COMPONENT-BASED FRAMEWORK FOR MOBILE DATA MINING WITH SUPPORT FOR REAL-TIME SENSORS

Taneli Rautio, Perttu Laurinen and Juha Röning
Intelligent Systems Group, Department of Electrical and Information Engineering, FI-900014 University of Oulu, Finland

Keywords: Mobile data mining, Component-based framework.

Abstract: The increasing use of various mobile devices has shown that there is a need for mobile data mining applications. While many existing data mining frameworks can be modified to handle data streams generated in real time, they are usually too complex and inflexible to be used in mobile devices. This paper presents Mobile Smart Archive, a component-based framework for data stream mining in mobile devices. The framework takes care of generic data mining operations, allowing the application developer to concentrate on implementing only application-specific functionalities. This reduces implementation time and generates fewer errors, since the underlying framework of the application is tested and robust. The presented framework is written in C++ and it extends the existing Smart Archive framework with support for mobile systems and real-time sensors. The benefits of framework-based applications in the mobile world are presented by building and testing a demonstration program in different computer architectures. In this paper we show that the MSA framework is suitable for building data stream mining applications for the hardware-oriented mobile environment.

1 INTRODUCTION

Ubiquitous data mining and time series data mining, among others, are research areas that are predicted to hold critical and future promise in the field of data mining (Hsu, 2002). While obviously a very potential research area, not much work has been done regarding mobile data mining frameworks, even though the benefits of such frameworks in software engineering have been recognized widely over 20 years ago (Johnson and Foote, 1988).

Frameworks are not very widely used in mobile application development, since they tend to need more system resources than applications that are made and optimized for a single purpose. Compared with desktop and server applications, the mobile environment has been notorious for its limitations and restrictions in memory capacity and processing power. This paper suggests that a well-designed and well-implemented framework can still speed up the development of a new application, even in the field of mobile systems.

This paper presents a component-based application framework, called Mobile Smart Archive (MSA). It is an extended C++ port of the Smart Archive framework, originally written in Java and reported in (Laurinen et al., 2005) and (Tuovinen et al., 2008). The port is designed and modified to serve as a generic data mining framework for mobile and embedded systems such as handheld and laptop computers and PDAs. It has been specialized with support for data stream mining using real-time sensors, but its use is not limited strictly to that field. Although MSA shares its internal design with the original Smart Archive, it has been modified to be better suited for environments with limited resources, such as memory and processing power.

MSA offers a tested and robust data mining framework for use in mobile applications with different sensors and data mining methods. It is not a framework for real-time data mining, although in favorable circumstances it can be use to build applications that perform almost in real time. Instead, it is a framework for data stream mining, and the sensors that produce the streams can produce real-time data. In (Thuraisingham et al., 2005) the major difference is explained: *In the case of real-time data mining, the goal is to mine the data and output results in real time. In the case of stream mining, the goal is to find patterns over specified time intervals.* The MSA framework guarantees that all the values from sensors are recorded in order, without losses, and handled as fast as the underlying hardware is able process them. The most significant advantage compared with data mining methods used before is that the data from the real-time sensors are collected at the same time as they are processed.

Rautio T., Laurinen P. and Röning J. (2009).
COMPONENT-BASED FRAMEWORK FOR MOBILE DATA MINING WITH SUPPORT FOR REAL-TIME SENSORS.
In *Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 208-213
DOI: 10.5220/0001657702080213
Copyright © SciTePress

In this paper we

- explain why there is a need for a generic mobile data mining framework
- discuss why the existing Smart Archive serves as a good basis for a mobile data mining framework
- show how the framework is modified so that it fits into mobile systems and can reliably mine data from real-time sensors

The paper is structured as follows: section 2 reviews previous work related to MSA and describes the motivation for creating a mobile data mining framework. Section 3 discusses why the Smart Archive framework servers a good basis for a mobile data mining framework and goes into the details of the internal design of the framework, especially related to the original framework. Section 4 then demonstrates the framework with a small example program. The conclusions are discussed in section 5.

