

CLIENT-SIDE USER MODELLING ACROSS MULTIPLE MOBILE APPLICATIONS

TECHNICAL REPORT 682

RAINER WASINGER, MICHAEL FRY, SIMON GERBER, OLLI IIVONEN, JUDY KAY, BOB KUMMERFELD

SEPTEMBER 2011

Client-side User Modelling across Multiple Mobile Applications

Rainer Wasinger, Michael Fry, Simon Gerber, Olli Iivonen, Judy Kay, and Bob Kummerfeld

School of Information Technologies,
The University of Sydney, NSW, 2006, Australia

{rainer.wasinger, michael.fry, sger6218, oiiv2253, judy.kay, bob.kummerfeld}@sydney.edu.au

ABSTRACT

Modern mobile devices commonly support applications that individually capture detailed information about a user with the goal to adapt and personalise the user experience to that of the individual. This paper describes research to go beyond this, by supporting client-side user modelling for context-aware and personalised applications. This work builds upon the PersonisJ framework, to tackle the challenges of supporting multiple mobile applications for the purpose of personalisation as well as user model reuse across applications. Our approach involved the creation of three substantial applications: a career organiser application that makes use of actual data from an online employment website; a casual employment application; and a university student information application. Also described in this paper are the key challenges in the creation and management of the model ontology to address the needs of developers of such applications. This work represents the first evaluation of the use of PersonisJ - or any other mobile client-side user modelling system - in which multiple mobile applications reuse a locally stored user model across applications.

Author Keywords

Mobile, client-side, user modelling frameworks; mobile personalisation; model reuse across mobile applications.

ACM Classification Keywords

D.2.13 [Software Engineering]: Reusable Software---reusable libraries, reuse models; H.5.2 [Information Interfaces and Presentation]: User Interfaces---evaluation/methodology, user-centered design.

1. INTRODUCTION

Desktop, web, and mobile applications are increasingly capturing and storing user information with the goal to adapt and personalise the user experience to that of the individual. Mobile devices, especially smartphones, offer particular promise for user modelling and personalisation. Smartphones will become increasingly common, with predictions of increases from 14% of mobile terminals in 2009 to 37% by 2012 (Cozza, 2009). Smartphones are also becoming increasingly powerful in processing and memory capability, as well as integrated sensing and communications technologies. Another important characteristic of smartphones that makes them an ideal choice for storing a user model is that they are always on and they are always with the user, thus making them easily accessible to the user even when they are out of an area with coverage. Smartphones are also typically

owned, configured, and controlled by a single user. This lays a solid foundation for user-owned and user-controlled user models on smartphones.

There are many benefits for mobile applications that can personalise the user experience. Personalisation can reduce information overload by making information relevant to the task at hand (Fisher, 2001; Weakliam et al., 2005); personalised systems can also actively alert users of useful information, as is demonstrated by (Krüger et al., 2004) in their Personal Navigator application.

There is also a close overlap between systems that are context-aware and systems that are personalised to the user; indeed, context can be seen as a potential solution to many of the usability problems associated with mobile information access. These problems extend beyond just formatting content for small screen devices. Past work has for example looked at tailoring user interfaces based on the current context – e.g. consider the Just-For-Us application (Kjeldskov and Paay, 2005), which uses context-awareness to personalise a user interface based on the user’s social context, and the SMMART e-commerce recommendation program (Kurkovsky and Harihar, 2006), which allows retailers to push recommendations to the customer based on their personal preferences. Past work has also looked at presenting contextual information within the user interface – e.g. (Debaty et al., 2005), in which geographical relationships (e.g. ‘next to’) are treated as hyperlinks providing contextual information as a browsable resource, and (Church and Smyth, 2008), in which search queries on a mobile device are augmented with contextual features like location and represented on top of a geographical map rather than as a list of results.

As outlined in (Shilton, 2009), the information required to personalise software does however bring with it much concern over user privacy. The dramatic rise of smartphones has only compounded the problem. Camera phones in particular have caused trouble, angst, and even unfairly ruined careers (Oei, 2009). The location-sensing technologies embedded in smartphones have also received much attention (Chen and Rahman, 2008; Tsai et al., 2010). In (Kärkkäinen et al., 2010), a field study on mobile phone activity and context logging was conducted with the purpose of life-logging in mind (i.e. the automatic and persistent collection of information about one’s life and behaviour). Results from that study show that although a little uneasy in the beginning, users were not disturbed by the logging, but did want to be in control of logging the most private information. The results also

showed that in order to make users feel more secure, information that is not shared should be stored locally. Local client-side storage of the user-model is a key focus of the PersonisJ framework and the applications described in this paper.

At present, personalisation of the information delivered to mobile phones is typically performed at the server-side, by services in the cloud. This has been a necessity in the past due to the limited computational power and memory of mobile devices. However, as these mobile devices become increasingly powerful, it has now become feasible to support client-side personalisation for these devices, in which the user's model is stored on their own device. By storing the information client-side, the user is afforded an opportunity for greater privacy and control over the data, and because the device is mobile and typically accompanies the user, contextual information can also be recorded into the user profile. To support the creation of new personalised applications for phones and other mobile devices, new tools are required, and PersonisJ is one such tool.

PersonisJ is a framework for building mobile, personalised, context-aware applications. In previous work (Gerber et al., 2010), the PersonisJ mobile client-side user modelling framework was described, including an outline of the framework's basic feature set, a description of its use in a demonstrator museum guide application, and results of a small scalability evaluation focusing on the 'ask' and 'tell' operations that are fundamental to the use of the PersonisJ framework. The purpose of this paper is to now go to the broader task of creating user models that are used by multiple locally-installed mobile applications as their primary user model, and for the purpose of personalisation and model reuse. This is the first evaluation of the use of PersonisJ - or any other mobile client-side user modelling system - in which multiple mobile applications reuse the user model across applications.

