```java
  List<String> vals = component.ask(personalInfoResolver, latestEvidenceFilter);
  ```

  - **resolver**: Resolvers are used to filter evidence into a preferred form. Default resolvers exist for all numeric types, for string, for Boolean, and for Location.
  - **filter**: Filters are used to filter the preferred evidence from the evidence list that is stored in the component. Two default filters exist: ‘LatestEvidenceFilter’ and ‘AllEvidenceFilter’.
    - Latest-filter returns the latest piece of evidence based on the timestamp that is stored in each piece of evidence.
    - All-filter returns the whole list of evidence.
• **tell**: To tell evidence to PersonisJ, one passes the evidence string and gives the ‘ValueType’ and ‘EvidenceType’ of the evidence.

    component.tell(evidence,
                   ValueType.String,
                   EvidenceType.Explicit);

– **ValueType**: The value type can be either Boolean, String, or numeric and is determined by the evidence.

– **EvidenceType**: This refers to the origins of the evidence. It can be the user, the device, or other.

• The PersonisJ API also exposes the ability to import and export partial ontologies:
  – The export method can be called from any context, returning a string containing the exported ontology encoded as JSON.
  – Import works in a similar fashion. The import method is provided by the Model class. It can load up a partial ontology from a JSON string.
PersonisJ: Sharing Data Between Apps

- An example of how different mobile applications can share information with one another (assuming that all applications have been granted access to the model):

![Diagram](image)
PersonisJ: Example Application

COMP5047PersonisJExample:

![Image of a file structure and Java Build Path window showing PersonisJExample project with its files and settings.](image-url)
PersonisJ: Example Application

- Two classes to look at:
  - PersonisJExample activity:
    - Checks if PersonisJ is installed and whether it is on the PersonisJ whitelist. It then calls AppModel to create the model.
      ```java
      appModel createUserModel();
      ```
    - Creates the UI, incl. the tell and ask buttons:
      - Tell:
        ```java
        appModel.savePersonalInformationEvidence (evidence.toString());
        ```
      - Ask:
        ```java
        String evidenceStr = appModel.
        retrievePersonalInformationEvidence();
        ```
  - AppModel:
    - This is where the real work is done.
PersonisJ: Example Application

- AppModel:
  1. Looking up the model:
     ```java
     Model pModel = Model.getPhoneModel(context);
     ```

  2. Passing PersonisJ this application’s model representation:
     ```java
     pModel.importFromJSON(readString);
     (based on the definition in the R.raw.usermodel_definition file).
     ```

  3. Telling PersonisJ a new evidence:
     ```java
     Component c = pModel.getComponentByPath("Applications/PersonisJExample/Profile/PersonalInformation");
     c.tell(evidence, Personis.Component.VALUE_TYPE_STRING, Personis.Evidence.TYPE_GIVEN);
     ```

  4. Asking PersonisJ for an evidence:
     ```java
     String evidence = c.ask(eResolver, eFilter);
     ```
PersonisJ: Example Application

PersonisJ Model JSON Format:

- At the top level context, the model is encoded in JSON as a dictionary:

  ```json
  {"contextinfo": {"Description": "Applications", "Identifier": "Applications", "resolver": ""}, "contexts": { "PersonisJExample": { "contextinfo": { "Description": "Example application that makes use of PersonisJ", "Identifier": "PersonisJExample", "resolver": "" } }, "contexts": {}, "components": {}, "views": {}, "subs": {} }
  ```

- **contextinfo**: is a dictionary containing information about the context, in particular: "Description", "Identifier", and "resolver".

- **thecontexts**: is a dictionary containing an entry for each subcontext: key is the context name, value is the subcontext encoded as above.

- **component**: An object encoded as a dictionary containing: "Description", "Identifier", "componentType", "evidenceList", "resolver", "valueType", and "value".

- **evidence**: An object encoded as a dictionary containing: "value", "evidenceType", "exp_time", "source", "time", "flags", and "comment".
**Lab Exercises**

- Completion of last week’s lab.
- **PersonisJ (Optional):**
  - Complete the assigned lab task on PersonisJ. See: