View metadata, citation and similar papers at core.ac.uk brought to you by CORE

Paper

Component-Based Architecture for Systems, Services and Data Integration in Support for Criminal Analysis

Jacek Dajda, Roman Dębski, Aleksander Byrski, and Marek Kisiel-Dorohinicki

Department of Computer Science, AGH University of Science and Technology, Kraków, Poland

Abstract—Criminal analysis processes is based on heterogeneous data processing. To support it, analysts utilize a large set of specialized tools, however they are usually designed to solve a particular problem are often incompatible with other existing tools and systems. Therefore, to fully leverage the existing supporting tools, their technological integration is required. In this paper we present original approach for integrating systems based on the component-driven paradigm. Firstly, a problem of supporting criminal analysis is described with a strong emphasis on the heterogeneity issues. Secondly, some theoretical information about integration is depicted followed by the details of the proposed architecture. Finally, the technological assumptions are discussed and prototype integration based on proposed concept is overviewed. om the experiments are discussed in the final part of the paper.

Keywords—criminal analysis, component-based systems, software integration.

1. Introduction

The continuous technological development affects various domains and aspects of life resulting in progressing informatization of organizations, procedures but also daily routines and habits. Interestingly, this process can be also observed in such area as police operation activities and criminal analysis. On the other hand, contemporary criminals use more and more sophisticated methods based on modern information technologies, which again, cause police analysts to develop new techniques. While these techniques often prove useful, every new information source or analyzing scheme requires additional time to be processed or executed.

Therefore, to support analyst in this process, various software tools are developed. They operate on different data and data formats as well as provide different set of features. These tools are developed by different software teams and often in different technologies and architectures. As result while specific activities are supported, there are still significant support gaps related to data handling and colligation as well as operation flows arrangement and execution. These gaps are also caused by the problem of new data sources and operations that are introduced continuously into the whole process. These sources and operations often remain unhandled by the existing tools that were designed for other purposes and are not generic enough.

Taking all of these under consideration, it must be assumed that the proper solution should be based on the concept of integration. The proper designed integration-driven architecture should be opened for adding new information sources and analytical operations provided by existing libraries and systems. On the other hand, it should remain lightweight and scalable so that can applied both to single desktop applications as well to larger systems.

The goal of this article is to present the concept of such architecture. To make the architecture flexible, it is proposed to take advantage of the component paradigm [1]. This paradigm assumes assembling applications from a set of independent components through well-defined functional contracts. By adding new components to the system, it can be extended to cover new requirements. What is more, the proposed architecture assumes the reuse of the component-based approach on all the architectonic levels from the data layer to the graphical user interface layer.

The structure of the paper is as follows. In the next section, the basics of the criminal analysis process are presented. Next section depicts current state of art. Section 4 is dedicated to the overview of the proposed architecture. In Section 5, the key mechanism behind the architecture are explained. Section 6 discusses the technological assumptions for architecture realisation. Section 7 presents an example implementation and preliminary evaluation of the proposed architecture. Finally, Section 8 contains the summary and plan for future work.

2. Supporting Criminal Analysis

The process of criminal analysis can be briefly described as a loop of the following intertwined activities: data-retrieval, processing, visualisation and interpretation. The rule of operation depends on a given scenario. The general concept is presented in Fig. 1.

It assumes that the analyst operates on various information sources such as police databases and Internet services and also on such operational data as phone billings. The data retrieval from a specific source requires a dedicated tool or parser. After the data is retrieved it can be filtered, searched, and analyzed with the use of available operations and algorithms such as pattern recognition algorithms [2] or hypothesis testing based on social networks

