An Active Database Framework for Adaptive Mobile Data Access *

Shiow-yang Wu and Chun-Shun Chang

Department of Computer Science and Information Engineering
National Dong Hwa University
Hualien, Taiwan, R.O.C.
showyang@csie.ndhu.edu.tw,
http://ravel.csie.ndhu.edu.tw/

Abstract. The ability to *adapt to changes* is among the most important issues in mobile information systems. We argue that active database system is the software architecture of choice as the foundation for adaptive mobile information services. We propose a modular framework and system architecture for building active mobile information systems by integrating off-the-shelf rule systems and database packages. For intelligent adaptation, a common *mobile event interface* is employed to detect the status changes. The rule system plays the role of a reactive component for making proper decisions in response to the changes. A general *database interface* ensures seamless integration of the rule system with any number of ODBC compliant database systems for flexible information access. We suggest implementation strategies and present performance results to demonstrate the feasibility of the proposed framework.

1 Introduction

In mobile computing environment, a mobile host is constantly subject to resource availability, changing mobile support stations and connection status, as well as terrain and weather. To cope with such a demanding environment, the ability to adapt to changes is among the most important system issues [5, 14]. Rudimentary mechanism for limited adaptation such as the power management facilities and "doze mode" operation on some portable devices are far from satisfactory. In many cases, optimal adaptation depends on both the resource and environment constraint, as well as the current status of operation. It may require instant and complex decision to be made on the fly.

As a motivating example, imagine the scenario in which a mobile user on a moving vehicle is approaching a mountain area where radio signal is expected to be weak or completely blocked. According to the current road and traffic condition and also from the speed and direction of the vehicle, no mobile support station will be available for the next half an hour before the vehicle passing

* This paper was partially supported by the National Science Council, Taiwan, R.O.C., under project NSC 86-2213-E-259-004.

through the mountain area, into the famous "semiconductor valley".[1] For assisting the user who must finish his/her work within half an hour for an important business meeting, the underlying information system has several choices. The easiest way is to take no specific action and run the risk of sudden disconnection, unfinished work, and a failed business trip. On the other hand, a highly adaptive system can sense the current situation, deduce key constraint mentioned above, and take the initiative in preparing for disconnected operation. Our goal is to apply intelligent techniques for adaptation in mobile information systems. In particular, we have identified the active database system concept as the building block for intelligent adaptation.

Active database systems have been shown to provide traditional databases with many desirable features [15]. For mobile information management, we found the active concept to be a power tool for intelligent adaptation. In particular, the event-driven interface is a natural candidate for sensing the resource availability as well as changes in the operating environment. The rule inference mechanism equips the system with sufficient arsenal to make intelligent decision and effective responses to any possible situation (such as intermittent disconnection). In this paper, we propose a modular framework for building adaptive mobile information systems based on the active database architecture.

Our framework is modular in the sense that a general software architecture is proposed to construct mobile information systems from existing enterprise databases using off-the-shelf rule systems and DBMS packages. Existing data can be used for active rule inference without any change to the DBMS. More specifically, we provide a *mobile event interface* as the active front-end. The detection and delivery of mobility, resource, and environment events enables an information system to be aware of current environment status and resource constraint. We construct a *database interface* on top of the widely accepted Open Database Connectivity (ODBC) standard [7] for seamless integration of rule languages with different database systems. This facilitates rule-based reasoning over any number of databases with ODBC driver support. A prototype implementation of our framework using CLIPS [2] and several PC database packages is reported in Section 6.

The rest of the paper is organized as follows. Section 2 enumerates research work on adaptation in mobile computing to position our work among others. Section 3 presents the proposed framework and the software architecture for its realization. Section 4 discusses the mobile event interface. Section 5 describes how we achieve easy access to multiple databases with the database interface. Section 6 provides our experience in prototype implementation and evaluation. Section 7 concludes the paper.

2 Related Work

Data management is emerging as one of the most promising area in mobile computing [1, 3, 9]. The need for adaptation is quickly identified as essential

[1] This is an imaginary scenario, therefore we avoid using any real place or name.

