Abstract

Implementation of data mining applications is a challenging and complicated task, and the applications are often built from scratch. In this paper, a component-based application framework, called Smart Archive (SA) designed for implementing data mining applications, is presented. SA provides functionality common to most data mining applications and components for utilizing history information. Using SA, it is possible to build high-quality applications with shorter development times by configuring the framework to process application-specific data. The architecture, the components, the implementation and the design principles of the framework are presented. The advantages of a framework-based implementation are demonstrated by presenting a case study which compares the framework approach to implementing a real-world application with the option of building an equivalent application from scratch. In conclusion, the paper presents a lucid framework for creating data mining applications and illustrates the importance and advantages of using the presented approach.

1. Introduction

A data mining (DM) application without an underlying framework is like a computer program without an operating system. Without a proper framework, a considerable portion of the development time of a DM application will be spent implementing functionality common to all applications in the DM domain. However, no redundant work would be necessary if a sufficiently generic framework that could be easily tailored to meet application-specific needs were available. The need to develop a component-based data mining framework is emphasized in [1]. The authors note that decision support systems have specific needs that cannot be properly addressed by conventional information systems. In this paper an application framework, called Smart Archive (SA), for implementing data mining applications utilizing continuously measured data streams is presented.

SA is a domain-specific but application-independent framework. This means that the framework supports operations required by most DM applications, but is not tied to any particular application. To create an application using SA, the designer needs to implement application-specific algorithms using the interfaces offered by the framework and configuring the framework to use the measurement data the application is designed to process. The framework takes care of transferring and storing data between the application components and implements some of the basic functionality required of application-specific filters. The full potential of SA can be realized in applications generating multivariate time series data simultaneously from multiple entities.

The benefits of using a framework for application generation are numerous, because a typical DM application has to handle a large number of variables and transformations. For one thing, the time spent implementing functionality common to most or all similar applications can be significantly decreased to the benefit of increased resources for application-specific development. The quality of the application is likely to be better, since the code of the framework has already been tested, and the application-specific code is called through well-defined interfaces. More advantages of using a framework are described in the case study presented in Section 3.

The architecture of SA is an instance of domain-specific software architecture (DSSA) [2]. The principle of DSSA development is to create a software architecture based on an analysis of the functional requirements of the target application domain. Based on the requirements analysis, components common to potential applications within the domain are identified. After that, a software architecture abstract enough for modelling the interoperability of the components is formulated. This version of the architecture is called the reference architecture.

The general principle of data mining is the measurement–pre-processing–feature extraction–modelling cycle. The basic elements of a generic reference architecture for data mining applications are presented in Figure 1; let us call it MPFM architecture based on the initials. The arrows in the Figure represent
data streams, and the boxes represent components transforming and storing data.

The field of data mining architectures is still rather unexplored, since only one earlier study on an architecture involving the MPFM principle was found in the literature. However, architectures that can be adapted to implement the MPFM do exist, as do architectures and frameworks in other fields overlapping some aspects of data mining architectures. A data mining architecture supporting the MPFM principle is reported in [3]. The work approaches the problem from a data warehouse (DW) perspective and does not describe a framework for implementing DM applications. Other DW-centric studies are presented in [4], [5]. Architectures for processing data streams or data feeds have been developed in [6], [7], [8]. Finally, the work most similar to this one, although in a different domain, is presented in [9]. The study presents a framework called VERTAF for the development of embedded real-time systems. The framework shares some of the motivation of this work, offering components and interfaces for implementing embedded systems.

This paper is organized as follows. The functional requirements analysis is presented in Subsection 2.1, the components of the architecture in Subsection 2.2 and the architecture itself in Subsection 2.3. Section 3 illustrates the advantages of using SA through a case study, in which an implementation of a data mining application for a steel mill was carried out from scratch and using SA. Conclusions are presented in Section 4.