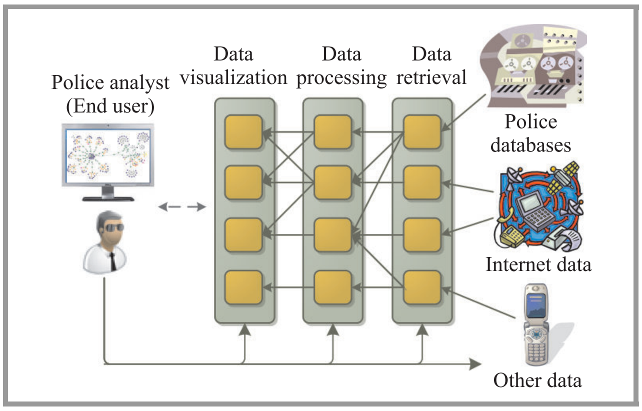

Fig. 1. Criminal analysis.

approach [3], [4]. Typically, there is a specific range of operations which can be executed on a data coming from a given source.

The result of the data processing stage is usually a subset of the initial data, which can be visualized using some predefined visualization type. The visualization stage assumes a considerable amount of human interaction in order to extract additional information based on such prerequisites as specific layout of data.

As stated in the beginning, the whole process is not a one-time sequence of operations but a continuous loop in which the analyst, based on his/her findings inspired by the specific visualization, can go back to specific stage and repeat the whole process by performing new visualizations or other operations on the actual data as well as adding new data or even new data sources to the analysis.

No doubt, the whole process can be time-consuming and error-prone. Thus, a proper software support is needed. While the results interpretation must be still carried out by the analyst (although in near future it can be also supported by artificial intelligence reasoning), the rest three activities can be greatly optimized by the proper software tool support. However, these tasks pose certain challenges.

First of all, the range of data sources is unlimited and can vary from Internet pages, through data-bases to simple text files containing data in structure or unstructured form. What is more, each data source may produce a unique set of data types which require specific processing. Once the data is obtained and the processing results are ready, they need to be visualized in order to be interpreted by the analyst. Again, each data type may require a dedicated visualization type, which, to make it more complex, may be linked with other visualization to form an efficient graphical user interface for the analyst.

What the data sources have in common is that they usually produce large quantities of information which is practically impossible to process manually. This constitute another challenge for the designers of supporting solutions.

Currently, there is a variety of tools and systems available that offer a different amount of support for certain data domains or operations. These solutions can be classified into two groups: specific and general-purpose ones. An example of the specific solution is the LINK platform [5] which allows for data analysis and visualization using object graphs. As for the general-purpose tool, Excel can be given as an example. During the process of analysis, these solutions are often utilized together. However, their integration is often limited or none. As result, while the individual solutions are of great help for the analyst, he/she spends a considerable amount of time on data transformations and moving the results from one solution to the other in order to continue the investigation plot.

Theoretically, the problem could be solved by constructing a large, scalable all-purpose analytical environment which will allow for handling all possible data sources and will provide all the necessary services and visualizations. However, from the practical and economical point of view this is impossible and cannot be accepted as a solution to the problem. Another utilized approach is the integration of the available legacy solutions and databases. In most cases, this assumes integration of specific products such as database and analytical system. However, in the discussed case, this is not a fully satisfactory solution due to the number of possible sources and utilized systems. What is more, new sources or systems can be added in future which emerges from the organizational and other aspects that are beyond the decisions of software architects.

All of this makes the problem even more challenging. It seems that the core of the problem lies in the heterogeneity. The heterogeneity can be observed onto 3 following levels:

– data sources,

– service providers (operations),

– technologies.

Data sources can vary in the data containers, formats and standards of the data. Service providers can differ in the rule of operation, semantics of the input arguments and produced results. As for heterogeneous technologies, this aspect must be considered on every stage of the analytical process which means data sources, service providers and visualization libraries.

No doubt, the heterogeneity should be approached and handled on the general architectural level. In the next section the current state of art in terms of software integration is overviewed.

3. Software Integration Styles

Application integration is nowadays a mature engineering discipline – good practices are cataloged in the form of patterns [6]. All the patterns can be generalized and divided into four main categories: *file transfer, shared database, remote procedure invocation* and *messaging*. The following sections describe briefly each of them.