Section 2 discusses related work that has a focus on user-modelling frameworks targeting mobile devices. Section 3 then outlines the existing PersonisJ framework upon which this work builds. Validity of our approach is measured along three axes. Firstly, in Section 4, we describe the three separately implemented mobile applications, each using the PersonisJ framework to contribute and retrieve information to/from the user model, and each with a strong focus on mobile client-side personalisation. These applications are: a long-term career management application, a casual employment application, and a University student application. Secondly, in Section 5 we analyse the challenges of supporting the ontologies of different third-party smartphone applications (and their associated developers) in creating, accessing, interpreting, and sharing the application models with other locally-installed applications. Section 5 reports on our design methodology for developers to create applications and shows user interfaces supporting end-users interacting with their model during application use. Thirdly, in Section 6 we discuss the ease of integration of the

framework by the developer, and performance issues when dealing with real-world datasets, particularly as the user model grows over time. Finally in Section 7 we outline our conclusions.

2. RELATED WORK

The PersonisJ framework (Gerber et al., 2010) can be seen to belong to the larger Personis ecosystem, which currently consists of a number of different user modelling framework implementations, ranging from both server-side – e.g. Personis (Kay et al., 2002), PersonaF (Niu and Kay, 2010), and PersonisAM (Assad, 2010) – to client-side (i.e. PersonisJ). Future applications will all utilise the Personis components relevant to them, including for example a mix of both server-side and client-side user modelling components, in order to create a seamless user experience, and in which the end-user is the owner of and is in control of the contents of their user model.

Most user modelling frameworks used in mobile computing scenarios focus on server-side solutions. Indeed, the original Personis (Kay et al., 2002) focused specifically on using a server to store user models and it demonstrates this in the context of an adaptive hypertext system. PersonaF (Niu and Kay, 2010) extended this implementation and focused on the use of the server-side system for ontological reasoning. Most recently, in PersonisAM (Assad, 2010), the focus was on active models for pervasive computing environments, in which the storage and processing of models could be distributed across the Internet. PersonisJ in contrast represents an entirely client-side solution to user modelling and more generally to providing a mobile framework to support personalised and context-aware applications.

In the SMMART e-commerce recommendation system (Kurkovsky and Harihar, 2006), the preferences of shoppers were stored on the mobile device and could be carried with them from shop to shop. Unlike the PersonisJ user-modelling framework however, SMMART stores the preferences in XML as a simple list of weighted keywords and these preferences are transmitted off the device to the tightly coupled SMMART server in order for it to match products to the customer's personal preferences.

Another relevant work is HP's Web Presence Manager (Debaty et al., 2005). In that implementation, the focus is on a distributed software framework that makes use of web technologies to enable mobile and context-aware access to personal contents, and the rendering of such content on local appliances in ubiquitous computing environments. The system focuses on the concept of a "web presence", i.e. independent software modules representing physical entities such as a printer, projector, stereo, paper book, and human users. Each web presence manages up-to-date information about its physical counterpart and presents it on the web for use by other web services or web users. Security and privacy of web presence data is however not addressed and instead left by the authors as future work.

In (Davidyuk et al., 2004), the focus is on creating a context-aware middleware for mobile multimedia

applications. The middleware provides functionalities like: the ability for services to locate other services and components; an event-notification mechanism based on the publish-and-subscribe model; and the ability to store and manage context data collected from various sources. Privacy and access rights to the user’s data are handled by a specific component called the ‘context-based storage’ module, which is included as part of the middleware architecture. This again differs to PersonisJ, in which privacy and access is controlled directly by the user on the mobile device.

In (Kobsa, 2007), a substantial review of generic user modelling systems covered shell systems, central server systems, and agent-based user modelling systems. Not one of the reviewed systems was designed specifically to run on mobile devices like smartphones. Similarly, in (Krüger et al., 2007), a review of mobile guides (museum, navigation, and shopping guides) was conducted, and it was found that although some applications used server-side user modelling frameworks like UbiWorld¹ (Heckmann, 2005), the majority implemented only simple user preference lists on the device.

3. THE PERSONISJ FRAMEWORK

PersonisJ is a client-side framework for building mobile, personalised, context-aware applications. PersonisJ represents a “model” as a hierarchy of “contexts” that can contain “components” (see also Figure 5). It is based on the accretion/resolution representation, and indeed each component accretes “evidence”. Evidences are essentially values with metadata indicating how and when the evidence was received. The hierarchy of contexts and components constitutes an ontology and a sub-tree within the hierarchy is a partial ontology.

PersonisJ allows locally-installed mobile applications to access the user model via an “ask” operation (i.e. to retrieve data from the model) and a “tell” operation (to supply data to the model), and it also provides an import/export operation that allows applications to import and export partial ontologies using a simple JSON format. In addition to the import/export, ask, and tell operations that PersonisJ provides to third-party applications, it also provides the ability to: broadcast Android messages indicating when a component has changed; monitor the user’s location; and (via the underlying Android OS) deny or grant applications access to the model via the PersonisJ-specific “ask” and “tell” permissions. Reasoning about a model is performed via the use of resolver functions, and first prototypes show that security can be even further enhanced by running the mobile applications within a restricted environment (Pink et al., 2010).