2. Proposed framework

2.1 Functional requirements analysis

As stated in the introduction, the development of the architecture is based on an analysis of functional requirements. The analysis was performed on the domain of data mining applications implemented for processing continuously observed measurements. The list of requirements is based on two general guidelines recommended for component-based data mining frameworks: transparency and usability [1]. The five most important requirements that were identified are:

a) The architecture should implement, and preferably extend, the reference architecture.
b) The architecture should be able to utilize history information.
c) The architecture should be customizable to suit application-specific needs. The components used for tailoring it to application-specific needs should be separate from the core architecture.
d) The architecture should be suitable for processing continuously observed time series data from multiple entities.
e) The architecture should be transparent and easily understandable to practitioners of data mining.

2.2 Components of Smart Archive

Before the presentation of the overall SA architecture, the individual components that make up the architecture are discussed. The components are divided into three categories:

a) components common to software architectures in general
b) components specific to data mining architectures

\(\) c) components specific to SA.

Data processing units common to most software architectures are ones that store, transform and transfer data. Storage components are also referred to as storage units, data sinks or data warehouses. The terms filters, transformations and operators are used about transformation components, and components transferring data are referred to as pipes and data streams. A component that is present in fewer architectures, but in most frameworks, is an interface component that makes other components less application-dependent. For the sake of coherence, in the remaining part of the paper, transformations will be referred to as filters, components transferring data as pipes and components storing data as sinks.

Components specific to data mining architectures consist of those present in the reference architecture. The responsibility of the data pre-processor is to perform elementary operations, such as data cleansing and integration, to ensure that only high quality data is fed into the subsequent components. The feature extractor filters the data to extract information that is not directly measurable but may improve the performance of the model. Examples of commonly
used features are averaged time series, principal components and Fourier coefficients of signals. The model is a component fitted on the feature data to extract the desired knowledge. The model is often used to anticipate the future behaviour of a phenomenon (predictive models); examples of modelling methods include statistical classifiers, neural networks and regression models.

The components specific to SA store and utilize information of the history of the phenomena being studied. The incremental history component archives completed measurement series in a non-redundant way. In this context, non-redundant data storage means storing data in such a manner that no two observations (or series of observations of an entity) resembling each other too closely are stored in the database. In practice, when observing quantities that are measured using real number precision, it is very unlikely that any two multidimensional observations turn out to be exactly the same, but the level of similarity can be calculated. The selective data storage component provides filters for determining the similarity between measurements already archived in the incremental history component and completed measurements considered as candidates for archiving. If the candidates are detected to resemble existing history observations too closely, they will not be archived. The algorithm for determining similarity can be, for instance, the k-nearest neighbours (kNN) algorithm, which is currently used in SA [10]. The component returning similar data compares on-going measurement data with archived data and pipes the most similar data found in the incremental history back to the component calling it.

The basic components presented so far are organized into larger units, which are used to build the SA architecture. The components (units) of SA are implemented according to the pattern shown in Figure 2. Each component consists of input and output pipes, an interface, a generic filter, an application-specific filter and a data sink. The data input into the component takes place via input pipes, after which the data is fed into the filter specific to the component through an interface common to all components. The interface gives the applier the freedom to customize the application-specific filter for application-specific purposes. From the filter the data is piped to the data sink and from the sink to the other components of the architecture. When standard data storage technology is used for implementation, the sink allows the applier to access the data in a manner independent of SA. Implementing the sink by using, say, a table in an SQL-compatible database allows direct access to the data through an ODBC / JDBC interface.

Finally, an example of adapting the generic component for the implementation of an application-specific component is presented in Figure 3. The example shows the implementation of the selective data storage component, and the other components can be implemented in a similar manner. The data is input into the selective data storage filter using pipes originating from the feature and history components. The filter provides the generic functionality of the selective data storage algorithm, such as fetching the data being currently processed from the feature component and archived data from the incremental history component. In case the similarity measure implemented in the application-specific filter finds that the incremental history does not contain data resembling the feature data being processed, the data is piped to the sink. Finally, the output pipe transfers the data to the incremental history sink.

