

DATA MINING APPLICATIONS FOR DIVERSE INDUSTRIAL APPLICATION DOMAINS WITH SMART ARCHIVE

Lauri Tuovinen, Perttu Laurinen, Ilmari Juutilainen and Juha Rönig
University of Oulu, Department of Electrical and Information Engineering
P.O. Box 4500, FI-90014 University of Oulu
Oulu, Finland

email: lauri.tuovinen, perttu.laurinen, ilmari.juutilainen, juha.roning@ee.oulu.fi

ABSTRACT

The increasing significance of data mining in many application domains of computational technology has resulted in a considerable body of work concerned with designing architectural models and collections of reusable software components to allow for more rapid deployment of data mining methods wherever they are deemed useful. Early work involved integrating machine learning algorithms and knowledge management features into class libraries; more recently the data mining research community has progressed towards application frameworks, which are based on the notion of a reusable high-level design rather than specific algorithms. Smart Archive provides such a design, fine-tuned for applications that process continuous measurements and make use of historical data. In this paper a pre-existing foundational framework is extended with a new layer of services and the overall system architecture established by the framework is examined. Two case studies drawn from diverse application domains—steelmaking and personal fitness products—are presented in order to validate the proposed design. The case studies show that Smart Archive is beneficial in integration, coding and testing tasks and suitable for both online and batch processing and for both localized and distributed applications.

KEY WORDS

data mining, software architecture, application framework, Java programming

1 Introduction

The advantages of high-quality application frameworks are widely recognized in the field of software engineering [1]. While obviously not a silver bullet, and although there are many development situations in which using a framework is not appropriate, the significance of application frameworks in general is not really contestable. A well designed and well implemented framework that is well suited to the task at hand can considerably boost the development effort by allowing the software engineers to concentrate on the details of their particular objective rather than dwelling on the basic mechanics common to a whole class of applications.

One class of software that has lacked dedicated frame-

work support is that of data mining applications. Naturally, general-purpose frameworks can be used to support the development of such applications, but there has not been much work on frameworks targeting the field of data mining applications specifically. Between library-based bottom-up solutions such as Weka [2] and XELOPES [3] on the one hand and packages of generic mining tools such as SPSS and ODM on the other, there is no established layer of object-oriented top-down approaches to building tailored data mining software. This is a significant deficiency: algorithm libraries, while potentially very useful, do not reduce the effort required by the creation of the execution logic of an application, whereas ready-to-run packages are too rigid for many purposes, as they bind the user to a predefined set of ways to manipulate and visualize data and models.

This paper is concerned with Smart Archive (SA), a component-based framework implemented in Java. It has been created as a response to the lack of frameworks designed specifically for programming data mining applications. It extends the work reported in [4], which introduced the key requirements and architectural design decisions of Smart Archive but made little headway into finer details. In this paper we establish a more detailed view on SA by taking a closer look at a few key points of class-level design and by introducing certain architectural concepts that did not appear in the introductory paper.

As described in [4], Smart Archive applications are made up of independent components coupled together with pipes. Each component, representing a step in the data mining process, encapsulates a filter, which transforms data passed to the component, and possibly a sink, which persistently stores data transformed by the filter. The network of components and pipes forms a refining chain (or more generally, a directed acyclic graph) that accepts raw input data at one end and produces interpretable results at the other. A central portion of the chain is devoted to similarity-based selective storage and intelligent retrieval of history data—the archive itself.

The previously reported version of SA explicitly implemented the static arrangement of filters, sinks and pipes. The runtime dynamics of the framework were, however, left largely implicit and therefore up to application code. In this paper we

- show how a new layer of execution logic and various services for data mining application programming can be built on top of the framework of components and pipes,
- present a parallel distributed architecture for a complete data mining application in which the processing chain communicates with three other main system components: a database, an input receiver and a user interface backend.

Section 2 reviews previous work related to Smart Archive and shows how a framework with the characteristics of SA contributes to the palette of available infrastructural solutions for data mining. Section 3 presents our extension of the Smart Archive concept discussed in [4]. Section 4 recounts how the extended design was used to implement two data mining applications for evaluation purposes. Section 5 discusses evaluation results and projected future developments. Section 6 concludes the paper.