3.1. File Transfer

This approach to integration utilizes files – universal storage mechanism available in all operating systems. One application (producer) creates a file that contains the information needed by the other applications. Next, the other applications (consumers) can read the content of the file (Fig. 2). Choosing this approach has the following consequences [6]:

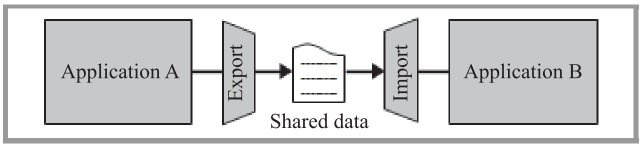

Fig. 2. File transfer.

- it is *data sharing oriented* (not *functionality sharing oriented*),
- files are (effectively) the public interface of each application,
- choosing the right file (data) format is very important (nowadays it is often XML),
- applications are decoupled from each other,
- applications are responsible for managing the files (creation, deletion, following file-naming conventions etc.),
- updates usually occur infrequently and, as a consequence, the communicating applications can get out of synchronization,
- integrators need no knowledge of the internals of applications.

3.2. Shared Database

In this pattern the integrated applications store their data in a single (shared) database. The stored data can be immediately used by the other applications (Fig. 3). Choosing this approach has the following consequences [6]:

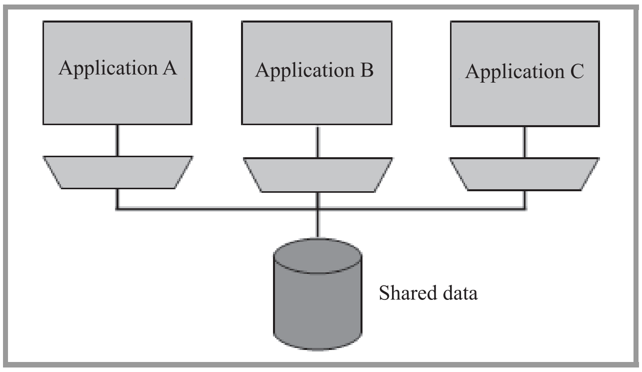

Fig. 3. Shared database.

- it is *data sharing oriented* (not *functionality sharing oriented*),
- data in the database are always consistent,
- defining a unified database schema that can meet the needs of many applications can be a really difficult task,
- any change of the shared database schema may have impact on all integrated applications (applications are strongly coupled),
- since every application uses the same database, there is no problem with *semantic dissonance* [6],
- shared database can become a performance bottleneck and can cause deadlocks.

3.3. Remote Procedure Invocation

In this approach each part of the integrated system (a set of cooperating applications) can be seen as a large-scale object (or component) with encapsulated data. Shared functionality of each application is accessible via its public interface[1] (Fig. 4). Choosing this approach has the following consequences [6]:

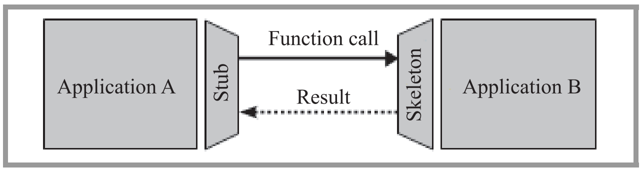

Fig. 4. Remote procedure invocation.

- it is *functionality sharing oriented* (not *data sharing oriented*),
- applications can provide multiple interfaces to the same data,
- applications are still fairly tightly coupled together (often each application of the integrated system perform a single step in many-step algorithms: in such a case one application's failure may bring down all of the other applications),
- communication is (usually) synchronous,
- developers often forget that there is a big difference in performance and reliability between remote and local procedure calls – it can lead to slow and unreliable systems.

[1] A number of technologies such as CORBA, COM, .NET Remoting, Java RMI and Web Services implement Remote Procedure Invocation (also referred to as Remote Procedure Call or RPC).

