Towards Service-Oriented Knowledge Discovery A Case Study

Jeroen de Bruin[1], Joost N. Kok[1], Nada Lavrac[2] and Igor Trajkovski[3]

[1] LIACS, Leiden University, Leiden, The Netherlands
[2] Jozef Stefan Institute, Ljubljana, Slovenia
[3] New York University Skopje, Skopje, Macedonia

Abstract. Due to advances in software engineering and architecture, as well as the increased popularity of scientific workflows, new and better ways of performing knowledge discovery experiments can be devised. In this paper we look into these technologies and reason how they can improve tradiotional knowledge discovery processes, and present an implemented use case that shows how these improvements work in practise. We also discuss the weaknesses of the new methods and look into directions that can be taken with these new technologies.

1 Introduction

Knowledge Discovery (KD) in data has proven to be valuable in many scientific fields over the last few decades. Setting up a KD experiment is no simple task. KD processes often consist of several algorithms connected together, whereby data flows from one algorithm to another. A common scenario for a KD process is as follows: the KD researcher either programs or obtains all algorithms and connects the in- and ouputs together, runs the process, and eventually gets a result as an output. We perceive this situation as far from optimal, as it comes with quite a few problems in process management and construction, and suboptimal performance.

Instead we can consider the following scenario: Suppose a researcher wants to create a certain KD experiment involving several algorithms. The researcher only has some of these algorithms on her own computer, knows that there are a few available at a location halfway around the world, and some that she needs but does not know where to find. The ideal situation for the researcher would be to just use a search engine to look up the missing algorithms, use a tool to connect all algorithms together, and then run the experiment.

The scenario presented above is not at all unrealistic. Due to advances in workflow management research, experiments can now be designed in such a way that individual parts of the experiment can be easily connected to each other, often in a simple graphical interface. Advances in software engineering and architecture have led to the Service Orientation (SO) paradigm that allows for relatively simple and secure remote computing, and easy lookup of publicly available services. Through combination of both technologies, we can make KD

processes better, faster, and easier to construct, which in turn leads to better and more reliable research.

Until recently the focus and application domain of SO technology has mostly been the commercial sector and large-scale business applications. In this paper we explore the benefits and drawbacks of SO applications in KD by conducting a case study. In Section 2, we will contrast the two scenarios briefly discussed above. We will also cover the merits of SO and workflow technology. In Section 3, we will discuss the specific algorithms and technologies used to implement the case study. In Section 4 we discuss the case study and compare its implementation and design to its non-web-based counterpart. Finally, in Section 5, we draw some conclusions on the use of SO in KD, and shed some light on future work and future investigations.

2 Problem statement

In this section we contrast the first scenario where the researcher *constructs* the KD process to the second scenario where the researcher *composes* a KD process using SO and workflows.

2.1 Scenario 1: Constructed KD process

In this first scenario the researcher constructs her own KD experiment by obtaining all required algorithms and connecting them manually. In practice, this usually means accessing diverse resources, for example the internet, to find all required algorithms and KD packages from diverse sources, and then run them one by one. If the algorithms are contained in a KD package like WEKA [Gar95], the researcher does not need to worry about intermediate data representation as the package does that for him, but if this is not the case, the researcher has to program this representation as well. Apart from this tedious construction process, there are also a number of weaknesses that can easily break this scenario:

- **Algorithm versioning**
  When a new version of an algorithm or KD package appears, the version that the scientist uses is often not automatically updated. Instead, the scientist has to keep track of modifications himself by regularly visiting the website. Another problem with versioning is that it sometimes breaks compatibility with previous versions, which may break the entire KD process.

- **Algorithm connection**
  While standard KD packages perform well on basic KD tasks, they usually lack algorithms that are tailored to a specific field of research like bioinformatics. When this is the case, the scientist has to connect all individual components of the KD process himself, usually by using a scripting language. This connection is often constructed in an ad-hoc fashion, making it less likely to cope with changes that might occur. Furthermore, the intermediate representations used in the connections are often suited only for the

process they were designed for, so if two processes were to be combined, it is likely that one representation would need change.