2 Motivation and Related Work

Given the already formidable and constantly increasing significance of data mining in many branches of science, engineering, business and industry, it is not surprising that several software architectures and libraries designed to support data mining have been developed. This section reviews these existing concepts and products along with the primary design objectives of Smart Archive, showing how SA is distinguished from its peers and occupies a niche of its own among them.

The leading motivation and design principle of SA is the process-oriented view of data mining illustrated in Figure 1. This view emphasizes the process of mining a collection of data on the one hand and the process of developing a data mining application on the other. The long-term goal of SA is to cover both these dimensions of data mining equally. In the current version the data mining process is more fully realized, but new developments that will address the software development process are already underway.

In the existing literature there are examples of data mining application frameworks designed for more constrained domains such as manufacturing systems [5] and text mining [6]. More generic views such as [7] tend to be more abstract as well, offering concepts rather than code. The SA approach is to combine concreteness with genericness in a software framework that makes few assumptions *a priori* regarding application domain.

The algorithm-centered approach to creating reusable building blocks for data mining applications provides many examples of mature developments. Early efforts in the field of data mining libraries include MLC++ [8] and MKS [9], the former of which at least is still publicly available and supported. Among more recent entrants Weka [2] and XELOPES [3] are prominent.

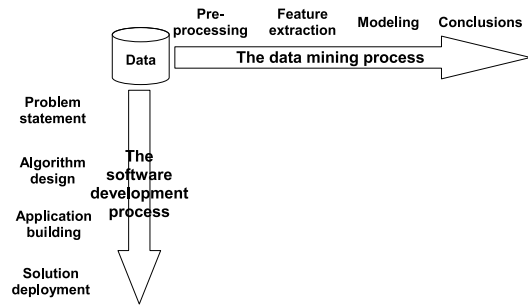


Figure 1. A simplified linear view of the processes involved in data mining, depicted as orthogonal dimensions. A full coverage on both axes is sought in the development of Smart Archive.

A module-centered approach has been adopted in D2K [10], Knime [11] and YALE [12], all of which provide a graphical user interface for manipulating data mining components. They share with Smart Archive an architectural model based on pipelining and reusable components, but SA trades off the convenience of a graphical editor for the greater flexibility and versatility of hands-on programming. Here again, SA takes the middle road, allowing the construction of fully tailored (in terms of user interfaces, for example) standalone applications while still providing considerable support in terms of application logic. Obviously, data mining libraries can be used in conjunction with SA, and a graphical application builder for SA could be implemented. In fact, the master view in Figure 2 (explained in Section 3) is a provision for just such a builder.

3 Structure and Services

Figure 2 shows the overall arrangement of top-level components in an SA-based information system. The core of the system is the component labelled 'Data processing', which is where the data passes through the network of filters and pipes and is transformed from raw input into interpretable results on the way. This component is fed by 'Input reception', which interfaces with the actual data sources and hands over their yield to the data processing component as it becomes available. At any point in the processing chain the transformed data can be written into the database, and at any point in the processing chain previously stored data can be retrieved from the database.

The 'UI backend' component is a server that acts as an interface between user interface views and the data processing component. One of the views is the master view that controls the framework itself, while other views control and visualize individual applications. There is no theoretical upper limit to the number of views the backend can serve. Currently the backend provides only certain rudimentary functions for controlling application threads, implemented in the master view as textual commands. Future developments will enhance the UI portion of the framework

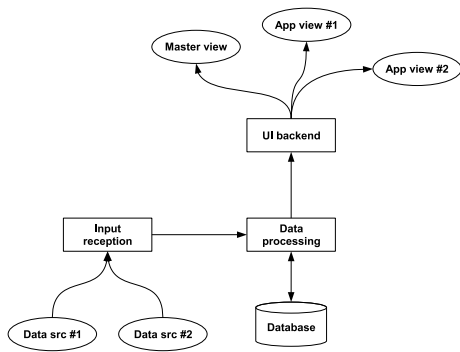


Figure 2. A top-level view of the system architecture of Smart Archive. The three main parts of the SA framework are drawn as boxes.

