Technical University of Vienna
Information Systems Institute
Distributed Systems Group

Web Services Interaction Mining

Schahram Dustdar, Robert Gombotz
and Karim Baïna
dustdar@infosys.tuwien.ac.at
e9906767@student.tuwien.ac.at
baina@ensias.ma

TUV-1841-2004-16 September 6, 2004

As Web services play a more and more important role in information technology, service-oriented systems can also be expected to grow larger in complexity. Such large systems demand for tools that allow for analyzing and monitoring of service-oriented systems in use. Our work attempts to apply data mining and process mining to Web services and their interactions in order to provide a means to analyze interactions between Web service consumer and provider. Firstly, we clarify the term of Web Services Interaction Mining (WSIM) and present three levels of abstraction on which WSIM could be performed: Web service operations, interactions and workflows. Then we outline some of the problems Web Services Interaction Mining (WSIM) could solve. Thirdly, we evaluate how WSIM could be done on these three levels and present a specification for Web service log records. We also discuss how these log records could be obtained using existing tools. We conclude the paper with suggestions on how to develop Web services which can be mined on all three levels of abstraction.

Keywords: Web service interactions, Web service logging, Web service mining

Argentinierstr. 8/184-1
A-1040 Vienna, Austria
phone: +43 1 58801-18402
fax: +43 1 58801-18491
URL: http://www.infosys.tuwien.ac.at/

Web Services Interaction Mining

Schahram Dustdar [1], Robert Gombotz [1], Karim Baïna [2]

[1] *Distributed Systems Group, Vienna University of Technology, Austria,*
{dustdar@infosys.tuwien.ac.at, e9906767@student.tuwien.ac.at}

[2] *ENSIAS, Université Mohammed V - Souissi, B.P. 713 Agdal Rabat, Morocco,*
baina@ensias.ma

Abstract. As Web services play a more and more important role in information technology, service-oriented systems can also be expected to grow larger in complexity. Such large systems demand for tools that allow for analyzing and monitoring of service-oriented systems in use. Our work attempts to apply data mining and process mining to Web services and their interactions in order to provide a means to analyze interactions between Web service consumer and provider. Firstly, we clarify the term of Web Services Interaction Mining (WSIM) and present three levels of abstraction on which WSIM could be performed: Web service operations, interactions and workflows. Then we outline some of the problems Web Services Interaction Mining (WSIM) could solve. Thirdly, we evaluate how WSIM could be done on these three levels and present a specification for Web service log records. We also discuss how these log records could be obtained using existing tools. We conclude the paper with suggestions on how to develop Web services which can be mined on all three levels of abstraction.

Keywords: Web service interactions, Web service logging, Web service mining

1 Introduction

Currently much research effort is going into the development of Web service technologies. The development of Web services (WS) themselves has already changed our view on system and application integration. It seems that we might have found a standard and widely accepted means of integrating formerly independent systems across the Internet. In the not so distant future, we might be looking at systems or applications that integrate and make use of numerous single Web services provided by just as many different suppliers. However, with the essential building block, the actual function/method call, now at hand, other challenges are faced. We begin this paper with a brief definition of key terms.

One of the challenges of making Web services more usable to both providers and customers is the coordination of interactions between WS. Consider a Web service that allows its users to order a product. It provides the operations `requestQuote-quantity)`, `orderProduct(quantity)` and `makePayment(quantity)`. It is obvious that a client application should always invoke these operations in a certain

sequence. The availability of the product in a given quantity should be verified before the actual order is placed and a payment should not be made before the ordering process has been completed successfully. Interactions of this kind are also called *conversations.* In a more complex scenario a Web service might not allow only a single possible conversation but rather a set of them. Therefore, a WS should have means of describing all allowable conversations a requestor may have with it. One attempt to standardize these descriptions is the *Web Service Conversation Language* (WSCL) [1].

The *Business Process Execution Language* (BPEL) deals with Web service composition and attempts to solve the problem of composing a number of Web services into business processes. The idea is to describe the aggregation of stand-alone Web services into a (larger-scale) workflow in BPEL, an XML-compliant standard, and to have a BPEL engine monitor the correct execution of the process. Although still emerging, BPEL appears to become a widely accepted standard for *Web Service Orchestration*, also because it is backed by many industry leaders. There are already many implementations of the BPEL specification. A BPEL engine should not only allow to define workflows using Business Process Execution Language, but also to monitor the execution of that workflow. It does so by logging activities, providing a (graphical) representation of the workflow and the state of execution of an instance of that workflow as well as informing the human user of possible exceptions or undesired or unexpected behavior in the business processes execution. Web services Interaction Mining (WSIM) can be seen as an extension to the BPEL approach, as our contribution is to allow also for the mining and monitoring of WS provided by a third party. A BPEL engine only allows the monitoring of WS that the user has ownership of.

Another essential term in WSIM is *mining*. Basically, mining describes the process of discovering knowledge in large amounts of data. An example we like to give when describing the goals of *data mining* is that of a supermarket chain "logging" all purchases made by its customers. After some time, the manager wants to increase sales by trying to identify which products are often bought in combination. The store's database can then be searched for patterns in customers' buying behavior, i.e., the data is *mined* for such patterns. Through mining, management might learn that its customers often buy a bottle of red wine and dinner candles in one visit to the store. Trying to optimize sales, the management could react by placing candles and red wine in shelves next to each other or even by offering – and, of course, advertising - a "romantic candle light dinner package" consisting of the two.

To go further, mining is also used to identify more complex patterns. *Workflow mining* or *process mining* describes the attempt to find workflow patterns in a given set of log data. Consider a number of entities or agents, each performing a single, independent task. The sum of these tasks might be a workflow or business process. Consider also that every entity involved in the process provides event-based data, which is logged. An event occurs, when an activity is started and when it is completed. In a simple case, the data provided should consist of the identifier of the *activity* that is being performed, the *event type*, which can be "start" or "completed", and a time stamp. After a sufficient amount of log data has been collected one can mine this logging information for patterns and thereby find that e.g. an activity A is always performed before activity B. Activity B in turn is always performed before

activity C, and sometimes, but not always before D. Activities B and D are also always completed before the execution of C starts. From this information one could derive the simple workflow model shown in Figure 1.

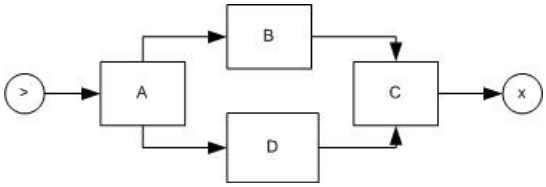

Figure 1. Simple workflow model

In more complex workflows the logged information about events must be slightly richer, but the above example should suffice for describing the idea behind process mining. An overview of research in the field of process mining is given in [2]. An algorithm for mining exact models of concurrent workflows as well as an implementation of that algorithm is presented in [3].