2 MOTIVATION AND RELATED WORK

The development of MSA originated from a specific need of our research group to apply data mining methods to time series data recorded from human movements. They are usually tracked with several different wearable sensors, which may be, for example, accelerometers, magnetometers, gyroscopes and other similar time series data-producing devices. Usually the sensors and the software used to collect the data have been proprietary and tied closely together, meaning that specific sensors can be used only with certain software within a certain project.

Furthermore, the ways to store the data are numerous: the data can be stored in the flash memory of the sensor, in a data file or in a database. In addition, the storage site can reside in a local computer or in a network. Only the first option allows data to be recorded without environmental limitations. Otherwise the sensors need a connection to the computer that records the data, which means data recording is restricted to the confines of the computer.

This has led to a situation where the data have to be collected using different sensors, processed so that they are uniform and transferred to a place where that are available to the actual data mining software before one can even think of starting the data mining process. The process is, at best, slow. With several different sensors and storage sites, it is also error-prone.

MSA speeds up the development of mobile data mining applications where data stream-producing sensors are used. The only two things the application developer needs to do are code an interface between the framework and the sensor(s) and define how the data are processed in the filters of the application. Everything else is handled by the framework. Even though the framework has been implemented with the data stream sensors in mind, the input to the application can be any kind of device or storage site that produces time series data.

Although data mining in a mobile environment is an emerging field of research, it appears that no research has been done regarding mobile data mining frameworks. However, there are some interesting applications for mobile systems which introduce different application areas for mobile data mining. For example, MobiMine (Kargupta et al., 2002) is client-server-based software for monitoring time-critical financial data from a handheld PDA. (Wang et al., 2003) proposes a distributed and mobile data mining system in which algorithms are encapsulated into SQL Server-stored procedures. An experimental mobile and distributed data stream mining system that allows real-time vehicle health monitoring and driver characterization is presented in (Kargupta et al., 2004).

Of course, existing non-mobile component-based data mining application frameworks such as the original Smart Archive (Laurinen et al., 2005), D2K (NCSA Automated Learning Group, 2003), Knime (Berthold et al., 2006) and YALE (Mierswa et al., 2006) can be modified to receive and mine real-time sensor data, but they do not work very well in mobile systems. First of all, they all are written in Java and therefore need Java Virtual Machine (JVM) to run. Many embedded and mobile systems are not capable of running JVM very well or it may not exist for these systems at all. Secondly, some of the aforementioned frameworks apply a graphical user interface (GUI) to program and visualize the relationships between the data mining application components, pipes and filters. The compatibility of the GUIs with different mobile systems, which can have output screens in various sizes, is questionable. Finally, if needed, MSA allows modifications anywhere in the code, not just in the component API of the framework.

3 DESIGN OF THE FRAMEWORK

The purpose of the MSA framework is to serve as a core for different data stream mining applications in mobile systems. To maximize the portability of the framework, it has been written using standard C++ as much as possible. Since it is very portable, the

main advantage of using MSA is the reduced implementation time of data mining programs. If the application classes are also written to conform to the C++ standard, the generated application can be easily transferred to a different computer architecture or a different operating system. On the other hand, the data mining software can also be written to take full advantage of the platform in which it is running by using platform-specific code. In both cases the developer is not required to implement the underlying data mining engine. On the contrary, the developer is free to start building application-specific functionalities.

As already stated, MSA borrows most of its internal design from the original Smart Archive, whose internal design is explained thoroughly in (Laurinen et al., 2005) and (Tuovinen et al., 2008). Compared with its predecessor, the structure has been modified to better suit the mobile environment. The first differences in design come from the fact that some of the features of the original Smart Archive are language-specific. Thus, some of the code could not be translated directly from Java to C++. However, most of the improvements are related to maximizing the efficiency of the framework in a limited environment.

The challenges of mining data streams in a mobile environment are introduced in (Kargupta et al., 2002). They include:

1. handling a continuous flow of data
2. efficient representation and communication of data mining models over a wireless network with a limited bandwidth
3. visualization on a small screen
4. minimization of power consumption.

In the development of MSA, emphasis has been placed on the first two issues listed above, with the third issue in mind.