As shown in Figure 1, PersonisJ is architecturally comprised of two components: the Core Framework and the public PersonisJ API. These are also explained in detail in (Gerber et al., 2010). Third-party applications access the user model via the PersonisJ API, though only

after such applications have been granted read and/or write access to the model. These security permissions are enforced by the underlying Android OS, which ensures that applications explicitly declare - on install - how they wish to interact with PersonisJ. At the heart of PersonisJ is an SQLite database. Access to this database is mediated by the ContentProvider, which specifies a unique “content URI” for content it exposes to the client application. To allow third party applications to interact with PersonisJ, the PersonisJ Core provides the PersonisService module. Applications intending to read from the database use the READ_CONTENT permission (which corresponds to the “ask” operation) and those intending to contribute to the database do so via the PersonisService module using the TELL_PERMISSION (corresponding to the “tell” operation).

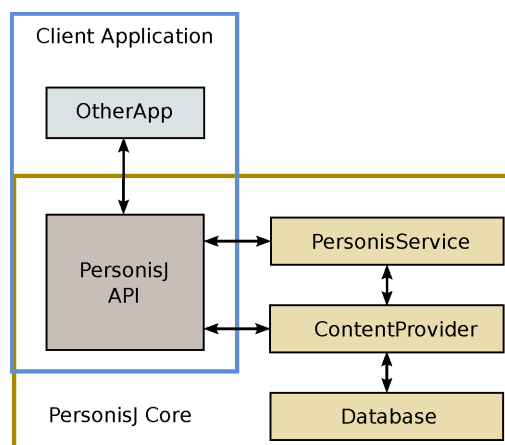


Figure 1. The PersonisJ architecture, showing a third-party Android application accessing PersonisJ locally on the mobile device via the provided PersonisJ API.

4. THE IMPLEMENTED MOBILE APPLICATIONS

This section describes the three mobile applications that highlight our focus on mobile client-side personalisation and user model reuse across applications. These are: the CareerOrganiser, a career management application designed to assist passive (and active) career-conscious workers to manage their long-term career prospects; the CasualEmployment application, which allows users to find jobs of a very casual nature; and the UniversityApplication, which demonstrates how relevant information can be delivered to students of a particular University.

First, we illustrate how content from the user model is reused across these applications. Consider a scenario in which Jon, a University student, installs the applications onto his smartphone and then begins to interact with them. Figure 2 shows how Jon’s semester timetable from the UniversityApplication (the second component in the application in the centre of the figure) is reused by the CasualEmployment application to help determine his time availability throughout the week. Similarly, course information from the UniversityApplication is reused by the CareerOrganiser application when Jon fills in his educational qualifications in his resume. The CareerOrganiser application also imports details from

¹ UbiWorld, URL: <http://www.ubisworld.org>

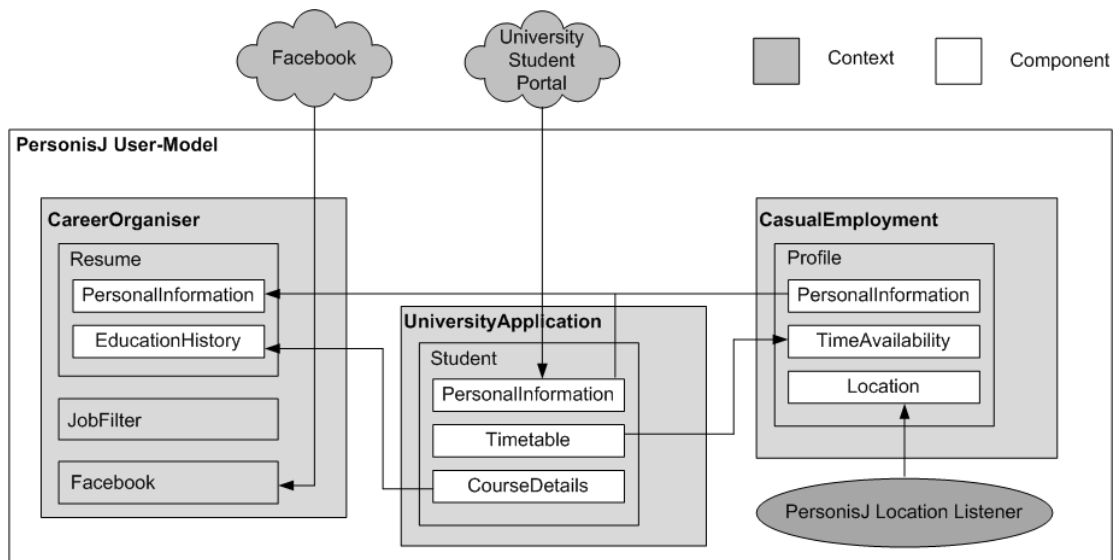


Figure 2. Client-side data flow of user model information between the multiple mobile applications. The shaded boxes represent “contexts” and the white boxes represent “components”.

Jon's Facebook² account, and this is reused, along with other personal information to help Jon complete his resume. Note that it is the resolvers (see also Section 5) that are responsible for determining what relevant information might exist in the user model, and so the order in which applications are used is not fixed to the sequence provided in the example above.

The Career Organiser Application

The CareerOrganiser (see Figure 3) allows passive and active career-conscious workers to manage their long-term career prospects, and to browse job positions currently advertised. This client-side Android application downloads live job market data from an online service associated with a website and stores this locally on the device in an SQLite database. Mobile personalisation aims to identify the jobs most relevant to the user. The application allows a user to configure multiple personas based on profiles, as in the example in Figure 3A - at the upper left, where the job search can use the “Graduate Profile” or the “Academic Profile”. Each such profile has an associated job filter and/or a resume. The job filters contain details of the type of job that the user is interested in; Figure 3B (lower left) shows that this user's “Academic profile” has specified “Locations”, “Sectors” and “Work Arrangements”, but not yet defined “Remuneration”. Resumes (Figure 3C,D) hold details on the user (including personal information, education, past employment, and skills). Resume creation is done via an in-application guide (see also Figure 7), which is able to prefill form fields for the user based on existing content in the PersonisJ user model. The multiple personas allows the application to retrieve job listings relevant to the user along a number of different career paths, e.g. a persona

for current job prospects, another for future job prospects; or a weekday and a weekend persona.