1.1 Fundamentals of Web Services Interaction Mining

We make use of the developments in the fields of *data mining* and *process mining* and apply them to Web services. We believe that this can be a vital contribution to the world of Web services in general as well as to future suppliers and users of Web services. We currently develop our Web Services Interaction Mining approach with regards to three levels of abstraction that represent three complementary Web services "views". Figure 2 depicts a stack of views on Web services. As one goes from the top to the bottom, the level of abstraction falls and we are looking at things in higher detail.

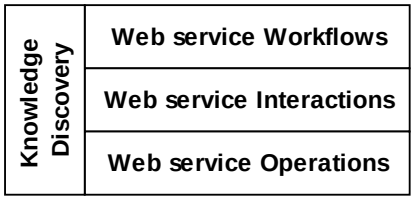

Figure 2. Levels of Abstraction

On the *Web service Operation level*, we want to examine only one single Web service and its internal behavior. We will not concern ourselves with a Web service's interactions with other Web services or applications, but rather focus on its functionality as if it were alone in the world. Furthermore, the focus might even be on just one operation of the Web service. However, we also want to examine the Web

service as a whole. Relating this to mining, a given log output of the Web service shall be analyzed to gain information about its behavior.

On the *Web services Interaction level*, we again direct our attention to one Web service, but also want to take into account its "direct neighbors". The term "direct neighbors" refers to other Web services that the examined WS interacts with. Such interactions may be explicit, i.e. defined in a Web Service Composition language, or implicit, i.e. calls to other WS from within the Web service's implementation. Explicit interactions are also said to be declarative, implicit interactions are also called programmatic. On this level we again want to mine log data for further information about the Web service's interactions with others. This information could reveal interesting facts about a Web service's interaction partners, such as critical dependencies.

The highest level of abstraction is the *Web service Workflow level*. As the name suggests, the focus now is on large-scale interactions and collaborations of Web services which together form an entire workflow. Technically, the number of Web services in a workflow is unlimited. On this level, we want to examine the execution of the entire process. Here we will be able to benefit from the results and findings of researchers in the field of process mining.

Even though our current focus is on mining, it must be stressed that once a mining effort is completed (with respect to its primary goals of building a model of a process), it should be continued and serve the purpose of *monitoring*. Therefore, future log data should constantly be analyzed and compared to the model established in the initial mining process. One might find exceptions in future behavior of the examined system, or – in an even more undesirable case – find that the initial model was built on false assumptions, possibly because of insufficient log data.

The remainder of this paper is structured as follows. In Section 2, we present a motivating case study which we will refer to throughout this paper. In Section 3, we present our approaches to WSIM in detail. In Section 4, we discuss possible implementations of these approaches. Section 5 presents some related work before we conclude with Section 6.

2 A Case Study – Managing a film crew

In this Section we present a real-life analogy to a service-oriented system. It is supposed to serve as an example that we can refer to throughout this paper in order to present our arguments more clearly to the reader.

Consider the process of creating a motion picture. This task involves the participation and cooperation of a large number of people, such as actors, camera crews, make-up artists, stuntmen, etc. Many of these people can be considered members of stand-alone teams or crews, e.g., the stunt crew or the make-up artists crew. Above all stands the director who manages all teams and actors to be in the right place at the right time.

The aggregation of individuals into teams allows the director to abstract from the complexity he is faced with. If he needs the stunt crew to be available to shoot a

certain scene, he will contact the head of the stunt crew, explain to him/her the details of the task and can then rely on all stunt crew members to do their job as desired.

Consider also, that the above mentioned crews may very well be long-term teams, who work together for much longer than what is the time span of the creation of a particular film. They participate in various movie projects over time.

Drawing analogies from the above described real-life situation to a service-oriented system is pretty straightforward. The individual, stand-alone, long-lived teams can be seen as Web services. For example, the "Stunt People Service" might provide operations such as `negotiateCooperation()`, `performMartialArts-Stunt()`, `performCarCrashStunt()` and `performWildAnimalStunt()`. These services are somewhat independent from the actual project they are currently used by. Of course, every scene in a film is unique, but the general service of `performCar-CrashStunt()` is provided independently of the particular movie.

Critics might say that the above real-life example is not a perfect analogy to how Web services work and that only an interface is provided which allows a client to request people who will then perform the stunt. We argue, however, that this detail is insignificant to the point we want to make. Imagine, a director only passing a detailed description of the desired stunt act to the stunt crew (i.e., a SOAP message) and the crew returning a video tape holding the completed scene. In this case, the stunt crew can serve as a perfect metaphor of a Web service.

The creation of a film is a *workflow*, or *business process*, within the service-oriented system. It combines the functionalities of numerous Web services in a somewhat loosely-coupled way. The director (producers) is the owner of the workflow, yet not the owner of each involved crew. The cooperation between the workflow owners and the crews is based on contract agreements, as it can (and probably will) be the case between the provider of a Web service and its clients.

The entire service-oriented environment can be either a large film studio, that itself employs make-up and stunt teams and provides them to its directors as "Web" services. Thinking in larger and more loosely-coupled terms, the service-oriented environment could also be the ecosystem of Hollywood. Within the Hollywood environment, numerous Web services are deployed and can be called by anyone who wants to make use of the functionality they provide. A producer, who plans to perform the business process of making a movie, may contact these Web services, lead negotiations, decide on which services actually fit his purpose, make contracts with the services and then trigger the execution of his workflow.

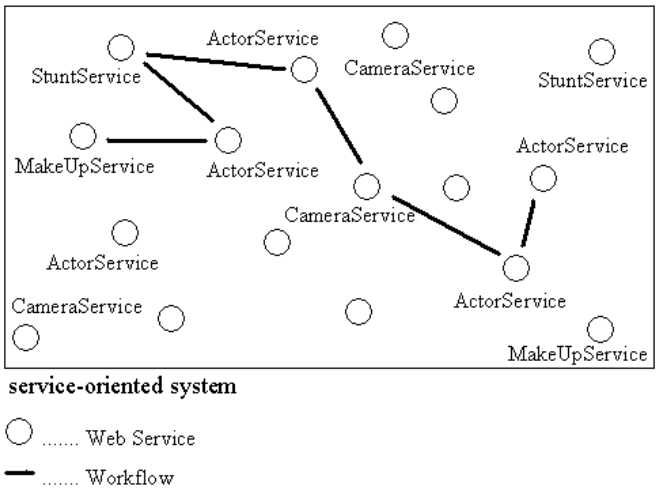

Figure 3. Making a film in a service-oriented system

Figure 3 depicts the scenario we presented as a case study. The rectangular box represents the service-oriented environment. The circles are single, stand-alone Web services. Note, that many services exist more than once. This is definitely the case in the motion picture industry – there are a number of stunt crews offering their services in the Hollywood area. However, each of them might provide slightly different, maybe even unique services. It is up to the requestor to decide which service provider can fulfill his needs best. The line represents the workflow which combines the functionalities of a number of services into a higher-level task of making a film. To be precise, the line only connects the Web services that are part of the workflow. The actual process will be a complex series of calls to the respective services which can happen sequentially, concurrently or in loops. Obviously, the camera crew will be needed throughout the entire duration of the filming, the make-up artists are needed usually in the mornings, but also during and between the taking of scenes and the stunt crew could be necessary only during a certain phase in the process. The schedule of the project shows when and where services are needed and can be seen as a detailed specification of an instance of a workflow.

