On The Design of Knowledge Discovery Services Design Patterns and Their Application In A Use Case Implementation

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. As service-orientation becomes more and more popular in the computer science community, more research is done in applying service-oriented applications in specific fields, including knowledge discovery. In this paper we investigate how the service-oriented paradigm can benefit knowledge discovery, and how specific services and the knowledge discovery process as a whole should be designed. We propose a model for the design of a service-oriented knowledge discovery process, and provide guidelines for the types of functionalities it requires. We also provide a case design to show the application and benefits of the proposed model and design pattern in practise.

1 Introduction

Knowledge Discovery (KD) in data can be a very intensive process in terms of computation and data transport, but also because the construction of a KD process can be quite difficult and time-consuming. Over time, many have tried to find ways to improve the quality of KD processes, for example by making them faster, easier to construct and/or less data intensive. When new technologies appear, it is interesting to see how they can be applied to improve performance in these areas.

In this paper we take a look at the Service Orientation (SO) paradigm and in what ways it can benefit KD. The SO paradigm allows users to design applications (in this context we will see a KD process as an application) in terms of individual components than can be connected to each other through standardized communication. These components can be either locally or remotely available, and can be found through public lookup facilities. The potential of the SO paradigm can make KD processes easier, faster, more understandable.

The focus of the SO paradigm has primarily been on bussiness components and the construction of distributed corporate applications, but SO seems to become quite predominant in the scientific world as well. In this paper we explore the benefits and drawbacks of the SO paradigm in KD, mainly focussing on the design of a KD service and process. We support the design theories by a small case study of two components, which are combined together to form a KD process.

This paper is organized as follows: In Section 2 we discuss diverse work related to service-oriented knowledge discovery, work that we used to generate ideas for research and that we compared to our own ideas. In Section 3 we discuss SO technology, thereby discussing standards and potential advantages in a scientific context. In Section 4 we discuss the design of the case study, and provide guidelines in the design of KD processes and indiviual components. In Section 4 we discuss the case study trial runs and give a few statistics in terms of performance compared to other implementations. Finally, in Section 5 we draw some conclusions on the implications of the use of SO in KD, and discuss some future work.

2 Related Work

Over the last few years distributed KD has become increasingly more popular, which generated research incentives in diverse fields of technology. In [DBG+06] distributed data mining is proposed by using peer-to-peer networks. The authors sketch a high-level introduction to peer-to-peer data mining and give some pointers and requirements for methods, as well as a theoretical example. However, a comparison with other techniques lacks, as does technological depth or formal models. The authors of paper [AC06] focus on the area of text mining, and give criteria and requirements which need to be supported by good text mining tools. While they focus on their tool being embedded in other applications and address issues such as security and statelessness, they seem to only brush the topic of service orientation and web services as part of the tool, and present restrictions and not solutions. In [CZW+06] the authors take a view quite similar to ours, but use Business Process Execution Language for Web Services (BPEL4WS) to achieve stateful long running interactions, and focusses on data security through gaussian models, while our focus lies on the design principles of web services itself within service-oriented knowledge discovery. Finally, [GJF06] describes a framework in which web services are used for knowledge discovery in databases, and describes the framework indepth, as well as supported algorithms, but not the web service design and construction methodology, and thus serves as a useful complement to this paper.

3 Background

In this section we discuss the SO paradigm, and present its benefits with respect to KD processes. We also discuss current standards in the SO paradigm. These standards will be used in the design and implementation of the use case, which will be presented in Section 3 and Section 4.

The SO paradigm is expressed in a Service-Oriented Architecture (SOA)[Gro]. A SOA is a layered 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 user 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] An overview of SOA is shown in Figure 1

