Adapting Ubicomp Software and its Evaluation

Malcolm Hall, Marek Bell, Alistair Morrison, Stuart Reeves, Scott Sherwood, Matthew Chalmers
Department of Computing Science,University of Glasgow, UK.
{mh,marek,morrisaj,stuartr,sherwood,matthew}@dcs.gla.ac.uk

ABSTRACT
We describe work in progress on tools and infrastructure to support adaptive component-based software for mobile devices—in our case, Apple iPhones. Our high level aim is 'design for appropriation', i.e. system design for uses and contexts that designers may not be able to fully predict or model in advance. Logs of users' system operation are streamed back in real time to evaluators' data visualisation tools, so that they can assess design problems and opportunities. Evaluators and developers can then create new software components that are sent to the mobile devices. These components are either integrated automatically on the fly, or offered as recommendations for users to accept or reject. By connecting developers, users, and evaluators, we aim to quicken the pace of iterative design so as to improve the process of creating and sustaining contextually fitting software.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: *design studies, measurement techniques*; C.5.3 [**Computer System Implementation**]: Microcomputers—*portable devices*; D.2.1 [**Software Engineering**]: Requirements/Specifications—*elicitation methods*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*evolutionary prototyping*; D.2.5 [**Software Engineering**]: Testing and Debugging; D.2.7 [**Software Engineering**]: Distribution, Maintenance and Enhancement—*restructuring, version control*; K.6.3 [**Management of Computing and Information Systems**]: Software Management—*software development, software maintenance*.

General Terms
Design, Experimentation, Human Factors

Keywords
Adaptive evaluation, contextual software, ubiquitous computing

1. INTRODUCTION
Ubiquitous computing (ubicomp) has emerged as a key area of computer science. It deals with systems that fit with user context and interaction, and takes a holistic view spanning technology, use and users, in which "the unit of design should be social people, in their environment, plus your device" [16]. Robin Milner recently reflected on the ubicomp 'vision' in [13] and advocated "exploratory projects that aim to define the kinds of experience that lie at the core of the vision. This requires experiments that create specific socio-technical environments and ask humans to enter them. [...] Here we look for synergy between the societal vision on the one hand, and the development of scientific models and engineering principles on the other." Our work is in accord with this, in that it supports and connects users, evaluators and developers engaged in the process of creating ubicomp systems of sustained contextual utility and value.

Understanding and improving this iterative socio–technical process is important for ubicomp because it is what creates and sustains systems' contextual fit. Making systems more adaptable or adaptive in this way is particularly important in ubicomp because contexts, needs and uses are often more dynamic, subtle and hard to predict than in other areas of computer science. Actual use of ubicomp systems may differ from designers' preconceptions when, for example, mobile users are interacting in the uncontrolled environment of city streets. Software based on such preconceptions may become increasingly unhelpful or inappropriate unless it adapts or is adapted with use. This openness of use exacerbates the problem of evaluation in ubicomp. Those aiming to create new ubicomp systems or adapt existing ones in a timely fashion need to understand users' changing contexts and uses, but it is prohibitively difficult for evaluators to be with users all the time in their everyday lives, observing and recording where they go, what they do, and their interaction with people nearby and—via networked mobile devices, for example—other people in quite different contexts [5].

Our work aims to make a contribution by increasing the pace of the iterative process of design, use and evaluation. Logged usage data is streamed to evaluators in near real time, who work with developers to make software changes that are quickly distributed among users, whose use of the new software is logged and starts the loop again. We describe our initial implementation of an infrastructure for system logging and for component distribution and integration, along with prototype tools for visualisation of multiple devices' system logs. We report on an ongoing pilot trial in which we use the dynamism of system structure to adapt the logging subsystem of FanPhoto, a mobile social networking application used by football fans and developed within the Augmented Stadium project (UK EPSRC EP/E04848X/1). In collaboration with Contextual Software (EPSRC EP/F035586), another project of the Social/Ubiquitous/Mobile Group, developers assisted evaluators by making analysis tools that were simple but specific to the ongoing issues raised by the trial, as well as by the ongoing changes to FanPhoto. We discuss some of the technical and experiential challenges as well as longer-term plans for addressing them.