[5, 14]. Similar ideas have been discussed under terms like context-aware [12], application-aware [8, 11], and adaptive [4] information systems. However, related issues have rarely been addressed from a software engineering point of view. Among the few exceptions is the work done at CMU on both application-transparent and application-aware adaptation [10]. The advantage of application-transparent adaptation [6] is that no changes are needed to existing applications. However, such strategy is necessarily application-independent which cannot take advantages of any application-specific feature for optimal adaptation. On the other hand, the potential improvement offered by the application-aware adaptation may very well come with higher complexity and software development cost. Our main contribution is to provide a modular framework for an event-driven rule abstraction on top of existing information systems. With easy integration of off-the-shelf software packages without changes, and the rule system as a responsive component, we offer the benefits of both worlds.

3 System Architecture

Our design is driven by the following considerations. For dynamically responding to changes, a reactive component is mandatory. We extend the active database concept to satisfy this requirement. For resource and environment awareness, an *event subsystem* can be used to detect the changes and trigger the reactive component. The framework must allow easy access to databases of different types and vendors to cope with the heterogeneity of mobile computing. Finally, it is impractical to require transformation of existing data or modification to existing DBMS softwares. The framework should require minimal changes to existing systems. To summarize our design objectives:

1. A flexible event interface for responsive mobile computing.
2. A general mechanism to integrate event delivery with rule inference mechanism to enable the rule system as a reactive component.
3. Seamless integration of rule languages with database systems such that the inference power of the rule system can be used for intelligent adaptation.
4. Multiple and heterogeneous databases access by conforming to standards.
5. Reuse of off-the-shelf rule languages and database packages such that all current investment can be preserved and no data reformation is required.
6. The framework should be modular such that major components can be easily upgraded or even replaced to satisfy user demand.

We suggest an extended active database architecture consisting of four modular components for constructing client side mobile information systems:(1) a mobile event engine, (2) a rule system, (3) a database interface, and (4) a database system which is composed of a DBMS and a DB cache. Fig. 1 illustrates the components and their integration. Fig. 2 depicts the software architecture for the realization of our framework. The main idea is the shared event interface and database interface to allow easy integration of any rule system (such as CLIPS, OPS5, ...) with multiple database systems (such as Microsoft Access, DB Maker, ...). Details are discussed in the following sections.

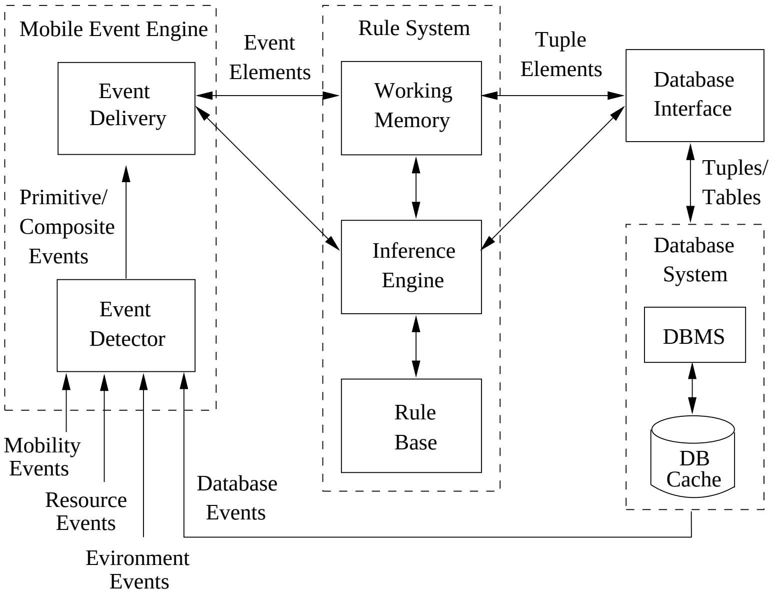

Fig. 1. A modular framework for building adaptive mobile information systems.