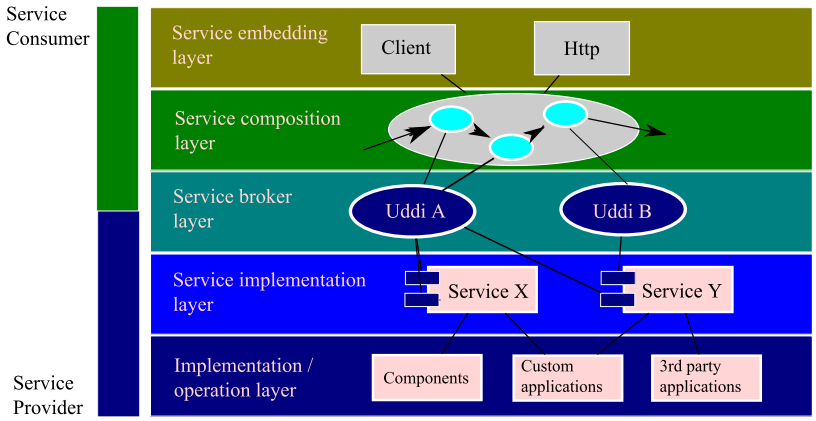

Fig. 1. SOA Layers

Without going into too much detail, there are a few key points in Figure 1 that we would like to draw attention to. First, notice how in the service provider layers service components can consist of not only custom software, but also existing solutions. This is possible because of the standardized messaging and interface formats that are part of the SOA specification. SOA incorporates a methodology that is called design by contract [Mey92]. In this methodology, implementation is decoupled from a program's interface, whereby and interface is 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 defining web service interfaces 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, what type of service it is, what kind of parameters it supports, and how the service interface is connected to the underlying implementation. We discuss WSDL in more detail in the next section.

Another area of interest are the service consumer layers. Notice that applications are no longer *constructed* but instead *composed* by putting together individual web services. This composability is partly the merit of the standardized interfaces, but also because SOA is message-oriented; communication between individual components proceeds through the use of uniformly defined messsages. A standard that is often used for web service message transport is the Simple Object Access Protocol (SOAP)[W3Ca], which is an XLM-based message format and transport protocol. Using both standardized ways of accessing and messaging makes an application decomposable into distinct, uniformly accessable units

of computation and processing, which allows for remote computing.

Finally, the last point of interest is the middle layer called the services layer. In this layer the interfaces of the web services are offered to the consumers who search for their underlying functionality. For a user it is impossible to know the location of each service, and similarly for a provider it is impossible to know the location of all its users. To meet both demands, the Universal Description Discovery and Integration (UDDI)[Dra] facility was created, which is a registry for web services offered by service providers containing all WSDL documents corresponding to interfaces of those services.

With all these protocols, standards and facilities in place, there are a plethora of advantages to be gained in the KD context. We summarize a few below:

- **Easier KD process setup**
  A scientist usually perceives a KD process as a workflow where data is continually modified in discrete computational steps. By using SOA and web service composition, the scientist's mental model is more closely approached. Process composition becomes even easier with the use of tools that offer a GUI like Taverna [MyG]. By offering the scientist an environment that matches his conceptual model of a KD process, the process becomes easier to understand and compose.

- **Easier KD process modification**
  Since web service interfaces are decoupled of their implementation, a user only needs to rely on the interface. If another webservice is available that adheres to the interface, that service could replace the forementioned one in a process without the need for any additional modification by the user.

- **Increased component availability**
  When a scientist searches for the solution of a specific step in his KD process, it might occur that an implementation is hard to find. With the UDDI in place, a scientist has a central location where specific solutions can be found. This reduces the time of process setup, and increases the chances of finding a solution.

- **Increased Performance**
  Since SOA and all related protocols are platform-independent technologies, each platform can potentially support it. This makes it easier for the service provider to use an implementation environment that is best suited for the web service, perhaps on specialized hardware which normally would not be available to the users. Moreover, if two services can be executed independently and are located on different physical domains, they can be executed in parallel.

4 Service-Oriented KD design

In this section we discuss the SO design model and patterns that we used to create the case study, and examine how these SO principles influence individual services and KD processes on a whole. First we discuss WSDL a bit more and explain how the standard influenced the design of the web service. After that, we discuss a design model for SO KD processes. Finally, we present guiding principles for individual KD service design, which were also used to design the use case.

4.1 WSDL and design implications

There are many views on the design of a KD process, ranging from a global view stating what functionality a service in a process has and what standard to use to design and implement it, to microdetails such as what message format to use. Since the use case was designed by using WSDL, this influences our further design of a web service and a KD process as a whole. In this paper we focus on the different operation types that are defined in WSDL, and its influence on the design of KD process and a KD service:

- **Request/Response**
  In this case, the client sends a message to the service, and the service sends a message to the client in response. This is the message equivalent of a function call.

- **Solicit/Response**
  This is the reverse case of the Request/Response type. The service sends a message to the client, and the client sends a message to the service in response. This is often used when a service needs to poll clients.

- **Client messenger**
  Here, the client sends a message but does not expect a message in return.

- **Server notification**
  Server notification is the exact opposite of the client messenger type. In this case, the service sends a notification to the client without expecting or waiting for an answer.