The personalisation of job listings is done in two phases. First, the set of jobs fitting the user's defined job filter are downloaded from the commercial server to the device (first the job title, and then via a background thread the entire job descriptions). Then, the list of jobs is ranked by relevance based on data from the resume profile, their previously favoured jobs, and keywords that the user selected while browsing through other job descriptions. Figure 3F shows the user adding a selected word “SQL” to his list of relevant keywords. The user is then able to browse job listings via both a sorted list, but also via a slide-show presentation mode (Figure 3E) at a configurable speed (e.g. one job every two seconds; but also via additional controls like pause/resume, and finger gestures).

The Casual Employment Application

This application (see Figure 4A,B) consists of a server-side casual jobs platform, a website front-end to allow job providers to administer job postings, and a smartphone application to allow job candidates to find relevant “one-off” casual jobs from their phones. In contrast to the CareerOrganiser, this application focuses on the casual job market, which is often drastically different from the blue- and white-collar jobs typically found in job platforms (e.g. consider handing out marketing pamphlets, or walking the dog). From the website, prospective employers can create an account, add new jobs, review applications, and accept applicants for their listed jobs. From the smartphone application, job candidates can set up their profile (including their preferred location and available times), search for casual jobs, view job details, and apply for jobs. The casual job listings are personalised to the user based on their

² Facebook, URL: <http://developers.facebook.com>

location and time availability, and the newest jobs relevant to the user are additionally shown at the top of the main application page (Figure 4A).

The University Smartphone Application

This prototype application (Figure 4C,D) addresses some of the needs of modern students on campus: viewing the

student's timetable, finding staff details, browsing course and subject information, viewing maps, and reading news feeds relevant to the user. This application allows a user to configure their semester timetable and then provides the user with contextually relevant links between data sources like building maps and locations, course and subject details, and lecturer contact details.

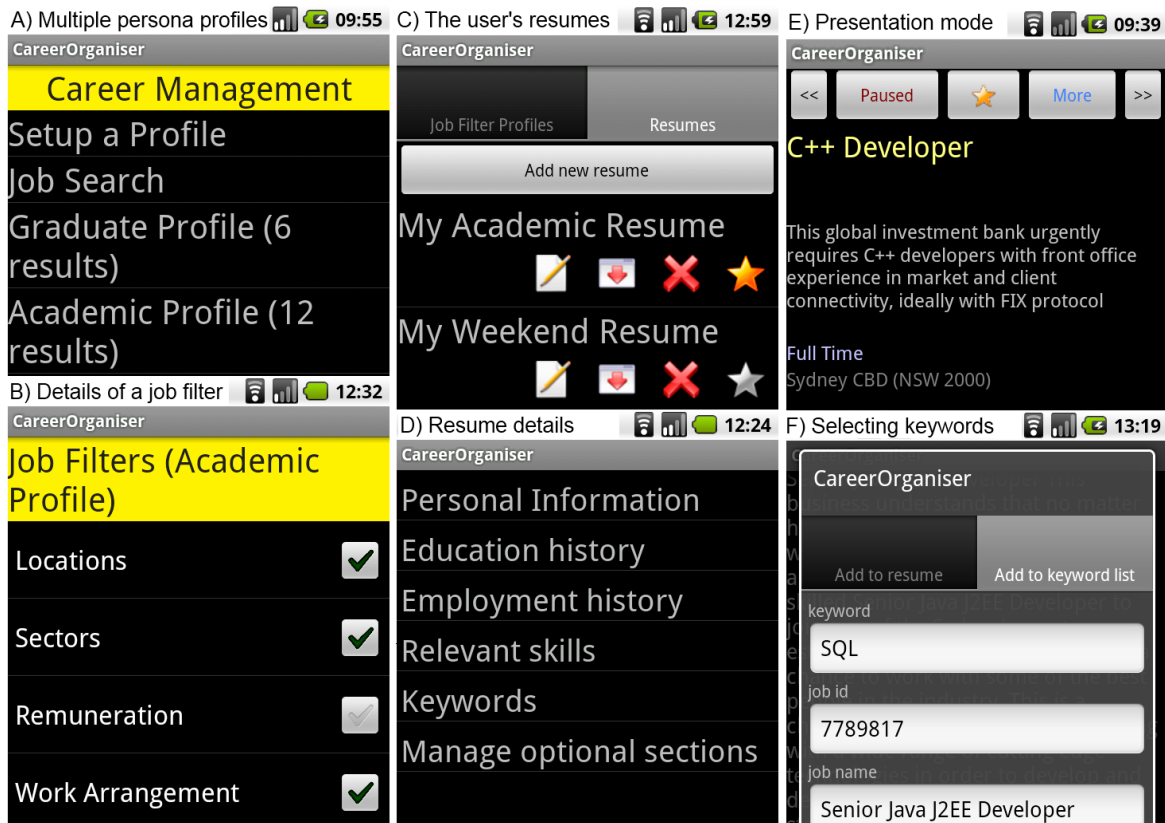


Figure 3. Partial screenshots from the CareerOrganiser mobile application, showing: multiple personas (A), job filter details (B, lower left), multiple resumes (C), resume details (D), job presentation mode (E), and keyword selection (F).