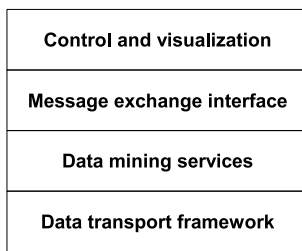


Figure 3. A layered model of the composition of Smart Archive. Application logic is coded at the two bottommost layers, whereas user interfaces are created at the two topmost layers.

with more advanced features.

Between them the input reception, data processing and UI backend components comprise the bulk of application logic in a Smart Archive application. The three top-level components are executed as parallel threads, independent of one another except when they synchronize to exchange data or instructions. The database may reside on a remote machine, and so may the user interface views. The data sources may be anything from sensors producing the data in real time to another database of previously gathered observations set aside for later use.

Another way to view the Smart Archive architecture is to model it as a set of layers as in Figure 3. The layers are progressively closer to the user of SA from bottom to top so that the top layer represents those parts of the overall system that are immediately visible to the user. In terms of Figure 2 the two upper layers form the user interface portion and the two lower layers form the data reading and processing portion. Application-specific code conforms to the API defined by the data transport framework layer and employs functions provided by the data mining services layer. In [4] the emphasis was on the data transport framework layer; in this paper we concentrate primarily on the data mining services layer.

The backbone of the subsystem concerned with ac-

cepting and processing input, as shown in Figure 4, consists of three classes: `InputReceiver`, which interfaces with data sources, `ExecutionGraph`, which encapsulates the network of pipes, filters and sinks, and `MiningKernel`, which mediates between the two. `InputReceiver`, in its main loop, retrieves input from data sources (1.) and hands it over to `MiningKernel` (2.), where it is placed into queues, one queue per input channel. Meanwhile, the main loop of `MiningKernel` takes data out of the input queues (3.), hands it over to `ExecutionGraph` (4.) and instructs `ExecutionGraph` to process it (5.). Thread boundary lies between `InputReceiver` and `MiningKernel`, with synchronization taking place by means of a concurrent access manager object residing within `MiningKernel`.

Although `ExecutionGraph` assumes principal responsibility for application execution once it has received the input data, the role of `MiningKernel` does not end there. The basic unit of data in Smart Archive is a data item, simply a set of N named values where N is the number of data variables. `MiningKernel` extends each data item by adding a set of control variables alongside the data variables. The control variables include timestamps as well as tag fields that help manage the data as a series of sequential groups of data items. This reflects the orientation of SA toward the mining of time series data, in which the data representing a single real-world entity is a sequence of observations rather than an individual observation. An individual observation, on the other hand, can be treated as a time series consisting of just one observation.

`MiningKernel` also offers a variety of services that are frequently useful in building application components. Most importantly, many components need to communicate with a relational database via a data sink, and `MiningKernel` supports this in two ways. First, it serves as a database connection manager, handing out connections at request and closing them when they are no longer needed. Second, it can automatically generate database table definitions from variable specifications. This ability reduces significantly the amount of manual coding required to set up a data sink that uses a database for persistent storage.

The construction of a Smart Archive application always follows the same general pattern. First, the filters and data sinks to be used are created and packaged as components. Next, these components and the pipes connecting them are registered with `ExecutionGraph`. One of the components is designated as the starter, which is the first component to be processed on each iteration of the `MiningKernel` main loop. Inputs to the starter component are specified in `MiningKernel` and an `InputReceiver` instance capable of supplying those inputs is created and set up. Carrying out this standard process can result in diverse types of applications, as illustrated in the next section.