As we shall see in the remainder of this section, operation types have an inpact on the entire KD process, so selecting the right type of operation is important in order to obtain a process with optimal performance.

4.2 KD process design

A KD process can be seen as a workflow, whereby data flows from one unit of processing to another. Conceptually we try to map these unists of processing

to web services. How successful this can be done depends on the understanding of the process and the functional discreteness of individual steps. We see the design of a KD process as a three-dimensional challenge containing the Logical, Funtional and Relational views, that al influence each other. We propose the following KD process design model for SO that incorporates all these views, which is illustrated in Figure 2 and described below. By applying this model in the design of an SO KD process, a better understanding of the process is achieved, which leads to a better design, until both understanding and design are optimal.

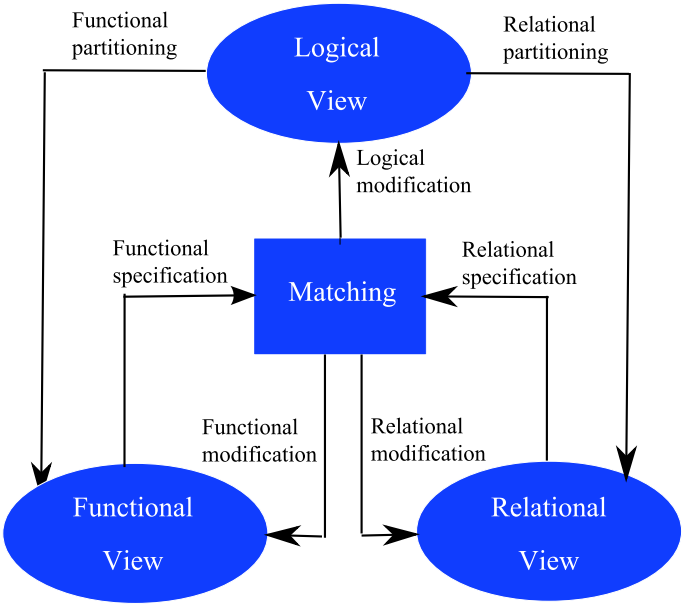

Fig. 2. KD process in SO

– **Logical view**
In this view, the entire KD process is being examined to identify all services and relations in the process. This logical view is not only guided by the designer's expertise, but also on the services already available, for example, services built earlier or services publicly available through a UDDI. Ideally all services fit together perfectly and are all available, but this is rarely the case. Therefore, choices have to be made if readily available services should be used, and how the unavailable process parts should be logically partitioned. Since different partitionings of a KD process yields different services and relations, the partitioning will affect the functionalities of each service as well as the relations among them.

– **Functional view**
  For each service identified all functionalities are recorded. These functinalities will serve as a guideline for interface design and operation type selection, and will determine the nature of the relations with other functionalities. In this stage similarities between services and dissimilarities within services can be uncovered on the basis of functionality, leading to a possible joining or splitting of services.

– **Relational view**
  In this aspect of design, relations should be identified for each service with other functionalities in other services. These relations should be annotated in two dimensions: direction and usage type. The direction indicates if messages will be flowing from a service or to a service, the usage type indicates if the relation is used only once, or continually until processing is done. Both dimensions will influence the functionality of a service, the operation type of the functionality's interface, and the content and format of the messages that will be transported. Similarities and dissimilarities in relations amongst services might also lead to a revision of the service partitioning.

– **Matching**
  This dimension is the feedback step of the model, and matches the outcome of all other phases to one another. It serves as a feedback phase for the design, and indicates if service partitionings, functionalities or relations should be modified or adapted in case of a mismatch.

4.3 KD service design

In this part we focus on the functionality design of a KD service, and how the design choices are expressed in the WSDL operation types.

As stated earlier, KD processes can be very time-consuming, especially when large data volumes are involved. This means that any error may result in the loss of a great amount of time. Therefore, individual KD services should be designed for interaction; A scientist should get regular feedback on the progress of the process, and should at all time be able to interact with the process.

We also mentioned that a KD process is often perceived as a workflow, a sequence of computational steps whereby data flows from one step to another. This does not mean, however, that one step should be completed in order for the next step to begin; the results that come from these actions sometimes can already be transferred to the next process phase without waiting for the service to finish processing all the data. To optimize performance as well, KD services functionality should be designed for streaming data where possible.

Having observed the facts stated above, we divided the functionalities of a KD service into three categories: Initialization, Feedback and Enactment. This classification forms a guideline for the design of a service's functionality using WSDL.