- **Algorithm availbility**
  It is not unlikely that some algorithms or specific algorithm implementations are not publicly available, espcecially when implementations are managed by commercial institutes. They want to keep their implementation propriety knowledge, and are unwilling to distribute it or only do so at a high cost. This leaves the researcher the choice to either implement the algorithm herself, or to revise the experiment.

- **Performance**
  KD processes sometimes involve terabytes of data. When performing KD on such high volumes of data, every speedup counts, and ideally the scientist uses the fastest machines with the best algorithms in their optimal implementation. However, the optimal implementations of algorithms are not always available or suitable for the platform that the scientist uses, making the KD process slower. Furthermore, some algorithms run faster on specialized hardware, that is also not always available to the scientist.

2.2 Scenario 2: Composed KD process

In this scenario the scientist composes her experiment through the use of a scientific workflow. We define a scientific workflow as a collection of components and relations among them, together constituting a process. Components in a workflow are entities of processing. They are connected by relations, which can either be data entities that transport data from one component to another, or control entities that impose conditions on the execution of a component. Scientific workflows have become increasingly popular over the last few years, since they allow a scientist to graphically construct a process of interconnected building blocks, allowing for easier experiment design.

Components in the composed KD process can be present either locally or at some distant location. These components interact with each other in a Service-Oriented Architecture [Gro]. A SOA is an architectural style that supports SO, where SO is a way of thinking in terms of services and service-based development and the outcomes of services. In practise, SOA is a distributed architecture that allows a researcher to build an application by means of composing individual components that potentially exist across separate (physical or logical) domains. These components are called Web services [HD06]

The combined use of SO and scientific workflows adresses the weaknesses of scenario 1 as follows:

- **Algorithm versioning**
  Keeping track of newer versions of an algorithm is no longer an issue for the scientist, since it is automatically updated on the side of the service provider. A scientist can be notified of an update, but updating can also

proceed completely transparently. Compatibility with previous versions is also guaranteed, for the service has to adhere to a certain interface, an annotation of the service's functionality that serves as a contract between the service user and the service provider. A widely used standard for annotating web services at the moment is the Web Service Description Language (WSDL) [W3Cb]. WSDL is an XML-based standard that describes for each web service how the service handles incoming messages and what type of parameters it supports.

- **Algorithm connection**
  Data transport between components in a scientific workflow proceeds through a standardized way, usually in some form of structured transport protocol. Web services are often accessed through messages written in the Simple Object Access Protocol (SOAP)[W3Ca], which is an XLM-based message format and transport protocol. Since message formatting is standardized, the KD process will not break as long as the components will adhere to the message content, which is guaranteed by the component's interface.

- **Algorithm availability**
  Since implementations of algorithms are now managed by the service provider and executed on their end, it is safe for the providers to offer their services without running the risk of losing proprietary knowledge. These services can automatically be polled and found by the scientist through a Universal Description Discovery and Integration (UDDI) [Dra], which is a registry for web services offered by service providers containing all WSDL documents corresponding to services of that provider.

- **Performance**
  Since SOA and all related protocols discussed above are platform-independent technologies, each platform can potentially support it. This makes it easier for the service provider to use the programming language and platform that is best suited, which usually leads to a performance increase. Moreover, if two services can be executed independently and are located on different machines, they can be executed in parallel in a scientific workflow, speeding up the entire KD process even more.

3 Experimental Setting

3.1 Algorithms

For our case study we used a KD scenario described in [TZTL06]. In this scenario microarray data is processed to identify differentially expressed genes based on a threshold score computed by the student's t-test. This set of differentially expressed genes, together with a selection of their non-differentially expressed counterparts (both expressed in Entrez id's [MOPT05]), are then annotated

with terms from the Gene Ontology (GO)[GO]. In the final step these annotations, together with information about interaction amongst genes, are represented as facts and supplied to the Relational Subgroup Discovery algorithm (RSD) [LZF02].