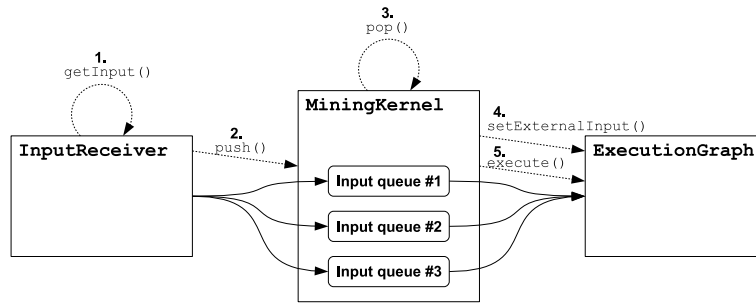


Figure 4. The main processing cycle of Smart Archive. Data flows are represented by solid arrows and control flows by dotted arrows. Control flows {1, 2} and {3, 4, 5} form two subcycles that are executed in parallel threads.

4 Two Application Cases

The first application constructed using the current version of Smart Archive is a system that accepts measurements from a steel production line as input and uses them to predict the yield strength, tensile strength and elongation of steel plates. It can also be used to train new models to replace the old ones when they no longer perform up to standards. The application is a subsystem of a work-in-progress system for supporting deployment and maintenance of predictive models at a steel mill.

The application consists of seven logical components, shown in Figure 5. The components are arranged into two processing sequences, corresponding to the two main functions of the application: prediction and training. In practice many of the logical components do not map directly into concrete SA components; the training tasks, for example, involve iterating two components in a closed cycle until a convergence condition is satisfied. The depiction of data flows is similarly simplified: intermediate results are actually written into the database at several points so that the sequence may be resumed at a later time instead of starting over from source data.

A thing worth noting here is that the SA application model as realized in `ExecutionGraph` does not actually allow deviations from the directed acyclic graph (DAG) model such as those used in model training. However, such deviations can be implemented by switching between different execution graphs, using a sink and an `InputReceiver` as the interface that mediates the switch. This allows the restrictions imposed by the DAG model to be circumvented without binding filter classes to contingencies of application structure, which would disturb their cohesion and therefore hurt their reusability.

It may be desirable to have this type of application do its processing either online, with every item of input data being processed as soon as it has been generated, or in batches, with new data being retrieved at regular intervals or upon user command. Both processing modes can be easily implemented in Smart Archive: the `InputReceiver` thread can be allowed to run indefinitely for online processing, or it can be made to terminate instantly once a

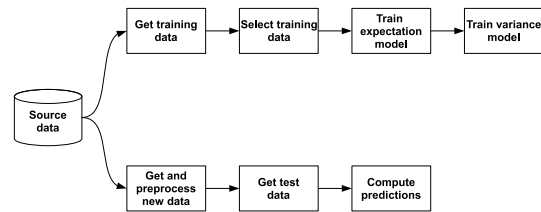


Figure 5. A Smart Archive application for predicting mechanical properties of steel. The boxes represent logical components, many of which break down into two or more concrete components at the implementation level.

batch of input has been delivered to the application. The `MiningKernel` thread can be suspended when there is no use for it and revived when there is new input data available for processing.

The second application is a multi-process distributed system for processing signals recorded from human subjects during physical exercise. Each process is an application instance: one broker, running on a desktop workstation, and several functionally identical workers, running on high-performance computing servers. The processes are largely independent of each other, communicating only indirectly through a shared data repository. Figure 6 illustrates this arrangement.

The input to the broker application is a time series, which the application breaks down into manageable chunks. These are then placed into the shared repository. The worker applications, when idle, poll the repository at steady intervals for new entries. If a worker finds one, it will tag it as 'being taken care of' so that other workers will not accept the job. The worker will then retrieve the chunk of data from the repository and process it further. Finally, the worker will yield the results of its computation, tag the processed entry in the shared repository as 'all done' and then proceed to scan the repository for another chunk of input.