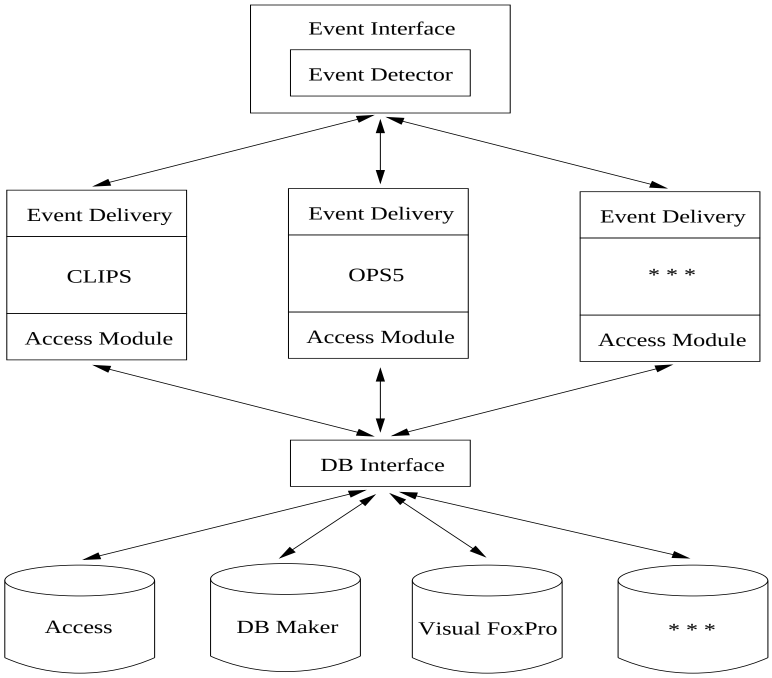

Fig. 2. Software architecture for building adaptive mobile information systems.

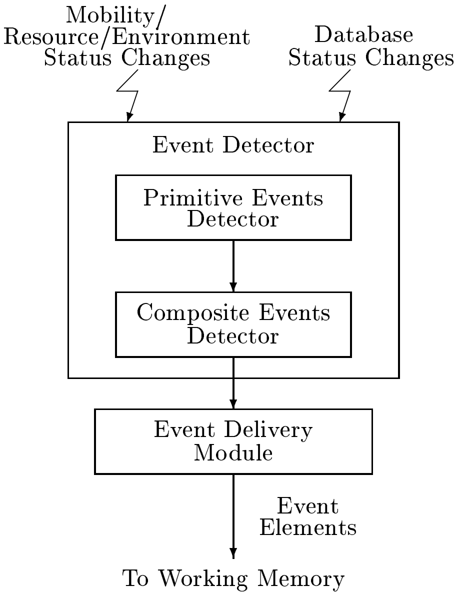

Fig. 3. The event interface.

4 Event Interface

The mobile event engine is the key component for resource and environment awareness. In additional to the traditional database and temporal events [15], we suggest the inclusion of mobility, resource, and environment events. Whenever a mobility variable (such as location, speed, etc.), the availability of a resource or the status of an environment variable has changed to a critical level, a corresponding event is triggered. These events are detected by the *event detector*. An event that is detected directly from a status change is called a *primitive event*. Primitive events can be combined to form *composite events*. When an event is detected, the *event delivery module* is responsible for transforming the event into an *event element* which is in exactly the same format as a *working memory element* of the rule system. In this way, the event engine can be easily integrated with the rule system without any change to the rule execution semantics or run-time system. The only modification to the rule system is a simple syntactic extension for users to specify the events of interest. Fig. 3 illustrates the structure of our event mechanism.

4.1 Mobile Event Specification

We classified the status changes in mobile hosts or the environment into three categories:

1. changes that are related or induced by the mobility of the mobile host;
2. changes in resource availability;
3. other status changes in the operating environment.

Events of the first two types usually occur with associated changes in certain measurable values such as the location change, low in battery power, and so on. The third type of event is for signaling other situations that may require immediate attention such as being in the process of hand-off. We use the names *mobility events*, *resource events*, and *environment events* for obvious reasons.