3.4. Messaging

This approach combines all the benefits of the previous three and is often considered [6] as the best one. Messages transfer packets of data frequently, immediately, reliably and asynchronously using customizable formats (Fig. 5). Choosing this approach has the following consequences [6]:

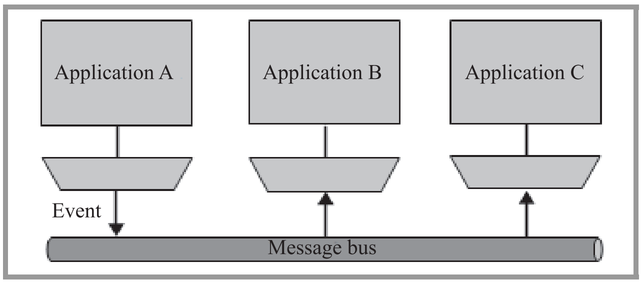

Fig. 5. Messaging.

- sending small messages frequently allows applications to collaborate behaviorally as well as share data,
- applications are decoupled (it has many consequences e.g., integration decisions can be separated form the development of the applications),
- the integrated applications (usually) depend on a messaging middleware.

4. Component-Driven Integration Architecture

Each of the presented integration styles has its advantages and disadvantages. Also, each one of them imposes specific requirements on the integrated systems. While fulfilling these requirements is possible for developers of the integrated systems, the same cannot be assumed when dealing with legacy, or worse, external systems. In this case, the choice of the proper integration style may vary on the specifics of given systems.

It seems that in the discussed problem a greater level of flexibility is needed. One that will allow to embed all the styles and utilize the one that suits best the integrated systems. To achieve this flexibility, it is proposed here to take advantage of the component paradigm.

Using the paradigm, it is possible to build an extensible architecture in which various integration styles can be realized by adding new dedicated components. In such approach, the integration can be view from the following two perspectives: low-level and high-level.

The high-level perspective is the one already discussed in the previous section. It means integration of data sources and systems by establishing data and service links. In the low-level perspective, integration refers to the individual components and their contracts. By providing proper components in the low-level, it is possible to achieve the high-level integration of desired style. For example, the shared database approach can be achieved by providing proper components dealing with the aspects of data retrieval (e.g., communication with the database) and conversion (e.g. object-relationship mapping of specific tables to a specific object model).

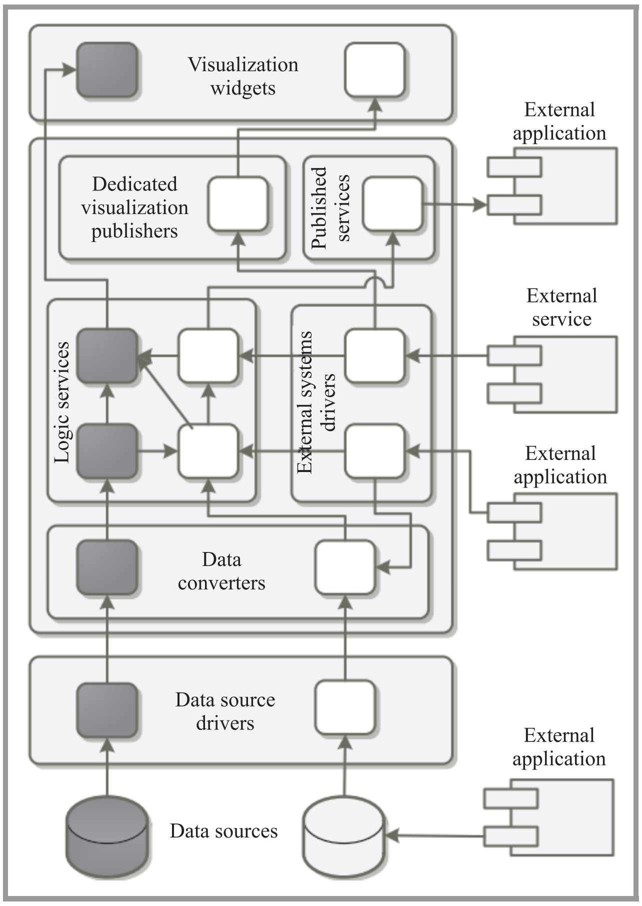

Fig. 6. Component-driven integration architecture.