– **Initialization**
Procedures designed in this class are expected to handle a continuous stream of messages that initialize this part of the experiment. Client messagers are usually best suited for these functions, unless initialization requires critical feedback, in which case Request/Response should be used.

– **Feedback**
In this category methods need to be designed that provide feedback to the service client. Both Notification or Solicit/Response method types can be used here, depending on if the feedback is used purely for informative purposes or if it is used to steer an interactive experiment through client intervention. Feedback is often provided iteratively, sending messages whenever an event occurs.

– **Enactment**
This category combines the actual functionalities of the service with the feedback functionalities that report on the service's progress. Since an experiment usually is expected to return a result, a Request/Response type method is usually chosen. However, if one does not need to wait on this service in order to continue with other processing steps, a combined Client messager and Server notification procedure could be used to let the service run asynchronously. Note that enactment can be done both atomically or iteratively, as we will see in the next section.

5 Use Case

In this section we present the use case. For our case study we implemented a KD scenario described in [TZTL06]. In this scenario two classes of leukemia are compared with each other. The microarray dataset from Golub et al. [GST+99] is processed to identify differentially expressed genes per class, 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 combined to find subgroups. We extended this scenario to also include the Kyoto Encyclopedia of Genes and Genomes (KEGG)[KEG] ontology, and used a tree-like rule miner that induces rules on the basis of maximal subsets that satisfy the user-defined support constraints.

We designed the use case using the model discussed in the previous section and compared it to the original design. Both implementations are done in C-sharp that use a C++ .Net back-end, using WSDL as an interface definition language, and use SOAP as transport protocol. All services and algorithms are performed on Microsoft Windows XP using an Intel Centrino duo processor 1.66GHz, and 1GB of main memory.

5.1 Use case process design

– **Logical view**
In the use case we identified two different web services that are used together to provide one composite service. The first service is the **GeneSelector** service that is used to compute a t-scores for all genes in the microarray data, and place it either in the differential or non-differential collection. The second service is the **GeneRuleInducer** service which takes the two lists and produces rules that describe subsets of these lists that share the same terms in the GO and KEGG ontology, which are also provided in the rule.

– **Functional view**
Per service we identify functionalities divided in the three forementioned categories.

- **GeneSelector**
  Initilialization functionalities
  * *Probe mapper*: Each row of the microarray data is annotated per probe and not per gene, so probes need to be mapped to entrez gene id's.
  * *Class mapper*: Each column of the microarray data is annotated with a label and not a class, so labels need to be mapped to classes.
  * *Cutoff initializer*: Initialises the t-score cutoff value for genes.

  Feedback functionalities
  * *Probe Feedback*: When a probe does not match any gene, a message is sent to inform on this.
  * *Class Feedback*: When a label does not match a class, a message is sent.

  Enactment functionalities
  * *T-test calculator*: With help of both mappings, t-test value for a gene per class is computed and compared with the supplied cutoff.

- **GeneRuleInducer**
  Initilialization functionalities
  * *Gene loader*: Loads genes and their scores in the ontology tree-mining structure.
  * *Support constraint initializer*: Initilaizes the minimum and maximum support constraints.
  * *Ontology loader*: Loads the ontology in a tree-structure.
  * *Gene to ontology mapper*: Loads the data that maps gene id's to ontology keys.
  * *Gene interaction mapper*: Loads the data that specifies interaction between genes.

  Feedback functionalities

  * *Gene List feedback*: When something is wrong with one of the lists, the user is sent a message that specifies the problem and the performed action.
  * *Rule feedback*: Presents periodical feedback on the progress of the rule miner.

  Enactment functionalities
  * *Rule miner*: Using the internal tree-structure of the ontologies which are annoted with differential and non-differential genes, rules are uncovered that satisfy the minimal differential support constraint and the maximal non-differential support constraint.

– **Relational view**
  Here we specify relations and if they are iterative or not. Iterative relations are denoted with *.
  Client to GeneSelector relations
  - *Probe map input*: Message containing probe and gene id's. Only needs to be supplied once.
  - *Class map input*: Message containing classes and labels. Only needs to be supplied once as well.
  - *Cutoff input*: Message containing the user-defined t-score cutoff.
  - *Data input**: Message containing a probe identifier and expressions per label.

  GeneSelector to Client relations
  - *Probe map feedback output**: Message that returns a problem with the probe mapping.
  - *Class map feedback output**: Message that returns a problem with the class mapping.

