# A Blackboard-Like Architecture for the Development of Evolving High Fidelity Mobile Application Prototypes

**Bodo Igler**
Department of Computer Science
bodo.igler@hs-rm.de

**Tobias Braumann**
Department of Computer Science
tobias.braumann@student.hs-rm.de

**Stephan Böhm**
Department of Media Management
stephan.boehm@hs-rm.de

RheinMain University of Applied Sciences, Wiesbaden, Germany

## ABSTRACT

The success of mobile applications depends on the incorporation of key features specific to their intended use. The identification of key features can be based on the iterative development of prototypes with varying features and increasing levels of fidelity. Mixed- to high-fidelity of the prototypes to both user interface and application/business logic features is required in order to support the normal usage context of mobile applications.

This paper proposes a framework for the development of evolving high-fidelity mobile application prototypes. A prototype family is produced in iterative-incremental cycles. In each iteration several prototype variants are built and evaluated. Feature selection and efficient creation of prototype variants are achieved with the help of a light-weight component model based on the blackboard architecture style.

A first version of the framework has been implemented and evaluated on the basis of one case study.

### Author Keywords

Mobile computing; design and evaluation methods; field studies; blackboard architecture; feature-oriented development; software product lines; SMAT

### ACM Classification Keywords

D.2.2. Design Tools and Techniques: Evolutionary Prototyping

### General Terms

Design; Human Factors

## INTRODUCTION

Modern smartphone platforms provide a number of innovative features which have prepared the ground for myriads of different mobile applications (apps) with novel functionality and unique user experience. While smartphone platforms keep on evolving, the variety grows even wider and developers are challenged to focus on the right features, as they want to guarantee the success of their apps.

There are extensive investigations on key features of mobile applications, ranging from general studies on user acceptance, quality of experience and usability [13, 3, 2] to research on success factors in specific application domains, e.g. [10]. [2] evaluate more than one hundred mobile usability studies from 2000 until 2010. They observe that most of these studies are based on laboratory tests and that "there is no usability evaluation framework that yet exists in the context of a mobile computing environment". Laboratory environments can only partially simulate the normal usage context of mobile applications (natural motion, interruptions, multitasking and noise) [16]. Mixed- to high-fidelity prototypes can be used in the laboratory as well as in field tests. This paper proposes a framework which facilitates the development of mixed- to high-fidelity prototypes whose iterative evaluation and re-design is the basis for the identification of key features.

The framework is required to satisfy three main objectives:

1. Field-Test Support: The prototypes can be used both in the laboratory and in the field.
2. High Fidelity: Fidelity in the context of this paper concerns the user interface properties as well as rich application and business logic of mobile applications.
3. Design Change Efficiency: Changes to the UI as well as to the application and business logic can be implemented in short iteration cycles.

The proposed framework is mainly based on the blackboard architecture style. A first version of this framework has already been implemented during the research project SMAT (Success Factors of Mobile Application Design for Public Transportation). An evolving set of app prototypes has been produced within the framework and on the basis of end-user feedback.

## BACKGROUND AND RELATED WORK

There are several process models and approaches which emphasize user satisfaction. All agile process models [9] and, more specifically, user-centered design [5] emphasize user involvement. [6] present a framework which comprehensively combines user-centered design with agile software development. Application of these concepts in the context of this



*PID-MAD 2013*, Aug 27 2013, Munich, Germany.
(In Conjunction with *MobileHCI '13*, Aug 27-30 2013, Munich, Germany.)

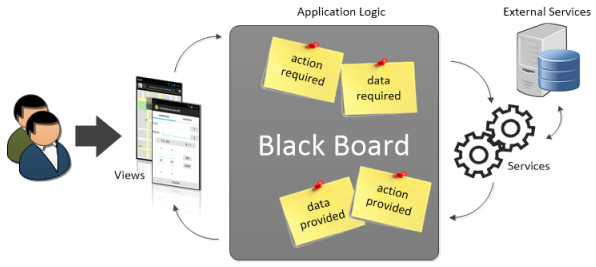


**Figure 1. Blackboard-Like Prototype Architecture**

paper leads to iterative development with the core activities *build* and *evaluate*.

Software Product Line Engineering (SPLE) considers the development of a product family (= software product line, SPL) [1]. All members (products) of a product family share a core of commonalities, while being distinguished by certain, varying features. This paper proposes to base the iterative development of a prototype on an **evolving** product family whose members are distinguished by different feature variants. There is little SPLE research for mobile applications. The existing research is mainly concerned with the variability introduced by different platforms [14]. The main interest of this paper with regards to SPLE lies in the development of a component model which covers the complete variability of evolving mobile app prototypes and which supports the efficient creation of prototype variants from feature choices.