Mobility and resource events are specified in two forms:

On $\langle Variable\_Name \rangle$ $\langle Op \rangle$ $\langle Value \rangle$
On $\langle Variable\_Name \rangle$ **Level** $\langle Level \rangle$

The $\langle Variable\_Name \rangle$ can be the name of a mobility variable (such as **Location**, **Speed**, and **Direction**) or a resource variable (such as **Power**, **Memory**, and **Bandwidth**). The $\langle Level \rangle$ is currently defined to be either **High**, **Normal**, or **Low**.[2] The first form is for fine grain detection of the status change of the named variable while the second form is for testing the current availability level. For example, the resource event

On Bandwidth < 10

will be triggered when the current bandwidth is lower than 10 Kb/sec. Similarly, the event

On Power Level Low

will be triggered when the remaining battery power is lower than a predefined level.

Environment events are simply specified as:

On $\langle Condition \rangle$

where $\langle Condition \rangle$ could potentially be any situation of interest but we limit it to include only **Hand-off** at present.

Mobility, resource and environment events are primitive events which can be further combined into composite events using arbitrary combination of the logical connectives **AND** and **OR**.[3] The occurrences of both primitive and composite events are detected by the event detector and delivered to the rule system by the event delivery module. It is important to note that, regardless of the rule system of choice, the event specification and detection module are shared among all rule languages as indicated in Fig. 2. In other words, all events are specified using the same set of syntax and detected with the same set of library modules. No porting or re-implementation is needed on any specific rule system. This is a good example of our modular design. The only thing that is rule-system dependent is the event delivery module, since events must be converted into a form understood by the inference engine of the corresponding rule system.

[2] It is certainly desirable to have user-defined levels, which are included in our plans for the immediate future.

[3] We excluded negation because activating a rule on the non-existence of an event is in general not in accordance with the semantics of active database rules.

4.2 Event Delivery

The separation of event detection and delivery is to:

1. support uniform event interface;
2. allow code sharing on common functionality;
3. enable easy integration of event mechanism with any rule system of choice.

Whenever an event is detected, it is presented in a standard format. The event delivery module of a particular rule system is to convert all detected events into event elements which are in the form of working memory elements of the corresponding rule system. In this way, the inference engine of the rule system can be used directly without modification to react to any event occurrence. Event elements are kept in working memory until either removed explicitly by an action of a rule or until the end of the current inference session. An inference session begins when the rule system is triggered by certain event occurrences and ends when no rule is executable. Note that a separate event delivery module must be developed for each rule system since different rule systems have different formats for working memory elements. Nevertheless, the structure and functionality of the module are the same across rule systems. It is therefore quite straightforward to port the event delivery module to suit a rule system of choice.

5 Database Interface

The database interface is responsible for integrating rule system with DBMS such that the tuples in the database can be used directly by the rule system for reasoning and the actions of a rule can perform arbitrary database operations. We also must have the system interact with multiple and heterogeneous databases without modification to existing databases and DBMS packages. Our approach is to separate the database interface into a rule-system dependent *access module* and a *common database interface module* built on top of the ODBC standard.[4] The access module provides a set of access functions as an extension of rule actions for database access. Tuples accessed from the databases are converted into *tuple elements* which are in the form of working memory elements of the rule system. Data generated within the rule system can also be saved into databases by the access functions. The access module is necessarily rule-system dependent because different systems have different language syntax and data formats. The common database interface module, on the other hand, is rule-system and DBMS independent. It is used to access one or more databases which may be built by DBMS packages from different vendors as long as the packages support ODBC. The structure and relationship between various components described above are depicted in Fig. 4.

[4] Any open database connectivity standard could be used in here. ODBC is chosen for accessing PC-based databases that are readily available on our machines.