We have now presented a motivating case study which we will refer to throughout the remainder of this paper. We described the making of a motion picture as an analogy to a workflow in a service-oriented system.

3 Approaches to Web Services Interaction Mining

In this Section we present our approaches on how the idea of Web Services Interaction Mining could be put into practice. First, we elaborate on the three levels of

abstraction we have identified in Section 1 along with references to the case study given in Section 2.

The lowest level of abstraction, or the most detailed view on a Web service, is the *Web service Operation level*. Here we focus on a single Web service's internal functionality or even only a single operation which is part of a Web service. Consider the case study of a film crew that was given in Section 2. When examining a film crew on the operation level, we direct our attention to one and only one crew (or Web service), e.g., a group of makeup artists. The crew offers a number of services (or operations). We assume there is one person who manages the crew and arranges contracts with outside clients. In a WS that could be an operation called `negotiateCooperation()` which should always be called by clients in order to initiate a cooperation. Another operation might be `requestMakupArtists()` which delivers a number of makeup artists who actually put make-up on actors. Other operations could be `requestHairDressers()`, `requestHorrorMakupArtists()` and maybe `requestChildrenMakupArtists()`. Together, these operations form the Web service `MakeupService`.

We now move up one layer in our stack of views on Web services: the *Web services Interaction level*. The focus is on one Web service and its interactions with other WS, or as we called them in Section 1, its *direct neighbors*. These are all Web services, which the focused WS interacts with. Such interactions occur in various ways.

There are four basic types (patterns) of WS interactions. They are *one-way*, *request-response*, *solicit-response* and *notification*. One-way and notification operations involve only one message that is sent between the interacting Web services. The requestor sends a message to the provided WS and does not expect a response. From the message sender's point of view such an interaction is considered to be of type *one-way*. From the message receiver's point of view it is considered a *notification* operation. Therefore, whether a transaction type is one-way or notification depends on the perspective, i.e., which WS the focus is on. If WS A calls WS B in a one-way manner, we say that WS A has knowledge of WS B, but not vice versa. An interaction of that type might be accomplished with a simple Logging WS within a service-oriented system that only waits to receive messages which contain logging information from other WS. Therefore, WS B is known to others, yet, it has no means of contacting them in return. Again, a service definitely has knowledge of a WS it sends messages to, but does not have to know about those who it receives messages from.

The two remaining types of interactions, request-response and solicit-response, involve the exchange of *two* messages between the interacting entities. A message is sent by the initiator of the interaction and the called service replies with a response message. Just as in the above cases, the two types of interactions vary only by who is the initiator of the interaction. When a service provides a two-way interaction to the outside passively, we call the operation request-response. If the situation is reversed, meaning that the service itself initiates a call to another WS and expects a response, the interaction is called solicit-response. A WS A needs to have knowledge of a WS B it wants to make operation calls to. In the other case of WS A only providing operations to the outside, it does not need to know about WS making calls to it.

Going back to our WS `MakeupService` and assuming this is the WS we want to analyze on the WS Interaction level, the four different kinds of interactions could be the following:

If the crew decides to lay off one of its employees, it might send a report to an agency stating it has fired person X. The crew does not expect a reply from that agency; it simply interacts with it in a one-way manner. In a different case, the crew might periodically receive magazines and product samples from a "Beauty Products Promotion"-WS. This is an example of a notification interaction. In this scenario, the make-up artists may or may not have knowledge of the service they are receiving notifications from. They could have subscribed to be notified of certain news or they could just be receiving them without having ever asked for them.

The ways in which a crew of make-up artists can interact with others in a two-way manner could be the following. The basic services the crew offers to its clients are examples of *request-response* interactions. The operations can be called from outside the service and a response will be given back to the requesting entity. An example of a *solicit-response* interaction is the crew calling a "Supplier"-Web service to order and receive beauty products which are needed on the set. Here again, the crew must have some means of contacting the Supplier-service and therefore must have knowledge of it. In other cases, the film crew may not know about other services it is used by. The crew might at any given time be contacted by a producer it has never heard of, who calls the operation `negotiateCooperation()` on the crew. If the crew is unavailable during the requested time, it replies in the negative and might not even keep a record of it. If we think again of Web services in a service-oriented system it will depend highly on the purpose of the Web service whether it has a persistent store of all calls to it or not. A company providing a WS "Get today's funny cartoon" would probably not log the calls to it. A "more important" WS within a company's information system which allows callers to query a database might very well decide to log all calls to it, both successful and unsuccessful. However, from a mining point of view, we have to request for *all* calls to be logged in order to be able to discover all entities interacting with a given WS. A more formal description of the four basic types of interactions is given in Section 3.2.

Having discussed the four basic types of interactions between WS with respect to the role they play in WSIM, we must also take a look at another categorization that can be applied to collaborations of WS. We can categorize interactions by whether they are *declarative* or *programmatic*. As we mentioned in the introduction of this paper, WS can be orchestrated using BPEL to declare the collaborations. A BPEL engine then guards the execution of these collaborations. Since BPEL is used to describe an entire workflow comprised of numerous WS, it somewhat exceeds the level of abstraction of the Web services Interactions level. We want to focus on one Web service and its direct neighbors. Therefore, we are not (yet) concerned with large-scale workflows but rather a small part of it.

We now take the last step up the stack of views and deal with mining for entire business processes, or *workflows*. On this level we might be looking at numerous Web services which in one way or another collaborate in the execution of a larger-scale task, i.e. a business process. Referring to our case study, we now look at the production of a whole movie. The completed motion picture is the result of inputs of numerous groups. These groups, or crews, can be seen as Web services; the making of

a movie is a business process. Since it is only a matter of time until the relatively new technology of WS will be used to perform more and more complex tasks and WS will be combined into entire workflows, the management of these workflows will be a major concern in the future. Our approach to provide mining of Web services on a workflow level can, therefore, be an important contribution to the field.

In the following subsections we go into further detail. Specifically, we will take a look at each level of abstraction. For each level, we present a - normative - format of log entries, or a *log specification*, as well as simple examples of log records. By normative we mean log formats the way we would like them to be. The focus is not yet on how to keep these logs or on who should provide them. This issue will be discussed in the next Section thereafter.