There are several existing component models which can serve as the basis for mobile SPLs. These range from sophisticated service-oriented approaches like OSGi [11] to experimental agent-based approaches (cf. e.g. [12]). However, the existing approaches are too heavy-weight for the efficient creation of prototypes. A relatively light-weight approach which suits the purposes of the above-stated objectives is presented in the next section. It is based on the blackboard architecture style. According to [15] the blackboard architecture style consists of a central communication center, the *blackboard*, and a group of cooperating *experts* which utilize the blackboard to achieve a common goal. This architecture style has the advantage of conceptual simplicity and a high level of changeability. Its major disadvantages are: difficulties with the initial structuring of the problem to be solved, improper complex behavior of the loosely coupled *experts* and complicated routing.

## PROTOTYPING FRAMEWORK

This section outlines the conceptual and technical architecture of the framework. It concludes with a brief discussion of the experience gained by implementing and using a first version of the framework.

### Component Model

The proposed prototyping framework is conceptually based on a blackboard approach (see figure 1). Each prototype is decomposed into independent components. Each component focuses on a particular task, e.g. presenting a form to be filled

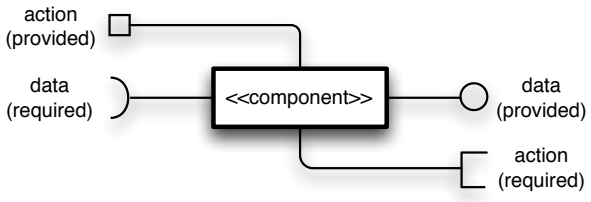


**Figure 2. Connector Types**

in by the user or connecting to a web service. The components communicate with each other via a communication center, the *blackboard*. The components are thus loosely coupled with respect to both control and data connections.

The component model comprises four connector types (see figure 2). Each component posts all the data it can provide (*provided data*) and all the events it accepts, i.e. all the *actions* it can perform (*provided action*), on the blackboard. Whenever a component needs data to perform a task, it posts a data request (*required data*) on the blackboard. It is the blackboard's responsibility to resolve the data dependency, i.e. to find the components which can provide the requested data. Possibly, the blackboard has to perform the data resolution recursively, as a component which provides the requested data may in turn request some other data to perform its task. A component can also post an event (*action required*). It is the blackboard's responsibility to route the event to the component which accepts the event and to resolve all data dependencies of the called component. Data resolution may again imply the invocation of several components.

The blackboard itself is not part of the component model. It is the instance which implements the underlying event routing and data resolution mechanism. This mechanism is illustrated with the help of an example:

Figure 3 displays a *SearchForm* view component which provides three pieces of information (*start*, *destination* and *dateTime*) and requires the action *displaySearchResults* to be performed. Figure 4 sketches the result of the corresponding routing/resolution procedure. The blackboard identifies the *SearchResultsView* view component as a component which provides the action *displaySearchResults*. However, before this component can be invoked its data requirements (*searchResults*) have to be satisfied. This data can be provided by the *SearchWebService* service component, whose data requirements are satisfied by the output of the *SearchForm* com-

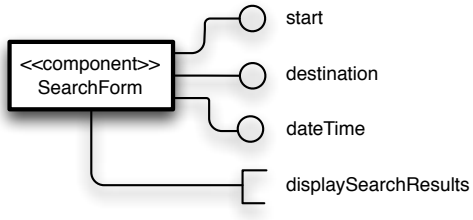


**Figure 3. Example Component**

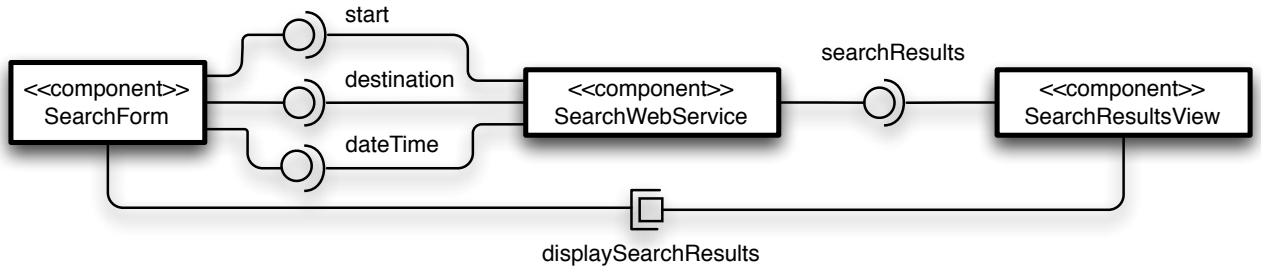


**Figure 4. Routing/Resolution Example**

ponent. After the blackboard has established this structure of matching components, it first invokes the *SearchWebService* component and then the *SearchResultsView* component.