2.3 Architecture and operation of Smart Archive

A software architecture extending the reference architecture was developed using the components...
presented in Subsection 2.2. The component architecture is based loosely on the architectural pattern of pipes and filters [11], which is suitable for organizing the co-operation between separate, highly data flow dependent components. The pattern divides data processing tasks into a number of sequential processing steps using filters to transform data and pipes to transfer the data between steps. The data processed by SA typically originates from entities producing sequences of observations. There are therefore three kinds of data in the system that are transferred through the pipes: on-going measurements, completed measurements and archived measurements.

The architectural layout of Smart Archive is shown in Figure 4. The architecture is divided into live and history sections. Units in the live section process data from on-going entities, that is, entities that can be expected to produce more measurement data. The organization of the data flow in the live section is compatible with the MPFM architecture, as explained in the introduction. The history section processes completed and archived measurement series. When the measurements of an entity are complete, the set of completed measurements are transferred from the feature component to the selective data storage component. In case the incremental history component does not contain too similar measurements, the completed measurements are archived. Archived data can be retrieved to be utilized in models using the component returning similar data.

The components of SA are updated in a sequential order. The data flow and the order in which the data is transferred during an update cycle is explained using the numbers and shapes above the pipes in Figure 4. The pipes transferring data from on-going measurements are marked with circles, the pipes transferring completed measurements with boxes and the pipes transferring archived measurements with diamonds.

The processing starts by checking from the measurement component if there are new entities or new data available from the on-going entities in the system. If so, the new data is transferred to the pre-processing unit (pipe no. 1), on to the feature extractor (2) and the model (3). In order to be able to utilize archived data, the model also needs data from the history sink. Therefore, the feature data is piped to the component returning similar history data (4) for the purpose of determining the subset of archived data most similar to the on-going data. Archived data is retrieved (5) and returned to the model (6 & 7). After that, the model gives its analysis based on on-going measurements and knowledge from archived measurements. The results of the model are then piped to the results unit (8).

The sequence so far was all about analysing data from on-going entities. The last three steps of the sequence account for the archiving of data. When the measurement sequence from an entity is complete, pipe no. 9 transfers the completed measurement series to the selective data storage component. The component then decides if the data will be piped (10) into the incremental history sink. The last pipe in the sequential order transfers the results of the completed measurements into the history sink (11).

The architecture meets the requirements laid down in the functional requirements analysis in Subsection 2.1. It instantiates and extends the reference architecture. The history section stores data selectively and provides data most similar to the on-going measurements to be utilized in models. The architecture can be configured to fulfill application-specific needs with its facilities for creating application-specific filters. It is suitable for processing

Figure 4. Architectural layout of Smart Archive. The units are shown as rounded boxes and the pipes connecting them as arrows. The sequential order in which data is processed during an update cycle is shown by the numbers above the pipes.
continuously observed data from multiple entities, and the operations of the architecture and its components are easy to understand.

3. Case study: implementing a data mining application for a steel mill

A data mining application for a steel mill was implemented using the developed architecture. Data from a steel slab re-heating furnace (a walking beam furnace) is collected on-line at the mill. Steel slabs are heated throughout the day to approximately 1100°C, and there are always some 40 slabs inside the furnace. Information of more than 150 variables is stored in four application-relevant database tables, and a subset of these variables, tracking the slabs in the furnace, is updated at one-minute intervals. The data mining challenge is to use these variables to predict the post-roughing mill temperature of the slabs while they are still in the furnace.