3.1 The Web service Operations level

On the Web service operations level we are concerned with a single Web service's internal behavior and its operations. We do not yet consider interactions with other WS. Take, for example, the make-up artists crew of Section 2. As stated before, the Web service provides the operations `negotiateCooperation()`, `requestMake-upStaff()`, `requestHorrorMakeupStaff()` and `requestHairDressers()`. Recall also that all activities performed by the crew are recorded, i.e. they are logged. A possible log kept by the operation `requestHairDressers()` of the "`Make-upService`" might therefore be :

```
Start – Ben – doingHair – 2003:02:03:06:45
Start – Lisa – doingHair – 2003:02:03:06:53
Complete – Ben – doingHair – 2003:02:03:07:33
Start – Ben  – doingHair – 2003:02:03:08:02
Complete – Laura – preparingWig – 2003:02:03:15:57
```

Listing 1. Log-information of `requestHairDressers()` in LOG-requestHairDressers.txt

Each entry in the above listing example consists of an event type specifying whether the record marks the start or the completion of an activity and identifier of the person performing the task (may be omitted if that information is irrelevant), a name of the activity, e.g. doingHair, and a timestamp. The information is stored in a file called "LOG-requestHairDressers.txt". Similar log-files exist for other operations. A formal model of a log entry is depicted in Figure 4.

On the provided data we can now perform data mining, which could reveal interesting information, such as extent of utilization of an operation or an individual person, idle-times of an operation, or bottlenecks within the operation. It is also possible that even within an operation a small workflow is executed. If so, workflow mining could be applied to the data in order to extract a model of the performed workflow.

We also assume there is a log-file provided by the operation `negotiate-Cooperation()`, which holds entries like the following:

```
Start – acceptOffer – customer05 – 2003:01:02:15:45
Complete – acceptOffer – customer05 – 2003:01:02:15:46
Complete – evaluateOffer – customer09 – 2003:01:07:10:23
```

```
Start - rejectOffer - customer09 - 2003:01:07:14:03
Start - executeCooperation - customer05 - 2003:01:28:06:00
Complete - executeCooperation - customer05 - 2003:07:27:18:00
```

Listing 2. Log-information of `negotiateCooperation()` in LOG-negotiateCooperation.txt

In the above example, we clearly deal with a workflow within an operation. Note that in this example we added an identifier to the log record that specifies which workflow instance the respective activity belongs to, e.g. customer09. This additional information is needed in order to be able to distinguish between workflow instances which should all follow a certain workflow model. The workflow model can then be discovered by examining all workflow instances and building a model which describes all these instances. This would be an example of process mining within an operation of a WS. However, data mining can also be applied to the above log. This could reveal information about a certain client being a very valuable customer or another one often posting offers that have to be rejected.

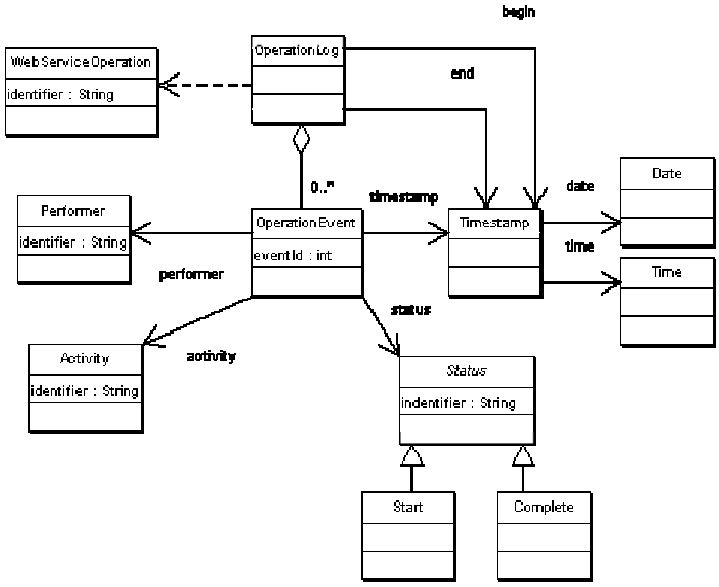

Figure 4. Class diagram of an operation log entry

To raise the level of abstraction, we now analyze the Web service as a whole. To do so, we may use all information provided by the operations in their respective log files. If we expect workflows to exist within the WS but across operations, we could apply process mining to the WS' logs. To do so, we would first generate a new log file "`LOG.temp-xyz-MakeupService.txt`". Into that file, we would merge information included in the original log files. For example, in the information provided by the operation `negotiateCooperation()` we would look for the activity

`executeCooperation` and then find the log-entries marking the start and the completion of `executeCooperation` with respect to a certain workflow instance, e.g., customer05. These entries could provide us with the information of when the task was started and when it ended through the included timestamps. After that, we would go through the logs of the other operations and extract all entries that were made within the given time span of the task. These records are extended by adding the identifier of the operation they were recorded by. The resulting log file would include all activities performed by any operation within a given task. This log could then be processed and mined for possible workflows.

An alternative approach would be to process the individual log files, e.g., "`LOG-requestHairDressers.txt`", first and use the extracted information when mining for larger scale workflows. For example, "`LOG-requestHairDressers.txt`" is first processed in a way such that for every activity the earliest start-record and the latest complete-record for every day are looked up. The timestamps of these records are used to generate new, higher-level log entries of the kind [Start | Complete] – activity – timestamp. Using these new log records, it might be discovered that a given activity within the operation is always performed before another activity. This is another example of doing workflow mining on a given activity. The resulting temporary logs of the individual operations could then be used to further analyze the Web service as a whole. The advantage of processing individual logs into temporary higher-level (or higher abstraction) logs is that the amount of data on even higher levels of abstraction can be dramatically reduced and therefore be analyzed in a smaller amount of time. Also, information in low level log records might be not needed, possibly even disturbing on higher levels of abstraction because the focus is on different aspects of the Web service's functionality and behavior. Note, though, that an occurrence of e.g. the operation `requestHairDressers()` in a higher level workflow model can always be replaced by the workflow model of that operation itself, which was discovered when the operation was mined on a lower level of abstraction. Therefore, a user is able to "browse" within the resulting high-level workflow model and view elements of that model on lower levels of abstraction.

We have now presented examples of how mining can be applied on the *operations level*. The approaches include techniques of both data and process mining. The strength of our suggestions is the variable level of abstraction when mining a Web service. However, a clear limitation of our approach is the obvious ad-hoc-ness of mining efforts on the Operational level. The questions mining can answer about a WS are highly dependent on the specific implementation and functionality of that WS. A more general and widely applicable solution is yet to be found.

3.2 The Web Service Interactions level

On the Web services interactions level, we focus on a single Web service and its relationships to its direct neighbors, i.e., other WS that the examined WS interacts with. Other than on the operations level, our attention now is directed primarily on these interactions rather than on functional aspects of the WS.