### Architecture

The proposed framework is technically based on the Android OS platform. [4] This is mainly due to the qualifications of the computer science students at the RheinMain University of Applied Sciences: Android apps are implemented in Java and the computer science curriculum comprises a thorough introduction to Java. Android OS is an open platform which is freely available. Development and deployment of Android apps poses less thresholds for student developers in a heterogeneous and federated development environment.

Figure 5 outlines the framework architecture in the form of an FMC (Fundamental Modeling Concepts, cf. [7]) block diagram. Prototype apps are embedded into the client-side of the framework. Part of the client structure is predetermined by the Android architecture: A native Android app is directly connected via so-called *intents* to the *Android Runtime Environment*. Intents are the basis of the Android event system (cf. [4] for more details). The framework architecture extends the Android Runtime Environment by the *Runtime Environment Supplement*. This supplement intercepts the intents sent by the prototype app and implements the blackboard-like event routing and data resolution mechanism.

Two types of components are distinguished within the framework:

1. *View component*: User interface.
2. *Service component*: Business logic.

The framework concept *view component* can be directly mapped to the Android concept *activity*. Activities are the main components of each Android app. Each activity consists of one view. There is no direct match for the framework concept *service component*. In order to prevent confusion with the existing Android concept *service*, the name *helper* has been chosen for the implementation of framework *service components*.

The *Runtime Environment Supplement* comprises the *Prototype Configuration*, the *Blackboard Controller* and the *Blackboard Data*. The *Blackboard Controller* routes events (intents) and resolves data dependencies according to the *Prototype Configuration* and the current status of the *Blackboard Data*. The *Blackboard Controller* invokes components indirectly by sending a corresponding intent to the *Android Runtime Environment*. Therefore existing native Android apps can be connected to the *Runtime Environment Supplement* with minimum effort.

The *Prototype Configuration* is implemented as an extension of the XML-based Android manifest file and contains the definition of the components and their connectors. The *required data* and *provided data* posts are implemented via a simple key-value access paradigm. The blackboard data is stored in a non-SQL database. A copy of the blackboard data and, depending on the configuration, the inter-component-communication is stored in a central database on the *Test Support Server*. This facilitates later usability evaluations.

### Evaluation

During the project SMAT a first version of the proposed framework has been implemented and has been used for the iterative development of an evolving mobile application prototype in the domain of public transportation. The resulting prototype is fit for lab and field tests. It captures the whole range of variability introduced during the evolution of its features, including high fidelity features (e.g. location-based information). The effort for the incorporation of changes and new features has been generally low.

Only one of the major disadvantages of the blackboard architecture style emerged: Routing can become complicated and make e.g. debugging difficult. Insidious tight coupling in the sense of [8] further complicates the development. There is a tight coupling between the component definition in the prototype configuration (XML file) and the component implementation (Java source code). The coupling is insidious, as the coupling is invisible to the existing development tools. Problems with complex routing and insidious tight coupling can be mitigated or even prevented by improving the development environment.

## CONCLUSION

This paper reports on work in progress regarding the development of a framework for evolving high fidelity mobile application prototypes. The framework architecture has been designed on the basis of the blackboard architecture style. This is, to the authors' knowledge, the first framework of this type based on this architecture style. A first version of the framework has been implemented and successfully utilized. This indicates that the chosen approach is (in the context