In Fig. 6 the proposed architecture is presented. The architecture is driven by the component paradigm. The architecture consists of the following types of elements:

Externals. They include all the external entities that operate within the architecture such as available data sources, services and systems.

Drivers. They are specific elements of the architecture that enable the low-level integration of the externals in the whole architecture. Drivers are responsible for handling communication with the externals and providing a model (data or service based) for other architectural elements that need to communicate with the externals. A JDBC driver or specific web service proxy are the examples of drivers.

Services. These elements are responsible for data processing and publishing to other elements. Processing includes data conversion, filtering (transformers) and execution of domain specific operations and algorithms (functional services). Also, there is a group of publishing services for both external systems (such as web services) and the visualization elements which can be utilized to prepare data for visualization purposes, in this way, improving the visualization performance.

Visualization elements. They are responsible for visualization of the data and processing results. An example of a visualization element, a graph-based view of the data can be given.

The layout of the elements and scheme of their dependencies presented in Fig. 6 are illustrative only.

There are two significant aspects of the presented architecture to be emphasized. First, all of the architectural elements (except externals of course) are software components, and thus are represented by identical symbols. Being components, they can be attributed with some contracts based on which they can be assembled and communicate with each other. This is represented by arrows showing how the communication and data can flow between these components. This shows how the whole integrated system can be extended with the support of new data source or service. In that case, a specific component or components (driver at least, maybe some transformer) need to be provided and linked with other existing components to provide new features.

The second aspect is how linking the components can provide new features. Obviously, a feature (in terms of user functionality) is usually built around several cooperating components. Let's take a closer look on the selected components which are distinguished in Fig. 6 by the dark colour. There are 6 interlinked components: starting from a data source, through its driver, data transformer, functional services with a visualisation component at the top. These components clearly form a kind of processing sequence. This sequence is a flow of certain operations performed on a given data. Therefore, it can be treated as a dataflow. One can notice that the operation of the whole architecture is based on various dataflows. The concept of dataflow is the core mechanism of the architecture and reflects the analytical process presented in Fig. 1. Next section describes the concept in more details.

5. Realization Assumptions

Having overviewed the architecture concept, the initial technological assumptions can be made.

When the technological assumptions are made there are always several aspects and levels on which the choice of proper technologies must be concerned. Here, the following aspects are taken under consideration: modularity, data persistence, communication, graphical user interface, semantic integrity.

5.1. Modularity

The presented concept emphasizes the system modularity which can be realized by a component-based approach. There are several available component frameworks available from simple and efficient PicoContainer [7], Guice [8] to complex and advanced ones such as Spring framework [9] and Eclipse RCP [10], which provide also a number of other advanced features.

5.2. Data Persistence

While the proposed architecture assumes data retrieval from external data sources, the integration infrastructure also requires internal persistence mechanism, for example for caching and versioning purposes. This makes the persistence a crucial aspect in which the performance is very important for efficient processing of massive quantities of data. Also, an important aspect is the flexibility which would allow for handling completely new data models without corrupting the existing data. For this reason, it is assumed that the persistence mechanism can be realized through ORM paradigm [11].

5.3. Communication

There are several methods of communication that might be required in the discussed architecture. In the simplest case, a component-based framework can be sufficient platform for plugin-based communication. However, in more distributed approach, in particular when services integration is considered, network-oriented technologies are required. Here the Java RMI [12] or SOA [13] concept and its implementations can be utilized such as Apache Geronimo [14] or Microsoft's Windows Communication Foundation WCF [15].