2. BACKGROUND AND RELATED WORK
A central but difficult issue for ubicomp system design is how to evaluate and model the changing contexts of users. Context and

EICS'09, July15-17, 2009, Pittsburgh, Pennsylvania, USA.

use are varied, dynamic and subjective, even though many systems rely solely on discrete geographical locations of users. The well-known paper by Dey et al. proposed that context includes "the location, identity and state of people, groups and computational and physical objects" [7]. This could include almost anything in human activity, but a system designer has to make some choice *a priori* as to what contextual features to model in the formal and objectifying medium of software. Gray and Salber, reflecting on [7], advanced system engineering for context-based systems by taking account of the inherent uncertainty of sensor data [11]. Chalmers described how history is a part of context, with consequences for theory as well as system design [4], and Dourish explained how, in their activity, people continually negotiate and adapt the contextual features that are significant or relevant [8]. People reflect on how they are modelled, i.e. the people using a system may take account of its model of their identities, locations and states, accommodating it or even appropriating it to their own ends rather as one controls and adjusts one's self–presentation to others in human–human communication. This reinforces the view that design should be treated as an iterative process that spans users, evaluators and developers, and that systems should be adaptive or adaptable.

Through repeated phases of design, implementation and evaluation, developers can change and improve their systems, and their understanding of use and context. However, evaluation in ubicomp can be complex in itself, and it can be difficult to communicate useful information back to the system developer. As Carter and Mankoff [2] put it, "Ubicomp systems [are] more difficult to evaluate than desktop applications. This difficulty is due to issues like scale and a tendency to apply Ubicomp in ongoing, daily life settings unlike task and work oriented desktop systems." Observational techniques founded in ethnography may be well suited in principle but in practice are often hampered because keeping up with the activity is difficult, small devices such as mobile phones and PDAs can easily be occluded from view, and people's use may be intimately related to and influenced by the activity of others far away [6]. System logs have often been used for debugging and for post-trial analysis, but streaming system log data back to evaluators in real time can be highly beneficial for letting them know where to focus their attention and resources [6, 15].

Some researchers have explored 'experience sampling' methods, that bring up a questionnaire on-screen when the mobile device detects that it is in a context of interest [10]. Carter and Mankoff recently developed Momento [3], which supports experience sampling, diary studies, capture of photos and sounds, and messaging from evaluators to participants. It uses SMS and MMS to send data between a participant's mobile device and an evaluator's desktop client.

Combining quantitative log data with qualitative data to give a more holistic view of ubicomp systems in use is the aim of the *Replayer* system [14], an integrated set of tools for display and interaction with different parts of the record of system operation, use and context. Log data visualisations combine many usage histories so as to support analysis of users acting in greater numbers, and at larger geographic and time scales, than one can directly observe. Complementary Replayer components support synchronised replays of audio and video recorded during user trials.

Ubicomp software is even harder to evaluate and analyse when it supports the adaptation and customisation needed to sustain contextual fit in long-term use. Such support is an especially important goal in ubicomp, as computers' use may extend beyond work activities, focused on pre-planned tasks, into leisure and domestic life. Recently, Humble et al. supported end–user adaptation in ubicomp via a distributed shared state model and a simple set of categories of transformation between physical effects and digital effects [12]. This was expressed using a 'jigsaw piece' editing metaphor, in which manual assembly of components is based on display of what a component could be connected to. *Speakeasy* [9] relied on a fixed set of interaction patterns and 'task templates' so as to allow a mobile component to be written with less prior knowledge of the particular components it will communicate with when deployed. In effect, many of components' combinations and uses are fixed in advance by the form of the template, and it was noted in [9] that a more dynamic approach would be preferable, i.e. one in which software structure is tailored to the user's context and history.

Using context and history in this way was the focus of our *Domino* toolkit [1]. Domino manages and supports adaptation of a set of .NET Compact Framework software components making up a ubicomp application. It uses mobile ad hoc networks and controlled epidemic algorithms to spread components and usage histories between users. It tracks and logs the current system 'context' in terms of the set of components currently running in the application. This set is used to filter usage histories in the course of making recommendations of new components to install and run. Therefore, instead of relying solely on predefined templates or patterns of use, like Speakeasy, Domino also takes advantage of emergent patterns of use in recommending and integrating components. This is done in a way that hides from the user much of the technical details of discovery and integration of new software, but reveals enough to let him or her maintain control over the system. When a user accepts a recommendation, Domino checks whether other currently running components satisfy the component's dependencies—required interfaces declared *a priori* by the developer—in an effort to ensure that its execution is technically feasible before trying to install it. If not, it suspends installation until it finds such components, which then can be recommended, accepted by the user, and installed. Dependencies specify objective constraints on component combination, like the connections in Humble's jigsaw editor and Speakeasy's templates, but Domino also takes advantage of the evolving patterns of use that represent users' subjective preferences about component combination.