Figure 4. CasualEmployment main page (A) and my jobs page (B). The UniversityApplication's main page showing functionality like the timetable, maps, staff details, and subject/course information (C), and the news page (D).

5. THIRD-PARTY CLIENT-SIDE APPLICATION DESIGN METHODOLOGY

This section discusses the challenges of creating multiple ontologies or namespaces within a user model, for use by each application. We also discuss the issues for reuse and the competing design goals for application developers, in maintaining application-specific parts of the user model, at the risk of duplication against the reuse of parts of the user model when they are available.

Designing the PersonisJ Application Models

In PersonisJ (Gerber et al., 2010), a model is represented as a hierarchy of “contexts” containing “components”, and components accreting “evidence”. This hierarchy of contexts and components is referred to in this paper as an “ontology”, though it can be noted that PersonisJ sets no restrictions, limits or requirements on the structure of an ontology. As such, client applications may define any structure that they like. This means an application can define domain-specific partial ontologies as required. 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.

The PersonisJ model representing the three applications described in Section 4 is shown in Figure 5. The figure shows that the CareerOrganiser has a “JobFilter”, “Resume”, and “Facebook” context defined. The names of the components in the JobFilter context are based on a predefined structure from a commercial online service; these include components such as location, sector, remuneration, and work arrangement. The components contained in the Resume context are based on a subset of the HR-XML³ specification that has been defined by the HR-XML Consortium. We have extended it to

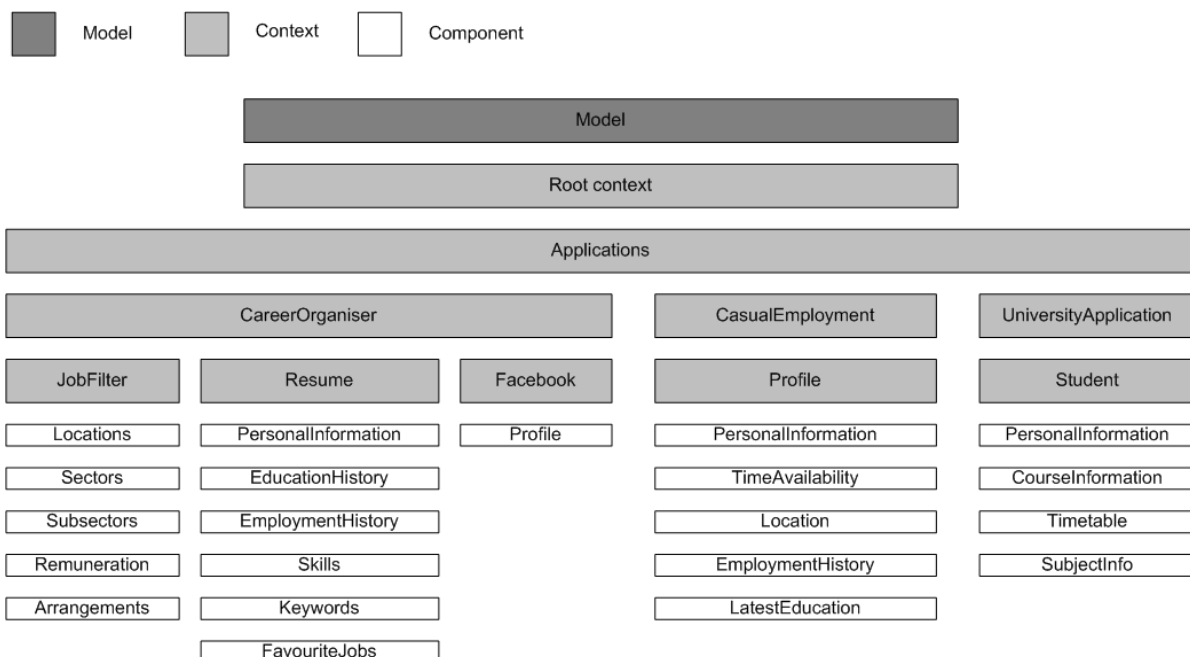


Figure 5. The PersonisJ client-side user model covering the multiple mobile applications discussed in this paper.

incorporate details of past keywords that the user selects while reading through job descriptions (i.e. the “Keywords” component; see also Figure 3F for the dialog that pops up when a user selects the word “SQL” from a job description) and jobs that the user has favoured (i.e. the “Favourite Jobs” component). The Facebook context contains data that the user has allowed the CareerOrganiser to import from the online social networking site.

As shown in the figure, a design decision was made to store each application's model separately under a new context node called “Applications”. This is in addition to the automatically created “Model” and “Root context” nodes of the tree. Our applications first check to see if the “Applications” context exists, and if it does not, they request PersonisJ to create it. Once this context exists, the application provides PersonisJ with its application ontology to be stored within the context.

In our approach, application developers may well decide to store their information in separate silos. This may lead to some duplication of information (e.g. “Personal Information”). While this might be stored under a separate PersonisJ-defined context alongside that of Applications, the separate silo approach has merit in terms of independence of developers, scalability across a large number of independent applications and flexibility, e.g. a person might use different names and aliases for different applications (see also Figure 7, left).

Another design decision was to name the individual application ontologies based on their actual package names to avoid conflicting application model names, e.g. au.examplecompany.CareerOrganiser for the Career Organiser application, though note that this is abbreviated in Figure 5 to save space.