There are two ways in which interactions between WS can be implemented. They can be either declarative or programmatic. Declarative interactions are described in a

specific (possibly standardized) language, such as BPEL, and then executed by a WS orchestration engine. Web Service Composition may also be accomplished using declarative transactions. From a mining point of view, declarative transactions are somewhat easy to discover, given they are declared in a standardized language that can be analyzed by a mining engine. The mining engine may traverse the deployment descriptor for entries concerning a certain examined WS.

The other possibility of implementing an interaction between WS is inside the program code, i.e., the transaction is programmatic. Interactions of this kind can only be discovered by a mining engine if they are logged carefully and completely. In order to be able to also distinguish between the four basic types of transactions we suggest a standardized format for log entries. We want an entry to consist of an identifier that marks the entry as an interaction so as to distinguish it from "regular" entries like those shown in the previous subsection. We refer to this identifier as *int*. The next portion of the log entry should be one of a set of standardized identifiers that shows details of what type of interaction it is. Furthermore, an activity, an identifier of the other participating WS, and a timestamp are needed. A log entry made by the `MakeupService` could therefore be "`int - oneWay - reportLayoff - someAgencyWS - 2003:05:22:10:00`". Note, that the entries concerning interactions do not require an extra log-file. The entries can be identified based on their structure which differs from that of other entries. The fact that the log entry is made into the files used in the previous subsection also gives us additional information about *where* within a WS, i.e., in which operation, an interaction with another WS takes place.

The result of the mining efforts on the Web services Interactions level should be an interaction graph for each WS and its direct neighbors. The graph should contain information about what type of interaction links the WS to a specific neighbor and from where in the WS the call is coming from. From the activity-part of the above described log-entry we can also derive a name for the interaction as is common in many modeling languages. Note, that a WS might have more than one interaction with a direct neighbor.

Log specification

In order to be able to mine for programmatic interactions between WS we need those interactions to be logged. The four basic types of interactions between Web services are *one-way*, *request-response*, *solicit-response* and *notification*. The log entries that are made by a WS should reflect these different kinds of interactions so they can be recognized and displayed appropriately. We have established a minimal, but sufficient set of log records that each corresponds to a different type of interaction. We start out with the case of two-way operations.

Two way operations are either of type request-response or solicit-response. The only difference between request-response and solicit-response is the perspective from which we look at the operation. Therefore, we can use the same type of log entry for both of them. During the mining it can be determined out of context which type of operation we are looking at.

Figure 5. Two-way interaction

Figure 5 depicts a two-way (synchronous) interaction between WS A and WS B. From WS A's point of view the operation is solicit-response. WS B, however, is providing a request-response operation to WS A. This detail is insignificant to our logging specification and only becomes important once we mine the logs for interactions. At that point, it will depend on which WS we focus on whether the interaction is identified as request-response or solicit-response.

As we stated before, log entries concerning interactions must contain the following information: an identifier marking it as an interaction entry, an identifier of whether the interaction is one-way (asynchronous) or two-way (synchronous), an identifier of where within the interaction we are, an identifier concerning the interaction partner, an identifier of the activity that is being performed, and a timestamp. The identifier marking the type of log entry could be simply "int", which is short for interaction. The following identifier can be "sync", stating that the interaction is two-way, or synchronous. The next identifier should mark the "state" of the interaction and can be one of the following: "sendRequest" indicates that the interaction was just initiated; "receiveRequest" (logged by the second entity involved) states that the request was received. That second entity would then declare it has replied by logging "sendResponse". Finally, the initiator of the interaction would log "receiveResponse". The next identifier should indicate the interaction partner. The initiator, WS A, would therefore log the target endpoint of WS B it makes a call to. WS B would log an identifier of WS A who is the requestor. The next part of the log entry would be an identifier of the activity that is being performed. The last part of the log entry is a timestamp.

With respect to the interaction between WS A and WS B (Figure 5) the log entries would look as follows. Recall that each of the Web services upholds its own (multiple) log files. The log entries are given in chronological order.

Just before making the actual call WS A would log the following entry:

```
   int – sync – sendRequest – <WS B target endpoint> – <activity
x> – timestamp
```

WS B would receive the call in form of a SOAP-message and create the following log entry in its log file:

```
   int – sync – receiveRequest – <WS A identifier> – <activity
y> – timestamp
```

After finishing performing the request WS B would send back a SOAP-message and create another log entry:

```
   int – sync – sendResponse – <WS A identifier> – <activity y>
– timestamp
```

Finally, WS A would receive the response and make another entry to its log file:

```
   int – sync – receiveResponse – <WS B target endpoint> -
<activity x> - timestamp
```

The log files now hold complete information about the interaction between WS A and WS B. Using the starting identifier *int* entries concerning interactions can easily be found in the log –files. The marker *sync* tells us we are dealing with a two-way operation. Upon the next part, e.g., *sendRequest*, we can determine whether the operation is request-response or solicit-response. The next identifier is used to match the interacting partners together. The activity description allows us to further describe the interaction. The above log entries are sufficient to mine for two-way interactions.

We now get to one-way, (possibly asynchronous two-way), interactions as shown in Figure 6.

Figure 6. Asynchronous interaction

Note, that in figure 6 the bottom arrow is drawn using a dotted line. That is because it may or may not exist in an interaction between two Web services. The top arrow symbolizes the initial call from WS A to WS B, a one-way interaction. Depending solely on the quality and semantics of the operation that is called in WS B, a response may or may not be expected by WS A. If no response is necessary, the interaction is simply *one-way* from WS A's point of view, or *notification* from WS B's perspective. If on the other hand a response is expected, we are dealing with an asynchronous two-way interaction. These different cases need to be reflected in the log entries concerning these interactions.

We begin with the case of an asynchronous two-way interaction, implemented by performing two individual one-way operations. In the above figure that means the interaction does consist of both arrows, each symbolizing one message being sent. The log entries for an asynchronous interaction differ only slightly from those corresponding to synchronous two-way interactions. Again, the log entries are listed in a chronological order.

When making the initial call WS A would log that step:

```
   int – async – sendRequest – <WS B target endpoint> –
<activity x> - timestamp
```

WS B receives the call and logs:

```
   int - async – receiveRequest – <WS A identifier> – <activity
y> – timestamp
```

When sending back the response WS B would create another log entry of the following format:

```
   int – async – sendResponse – <WS A identifier> – <activity y>
– timestamp
```

As soon as it has received the response WS A would finally record:

```
      int – async – receiveResponse – <WS B target endpoint> –
<activity x> – timestamp
```

These log entries are similar to those of a synchronous interaction. They only differ in the identifier for the kind of interaction which is now *async*. However, during the mining process asynchronous interactions may be harder to identify. Synchronous interactions are terminated within one step of their initiation, because the execution of the requesting process halts until a reply is received. That means that the sendRequest-log entry will be followed by the receiveResponse-entry. Asynchronous interactions, however may take much longer to process. Also, the sendRequest-entry will typically be found in a different operation's log file than the receiveResponse-entry. That is because the response is delivered to the requesting WS A by an operation call by WS B. Therefore, when mining for interactions we will first create an intermediary log file for a whole WS by combining the original logs kept by each operation.