3. INFRASTRUCTURE DESIGN

We now describe a new version of the Domino infrastructure, in ObjectiveC for the Apple iPhone, which is at the core of our ongoing work. We utilise the NSBundle feature of the Cocoa framework, taking advantage of its dynamic abilities. In essence, NSBundle is a folder containing program resources and executable code that can be loaded and unloaded at runtime. This folder is zipped for easy transportation over a network. A bundle contains an XML file (Info.plist) that specifies the principal class of the bundle. A pointer to the principal class can easily be obtained in code, so we set this class to be the Domino Module subclass in the bundle.

The Domino Module superclass has the ability to check for remote updates to the bundle, any time an instance of the subclass is created. The update compares the version number of the deployed component with a version number read from a remote server. If a newer version is found then the corresponding bundle zip is downloaded and extracted to the application's library directory, replacing the existing version. Currently the new version of the bundle will be used at the next application start, so the current launch isn't delayed by the check and download, however another strategy would be to use the old version initially then replace any references other classes have obtained to the module with a newer version later. A lightweight DominoBootstrap class is the only part of Domino that is built into an application. It has one method: a factory method for creating an instance of the module class from the latest version of the NSBundle. All other parts of the Domino framework can be updated with new modules.

SGLog is a generic logging framework we built for the iPhone. We use Domino to update the features of this implemented logging framework while trials are ongoing so as to record things we might have forgotten or whose importance became apparent only after a trial has started. SGLogItems contain timestamp, type, data, device ID, application ID, bundle ID, and app and bundle version number fields. The type field is a unique identifier that represents the type in the data field, e.g. SGLogLocation is used as the type when the data is an XML representation of the iPhone's location information obtained from the Core Location framework. The data field can be application-specific log strings. When a log entry is created, it is added to an in-memory array, so the application's current thread is not delayed, and a background thread is spawned to write the log to disk. SGLogItems use the NSDictionary's property list serialization methods for encoding and decoding to and from disk and offer both binary and XML representations. The log file is simply a concatenation of a length field and the binary property list. The aim is to have minimal data loss if the application crashes, and it also writes out the log on the application terminating event. Five seconds after a log has been created any logs on disk are read in and uploaded to a server, this time using the XML representation of the SGLogItem. The reason for the slight delay is to reduce network connection overhead if many items are created in quick succession. SGLog has a number of drop-in classes that developers can use to quickly log common data such as telephony events, like SMS sent/received, phone calls made/received, cell signal strength, cell ID, and 3G or 2G data modes. There are also classes for logging battery level and accelerometer data, and we are continually adding more.

The SGLog server backend is a MySQL database and Doctrine ORM (www.doctrine-project.org) that uses PHP running on OS X Server. We built an OS X application called Device Monitor that can be used to visualise the SGLog data. The application displays device status, such as when the last log was received, and also map and graphing features that can be used by evaluators. Device Monitor receives data from the SGLog server using a PHP web service, using the same property list serialization technique the iPhones use. We are also using a push technology design pattern called Comet so that Device Monitor can receive real time updates pushed from the server whenever the server receives new data from an iPhone. Our intention is that this real time infrastructure should support monitoring of possibly hundreds of users, although we have yet to investigate this in practice. We also use the Comet technique to push messages from Device Monitor to the iPhones, which lets us send messages to users.

4. A PILOT APPLICATION: FANPHOTO

This section describes FanPhoto, a simple photo and comment sharing application that was used as a pilot study for the adaptive component-based infrastructure described above. FanPhoto permits groups of friends to take photos, which are then automatically shared with other members of the group. These group members can also comment on one another's photos. FanPhoto is designed for use in and around stadium-based sporting events, particularly football events. Taking photos as souvenirs of a trip to a match is extremely common, as is sharing such souvenirs with fellow fans that were unable to attend a trip. It is often the case that fans are unable to attend a game as a complete group, with friends or family 'left behind' to watch on TV. FanPhoto supports such distributed groups through instant sharing of data.