5.4. Graphical User Interface

The choice of proper GUI technology is always a difficult task as it strongly depends on the user requirements and preferences, which are often very individualistic. The situation is also aggravated by the fact that currently at the market there is a number of libraries and technologies available.

Another aspect refers to the architecture type of the system: whether is it web-based application, server-client application (with heavy client) or desktop application for off-line work. Another issue is the target platform whether this is Windows, Linux or Mac.

With a high level of uncertainty concerning above aspects, it seems that the following assumptions need to be taken:

- It is preferable to use a technology which is portable. With a use of web-based architecture this requirement is of course easier to fulfill than for desktop application. As for example, solution J2EE technologies with Ajax support (such as RichFaces [16] library) might be pointed out.
- It is preferable to use a technology which is flexible with respect to the way of execution (web application or desktop). Currently, this assumption is the latest

trend realized by such technologies as Windows Presentation Foundation WPF [17] and Silverlight [18], Adobe AIR [19] or JavaFx [20].

A careful look should be given to WPF library as it is aimed at clear presentation of the data transformations that may be performed in the system.

5.5. Semantic Integrity

To achieve the semantic integrity between various models emerging from different data sources the ontology paradigm might be used. As for technological choices, one of the option can be Web Ontology Language (OWL) and Resource Description Framework (RDF) graphs [21].

6. Selected Implementation Aspects

To validate the proposed architecture, a prototype implementation is realized. For experimental purposes, a simple integration scenario is proposed. The goal of the scenario is to integrate two analytical applications. The first one is LINK tool [5] which allows for importing, preprocessing and visualising data coming from file-based sources. The second application is Mammoth [22] which offers specialized pattern recognition and discovery functionalities.

Both applications have their strong an weak points, however when utilized together they can form an advanced analytical platform. The integration scenario assumes that LINK application can be used as data import, preprocessing and visualization tool, while Mammoth can provide routines for finding the patterns in the imported data, which can be visualized in a form of graph. To make the integration more difficult, the applications are realized in different technologies: LINK is realized in Java technologies while Mammoth is developed in .NET technologies.

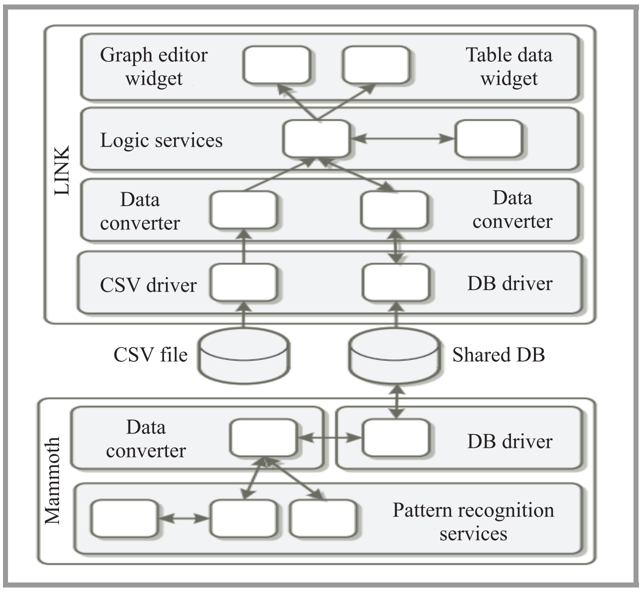

Fig. 7. Architecture overview of the prototype integration.

Figure 7 illustrates how the proposed component-drive architecture is applied to the discussed scenario. First thing to be noticed is that the chosen integration style is the shared database approach. There is also another data source which is a CSV file. The idea is that LINK provides dedicated components for reading data coming from CSV files and preprocessing the data. Once the data is ready, it is put into the shared database and can be visualized in form of tables or visual graphs.