³ HR-XML v3.1 (2010-09-21), <http://www.hr-xml.org>

Employment History - Evidence list:
<pre> {"resumeId":"My academic resume","employment":[{"title":"Summer scholar","employer":"Example company" ...}, {"title":"Java programming tutor","employer":"Example company" ...}]...} </pre>
<pre> {"resumeId":"My industry resume","employment":[{"title":"Java programmer","employer":"Example company" ...}, {"title":"software engineer","employer":"Example company" ...}]...} </pre>

Figure 6. Evidence located in the Employment History component. The resumeId value differentiates the evidence between different resumes.

A final design decision regarding model design was to offload some of the ontology complexity into the JSON evidence structures stored by components. “Evidence” is essentially a value stored as a string with some metadata indicating how and when it was received. In our three applications, we store the evidence as a JSON string, and outline in Section 6 how this approach minimised the number of “ask” operations required by the CareerOrganiser application, thus improving its performance. Storing part of the application ontology as JSON-formatted evidence was also useful when dealing with the need to model multiple dynamic instances of a particular context. For example, in the CareerOrganiser, it is possible for a user to create multiple “Job Filters” and multiple “Resumes”, though just how many is for the user to decide. The security restrictions defined in PersonisJ (and enforced by the Android platform) currently prevent third-party applications from changing the contexts and components that they have provided the PersonisJ framework using the import feature. Figure 6 shows a practical example of our modelling of multiple dynamic instances of a context (in this case employment history evidence belonging to different resumes), in which it can be seen that all resume evidence is identified by an additional key-value pair containing the resumeId to which the evidence belongs.

A Practical Example demonstrating how Applications access Data in the PersonisJ Model

PersonisJ requires that applications using the framework have knowledge and an understanding of the underlying user model context that they want to utilise. They must know the preferred context names and the paths to the component elements where the reusable information is stored. It is conceivable that an online resource listing all PersonisJ-compliant applications exist, with links to the developer’s own documentation. With direct paths to the components, the navigation to the preferred source by a third-party application is a fast and simple process. The code sample below from the CareerOrganiser application illustrates this process, using the “Model” object provided by the PersonisJ API to retrieve information from the CasualEmployment application’s model.

```
Model pMod =
```

```

    Model.getPhoneModel(pContext);
Component pCom =
    pMod.getComponentByPath("/Applications/CasualEmployment/Profile/PersonalInformation");
List<String> pVals =
    pCom.ask(personalInfoResolver, latestEvidenceFilter);

```

Reasoning about a model in PersonisJ is achieved through the use of the resolver function. Resolvers are used to interpret evidence, and evidence passed to a resolver can optionally also be passed through an evidence filter. The PersonisJ API contains two pre-defined evidence filters, one that returns just the latest piece of evidence, and the other which returns the list of evidence without any filtering. The API also contains a number of resolvers for resolving most of the basic data types.

As an example of the use of filters and resolvers, in the CareerOrganiser, a user is able to create a new resume from within the application, and data for guiding the user in filling in such a resume is retrieved by the application from the PersonisJ model. While filling out the resume, the default LatestEvidence filter is used to retrieve relevant evidence from both the CasualEmployment application (see code sample above) and the UniversityApplication. In this case, the latest evidence is also the most relevant evidence.

After filtering the evidence, a custom resolver is used to resolve a certain value from the evidence. The CareerOrganiser application has a separate resolver for each resume section (e.g. personalInfoResolver and educationHistoryResolver). These resolvers transform the evidence from the UniversityApplication and the CasualEmployment application from that of a JSON string to a list of single values. The list is created by matching keys between the CareerOrganiser and the other applications. For each separate resume section, the CareerOrganiser has a defined set of key values which it looks for across the application models (e.g. for personal information: first_name, last_name, phone, and so on). Figure 7 (left) shows how these values (i.e. the values for “first_name”) are then tied to the UI elements to help guide the user in filling out their resume.