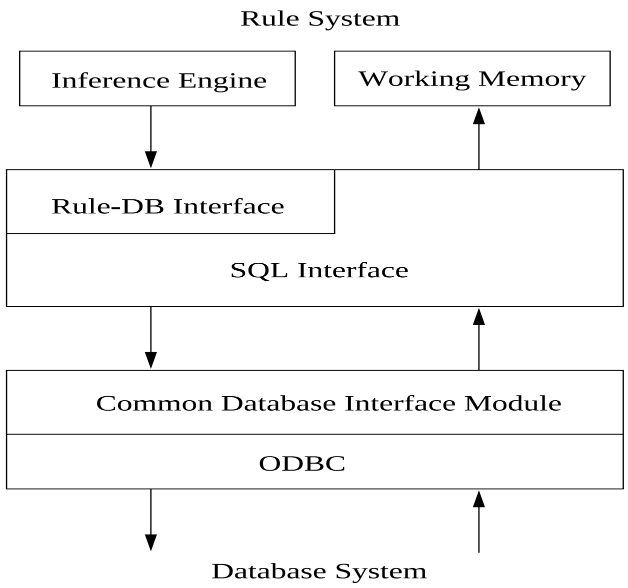

Fig. 4. The database interface.

5.1 Access Module

The design of the access module must satisfy two seemingly conflicting goals. It must be flexible and powerful enough to allow arbitrary database operations. At the same time, it must be simple and easy to use so that rule programmers with little knowledge about databases can still use it without any difficulty. Furthermore, it is desirable to have a common approach to database access to maintain a reasonable extent of uniformity across the access modules of different rule systems. We therefore provide two sets of functions for each access module:

1. *SQL interface*, for power users who want the full control and flexibility of database access; and
2. *Rule-DB interface*, for general rule programmers who may not know much about SQL.

The SQL interface provide a direct way of manipulating databases from rule actions using the SQL language. Each access function accepts a database name and an SQL statement. The programmer is free to use arbitrary SQL statements but is also responsible for their correctness.

The rule-DB interface consists of a **Load** and a **Save** function. The **Load** function accepts a database name and a table name. It reads the table from the designated database and inserts the tuples into working memory of the rule system. Tuples are inserted as elements of the working memory class with the same name as the table name. The **Save** function saves all elements of a working memory class into the table that bears the corresponding name.

Since access module is rule-system dependent, the syntax of both the SQL interface and rule-DB interface functions must be designed in the style of the corresponding rule system.

5.2 Common Database Interface Module

The access functions discussed above are necessarily rule-system dependent. Their functionality, however, are system independent. This is why a common database interface is possible and desirable. The interface can also be made independent of the underlying DBMS packages if a standard interface exists such that the actual database calls can be issued in a common format. We choose ODBC for such purpose. More specifically, we provide a shared library of functions that: (i) perform the necessary data conversion and aggregation; and (ii) issue ODBC calls to accomplish the desired functionality. In this way, we can access not only different databases with the same interface, but also multiple databases at the same time. The same interface can be used with any access module of the chosen rule system to communicate with any number of databases as illustrated in Fig. 2. No modification is required on either the existing databases or the underlying DBMS packages. This facilitates easy integration of existing data into the mobile information system as well as the objective of modular design.

6 Prototype Implementation

We have built a prototype system for evaluating the feasibility and potential of our framework. A Windows95 PC was chosen as the implementation platform since most of the existing portable computers are PC-based. Because of the space limit, we only discuss the database interface.

For the rule system, we chose wxCLIPS [13], a Windows version of the well-known CLIPS [2] expert system tool. For the database systems, any package with ODBC driver support would do. We selected Microsoft Access, Visual FoxPro, and DBMaker for their immediate availability on our machines.

The rule-DB interface is implemented by two functions: **LoadTable** and **SaveTemplate**, both in CLIPS syntax. The **LoadTable** is for loading database tuples into the rule system while **SaveTemplate** is for saving CLIPS data objects back to the database. The SQL interface is also implemented by two functions: **SQLtoFACT** and **SQLdo**. This is for power users who can use arbitrary SQL statements for better control over database operations.

The common database interface module is implemented by providing a layer of abstraction on top of ODBC. The layer is designed to hide the complexity of ODBC API so that the access functions could be implemented more easily.