Figure 1 shows the overall arrangement of the main classes in the framework. Compared with the original Smart Archive, all of the illustrated classes

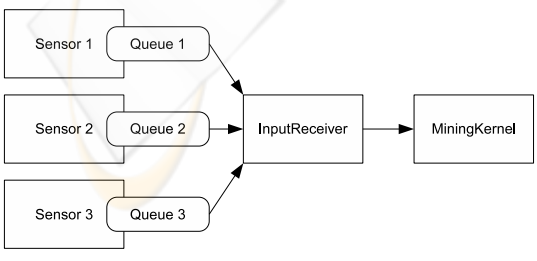

Figure 1: Threaded classes in the framework are illustrated as rectangular boxes. Arrows illustrate data flow within the framework. The boxes with rounded corners represent data buffers of the sensors.

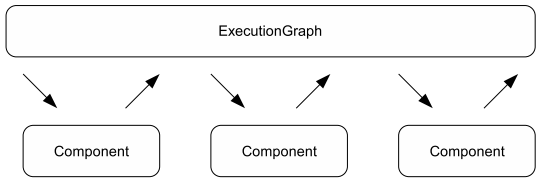

Figure 2: ExecutionGraph class is responsible for transfering data between components in the right order.

run in separate threads. A continuous flow of data is ensured by presenting every real-world sensor as a separately threaded object with a queue of its own for storing sensor data. The `InputReceiver` class collects all the data in a single storage site and the actual data processing class, `MiningKernel`, fetches the data to be processed in the components of the framework.

MSA uses pipes-and-filters architecture, where the input data are processed and transferred from one component to another via distinct pipes. A single component includes a filter and optionally a sink. The former takes care of data transformation and the latter writes the transformed data into a permanent storage site. A data mining graph, `ExecutionGraph`, resides inside the `MiningKernel` class and is responsible for transferring data from one component to another. The data mining graph is illustrated in Figure 2.

Attaching new physical sensors to the framework has been made easy by offering classes that implement serial port communications to applications. As mentioned before, the sensors in the sensor classes do not need to be actual physical devices. However, in the domain of mobile data mining, the use of abstract input devices may mean significant performance loss. A data mining application that uses for example a database table as its input may be more efficient when implemented in a non-mobile domain.

Some compromises were made to ease the use of the framework code and enhance its compatibility with different platforms. Most mobile applications are not portable because their code is often optimized for a certain device architecture. MSA uses standard C++ whenever possible, but in order to maximize the portability of the framework, free peer-reviewed portable Boost libraries (http://www.boost.org) are used in several places in the code. This means applications made with MSA may be not as fast as single-purpose, platform-optimized applications, but the framework significantly eases and speeds up application assembly, as proved in a case study in (Laurinen et al., 2005).

Due to the openness of the MSA framework, further threading of the framework components is possible if the data mining graph in the MiningKernel thread becomes very large. Otherwise, in some cir-

cumstances a situation may arise in which execution of the program is still in the middle of the MiningKernel when the program should be receiving incoming data. Currently the MSA framework only supports standard C++ data types, i.e. one cannot send pointers to objects through the pipes. Since the data from physical sensors are usually numeric, this should not pose a very significant problem.

The most interesting detail from the point of view of the source code is the framework's use of smart pointers provided by the Boost library. In an embedded environment, dynamic memory allocations are traditionally frowned upon by programmers. Since MSA is directed more towards mobile systems with a little larger memory capacity, the use of smart pointers is acceptable. The structure of the data mining graph is built automatically during the execution of the MSA application, which also makes the use of smart pointers essential. It should be noted that the use of raw pointers as members in application classes is nowadays considered very harmful (Smirnov, 2007). This means the application programmer can forget the error-prone `new` and `delete` commands for C++ dynamic memory allocation. The usage of smart pointers will not eliminate the fact that it is still the application programmer's responsibility to be sure the target platform has enough memory to run the application.

The other Boost libraries used extensively in the framework are a threading library and a variant value type library. The former allows portable threads, while the latter allows any value to be stored into the `any` variable.