4.1 FanPhoto and Domino

Studies of ubicomp systems can benefit from 'real-world' settings that are more representative of general day-to-day use than a purely lab-based evaluation. One of the main challenges in conducting evaluation in such a way is the problems created by time and space [15]. Trials that span larger periods of time and larger regions of space bring several logistical implications, in that they require more effort to deploy, manage and collect evaluation data. For evaluators, remaining in touch with the participants and studying the use of the system becomes challenging as evaluators cannot directly observe system use. This makes relying on log data and evaluation tools that support awareness very important. In running system trials for a long period there are two main issues that arise: ensuring a smooth technical experience for participants, and developing issues for research. So, ongoing observation of log data may result in interventions in order to fix technical problems and ensure the smooth running of a trial. On the second issue, there is potential for research questions to adapt based on these ongoing observations of interesting system use. As a result, it is often desirable for systems to be amended both to fix unforeseen technical errors and to gather data on the new aspects of system use under consideration.

Since user trials of FanPhoto operate at a city scale and last for weeks, it shows such requirements for adaptation. Therefore the Domino component architecture described above is a good match with FanPhoto. In our initial trials, we have not yet experimented with the addition of new components to extend system functionality for end users, but we have used Domino to update the components used in logging usage data. Remotely installing and dynamically loading new logging components at runtime has enabled evaluators to gain new insights and examine different aspects of system use, while reducing the need for redeployment and the subsequent interruption to the user experience.

In its initial deployment, FanPhoto was created with a minimal stub of a Domino module shell that didn't do anything, i.e. a seed to be used to log more interesting things. This module, the SGLog infrastructure, and Device Monitor were all evolving at a fast pace while the trial was in progress. Module updates were done silently so that the users were not aware anything had changed. The first update to the module was to utilise SGLog to log application start-ups, so that the evaluators could view a graph of participants'

activity in Device Monitor and could then contact users who had generated no usage data to find out why. The second update to the module was to log when the application quit, with the aim to visualise session length as a time series. A problem developed in the analysis of this log data with regard to calculating the session duration. If FanPhoto crashed then an application quit message was not logged, thus pairs of 'start' messages would appear in the log. Since we have the ability to fix such problems in situ on the phone, we decided to update again - this time logging a 'session' object when the application quits, which logs the start time, quit time, and also the duration. Device Monitor could now visualise sessions with no pre-processing of the data, which helped keep the SGLog PHP service that feeds the graphs in Device Monitor very simple and generic.

5. INITIAL EXPERIENCES

This section reports on ongoing trials of the FanPhoto system which we have used to pilot our infrastructure, exploring logging, updating software, and communication between users, evaluators and developers, and also piloted our approach of building simple analysis tools on the fly to address specific evaluation issues. We examine the findings from the trial from the perspective of developers, users and evaluators.

5.1 The Developer Experience

One of the challenges we encountered in FanPhoto was the issue of testing module updates before deploying them to users. Because some users could be upgrading from various different versions of the module, depending on when they last ran the application, we considered testing every upgrade path. This would be adequate for testing the release of version 1.2 with upgrades from 1.0 to 1.2 and 1.1 to 1.2, but of course this approach does not scale to large numbers of versions. Another issue was how to test the update mechanism with our development iPhones before the trial users had access. Initially we created a development version of the module that uses a test server URL to check for updates instead of the users' 'real' URL. While simple variants could use parameters, e.g. a passed-in URL, the general approach of building additional 'development' versions of each module adds even more complexity to the developers' work.

We decided to build an access control system for the update server to overcome these issues. We can configure which device IDs have access to what update version. So, when a new module deployment needs to be tested, it can be made available to only the test devices. Then, once it is considered safe, it can be made available to the users. We can also control which version each device should upgrade to, which gives us the ability to try out different features on different user subgroups. A problem we envisage happening in the future is testing upgrades when users have multiple modules that have dependencies on others. Because of the multiple unique combinations we will need a good strategy for providing smooth updates. We plan to offer a feature whereby a user can roll back the state of his/her application, in the case that problems occur.

5.2 The User Experience

There have been two trials of FanPhoto, examining two separate categories of user: those who regularly attend football matches and those that do not. Both trial groups were told that the application was explicitly designed for fans, however the two groups used it in significantly different ways. For those who did not attend football matches, the application became a way of filling time when at work, exposing their daily activities to others, sharing jokes, and maintaining awareness between members of the group. The second group oriented their use around football matches, using photos of their experiences at matches or watching matches on TV in order to co-ordinate talk about the game when in physically different locations. In the football fan study, we deployed the system to two distinct groups of fans (a group of 6 and a group of 4, all males in their 20s and 30s). It was important for our concerns for naturalistic study that we test the system with pre-existing friendship groups, such that members of each group already regularly went to matches together.