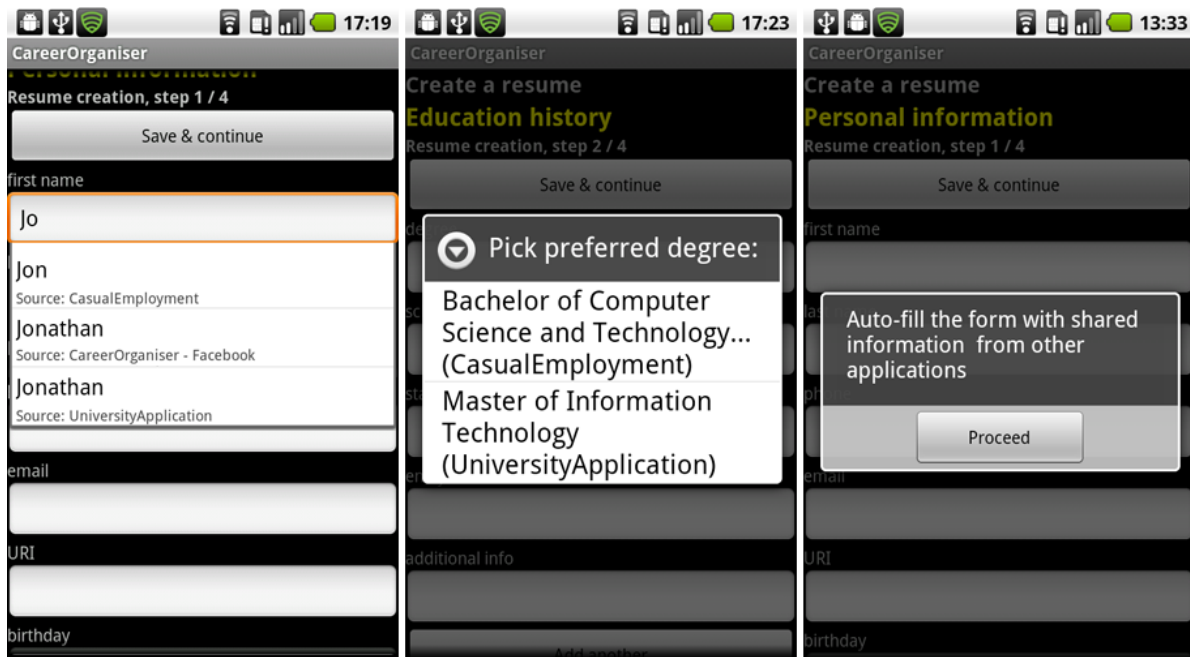


Figure 7. Three ways that the CareerOrganiser application presents the user with relevant data found in the user model: auto-complete (left), pop-up (middle), and auto-fill (right).

In more detail, Figure 7 shows the different ways in which the CareerOrganiser is able to resolve and present the user with relevant data that it has found in the user model during resume creation. The resume creation process in the CareerOrganiser lets users create a resume section-by-section. Before asking input from the user for each of the sections, the CareerOrganiser's PersonalisationService class (which manages the application's resolvers that were described above) accesses the user model and links this information to the user interface components. Figure 7 (left) shows the auto-complete method, which provides values found from the other models as the user types in the text field. Users also have the ability to view all found values for an attribute by holding down on the text field (Figure 7, middle), which is particularly useful when there are many different values. Figure 7 (right) shows the auto-fill option, which combines information from all available sources as best it can. In a formative usability study with seven participants, it was found that on a scale of 1=completely disagree to 5=fully agree, subjects said that “I could imagine creating the resume with the provided interface” (4), and that “it was easy to modify existing values” (4), “it was easy to add new values” (4.1), and “it was easy to add new sections to the resume” (4.1).

6. EASE OF INTEGRATION AND PERFORMANCE

Regarding ease of integration, an application like the CareerOrganiser typically interacts with the PersonisJ user model(s) at three different points during application use. Firstly, at start-up, the application is required to check if the user model already exists and to create the model if it does not already exist. The next time an application interacts with the model is when it has new user information to contribute to the model (e.g. when a new resume is created or when an existing job filter is modified). Such information is generally stored in an application's internal database (e.g. SQLite) and it is at

this point that the changes are also typically added to the PersonisJ user model. For both the creation of the application's model (i.e. via the PersonisJ import function) and the adding of new user information (i.e. via the tell operation), integration is simple, requiring little more than a few lines of code. The third point at which an application typically accesses the user model is when the application wishes to retrieve information from the model. Integration here can be more demanding. An application first needs to check that the model it wants information from actually exists (i.e. via the GetComponentByPath method shown in Section 5). The application may also need to implement a custom resolver function if the default resolvers do not perform the required task, e.g. as is the case when retrieving evidence like that shown in Figure 6 consisting of dynamic lists of lists (i.e. multiple past employment evidence contained within multiple resumes). However, once a resolver is created, the developer will likely reuse it within other applications (e.g. the resolver created for the CareerOrganiser to resolve personal information is also used within the UniversityApplication and CasualEmployment applications).

The performance of PersonisJ was tested using a 1GHz HTC Nexus One Android smartphone using the profiler built into the Android framework. The performance of the “tell” operation was already shown in (Gerber et al., 2010) to be sufficient for use on mobile devices (nearly 2.5x faster than the time to inflate a trivial XML user interface in Android consisting of one button). Simple “ask” operations in comparison were shown to take three times as long as the “tell” operation. As stated in (Gerber et al., 2010), the execution time of an ask operation is dependent on the supplied parameters (e.g. the complexity of the model being queried and the complexity of the resolver and filter that is being used). Using the model described in Figure 5, we demonstrate

via an example below, the performance gains that can be achieved by offloading part of the model schema into the structure of the accreted evidence.

In particular, consider the “Personal information” component in Figure 5, i.e.:

```
"/Applications/CareerOrganiser/Resume/PersonalInformation".
```

The evidence that is accreted by this component (which we call Model 1) contains eight individual attribute-value pairs (first_name, last_name, phone, email, website, gender, dob, and work-permit) represented as a single JSON string as follows:

```
{"first_name": "Jon", "last_name": "Smith" ... }
```

Consider now a second model, Model 2, in which personal information is stored as a context (rather than a component) and each of the 8 attribute-value pairs are themselves represented as components and evidence respectively. Retrieving the personal information from Model 2 requires eight different “ask” operations. Using a similar setup to that used in the previous study (Gerber et al., 2010) – i.e. ask operations that use the default String resolver and latest-evidence filter, and averages calculated across 200 repeats – we calculated that the time to execute these eight ask operations was 523.27ms in total. In comparison, the time to execute the single ask operation required in Model 1 was calculated to take only 66.53ms. The difference (in this case a factor of 7.86, i.e. ~8) is further magnified when information from multiple models is being requested. As shown in the user interfaces in Figure 7, the CareerOrganiser application guides the user in filling out their resume by first retrieving the personal information contained within the UniversityApplication, CasualEmployment, and Facebook applications. Assuming similar-sized models for these other three applications, the time difference is then 200ms versus 1,570ms, i.e. a vast improvement. Although hardware technology advances will improve performance, life-long user models will also grow over time; this performance evaluation shows one way in which large client-side user models based on the accretion/resolution modelling approach can be optimised to avoid unresponsive application behaviour.