The framework itself does not place any restrictions on the GUI, so application developers are fully free to create such a graphical interface as they like. Since use of the applications created with MSA is intended to be quite short-term, the power consumption aspect of the framework has been left in the background. Often the external sensors have considerably shorter uptime than does the actual mobile device.

4 DEMONSTRATION PROGRAM

While this paper is concerned with the MSA framework, its use cannot be demonstrated without a demonstration program. Therefore, a very simple application for testing the portability of the Mobile Smart Archive framework was created. The application was tested in three different target systems. The first two were desktop systems (Windows and Linux) and the third was a mobile device with Linux (Nokia N800). All of the aforementioned systems used a wireless sensor box to record real-time data.

The N800 is a rather powerful handheld computer with a 330 MHz ARM11 processor and a 128 MB memory. It uses a Linux-based operating system and is therefore a suitable platform for demonstrating the MSA framework. Shake (SAMH Engineering Services, 2007) is a matchbox-sized sensor box that includes an accelerometer, a magnetometer, a gyroscope, capacitive sensors, a compass and a thermometer. It can be connected to a computer with a Bluetooth connection or a USB cable.

The aim of the application was to prove that applications made with Mobile Smart Archive are transferable to different processor architectures and operating systems. Especially the speed and reliability of the software in the N800 was observed and compared with those of the desktop PCs. After all, the processing speed of the demonstration application is only one aspect of comparison. The MSA framework reduces the actual implementation time of data mining applications in many ways, which are also explored in the program under discussion. In the course of building the demonstration application, the following assumptions were proven to be correct:

- The robust and tested framework engine gives the developer the freedom to focus on application-specific functionalities
- No target machine-specific coding is required; standard C++ is sufficient for building an application
- If the application-specific classes are coded in a portable way (i.e. using standard C++), the same application can be easily transferred from one target system to another without extra coding

The demonstration program worked as follows: a Shake sensor box was used to collect three-axis accelerometer data at a user-defined frequency. The collected data were relayed to the software via a Bluetooth connection. The application received the data, attached time stamps and ordering numbers to each data packet and forwarded them to a specified sink, which in the demonstration program was either a local text file or a table in a MySQL database.

Execution times were recorded with a built-in timer that used a clock class from the Boost library. The stored times include not only the actual time required to acquire the data with the sensor, they also include initialization of the components, filters and sinks and assembly of the data mining graph.

The test cases consisted of processing 100, 1000 and 10000 accelerometer data packets gathered using a 100 Hz frequency. When the local text file in the desktop PCs was used as a storage site for processed

data, data flow through the application was close to real time in all the tests, when the initialization and set-up time of the application was ignored. With the limited processing power of the N800, data mining was roughly 2-3 times slower as the Table 1 indicates.

Table 1: Execution times of the demonstration program when accelerometer samples were recorded into a flat file.

	Amount of data packets		
Platform	**100**	**1000**	**10000**
Linux PC	3,26 s	11,49 s	98,96 s
Windows PC	2,69 s	10,67 s	98,76 s
N800	4,21 s	29,59 s	278,19 s

In practice this means that when the sensor has stopped gathering raw data, the application spends the rest of its running time processing the queued raw data. Since the applications made with the framework can be hugely different, there are no implicit frequency rates for the sensors in different platforms where data can be collected in real time.

However, an estimation of the memory consumption of the application can be made when the structure of single datapacket is observed. In the demostration application one datapacket consisted of four variables from the accelerometer of the Shake sensor box. When the application was compiled with GNU G++-compiler and used in the N800, the size of the resulting package was 32 bytes as seen in Figure 3. This means that with 100 Hz frequency 3200 bytes of memory was used every second.

When the MySQL database acted as a sink, the processing speeds of the applications were more similar, although the applications run on the desktop PCs were generally a bit quicker. The execution times were not evaluated, since the MySQL sink implementation was unoptimized, which could be observed as slowness in all platforms. It was only noted that different sink implementations worked and the MySQL connection from the framework needed more optimization.