  GeneSelector to GeneRuleMiner
  - *Data return output**: Message that returns the score of the gene and if it's in the differential set or not.

  Client to GeneRuleMiner
  - *Differential support input*: Message that supplies the cutoff for the minimal number of differential genes a rule has to support.
  - *Non-differential support input*: Message that supplies the cutoff for the maximal number of non-differential genes a rule may to support.

  - *Enactment input*: Message that enacts the mining process.

  GeneRuleMiner to Client
  - *Gene list feedback output**: Message specifying feedback if anything goes wrong loading the specified lists.

- *Rule miner output*: Message that returns rules uncovered by the algortihm.

A complete overview of service connectivity and data flow is presented in Figure 3. Note that only those functionalities that require interaction with the user or another service are displayed.

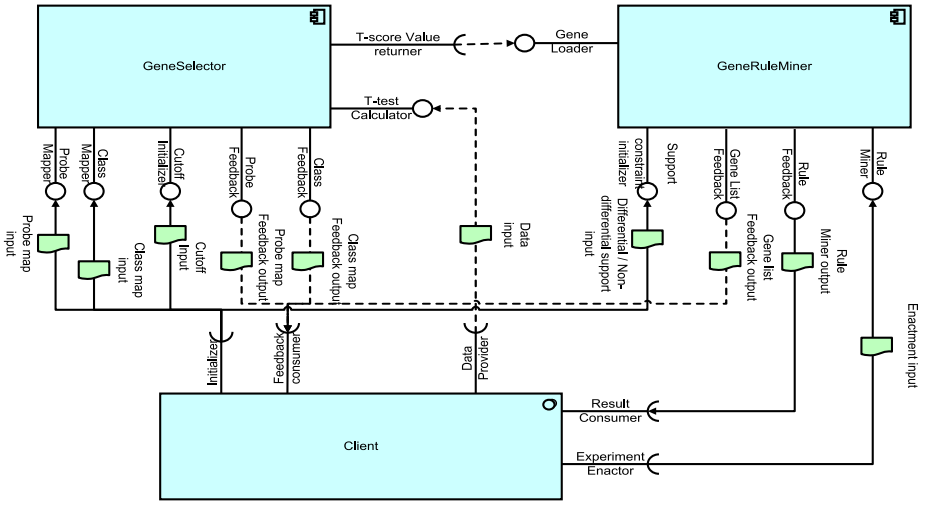

Fig. 3. SO KD use case design

5.2 Use case process comparison

The original process was divided in the same service partitioning as the one that our model yielded, but processing of individual services was done one by one, and results did not transfer before processing was completed. Furthermore, in the original implementation feedback was not supplied upon occurence of the event, but as a return value after processing, which is a less interactive way. A complete list of differences per service are listed in Table 1 and Table 2

Category	Original process	Use case process
Service feedback	as return values	iterative on occurence of event
Service initialization	Supplied as a whole	Iterative data supply
Service processing	all	per element
Service outputs	at end of processing	continuous outputting per element

Table 1. Design differences in GeneSelector service

Category	Original process	Use case process
Service feedback	as return values	iterative on occurence of event
Service initialization	Supplied as a whole	Iterative data supply
Service processing	all	all
Service outputs	at end of processing	at end of processing

Table 2. Design differences in GeneRuleMiner service

Finally, we took some benchmarks for the performance of the original process and the re-designed use case. As input for the selector we took respectively cutoffs of t-score 15, 10 and 8. For the GeneRuleMiner, we took supports of minimally 10% differential genes and maximally 5% non-differential genes. Results of the original process are displayed in Table 3, and those of the re-designed use case are in Table 4. Note that the measurements of each phase in the table indicate after how much time since the *process* started this phase ended. All measurements are averages over 50 consecutive runs, and are min milliseconds. Since in the GeneRuleMiner processing is no speedup to be gained due to the return of all results at once, we only show the benchmarks of the phases preceeding the GeneRuleMiner processing phase.

Phase	Cutoff 15	Cutoff 10	Cutoff 8
GeneSelector initialization	74	69	73
GeneSelector processing	1331	1291	1328
GeneRuleMiner initialization	3388	7122	13318

Table 3. Benchmarks of the original process

Phase	Cutoff 15	Cutoff 10	Cutoff 8
GeneSelector initialization	73	69	74
GeneSelector processing	1260	1228	1257
GeneRuleMiner initialization	2139	5841	11998

Table 4. Benchmarks of the re-designed use case

To make the difference between the original process and the re-designed use case more clear, consider Figure 4. Here, the cutoff 15 scenario was worked out more thoroughly. The top part displays how the original process parts consecutively get processed. The bottom part shows how the re-designed process part itera-

tively get processed in parallel where possible.