The RSD algorithm takes a set of labelled data items and a class (in this case the classes differentially expressed and non-differentially expressed) and tries to find descriptions of subgroups of target class examples that are as large as possible, and have a significantly different distribution of the target class examples. However, to avoid that a certain set of data items dominate the entire rule-space, an iterative weigthed covering algorithm is used to decrease weights of those items once they are collected in a rule. Based the number of items in that rule belonging to each class and their individual weights, the quality of a rule is measured.

As a post-processing step, rules are uniformly formatted using the GO descriptions, improving readability for expert reviewers. As an example, consider the rule below:

Rule 1: Support 6, Weight: 12.0
Differential participants: [119391,1375,5287,1021,6011]
Non-differential participants: [9950]
molecular_function(A,catalytic activity), cellular_component(A,cytoplasmic part)

In this rule, a total of six genes were involved; five of them were differentially expressed, one was not, giving the rule a total weight of twelve (weigths of individual genes are not mentioned in the rule). The rule itself states the common factors of all genes, whereby the genes are designated as group 'A'. The different predicates like 'molecular_function', as well as descriptors like 'catalytic activity' all come from from GO. In this case, the correct interpretation of the rule would be:

"Subgroup 'A' of the differentially expressed genes resides in the cytoplasmic part of the cell and has as primary function catalytic activity, whereby 'A' consists of genes with Entrez ids 119391, 1375, 5287, 1021, 6011 and 9950"

3.2 Workflows

A number of workflow designer tools have been developed over the last few years, such as the orange toolkit [JMD+05] and Taverna [MyG]. For our case study, we chose Taverna because it has the capability to execute web services. In Taverna, components are called processors, and apart from local services and WSDL services, Taverna also supports BioMoby [WL02] and SoapLab [KFH+06] web service interfaces. Connections in Taverna are pretty straightforward; data connections are called data links, and control connections are called contol links. After a process has been designed and composed, the user can supply the input parameters of the process and execute it. When the process is done, Taverna will present the user the results, or give an error message if something went wrong.

3.3 Implementation

Implementation of the original algorithms was done in the Python language and run on Python 2.5.2. The web service implementations were done in Microsoft C++ .Net 2005 and Microsoft C-Sharp 2005 . All experiments were performed on MicroSoft Windows XP using an Intel centrino duo processor 1.66GHz, and 1GB of main memory.

4 Use Case

In this section we present the use case. The original program was already split up in a selection and a rule mining part, so we implemented both as distinct services. We chose three different implementations to compare: The first implementation was a simple webservice that executed a command in the windows command-line interpreter, the second implementation was a python web service implementation that uses the original python code, and the last implementation was a C-sharp web service implementation that used a C++ .Net re-implementation of all python code. All implemenations still use the RSD algorithm, which was left unmodified.

4.1 Shell web service

For our first use case, we made a web service in C-Sharp that takes as an input the parameters of the program or executable that needs to be executed, and simply forwards them to the command-line interpreter. While this is the easiest implementation of a web service and introduces the possiblily of remote processing, it still shares some of the weaknesses of a constructed KD process, since the underlying program remains prone to unexpected change in versioning, thereby possibly breaking the web service shell.

4.2 Python web service

For our second use case we tried to modify the original program code as little as possible, to show that any program can be transformed into a web service within its own implementation domain. For the Python webservice implementation we used the Python Webservice Module, and coupled the input of both services to this module. Furthermore, we extended the algorithm to use the KEGG ontology [KEG] as well, to show how updates can be performed without modifying the interface of a web service, making users completely oblivious of the service's implementation and updates.