For performance evaluation, we constructed a set of student databases using three different DBMS packages and wrote a rule program in CLIPS for computing GPA. Fig. 5 demonstrates the performance difference between using the CLIPS system alone and using the CLIPS integrated with a database (Visual

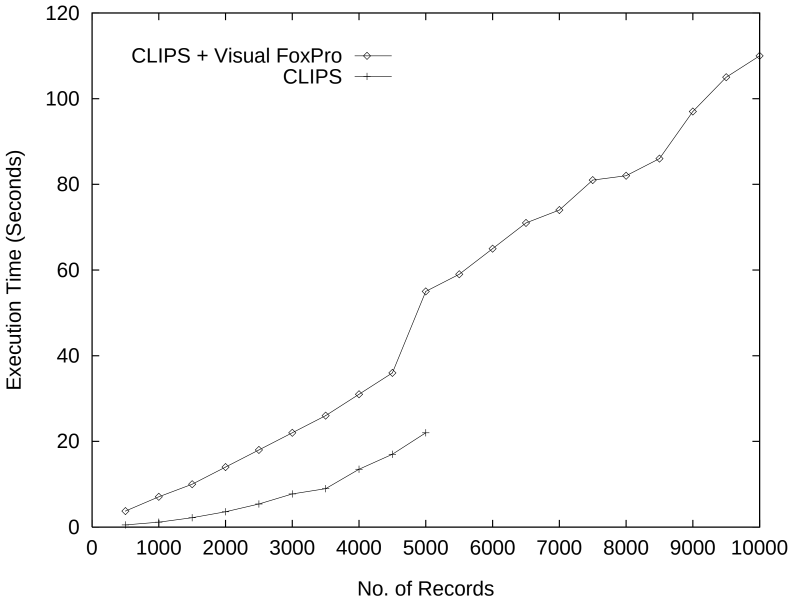

Fig. 5. Rule system (CLIPS) vs. database-enabled rule system (CLIPS + Visual FoxPro).

FoxPro). It is conceivable that without the additional overhead, the CLIPS system should be more efficient when the volume of data is small. However, the benefit of using database reveals itself when the data volume becomes larger. In particular, while the CLIPS system can only handle up to 5000 records before saturating the system resource, our implementation can handle a potentially unlimited number of records. We note that the processing time grows linearly with data size, which demonstrates the efficiency of our prototype implementation.

Fig. 6 compares the performance of the SQL interface and the rule-DB interface. The SQL interface is clearly more efficient. However, the rule-DB interface is much easier to use since it does not require prior knowledge about SQL.

Fig. 7 illustrates the performance of the database interface and also demonstrates the ability of our implementation to integrate the CLIPS system with different DBMS packages. All three packages exhibit linear growth of execution time with increasing data size, which indicates the limited overhead of our approach and the prototype implementation.

The performance results suggest that it is possible to achieve modularity and flexibility without loss of functionality and efficiency. We provide an easy integration mechanism to allow immediate extension of existing rule systems and database systems into adaptive mobile information systems. The integration can be achieved with low software development cost and good performance.

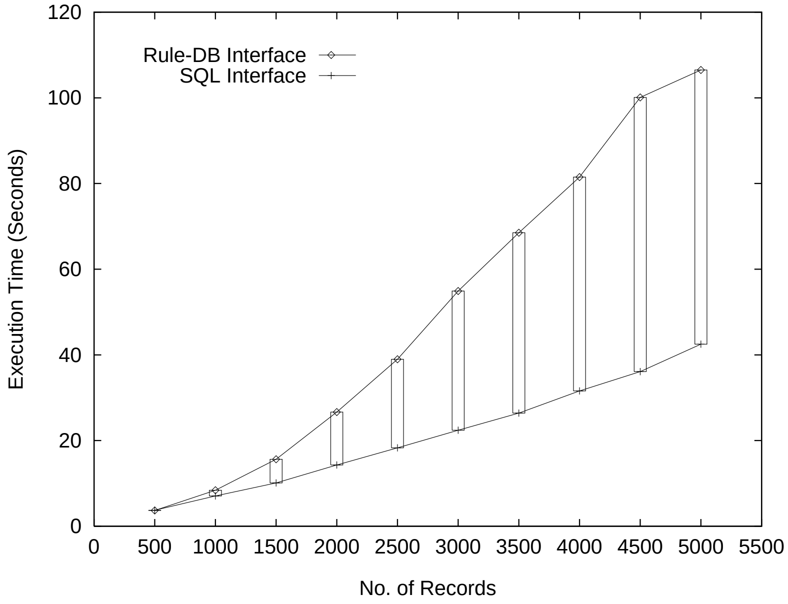

Fig. 6. SQL interface vs. rule-DB interface.