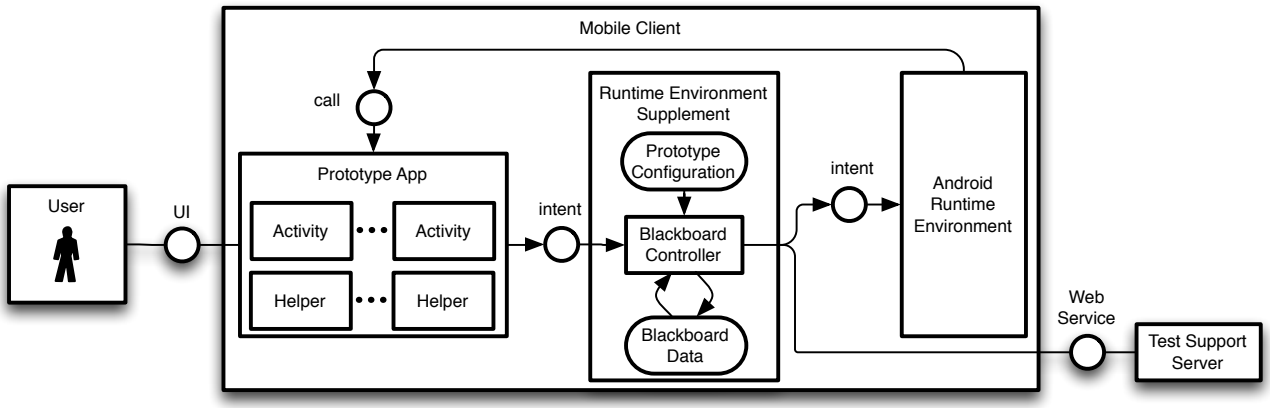


**Figure 5. Framework Architecture (Outline)**

of high-fidelity mobile application prototypes) a promising light-weight alternative to existing component architectures.

Further research and development will focus on enhancements of the framework and on the incorporation of usability evaluation features. The enhancements comprise better support for prototype app developers (e.g. make all relevant types of coupling visible to the tool chain) and for test managers (e.g. feature-based configuration interface). The incorporation of usability evaluation features will be based on the existing interceptor architecture.

## ACKNOWLEDGMENTS

Part of this paper is based on findings of the research project SMAT, which has been partially funded by the HMWK (Hessen State Ministry of Higher Education, Research and the Arts) under the program "Forschung für die Praxis".

## REFERENCES

1. Clements, P., and Northrop, L. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.
2. Coursaris, C. K., and Kim, D. J. A Meta-Analytical Review of Empirical Mobile Usability Studies. *J. Usability Studies 6*, 3 (May 2011), 11:117–11:171.
3. De Moor, K., Ketyko, I., Joseph, W., Deryckere, T., De Marez, L., Martens, L., and Verleye, G. Proposed framework for evaluating quality of experience in a mobile, testbed-oriented living lab setting. *Mob. Netw. Appl. 15*, 3 (June 2010), 378–391.
4. Google. Android developers. `http://developer.android.com`.
5. Gulliksen, J., Göransson, B., Boivie, I., Blomkvist, S., Persson, J., and Cajander, Å. Key principles for user-centred systems design. *Behaviour & IT 22*, 6 (2003), 397–409.
6. Humayoun, S. R., Dubinsky, Y., and Catarci, T. A three-fold integration framework to incorporate user-centered design into agile software development. In *Proceedings of the 2nd international conference on Human centered design*, HCD'11 (2011), 55–64.
7. Knoepfel, A., Groene, B., and Tabeling, P. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. Wiley, 2006.
8. Lewis, B. Insidious tight coupling. `http://www.drdobbs.com/architecture-and-design/insidious-tight-coupling/196802793`.
9. Martin, R. C. *Agile software development: principles, patterns, and practices*. Prentice Hall, 2012.
10. Niklas, S., Böhm, S., and Strohmeier, S. Mobile Job Board Applications – Which are the Key Success Factors? A Literature Review and Conceptual Framework. In *Proceedings of the 4th European Academic Workshop on Electronic Human Resource Management* (Nottingham, UK, 2012).
11. OSGi Alliance. `http://www.osgi.org/`.
12. Padovitz, A., Loke, S. W., and Zaslavsky, A. The ECORA framework: A hybrid architecture for context-oriented pervasive computing. *Pervasive Mob. Comput. 4*, 2 (Apr. 2008), 182–215.
13. Platzer, E., and Petrovic, O. Development of technology acceptance research for mobile services. In *MIPRO, 2010 Proceedings of the 33rd International Convention* (May 2010), 1154–1159.
14. Quinton, C., Mosser, S., Parra, C., and Duchien, L. Using multiple feature models to design applications for mobile phones. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC '11, ACM (New York, NY, USA, 2011), 23:1–23:8.
15. Shaw, M., and Garlan, D. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.
16. Tamminen, S., Oulasvirta, A., Toiskallio, K., and Kankainen, A. Understanding mobile contexts. *Personal Ubiquitous Comput. 8*, 2 (May 2004), 135–143.