4.3 C-Sharp web service

For our last use case we re-imlemented all the Python code into C++ to show how web services can increase performance. The performance gain is in several

dimensions here. First, applications made in programming languages like C++ and C-Sharp tend to execute faster than those made in scripting languages due to rigorous compile-time optimizations. Second, the authors are more proficient with C++ than with Python, which also yields in a performance increase. To compare performance, we took the microarray data used in [GST+99] and reran the experiment that was done in [TZTL06], whereby Acute Lymphoblastic Leukemia (ALL) was contrasted against Acute Myelogenous Leukemia (AML). The results of the benchmark test can be seen in Table 4.3. All measurements are in seconds and are the averages of 50 consecutive runs

	Selection service	Mining service
Original implementation	3.08	99
Shell implementation	3.20	102
Python implementation	3.23	100
C++ implementation	1.53	66

Table 1. Web service implementation benchmarks

4.4 Taverna workflow

In Taverna we constructed the workflow as shown in Figure 1. To set up the workflow, the user loads the data files to the inputs created, and then runs the workflow. When succesfull, the workflow will display the returned file. Because all parameters are bundled together in one SOAP message, we used input-splittlers and output-splitters to join and disjoin them. A few observations on this workflow:

- **Monolithic sequential processing**
  Subprocesses that are in sequential order in Taverna need to finish first before the next subprocess can start; when sending data to a remote component, all the data is uploaded first before data mining can begin. While it is very intuitive to separate these processes completely, it can also be faster to let the processing begin while the data transfer is still in progress. In case of our Selection service, which can already start calculating t-values of individual gene entries when the upload is still in progress.

- **Stateless services**
  When executing our mining service, first all ontologies have to be loaded into memory, then annotation of the genelists is performed, and finally RSD starts processing. Every time the web service is executed, the same process is repeated. This is because this service is stateless, meaning that the web

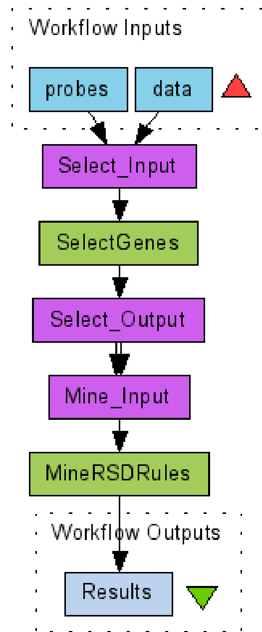

Fig. 1. Taverna workflow of web services

service has no knowledge of previous executions. We argue that if the service would preserve some form of state, execution could proceed faster. For example, if the mining service would keep the ontologies in memory and only change the gene assignments, the process would speedup considerably, being freed from most startup costs after the first run.

- **String representation**
  Message contents in SOAP are usually represented in text form with the type String. While this suffices for small messages, it is a redundant representation for large volumes of data. We argue that compression and decompression of data segments in the SOAP messages could speed up a KD process .

Notice that these observations may also hold for the KD construction scenario, and that solutions to these problems require reimplementation of services and underlying algorithms, and possibly revisions or extensions to the SOAP protocol and Taverna messaging.

5 Conclusions and Future Work

In this paper we discussed the transition from construction of a KD process to the composition of a KD process through the use of SOA and workflows. We have contrasted two scenarios that indicate the weaknesses of process construction and execution on a single machine, weaknesses that were adressed in the second scenario by composing processes in an SO fashion.

By using web services, versioning, connectivity, availability and performance of individual services are improved; versioning is improved by transparent updating of algorithms and the use of fixed, WSDL standardized interfaces, connectivity is improved by using the standardized SOAP transport protocol, availability is improved by better guarantees in service safety and the availability of the UDDI lookup service, and performance is improved by the option of parallel programming and platform specific implementations.

Apart from the improvements on individual services, the design and performance of the entire KD process can be improved by using scientific workflows, which can be designed and executed with the Taverna workbench. By using workflows, design of KD processes is easier and more intuitive, since it splits KD processes in processing components and connections. A KD process designer just needs to import the components and connect them together in order to gain a valid KD process, which can then immediately be executed to gain a result.