7. CONCLUSIONS

This paper represents the first evaluation of the use of PersonisJ - or any other mobile client-side user modelling system - in which multiple mobile applications reuse the user model across applications and for the benefit of mobile personalisation. It outlined three separate mobile applications, each novel in its own right, and each with a strong focus on mobile personalisation and model reuse across the applications. The paper provided a design methodology for integrating, into mobile client-side applications, models based on the accretion/resolution approach. We provided practical examples showing how to build PersonisJ-compliant applications and demonstrated via performance testing and model optimisation how PersonisJ is suitable for mobile device use, even when used on resource-limited devices like

smartphones and across complex user models covering multiple application schemas.

ACKNOWLEDGMENTS

This work is partially funded by the Smart Services CRC, as part of the Multi-channel Content Delivery and Mobile Personalisation Project.

REFERENCES

- Assad, M. Active Models for Pervasive Computing. Ph.D. thesis, The University of Sydney, School of IT (2010).
- Chen, G., Rahman, F. Analyzing Privacy Designs of Mobile Social Networking Applications. In: IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (2008), 83-88.
- Church, K., Smyth, B. Who, What, Where & When: A New Approach to Mobile Search. In: Proceedings of the 13th International Conference on Intelligent User Interfaces, ACM (2008), 309-312.
- Cozza, R. Gartner Says PC Vendors Eyeing Booming Smartphone Market. Gartner Report, URL: <http://www.gartner.com/it/page.jsp?id=1215932> (2009).
- Davidyuk, O., Riekkki, J., Rautio, V-M., Sun, J. Context-Aware Middleware for Mobile Multimedia Applications. In: Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia, ACM (2004), 213-220.
- Debaty, P., Goddi, P., Vorbau, A. Integrating the Physical World with the Web to Enable Context-Enhanced Services. In: Mobile Networks and Applications, Springer (2005), 10 (4): 385-394.
- Fischer, G.: User Modeling in Human-Computer Interaction. In: User Modeling and User-Adapted Interaction, Kluwer Academic Publishers (2001), 11(1-2): 65-86.
- Gerber, S., Fry, M., Kay, J., Kummerfeld, B., Pink, G., Wasinger, R. PersonisJ: Mobile, Client-Side User Modelling. In: Proceedings of the 18th International Conference on User Modeling, Adaptation and Personalization, Springer-Verlag (2010), 111-122.
- Heckmann, D. Ubiquitous User Modeling. Ph.D. thesis, Department of Computer Science, University of Saarland (2005).
- Kärkkäinen, T., Vaittinen, T., Väänänen-Vainio-Mattila, K. I Don't Mind Being Logged, but Want to Remain in Control: A Field Study of Mobile Activity and Context Logging. In: Proceedings of the 28th International Conference on Human Factors in Computing Systems, ACM (2010), 163-172.
- Kay, J., Kummerfeld, B., Lauder, P. Personis: A server for user models. In: Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Springer-Verlag (2002), 203-212.
- Kjeldskov, J., Paay, J. Just-For-Us: A Context-aware Mobile Information System Facilitating Sociality. In:

- Proceedings of the 7th International Conference on Human Computer Interaction with Mobile Devices and Service, ACM (2005), 23-30.
- Kobsa, A. Generic User Modeling Systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) Chapter in: *The Adaptive Web: Methods and Strategies of Web Personalization*, Springer-Verlag (2007), 136-154.
- Krüger, A., Butz, A., Müller, C., Stahl, C., Wasinger, R., Steinberg, K-E., Dirschl, A. The Connected User Interface: Realizing a Personal Situated Navigation Service, In: *Proceedings of the 9th International Conference on Intelligent User Interfaces* (2004), 161-168.
- Krüger, A., Baus, J., Heckmann, D., Kruppa, M., Wasinger, R. Adaptive Mobile Guides. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) Chapter in: *The Adaptive Web: Methods and Strategies of Web Personalization*, Springer-Verlag (2007), 521-549.
- Kurkovsky, S., and Harihar, K., Using Ubiquitous Computing in Interactive Mobile marketing. In: *Personal Ubiquitous Computing*, (2006), 10 (4): 227-240.
- Niu, W.T., Kay, J. PERSONAF: framework for personalised ontological reasoning in pervasive computing. In: *User Modeling and User-Adapted Interaction* (2010), 20: 1-40.
- Oei, T-Y. My Students, My Cellphone, My Ordeal. *Washington Post* (19.April.2009).
- Pink, G., Gerber, S., Fry, M., Kay, J., Kummerfeld, B., Wasinger, R. Safe Execution of Dynamically Loaded Code on Mobile Phones. In: *Proceedings of the 7th International ICST Conference on Mobile and Ubiquitous Systems* (2010).
- Shilton, K. Four billion little brothers? Privacy, mobile phones, and ubiquitous data collection. In: *Queue* (2009), 7 (7): 40-47.
- Tsai, J. Y., Kelley, P. G., Cranor, L. F., Sadeh, N. Location-Sharing Technologies: Privacy Risks and Controls. In: *A Journal of Law and Policy for the Information Society* (2010), 6 (2): 119-151.
- Weakliam, J., Lynch, D., Doyle, J., Min Zhou, H., Aoidh, E.M., Bertolotto, M., Wilson, D.: Managing Spatial Knowledge for Mobile Personalized Applications. In: *Proceedings of Knowledge-Based Intelligent Information and Engineering Systems*, Springer-Verlag (2005), 329–335.

School of Information Technologies
Faculty of Engineering & Information
Technologies
Level 2, SIT Building, J12
The University of Sydney
NSW 2006 Australia

T +61 2 9351 3423
F +61 2 9351 3838
E sit.information@sydney.edu.au
sydney.edu.au/it

ABN 15 211 513 464
CRICOS 00026A