The second case of one-way operations is the one where no response is expected by the initiator of the interactions. Recall the example we gave of the film crew reporting to an agency the layoff of one of its employees. Referring to Figure 6 the bottom arrow would be omitted in a one-way operation. Such operations are called one-way (from WS A's perspective in Figure 6) or notification (from WS B's point of view. The format of a log entry concerning that specific interaction should be as follows:

When initiating the interaction WS A would log:

```
   int – oneWay – <WS B target endpoint> – activity – timestamp
```

When receiving the call WS B would create the following entry in its log file:

```
   int – notification – <WS A identifier> – activity – timestamp
```

A formal depiction of interaction log entries is given in Figure 7. The fact that the interaction is logged twice, i.e. by both WS might seem redundant at first. However, it is the only way to be able to discover the interaction from both WS perspectives when mining their logs. If, e.g., only the initiating WS were to log the interaction, there would be no way to find out about the interaction when mining the other Web service's log. Therefore, both participating entities need to keep record of the

interaction. The initiator marks the entry as *oneWay*, the notified WS keeps a *notification* record. Another reason why also the *notified* party in a one-way transaction needs to uphold a record is the possible unavailability of the initiator's log record. That is the case of the initiating WS is owned by a third party. Our mining engine should also discover such cases and display a Web service's dependency on a third party WS.

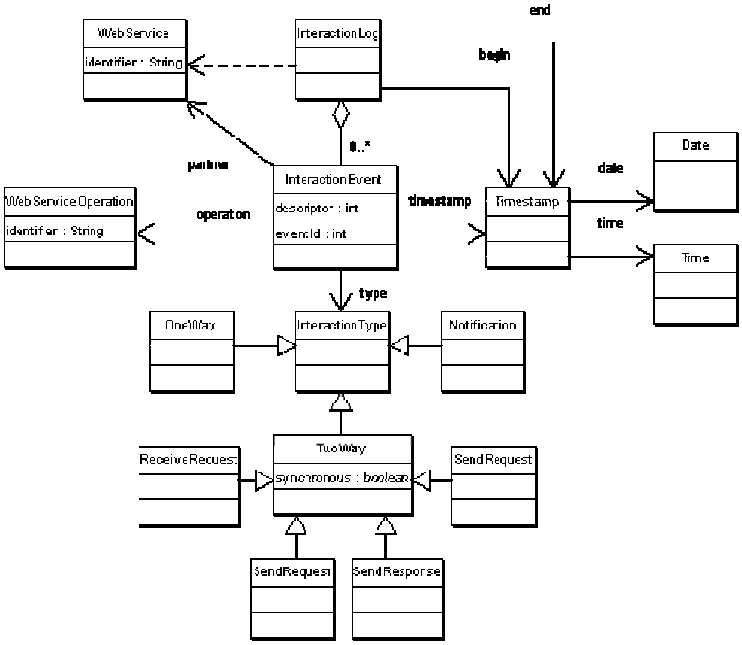

Figure 7. Class diagram of an interaction log entry

Example of WSIM on the Interactions level

In this subsection, we give an example that will further clarify our mining approach on the Web services Interaction level. We present example log records and the resulting interaction graph, which is the output of mining on the Interactions level.

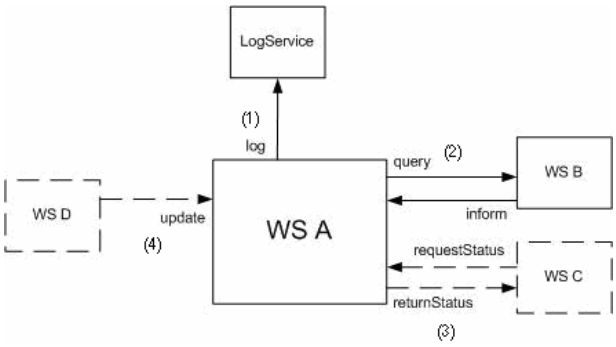

Figure 8: Example interaction graph

Figure 8 depicts an interaction graph for Web service A, i.e., the result of mining on the Web services Interactions level with the focus on Web service A. It contains one example of each of the four basic types of interactions between WS. The interactions are numbered from one to four, to which we refer to when we present the corresponding log entries in the following. For simplicity, the timestamps only show a time, while the date is omitted. Imagine the entries were all made on the same date.

WS A interacts with the LogService in a one-way manner. The interaction is marked with (1) in the interaction graph. WS A's log file would contain an entry such as

```
int – oneWay – LogService – log – 7:03
```

Consequently, the LogService's logs would contain an entry

```
int – notification – WS A – loggingRecord – 7:04
```

Note, that for the interaction graph in Figure 8 we only used the entry in WS A's log. WS A recorded the performed activity by the identifier *log* which is also depicted in the graph as a more detailed description of the arrow symbolizing the interaction. We could have added the LogService's activity, i.e., *loggingRecord* on the other end of the arrow, but we see that as a detail that is insignificant since the focus is on WS A.

The one-way interaction is depicted as a unidirectional, lined arrow pointing from the initiator to the called WS. The interaction between WS A and WS B is of type solicit-response. It is marked with (2) in Figure 8. The log entries are

```
int – sync – sendRequest – WS B - query – 8:01
int – sync – receiveResponse – WS B – query – 8:04
```

in WS A's logs and

```
int – sync – receiveRequest – WS A – inform – 8:02
int  - sync – sendResponse – WS A – inform – 8:03
```

in WS B's log files. The interaction is depicted as two line arrows going in opposite directions indicating the direction the messages are passed. Both arrows show the activity that is being performed on the respective interacting partner's side, i.e., *query* and *inform*.

The interaction of WS A with WS C is of type request-response, i.e. the interaction is initiated by WS C. It is marked with (3) in Figure 8. These are the corresponding log entries, starting with those in WS A's logs.

```
int – async – receiveRequest – WS C – returnStatus – 9:02
int – async – sendResponse – WS C – returnStatus – 9:03

int - async – sendRequest – WS A – requestStatus – 9:01
int – async – receiveResponse – WS A – requestStatus – 9:04
```

The fact that WS A is only a "passive" participant in the interaction is emphasized by drawing both the interaction and the other WS using a dotted line. Other than that, the visualization is similar to the previous case. Two arrows symbolize the messages that are passed between the WS. The interactions are further described by the terms *requestStatus* and *returnStatus* which refer to the activity-part of the log entries.