To illustrate the merits discussed above and to show how web services can be implemented, we created three different use cases. We showed how each use case contributed to the KD process, and showed how web-services can yield a performance gain, sometimes cutting exectuion time by 50%

Apart from merits we also addressed weaknesses in SOA and scientific workflows as they are right now, weaknesses such as monolithic sequential processing, redundant message formats and staelessness. These weaknesses can be adressed by further re-implementing algorithms as staeful services, and my modifying or extending SOAP.

Apart from SOAP extension, we also see future work in the combination of web services and databases. Our vision is to use web services to access databases to perform remote datamining and to construct queries. This vision fits in the vision expressed in our earlier work [dBK06,dB06]

References

[dB06] Jeroen S. de Bruin. Towards a framework for inductive querying. In Floriana Esposito, Zbigniew W. Ras, Donato Malerba, and Giovanni Semeraro, editors, *ISMIS*, volume 4203 of *Lecture Notes in Computer Science*, pages 419–424. Springer, 2006.

[dBK06] Jeroen S. de Bruin and Joost N. Kok. Towards a framework for knowledge discovery. In Jean Ponce, Martial Hebert, Cordelia Schmid, and Andrew Zisserman, editors, *IFIP PPAI*, volume 4170 of *Lecture Notes in Computer Science*, pages 219–228. Springer, 2006.

[Dra] Uddi Open Draft. Uddi version 2.0 api specification.

[Gar95] S. Garner. Weka: The waikato environment for knowledge analysis, 1995.

[GO] The gene ontology. http://www.geneontology.org/.

[Gro] The Open Group. Definition of soa, version 1.1. http://opengroup.org/projects/soa/doc.tpl?gdid=10632.

[GST+99] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, October 1999.

[HD06] Jeffrey Hasan and Mauricio Duran. *Expert Service-Oriented Architecture in C# 2005, Second Edition*. Apress, Berkely, CA, USA, 2006.

[JMD+05] Aleks Jakulin, Martin Možina, Janez Demšar, Ivan Bratko, and Blaž Zupan. Nomograms for visualizing support vector machines. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 108–117, New York, NY, USA, 2005. ACM.

[KEG] Kegg: Kyoto encyclopedia of genes and genomes. http://www.genome.jp/kegg/.

[KFH+06] G. Kandaswamy, L. Fang, Y. Huang, S. Shirasuna, S. Marru, and D. Gannon. Building web services for scientific grid applications. *IBM J. Res. Dev.*, 50(2/3):249–260, 2006.

[LZF02] Nada Lavrac, Filip Zelezný, and Peter A. Flach. Rsd: Relational subgroup discovery through first-order feature construction. In Stan Matwin and Claude Sammut, editors, *ILP*, volume 2583 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 2002.

[MOPT05] D. Maglott, J. Ostell, K. D. Pruitt, and T. Tatusova. Entrez gene: gene-centered information at ncbi. *Nucleic Acids Res*, 33(Database issue), January 2005.

[MyG] MyGrid. Taverna workbench 1.7. http://taverna.sourceforge.net/.

[TZTL06] Igor Trajkovski, Filip Zelezný, Jakub Tolar, and Nada Lavrac. Relational subgroup discovery for descriptive analysis of microarray data. In Michael R. Berthold, Robert C. Glen, and Ingrid Fischer, editors, *CompLife*, volume 4216 of *Lecture Notes in Computer Science*, pages 86–96. Springer, 2006.

[W3Ca] The World Wide Web Consortium W3C. Soap version 1.2 part 0: Primer (second edition). http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.

[W3Cb] The World Wide Web Consortium W3C. Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl.

[WL02] Mark D. Wilkinson and Matthew Links. Biomoby: An open source biological web services proposal. *Briefings in Bioinformatics*, 3(4):331–341, 2002.