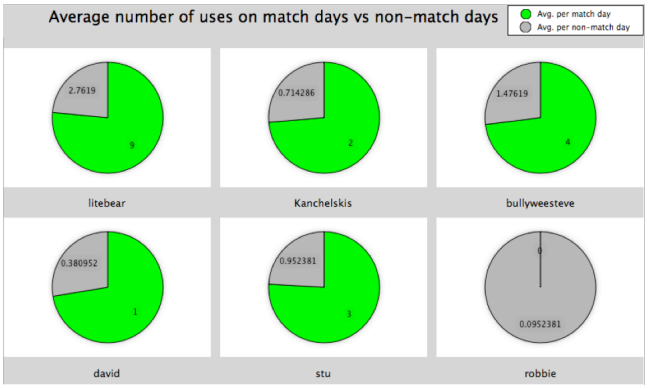

Figure 1. Pie charts for each participant, showing the numbers of photos taken on match and non-match days

Figure 1 shows usage data from the trial of the system involving football fans, graphed in Device Monitor. Pie charts are shown for each trial participant, showing the average number of photos taken on match days compared to the average number of photos taken on non-match days. It can be seen that for the majority of users, more photos were taken on the day of the games. Interestingly, most of the participants were not in attendance at the stadium, yet still took photos to share their experiences supporting their team from afar.

5.3 The Evaluator Experience

Integrating Domino into FanPhoto provided the evaluators with greater opportunity to explore the use of the system as it unfolded. As the trial progressed, evaluators developed new requirements for tools that would aid in maintaining awareness of the state of the trial and the system use. This involved the development of both new logging modules, as mentioned above, and visualisation components to explore the generated logs.

In initial studies of the use of FanPhoto, evaluators looked at the numbers of photos and comments made by each participant, to examine how active each participant was in using the system, and to look at features such as the relative proportions of photos taken to comments made. In a departure from normal evaluation practice, we chose to develop visualisation tools that, while simple, were specifically developed to examine such data and address evaluators' questions – for example that shown in Figure 2 (below).

Over the course of the trial, evaluators were aware that examining only counts of photos and comments did not provide an accurate reflection of participant usage. Certain users appeared to be less

active than others, but evaluators were unable to ascertain whether these users were actively viewing the content generated by others or simply not running the system at all. Having identified the requirement to look more thoroughly at participant activity, developers created the logging component described above, which noted every time that the application was launched or closed. As FanPhoto (in common with general iPhone applications) is unable to run as a background process, these logged times provide a timeframe at which FanPhoto was the active application on the phone. A new visualisation was also created in Device Monitor to view this data, as shown in Figure 3.

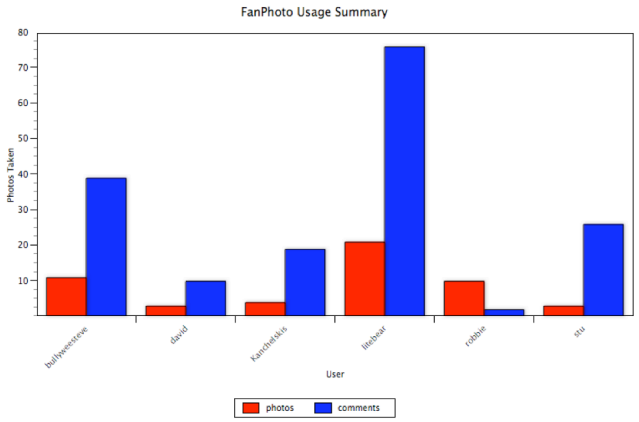

Figure 2. A bar chart showing photo and comment numbers

The *x*-axis of Figure 3 represents time and each trial participant is given a separate row on the *y*-axis. A blue line is drawn in the participant's row during the times at which the trial application was running. The analyst can view the full trial timeframe, as in the figure, or zoom in to see a single day's activity in detail.