Finally, WS A interacts with WS D in a notification manner, i.e., WS D sends an unanswered message to WS A. The interaction is marked number 4 in the interaction graph. The log entries are

```
int - notification – WS D – update – 10:02
```

on WS A's side and

```
int - oneWay – WS A – sendUpdate – 10:01
```

Note, that again the box symbolizing WS D and the arrow symbolizing the interaction are drawn using dotted lines to emphasize WS A's passive role in the interaction. Also, only WS A's role in the interaction is visualized, i.e., *update*. This example can be used to clarify the need for log entries on both sides even in one-way interactions. Imagine, WS D were not in our ownership and we would have no means of accessing its logs. In that case, the log entry in our own WS A would be sufficient to discover that an interaction has happened.

3.3 The Web Service Workflows level

On the highest level of abstraction, the workflow level, an entire workflow, or business process, comprised of a number of Web services is to be examined. The result of the mining effort on this level is a model of an entire workflow.

The mining on the workflow level is a stand-alone procedure, i.e. the logs need to be mined specifically for workflows. At first glance, one might think a workflow model could be constructed by simply combining the interaction graphs of the previous Section into a workflow model. Unfortunately, it is not that simple.

The interaction graphs built through WSIM on the Interaction level do not contain any information on workflows. They display *all* interactions of a WS, no matter what workflow they belong to. Consider our case study, for example. A camera crew might interact with a stunt crew when involved in a workflow called *makeFilm*. However, during another workflow, e.g., *makeNatureDocumentary*, there are no interactions with a stunt crew, but rather with a team of nature experts, who assist the camera crew in selecting the appropriate equipment. The WS "Nature Experts" might inform the camera crew that frogs are amphibians, i.e., they live on the land *and* in the water. The camera crew would then know that regular as well as water proof cameras need to be used during the production of a documentary on frogs. This example shows that some interactions are part of a given workflow, others are not. This has important implications on WSIM, for it shows the requirement of workflow information when WSIM should be performed on a workflow level.

Log specification

On the workflow level, we want to apply process mining to a service-oriented system. To do that, our log specifications need to be extended. In order for workflow mining to be possible, log entries need to include workflow information, i.e., a process-ID and an instance-ID. The process-ID specifies the workflow, or business process that is currently being executed. With respect to our case study, the business process might be called *makeFilm*, hence all log entries would include the starting identifier *makeFilm*. The instance-ID refers to one single execution of the business process. During mining this is needed to compare multiple instances of a workflow to each other and find a workflow model that best describes *all* those instances. In our case study an instance could be "Lord of the Rings part 1" or "The Village". The last extension to the log specification is therefore to add

```
<process-ID> - <instance-ID> -
```

to the beginning of every log entry.

Example of WSIM on the workflow level

We now present possible results of WSIM on the Workflow level. We will not present the reader with complete sets of log records belonging to a workflow. Remember, a workflow may involve the execution of a very large number of activities performed by many different entities, i.e., Web services. Therefore, we prefer to show the output of WSIM on the workflow level for two different workflows. The examples shown should raise awareness of the challenges that are faced during process mining and further emphasize the importance of workflow information in a Web service's logs.

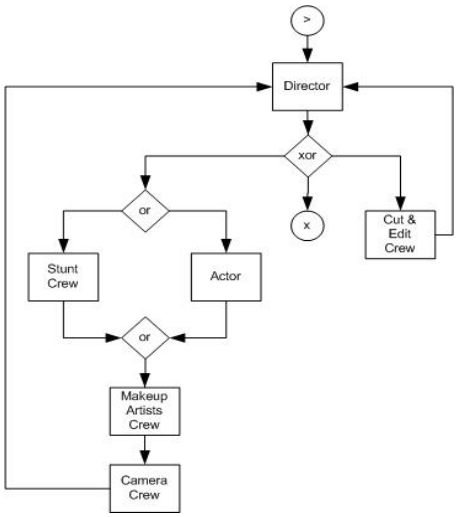

Figure 9. Workflow model "makeFilm"

Figure 9 shows the possible output of WSIM on the workflow level when mining for the workflow *makeFilm*. The top circle indicates the initialization of the workflow. At first, the WS "Director" is called. The WS then has one of three options. It may make a call to the WS "Cut & Edit Crew" or it may terminate the workflow (indicated by the circle marked X). The third option is to shoot another scene for the film, represented by the branch on the very left. The making of a scene starts by a call to the WS "Stunt Crew", or to the WS "Actor", or to both. After that, there is always an interaction with first the Makeup Artists Crew and then the Camera Crew. When the branch of the workflow is finished, the control flow goes back to the Director. The above example is, of course, strongly simplified – the creation of a motion picture involves many more entities (WS). A general model of the making of a movie would be more complex. However, the presented example is sufficient to provide an idea about WSIM on the Workflow level.

In order to show the large scope of WSIM on the Workflow level, we provide another example workflow. Figure 10 is a graphic representation of the workflow *makeDocumentary*. Note, that some of the WS which are part of the workflow *makeFilm* are also involved in making a documentary, e.g., the WS Director, the Camera Crew and the Cut & Edit Crew. However, there are also other WS which play no role in the making of a film, e.g., the WS "Historian".

These two example workflows show nicely, how *loosely-coupled* Web services can be. There are basically no limitations as to how many workflows a single WS might be part of. The examples also emphasize the need for workflow information in log records if WSIM should be performed on a workflow level. Without the identifiers we discussed in the previous subsection, i.e., process-ID and instance-ID, a log entry is practically worthless to workflow mining. Therefore, WSIM on the workflow level is only possible if sufficient workflow data is available.

In this Section we have presented log specifications and example log records which would make WSIM possible. In Section 4 we will discuss how these logs could be obtained by a WSIM implementation

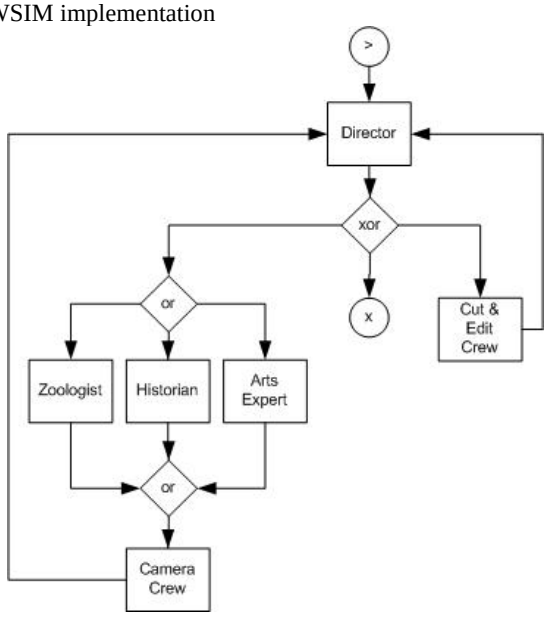

Figure 10. Workflow model "makeDocumentary"

4 WSIM Implementation Architecture

In the previous Sections the basic concepts of WSIM have been presented. In this Section we focus on discussing implementation aspects. Figure 11 details WSIM components and their coordination as follows:

(a) Starting from Web services execution, events are gathered into logs;

(b) To keep these logs, created by different log monitors (see later), homogeneous, and usable within a mining process, these logs are wrapped into a common 1st order logic format, compliant with UML class diagrams shown in figures 4 and 7;

(c) Mining rules are applied on resulted 1st order Log events to discover Web service interaction patterns. We use a Prolog-based presentation for log events, and mining rules using the XProlog system 6];

(d) Since interaction patterns are discovered, the Web service designer will have a look on the developed web service to restructure or redesign his Web service interactions.