In the end the final result was that both test cases were run successfully: no data packets were dropped during the process and the execution times did not increase to disproportionate dimensions. During the tests the processor load of the programs in the N800 with one Shake sensor was constantly around 30%. With two sensors the processor load was 70-95%, implying that with a little optimization the N800 could serve as a functional platform for actual MSA-based applications in the future.

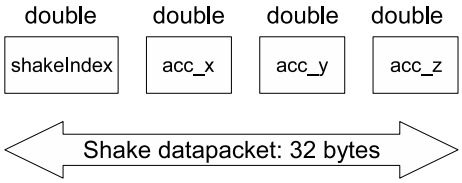

Figure 3: An example datapacket.

5 CONCLUSIONS

This paper is the first step in exploring the area of mobile data mining frameworks. It presented Mobile Smart Archive, a data mining application framework for mobile systems. It is designed for building data mining applications for mobile platforms. Especially data stream mining using one or more physical sensors is supported. The framework is based on the existing Smart Archive framework, but it is implemented using C++ programming language and it has been adapted and modified to utilize mobile devices better.

With MSA, writing mobile data mining software for data stream mining is significantly easier. There is an underlying framework for looking after generic data mining functionalities, thus the application developer can concentrate on implementing application-specific classes and functions. This reduces the implementation time of a new application and generates fewer errors, since the framework is thoroughly tested and robust.

A demonstration application was created for different platforms to test the portability of the framework. The execution speeds of the programs were tested and it was observed that the data were collected successfully. In this paper we demonstrated that, compared with building a mobile data mining application from scratch, the MSA framework allows an application to be built faster and in a less error-prone way.

REFERENCES

Berthold, M. R., Cebron, N., Dill, F., Fatta, G. D., Gabriel, T. R., Georg, F., Meinl, T., Ohl, P., Sieb, C., and Wiswedel, B. (2006). Knime: The Konstanz Information Miner. In *Proceedings of the 4th Annual Industrial Simulation Conference*, pages 58–61.

Hsu, J. (2002). Data Mining Trends and Developments: The Key Data Mining Technologies and Applications for the 21st Century. In *The Proceedings of 19th Annual Information Systems Education Conference (ISECON 2002)*.

Johnson, R. E. and Foote, B. (1988). Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1(2):22–35.

Kargupta, H., Bhargava, R., Kun, L., Powers, M., Blair, P., Bushra, S., and Dull, J. (2004). VEDAS: A Mobile and Distributed Data Stream Mining System for Real-Time Vehicle Monitoring. In *Proceedings of the fourth SIAM international conference on data mining*, pages 300–311.

Kargupta, H., Park, B.-H., Pittie, S., Liu, L., Kushraj, D., and Sarkar, K. (2002). MobiMine: monitoring the stock market from a PDA. *ACM SIGKDD Explorations Newsletter*, 3(2):37–46.

Laurinen, P., Tuovinen, L., and Röning, J. (2005). Smart Archive: A Component-based Data Mining Application Framework. In *Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, pages 20–26, Wroclaw, Poland. IEEE Computer Society Press.

Mierswa, I., Wurst, M., Klingenberg, R., Scholz, M., and Euler, T. (2006). YALE: Rapid Prototyping for Complex Data Mining Tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–940.

NCSA Automated Learning Group (2003). *D2K Toolkit User Manual*.

SAMH Engineering Services (2007). *SHAKE Sensing Hardware Accessory for Kinaestethic Expression Model SK6*. Dublin, Ireland, revision f edition.

Smirnov, I. B. (2007). Raw Pointers in Application Classes of C++ Considered Harmful. *ACM SIGPLAN Notices*, 42(4):23–31.

Thuraisingham, B., Khan, L., Clifton, C., Maurer, J., and Ceruti, M. (2005). Dependable Real-time Data Mining. In *Proceedings of the Eight International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 158–165.

Tuovinen, L., Laurinen, P., Juutilainen, I., and Röning, J. (2008). Data Mining Applications for Diverse Industrial Application Domains with Smart Archive. In *Proceedings of the IASTED International Conference on Software Engineering*, pages 56–61.

Wang, F., Helian, N., Guo, Y., and Jin, H. (2003). A Distributed and Mobile Data Mining System. In *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 916– 918.