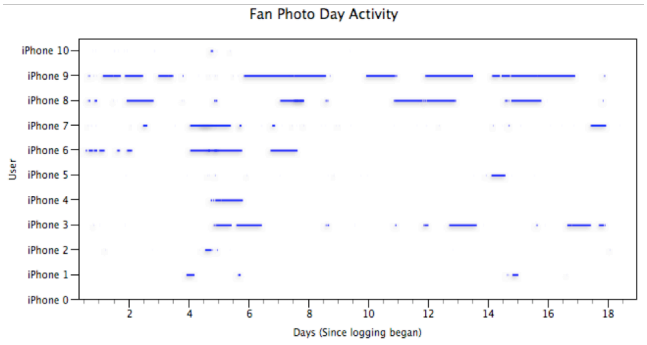

Figure 3. Overview of FanPhoto usage data, with a horizontal line drawn for each session of each user

The logging component did not take into account when the phone sleeps with the application still open, so a further iteration was made so that sleep and wake events were logged and taken into account in the analysis. Thus the adapted system supported an even more accurate session duration.

The timeline can also show the times at which photos were taken, linked to a photo browser, as shown in Figure 4. Here, each taken photo is represented as a grey circle on the timeline, with the one currently shown on the right represented as a red triangle. Analysts can click on the circles to browse the photo collection and the timeline shows an overview of the photos taken by each user and their temporal distribution. We are currently adding a map view based on GPS log data, to show spatial distribution too.

Figure 4. The FanPhoto image browser, based on a timeline of users' photo taking, shows the distribution of photo activity

The image browser in Figure 4, and perhaps the overview tool of Figure 3, begin to go beyond the simple facilities of everyday data analysis tools such as Excel. At the time of writing, this user trial is still ongoing, and so far these tools have proved to be a valuable resource in keeping in touch with the trial and monitoring the system's use. However, as we go further into the user experience, create richer system logs, and create more complex assemblies of software, we expect to have to go further with regard to visualisation tools that let us understand and orchestrate both the structure of our systems and the structure of our evaluation.

6. ONGOING AND FUTURE WORK

We thought it prudent to make our first experiments with our new infrastructure less apparent to users. By adapting our system loggers, we were able to pilot some of the key mechanisms of that infrastructure, and to develop new subsystems that made our work easier, such as the access control system. However, we will soon run another system trial in which the usage data, along with comments from and interviews with users, is used to create new modules to alter or add to the system functionality, which will be distributed among users and offered as recommendations.

On the developer side, we expect several challenging issues to become apparent. Firstly, having multiple versions of code modules active among a user population will further complicate testing and debugging. Log viewers that let us look at system log data as it streams in across the network in near real time will have to show not just which modules but which versions of modules are being used on each device. Since each device may have different sets of modules as well as different versions of each module, we are in the process of designing an analysis tool that makes an interactive 2D layout of devices being used in the trial, with devices with similar sets of components/versions clustered together while dissimilar ensembles are far apart. By cross-linking this tool with the log viewer, we hope that this will let us answer questions such as which log entries (and which bugs, module interdependencies and other associated design issues) are associated with which combinations of modules/versions. We also note that our system uses open source products, and we will make our code open source, for others to experiment with and extend.

On the user side, the fact that the system is dynamically adaptable and/or adaptive may potentially be difficult to handle, for some, but we see this as a research issue and a design challenge. Some mechanisms may be familiar, such as the Apple App Store's

updates, but we aim show users more of the contexts that led to a particular module's recommendation, beyond patterns of combination with other modules. While current trial participants are aware and accepting of the fact that full log data is sent to evaluators, when such data is used for recommendations then it will also be shared between users. We could hide it within the system, and avoid privacy concerns to do with such inter-user sharing, e.g. use a ranked list of recommendations that shows nothing of who they came from. However, since we intend to offer users a richer basis for choosing what software to trust and use, we prefer an open approach in which such sharing is explicitly treated as self-presentation. Tools to control how one's usage data is presented to others will thus let users affect others' recommendations in ways appropriate to the community of use.

On the evaluation side, developing custom analysis tools has helped us focus more clearly on what a particular trial needs, rather than blindly using more standard tools and techniques. The log viewer and component ensemble visualisation mentioned above may be useful for getting underlying technical detail on problems and issues that users have, but in the long run we expect to develop other tools that support remote interaction with users, letting them add comments and annotations to their own system logs as well as supporting more direct communication between evaluators and users.