A distributed system was deemed appropriate for this task because it was possible to break down the work to be done into independent units that can be executed in par-

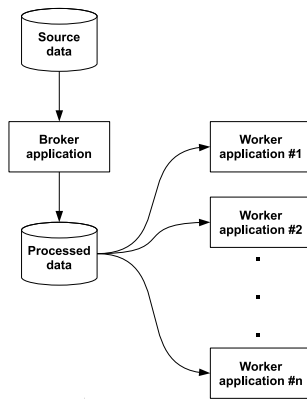


Figure 6. A Smart Archive application for analyzing measurements of physical activity. This is a higher-level view, with boxes representing distinct SA instances.

allel, and because of the heaviness of the computation involved in processing each work unit. The manual distribution strategy, in which the system operator individually sets up each worker process, served this purpose well, allowing the system to make good use of available computing resources without the need to implement a special distribution algorithm in the broker process. The inherent ability of Smart Archive applications to interface with data sources enabled sufficient coordination between processes, eliminating the need to define a message-based communication protocol for the purpose.

5 Evaluation and Future Work

First of all it is worth noting that each of the case-study applications is a useful and important deliverable in a research and development project, not a contrived example of no practical consequence. Statistical regression modelling in steel production is a technique of considerable potential, as [13] clearly demonstrates. Methods for mining physical activity data, in turn, are necessary for the creation of advanced fitness products whose capabilities extend beyond simple monitoring and tracking functions. Smart Archive allowed both applications to be completed within the performance, reliability and development time constraints imposed by the parent project, regardless of the differences in application domain and overall system architecture.

Using the Smart Archive framework clearly shifted the workload balance of writing and testing code away from high-level application structure and toward low-level computing algorithms. There is obviously an overhead from having to write the algorithms to conform to the interfaces of SA, but this is compensated for by the fact that once the algorithms are implemented, assembling them into a computation graph is rapid. Thus, resource savings are gained in application integration. Even faster integration could have been achieved with one of the more mature graphical frameworks, but this would have restricted the

ability of the developers to build an application with a custom user interface and many complementary functions besides training and applying models, which were important requirements of the steelmaking application.

Building the steelmaking application demonstrated that algorithm coding also can be made more rapid with SA, as the framework encourages application elements to be designed for reuse. The predictive models, for instance, were implemented by creating an abstract filter class for evaluating regression functions and then subclassing it with little extra effort to fill in some model-specific details. The partitioning of functionality into independent components and the automatic storage of results into a database via data sinks brought the additional benefit that many defects could be conveniently identified and traced back to their source by inspecting the contents of the result tables.

The application for processing physical exercise signals, unlike the steel application, was developed entirely by someone largely unfamiliar with the internal details of Smart Archive and therefore requiring some training in its use. Nevertheless, also in this case the application-independent execution logic with support for arbitrarily complex processing cycles was found useful and time saving. The functional partitioning proved especially good for isolating update points when, after initial test runs, it became necessary to modify filter code and to introduce alternative processing paths.

Predictably, there were certain problems due to the framework being a work-in-progress rather than a production-quality system. Most importantly, the level of attention to detail required in coupling components was the source of several programming errors that proved somewhat difficult to trace. This, however, reflects immaturity of implementation rather than a design-level flaw. The overall success of the case studies indicates that the framework is ready for another round of development and testing. The inconveniences identified in the case studies will be eliminated or alleviated in the process.

One more advantage of SA worth mentioning is that the hiding of implementation details of filters allows them to assume roles other than the standard one of wrapping an algorithm written in Java. For instance, one can create stub filters that simulate the functioning of actual algorithms, allowing rapid construction of a mock-up model of the final application. Such a model can be used to reason about runtime issues such as performance at an early stage in application development. Another possibility is to use a filter as a gateway to an algorithm implemented using a technology other than SA; creating such gateways is the most imminent future extension to SA, as they are a key element in supporting the algorithm design phase of the software development process shown in Figure 1.

6 Conclusion

Smart Archive is a data mining application framework based on the pipes-and-filters architectural style. Smart

Archive applications are composed of independent components connected by pipes. Each component contains a filter for transforming data and optionally a data sink for storing results. Data is refined in a stepwise manner as it passes through the network of filters, each of which performs a data mining task such as preprocessing or classification.

The work reported in this paper built on an introductory paper in which the static structure of a Smart Archive application was presented. A new layer of functionality, referred to as data mining services, was implemented on top of the original pipes-and-filters architecture or data transport framework. The purpose of the new layer is to support Smart Archive application development by assuming control of application execution and by providing services that are useful in component programming.