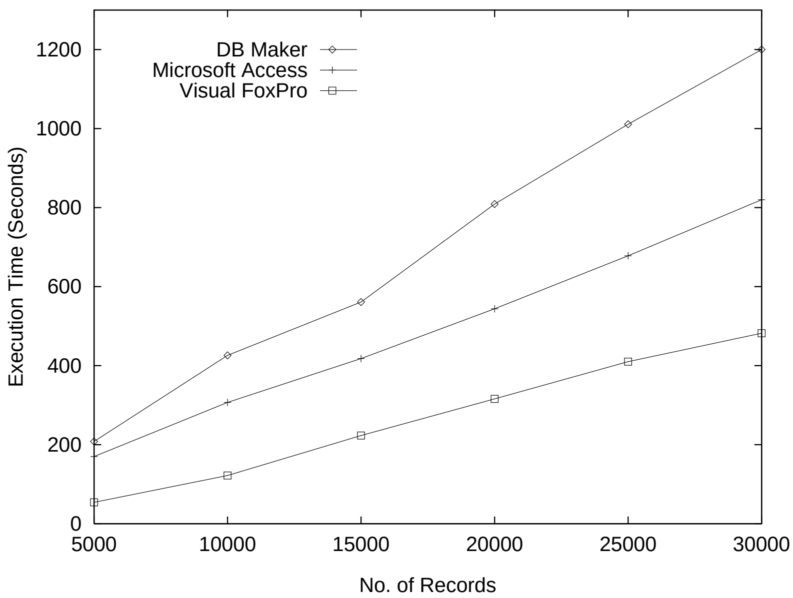

Fig. 7. Performance comparison of using different database packages.

7 Conclusions and Future Work

We proposed an active database framework for building adaptive mobile information systems. The key to intelligent adaptation is a mobile event interface and the integration of rule system with databases. Modularity is achieved through software architecture and techniques that can easily integrate off-the-shelf system components with existing data. A prototype implementation and evaluation results demonstrate the feasibility and potential of our approach.

This work is part of an ongoing project at National Dong Hwa University, Hualien, Taiwan, R.O.C. The goal is to develop necessary infrastructure, middleware, and software technologies for responsive mobile computing. We plan to apply the technologies on an experimental hospital information system and a campus information system.

References

[1] B. Bruegge and B. Bennington. Applications of mobile computing and communications. *IEEE Personal Communications*, 3(1), February 1996.

[2] Lyndon B. Johnson Space Center. CLIPS: A tool for building expert systems. (`http:// www.jsc.nasa.gov/~clips/CLIPS.html`).

[3] T. Imielinski and B. R. Badrinath. Wireless mobile computing: Challenges in data management. *Comm ACM*, 37(10):19–28, October 1994.

[4] T. Imielinski and S. Vishwanathan. Adaptive wireless information systems. Technical report, Department of Computer Science, Rutgers University, 1994.

[5] R. H. Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, 1(1):6–17, 1994.

[6] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[7] C. Lambert. ODBC: Architecture, performance, and tuning. ODBC Product Group, Microsoft Corporation, 1995.

[8] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility. In *16th ACM Symposium on Operating System Principles*, 1997.

[9] E. Pitoura and B. Bhargava. Building information systems for mobile environments. In *Third International Conference on Information and Knowledge Management*, pages 371–378, November 1994.

[10] M. Satyanarayanan. Mobile information access. *IEEE Personal Communications*, 3(1), February 1996.

[11] M. Satyanarayanan, B. Noble, P. Kumar, and M. Price. Application-aware adaptation for mobile computing. *Operating System Review*, 29, January 1995.

[12] B. Schilit, N. Adams, and R. Want. Context-aware mobile applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., December 1994.

[13] Julian Smart. wxCLIPS. (`http:// web.ukonline.co.uk/julian.smart/wxclips/`).

[14] T. Watson. Application design for wireless computing. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., December 1994.

[15] J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann, San Francisco, California, 1996.