7. CONCLUSION

We have outlined a dynamic approach to not only the structure of ubicomp systems, but the tools used for orchestration and analysis in user trials. Adapting the logging in a mobile application led to changes for developers and evaluators too. New log data led to new analysis tools that assisted understanding of user activity as well as the development of new software—which in turn led to new questions and issues to resolve. While the work described is preliminary, it revealed some concrete technical issues and solutions that we have addressed or are addressing, such as the utility of making logging software adaptable, of access control systems for managing the testing and release of new module versions, and the need for new development/evaluation tools to handle the complexity of software used in a wide geographic area for a substantial period of time, and made up of many different sets of related modules. As we move towards supporting adaptation in end user applications, we maintain our belief that it is appropriate, feasible and useful to take a holistic view, accepting that change in one part of this complex ensemble of devices, tools, users, evaluators and developers should lead to changes in other parts. We see this as a promising way to achieve ubicomp's aim of sustaining the contextual fit of software despite the inevitable changes to system contexts and uses, although we readily accept that further experimentation and research is necessary in order to settle such an issue.

8. REFERENCES

[1] Bell, M., Hall, M., Chalmers, M., Gray, P. and Brown, B. 2006. Domino: Exploring Mobile Collaborative Software Adaptation. In Proceedings of the International Conference on Pervasive Computing. 153-168.

[2] Carter, S. and Mankoff, J. 2005. Prototypes in the Wild: Lessons Learned from Evaluating Three Ubicomp Systems. IEEE Pervasive. 4(4). 51-57.

[3] Carter, S., Mankoff, J. and Heer, J. 2007. Momento: Support for Situated Ubicomp Experimentation. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI2007. 125-134.

[4] Chalmers, M. 2004. A Historical View of Context. Computer Supported Cooperative Work. 13(3). 223-247.

[5] Crabtree, M., Benford, S., Greenhalgh, C., Tennent, P., Chalmers, M. and Brown, B. 2006. Supporting Ethnographic Studies of Ubiquitous Computing in the Wild. In Proceedings of the ACM Conference on Designing Interactive Systems. DIS2006. 60-69.

[6] Crabtree, A., Benford, S., Rodden, T., Greenhalgh, C., Flintham, M., Anastasi, R., Drozd, A., Adams, M., Row-Farr, J., Tandavantij, N. and Steed, A. 2004. Orchestrating a mixed reality game 'on the ground'. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI2004. 391-398.

[7] Dey, A., Salber, D. and Abowd, G. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human Computer Interaction. 97-167.

[8] Dourish, P. 2004. What We Talk About When We Talk About Context. Personal and Ubiquitous Computing. 8(1). 19–30.

[9] Edwards, W. K., Newman, M., Sedivy, J., Smith, T. and Izadi, S. 2002. Challenge: Recombinant Computing and the Speakeasy Approach. In Proceedings of the International Conference on Mobile Computing and Networking. ACM MobiCom. 279-286.

[10] Froehlich, J., Chen, M. Y., Smith, I. and Potter, F. 2006. Voting With Your Feet: An Investigative Study of the Relationship Between Place Visit Behavior and Preference. In Proceedings of the International Conference on Ubiquitous Computing. Ubicomp 2006. 333-350.

[11] Gray, P. and Salber, D. 2001. Modelling and Using Sensed Context Information in the Design of Interactive Applications: EHCI. LNCS 2254. 317–335.

[12] Humble, J., Crabtree, A., Hemmings, T., Akesson, K.-P., Koleva, B., Rodden, T. and Hansson, P. 2003. 'Playing with your bits': user-composition of ubiquitous domestic environments. In Proceedings of the International Conference on Ubiquitous Computing. Ubicomp 2003. 256-263.

[13] Milner, R. Ubiquitous Computing: Shall We Understand It? 2006. The Computer Journal. 49. 383-389.

[14] Morrison, A., Tennent, P., Williamson, J. and Chalmers, M. 2007. Using Location and Motion Data to Filter System Logs. In Proceedings of the International Conference on Pervasive Computing. 109-126.

[15] Sherwood, S., Reeves, S., Maitland, J., Morrison, A. and Chalmers, M. 2009. Adapting Evaluation to Study Behaviour in Context. International Journal of Mobile Human Computer Interaction. 1(2). 37-55.

[16] Weiser, M. 1994. Creating the Invisible Interface. Invited talk. ACM Symposium on User Interface Software and Technology. UIST.