The overall system architecture of a Smart Archive application has a star topology, with the data processing components at the center. These communicate with an input reception component, an application database and a user interface backend. Most of the application logic involved in executing an SA-based system is contained in three classes, `ExecutionGraph`, `MiningKernel` and `InputReceiver`. The features and interactions of these classes were discussed in some detail. Two realistic example applications were then described, illustrating the ability of Smart Archive to support such activities as reusable component programming and distributed computing.

Based on experience from the development of the example applications, the suitability of Smart Archive for implementing data mining solutions for diverse application domains was evaluated. The results encourage further development and trials, as the framework was found to support application integration, coding and testing in helpful ways. Immediate concerns for future modifications include facilities for using algorithms implemented on platforms other than Java.

Acknowledgements

The research reported in this paper was funded by the Finnish Funding Agency for Technology and Innovation (Tekes), Rautaruukki Corporation and Polar Electro. Rautaruukki and Polar also provided data and domain expertise that were indispensable for the development of the case applications described in Section 4. L. Tuovinen wishes to thank the Graduate School in Electronics, Telecommunications and Automation (GETA) for the personal funding he has received for his dissertation work.

References

- [1] M.E. Fayad & D.C. Schmidt, Object-oriented application frameworks, *Communications of the ACM*, 40(10), 1997, 32-38.
- [2] I.H. Witten & E. Frank, *Data mining: practical machine learning tools and techniques* (2nd ed., San Francisco, CA: Morgan Kaufmann, 2005).
- [3] *Xeli's intro. Introduction to XELOPES* (technical manual, Prudsys AG, 2007).
- [4] P. Laurinen, L. Tuovinen & J. Röning, Smart Archive: a component-based data mining application framework, *Proc. Fifth International Conf. on Intelligent Systems Design and Applications*, Wroclaw, Poland, 2005, 20-25.
- [5] H. Ma, J. Zhang, Y. Zhao, C. Tan & R. Qu, Knowledge discovery application framework for manufacturing execution, diagnosis and optimization, *Proc. First IEEE Conf. on Industrial Informatics*, Banff, Canada, 2003, 141-146.
- [6] L.E. Holzman, T.A. Fisher, L.M. Galitsky, A. Kostostathis & W.M. Pottenger, A software infrastructure for research in textual data mining, *Proc. 15th IEEE International Conf. on Tools with Artificial Intelligence*, Sacramento, CA, USA, 2003, 112-121.
- [7] I. Geist, A framework for data mining and KDD, *Proc. 2002 ACM Symposium on Applied Computing*, Madrid, Spain, 2002, 508-513.
- [8] R. Kohavi, D. Sommerfield & J. Dougherty, Data mining using MLC++: a machine learning library in C++, *Proc. Eighth IEEE International Conf. on Tools with Artificial Intelligence*, Toulouse, France, 1996, 234-245.
- [9] S.S. Anand, B.W. Scotney, M.G. Tan, S.I. McClean, D.A. Bell, J.G. Hughes & I.C. Magill, Designing a kernel for data mining, *IEEE Expert*, 12(2), 1997, 65-74.
- [10] *D2K toolkit user manual* (technical manual, Automated Learning Group, NCSA, 2003).
- [11] M.R. Berthold, N. Cebron, F. Dill, G. di Fatta, T.R. Gabriel, F. Georg, T. Meinl, P. Ohl, C. Sieb & B. Wiswedel, Knime: the Konstanz information miner, *Proc. 4th Annual Industrial Simulation Conf., Workshop on Multi-agent Systems and Simulation*, Palermo, Italy, 2006, 58-61.
- [12] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz & T. Euler, YALE: rapid prototyping for complex data mining tasks, *Proc. 12th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, Philadelphia, PA, USA, 2006, 935-940.
- [13] I. Juutilainen, J. Röning & L. Myllykoski, Modelling the strength of steel plates using regression analysis and neural networks, *Proc. International Conf. on Computational Intelligence for Modelling, Control and Automation*, Vienna, Austria, 2003, 681-691.