In order to perform an advanced analysis, user can take advantage of the features of Mammoth which connects to the shared database using dedicated components. After the data is converted to a proper format for the available algorithms, it is processed and the results (found patterns) are written back to the database, from which, the found patterns can be read, converted and visualized in a form of graph.

One can notice that the rule of operation which is described in the above paragraph is a plain data-flow. This flow is illustrated in greater detailed in Fig. 8.

As for the technological choices, the shared database was realized using SQLite. As for LINK application it utilizes Eclipse RCP component platform and dedicated ORM layer based on JDBC. The visualization was realized with SWT and GEF libraries [5]. As for Mammoth application, it utilizes MEF as component platform and ADO.NET for database integration. The logic services were realized using various algorithm models for pattern recognition [22].

This prototype implementation shows that by utilization a component-driven approached, it is possible to integrate existing systems and applications and provide analyst with the more advanced environment that consists of multi-data sources and applications, as presented in the conceptual architectural scheme in Fig. 6.

7. Conclusions and Further Work

In this paper, a component-driven architecture is presented. This architecture is designed as a solution to the integration problem which occurs during development of maintainable software support for criminal analyst. To show how this architecture can be realized, the paper discusses the technological assumptions and provides selected details from the prototype implementation, which follows one of the discussed integration styles.

Further work should proceed into two following directions. The first one will be examining other integration styles. The second one is experimenting with larger integration examples that include more heterogeneous technologies and data models.

Acknowledgements

The work described in this paper was partially supported by The European Union by means of European Social Fund, PO KL Priority IV: Higher Education and Research, Activ-

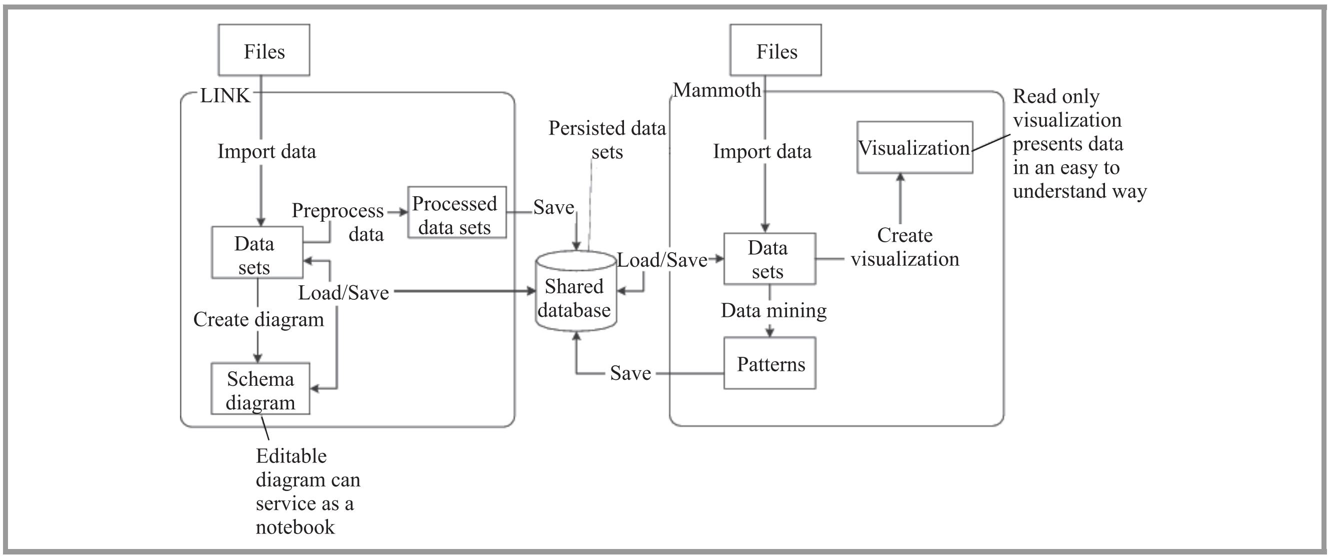

Fig. 8. Dataflow in the prototype integration.

ity 4.1: Improvement and Development of Didactic Potential of the University and Increasing Number of Students of the Faculties Crucial for the National Economy Based on Knowledge, Subactivity 4.1.1: Improvement of the Didactic Potential of the AGH University of Science and Technology “Human Assets”, no. UDA-POKL.04.01.01-00-367/08-00.