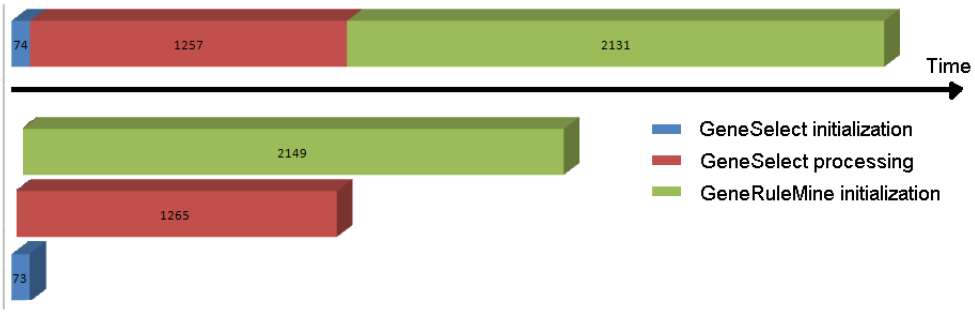

Fig. 4. Benchmark comparison of t-score 15 cutoff process

6 Conclusions and Future Work

In this paper we discussed the design of KD processes in an SO environment. We discussed the SO technology and related standards, and showed how KD processes can benefit from SO technology; It makes process setup and modification easier, increases component availability, and can potentially have a positive inpact on performance.

By designing a SO KD process workflow using a design model that combines logical, functional and relational views, a better understanding of a KD process can be gained iteratively due to the matching and mismatching of entities in these views, whereby each iteration yields a better SO KD process design and a closer match of relations, services and functionality. An important factor that influences the partioning of services are the services already available, thereby promoting software reuse.

When designing individual services in KD, interaction and feedback are importants aspects to keep in mind. Interaction and regular feedback are important for the scientist to steer the KD process in a correct way, for KD processes are often time-consuming and thus any process incorrectly set up could possibly result in a considerable loss of time. Another important aspect is performance through paralellization. Since web services can be distributed across different logical or physical platforms, their execution could possibly proceed in a parallel fashion. To support paralellism, streaming data is preferred over monolithic data transport where possible. By combining these aspects and the functionality types in WSDL, we created a guideline for the design of a KD service's functionality that is otimized for streaming data where possible and incorporates the need for feedback.

To illustrate the merits of SO and our developped design model and guidelines, we implemented a use case according to our model, and compared it to a web service implementation using monolithical data transfer and periodical step-by-step execution. In some cases, processing times for the initialization of a process were reduced up to 37%, and with 22% on average.

The design principles stated in this paper are but a minor step to incorporating SO technology in the field of KD. However, by assuring that the design of a KD process and individual services is optimized for feedback and paralellism, a researcher can enact a process and conclude it successfully with minimal error and maximal performance. For further research we would need to study more use cases to ensure the research principles have maximum support in the KD scientific field. Furthermore, this design needs to be supported by grapical workflow tools that support iterative relationships instead of just monolithical data transport.

References

[AC06] Juan Jos Garca Adeva and Rafael A. Calvo. Mining text with pimiento. *IEEE Internet Computing*, 10(4):27–35, 2006.

[CZW+06] William K. Cheung, Xiao-Feng Zhang, Ho-Fai Wong, Jiming Liu, Zong-Wei Luo, and Frank C.H. Tong. Service-oriented distributed data mining. *IEEE Internet Computing*, 10(4):44–54, 2006.

[DBG+06] Souptik Datta, Kanishka Bhaduri, Chris Giannella, Ran Wolff, and Hillol Kargupta. Distributed data mining in peer-to-peer networks. *IEEE Internet Computing*, 10(4):18–26, 2006.

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

[GJF06] Dorgival Guedes, Wagner Meira Jr., and Renato Ferreira. Anteater: A service-oriented architecture for high-performance data mining. *IEEE Internet Computing*, 10(4):36–43, 2006.

[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.

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

[Mey92] Bertrand Meyer. Applying "design by contract". *IEEE Computer*, 25(10):40–51, 1992.

[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 Železný, 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.