A neural network model for predicting the temperature accurately was first developed [12]. The production line implementation of the model was started out by building the first version of the software from scratch. Although the software used the components (excluding the components for utilizing history data) described in Section 2, almost all of the code was fully application-tailored. Java was used to implement the filters, MySQL and Oracle were the supported data sinks, and SQL queries (through a JDBC connection) were used to pipe the data. During the laborious implementation of the model, the need for a generic framework for constructing individual data mining applications became more and more evident. However, the tailored implementation was finalized before the development of the Smart Archive was started and the application was re-implemented using it. The tools mentioned above were also used to implement SA. The benefit of carrying out the implementation twice with different approaches is that it provides an excellent chance for comparison.

The two approaches are compared using criteria reflecting the requirements analysis of Subsection 2.1. The following list presents the evaluation, with the implementation from scratch denoted with (I) and the implementation using SA with (II).

- **Reference architecture:** (I) The implementation is an application-specific adaptation of the reference architecture. Feature extraction filters are hard-coded. The model is hard-coded. (II) The reference architecture is the backbone of the system and clearly present. The amount of work needed for data pre-processing is approximately the same as in the other implementation, although the implementation of the filters can be kept separate from the core architecture using the interfaces. Feature extraction and model filters are adapted to application-specific needs using the interfaces.
- **Use of history information:** (I) No support for history information. The user must implement his or her own components for utilizing history information, which raises the threshold for utilizing such information. (II) SA supports the utilization of history information. The user can take advantage of the facilities for identifying similar measurements from observation periods that may be long.
- **Level of transparency:** (I) Almost no transparency. It is quite hard, even for a person familiar with the reference architecture, to understand how the components of the system relate to each other. The documentation of the software requires all operations used in transforming data (~ 10 000 lines of code) to be explained. (II) High. A person familiar with the reference architecture can understand the operation of the system at a glance. Only application-specific filters need to be documented.
- **Usability:** (I) Fulfils the application-specific needs, works well as long as no major changes are made in the furnace data system. Upgrades must be carried out via the application-specific framework. (II) Tailored for the application-specific needs, operates well. Easy to adapt to new requirements, with an option to use third party components developed for the SA framework.
- **Implementation time:** (I) It took about 6 months to implement the solution. (II) It took about 1.5 months to implement the application-specific filters and a week to configure the architecture for the application.
- **Software quality:** (I) Bugs are equally likely to appear in all parts of the code. All of the code has to be tested. (II) Bugs are likely to be present only in the application-specific filters. Only application-specific filters need to be tested.

From the implementation viewpoint, the most important advantage was the decrease in implementation time and the increase in quality. The amount of time spent tracking bugs decreased, and the bugs could be traced by default to the application-specific code. From the modeling viewpoint, the ability to use history data is an important feature. In this particular application the production conditions vary so
much that the use of a training set containing data from a limited time frame does not necessarily lead to optimal results. When the training data is extended with similar measurements from longer periods of history, it is more likely that measurements resembling the current production conditions will be included. Finally, it is much easier to explain the application logic to a customer using the SA framework than a fully tailored approach.

4. Conclusions and future work

Because of the limited space available, this paper is merely an introduction to the architecture and framework provided by SA. Compromises in presenting the architecture and its implementation principles at a detailed level had to be made, and the emphasis was placed on presenting the research from multiple points of view in order to provide a general idea and to raise interest in the topic among people approaching it from different aspects. Particular effort was dedicated to illustrating the architecture from the viewpoint of an applier who is familiar with the reference architecture to some extent. The main points of the functional requirements analysis were enumerated in order to present the motives behind the design solutions of SA. The components and architecture of SA were explained at such a level that people interested in experimenting with the architecture can implement it and adapt it to their applications and tools. Finally, the benefits of SA in application generation were outlined in a brief case study. An implementation of a data mining application for a steel plant using a fully tailored approach was compared to an implementation using SA. The major benefits were a reduction in development time, higher quality, extensibility and a more transparent system structure.

Acknowledgements

The authors wish to thank Ruukki Hot Strip Mill in Raahé, National Technology Agency of Finland (TEKES) and graduate school GETA for supporting this work.

5. References