The research leading to these results has received partial funding from the European Community’s Seventh Framework Program – Project INDECT (FP7/2007-2013, grant agreement no. 218086).

References

[1] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[2] Pei J. Han J. and Yin Yiwen, *Mining Frequent Patterns without Candidate Generation*. ACM, USA, 2000.

[3] R. A. Hanneman and M. Riddle, *Introduction to Social Network Methods*. University of California Press, 2005.

[4] A. Kirschner, “Overview of common social network analysis software platforms”, Tech. Rep., Monitor Group, San Francisco, 2008.

[5] R. Dębski, M. Kisiel-Dorohinicki, T. Milos, and K. Pietak, “Link – a decision-support system for criminal analysis”, in *Proc. IEEE Int. Conf. Multim. Commun. Serv. Secur. MCSS 2010*, J. Danda, J. Derkacz, and A. Głowacz, Eds., Kraków, Poland, 2010, pp. 110–116.

[6] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2003.

[7] P. Hammant, K. Pribluda, and J. Schaible, “Picocontainer: a highly embeddable, full-service, inversion of control (ioc) container for components honor the dependency injection pattern” [Online]. Available: http://www.picocontainer.org/

[8] R. Vanbrabant, *Google Guice: Agile Lightweight Dependency Injection Framework*. Apress, 2008.

[9] C. Walls, *Modular Java: Creating Flexible Applications with Osgi and Spring*. Pragmatic Bookshelf, 2009.

[10] C. Aniszczyk J. McAffer, J. M. Lemieux, *Eclipse Rich Client Platform*. Addison-Wesley Professional, 2010.

[11] T. Halpin and T. Morgan, *Information Modeling and Relational Databases*. Morgan Kaufmann, 2008.

[12] W. Grosso, *Java RMI*. O’Reilly Media, 2001.

[13] T. Erl, *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, 2005.

[14] K. Kumar, *Pro Apache Geronimo*. Apress, 2006.

[15] J. Lowy, *Programming WCF Services*. O’Reilly Media, 2007.

[16] M. Katz, *Practical RichFaces*. Apress, 2008.

[17] A. Nathan, *Windows Presentation Foundation Unleashed*. Sams, 2006.

[18] M. MacDonald, *Pro Silverlight 3 in C#*. Apress, 2009.

[19] B. Gorton, R. Taylor, and J. Yamada, *Adobe AIR Bible*. Wiley, 2008.

[20] S. Morris, *JavaFX in Action*. Manning Publications, 2009.

[21] D. Allemang and J. Hendler, *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann, 2008.

[22] P. Włodek, A. Świerczek, and B. Śnieżyński, “Pattern searching and visualization supporting criminal analysis”, in *Proc. IEEE Int. Conf. Multim. Commun. Serv. Secur. MCSS 2010*, J. Danda, J. Derkacz, and A. Głowacz, Eds., Kraków, Poland, 2010, pp. 212–218.

Jacek Dajda received his Ph.D. degree in Computer Science from AGH University of Science and Technology in 2008. At present, he works as Assistant Professor at the Department of Computer Science of AGH-UST. His research interests involve software engineering with an emphasis on software architectures and frameworks.

E-mail: dajda@agh.edu.pl
Deptartment of Computer Science
AGH University of Science and Technology
Mickiewicza Av. 30
30-059 Kraków, Poland

Roman Dębski, **Aleksander Byrski**, and **Marek Kisiel-Dorohinicki** – for biographies, see this issue, p. 22.