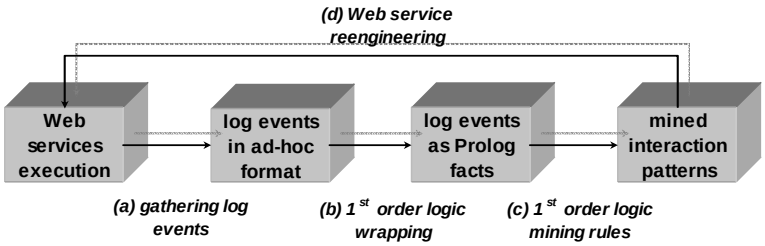

Figure 11. WSIM Components Overview

Concerning the creation of the log event entries, which is a critical phase within our WSIM approach, the question that arises is where to get the necessary log information from. Let us first take a look at the situation we face before we begin with WSIM.

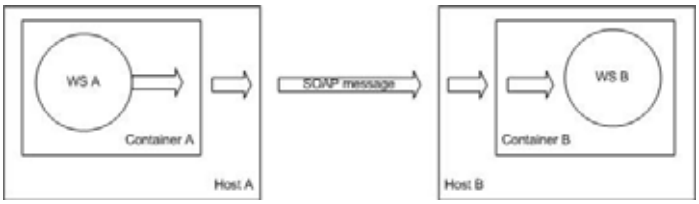

Figure 12. Interaction between WS

Figure 12 gives a more detailed look on an interaction between two WS. It also contains the involved entities, i.e., the Web services, the WS containers that manage the WS, the hosts the WS containers are running on, and the SOAP message that is being exchanged between the Web services. One of our core assumptions is therefore that the WS we want to mine use SOAP (over HTTP) to communicate, or interact.

We now want to find a way to make WSIM possible, i.e., to ensure the availability of log records as we described them in the previous Section. Basically, there are two possible situations WSIM might be confronted with. Option (1) is that a set of WS were designed with the intention to make them "*minable*", or "fit for WSIM". This means that the desired log records could be provided by the WS itself. Therefore, the Web service's program code would contain additional statements that deal with the logging of activities and interactions. From WSIM's point of view, this is clearly the desired situation, as it guarantees the availability of logs that conform to our specification. The available information would therefore be complete and sufficient to perform WSIM on *all* three levels of abstraction. However, there are two main

downsides to this assumption. For one, the logging of events by the WS would require additional development effort. Although this additional effort could be kept relatively small if an appropriate API were available, it can not be expected that all WS developers will have WSIM in mind during the design and development phase of their service-oriented systems. The second weakness of the assumption that WS are "built for WSIM" is that WS developed *prior* to the appearance of WSIM would not be "*minable*".

This describes exactly the second kind of situation one might face before starting Web Service Interaction Mining. Option (2) is that a collection of WS that have not been developed to be ready for WSIM. In this case we need to find other ways of creating the log files. However, it turns out that with the tools available today it will be impossible to keep log records that fulfill our log specifications on *all* levels. However, a partial compliance with our specifications might be reached.

As we have stated before, WSIM on the *Operation level* can be closely tied to the Web service's internal structure and functionality. Therefore, it is not subject to standardization and strongly depends on the developer's goals and needs. For example, data mining might be applied to the WS' output data if operational data is logged in e.g., a database. Without any logging that is done "from the inside" of the WS, WSIM on the O*peration level* is not possible.

Log records that allow WSIM on the *Web services Interaction level* may be acquired using tools which are already available. Since all interactions between WS happen through the exchange of SOAP message, it would be possible to intercept those messages and extract the information needed. The extracted information can be used to build logs that partially conform to our log specification on the Web services level. A SOAP messages contains both an identifier of the Web service provider (the receiver of the message) and an identifier of the Web service consumer (the sender of the message). Using the target endpoint specified as the receiver of the message, the WSDL file of the Web service can be retrieved to find out what type of operation is being invoked. Using this information a log entry of the following format can be created:

```
  int - [type of operation] - ID requestor - ID provider -
[name of operation] - timestamp
```

Note, that when using this technique an interaction is only logged once as opposed to logging it on the side of each interaction partner as we demanded in the previous Section. However, from the above information the log entries for each interaction partner can be easily generated. If the type of operation in the above example were e.g., *oneWay* (from the requestor's point of view) the corresponding log entry on the provider's side would be *notification*. What is missing from our log specification is an identifier of the *activity*. Therefore, the name of the invoked operation will need to serve as the identifier of the activity. If an operation's name is semantically meaningful, this approach is justified.

There are tools available which can be used to intercept SOAP messages. The Apache Software Foundation provides a TCP Tunnel/Monitor tool as part of the Apache SOAP package [7]. This tool, intended mainly for debugging purposes, can be used to tunnel SOAP traffic to a remote host. The tunneled traffic, i.e. the SOAP

messages are then displayed in a simple graphical user interface (GUI). The technology used by this tool could be extended to log SOAP message content in the way we described earlier.

Another interesting tool available is Mindreef's SOAPscope 3.0 [8]. The vendor calls it "a toolkit-independent Web services diagnostic system for examining, debugging, testing and tuning Web services." It intercepts SOAP-messages in one of three ways: through sniffing the network traffic for SOAP-messages, through an HTTP-proxy, or through port forwarding. It provides logging and the display of statistics concerning the analyzed messages.

Another possibility to intercept SOAP messages is on the container level. An example of a Web service container is Apache Tomcat [9]. Since all SOAP messages are passed to the WS by the container, the container could be extended to provide for our logging needs. The advantage of this approach would be that messages to and from every WS deployed in the container could be intercepted, analyzed, and logged. The resulting log file, containing information about all Web services, could be preprocessed to comply with our specification.

On the highest level of WSIM, the *Workflow level*, we face serious difficulties. As we discussed in the previous Section, mining for workflows requires the availability of workflow information. However, in traditional Web services there is no means of obtaining information about which workflow is currently being executed. WSIM on the workflow level requires additional development effort during design and implementation time and will therefore be discussed in the following subsection.

Building WS for WSIM

We conclude this paper with suggestions regarding the design of Web services with the intention to make them fit for future WSIM. Basically, we recommend that Web services provide the logs according to our specification of Section 5. We are in the process of developing an easy to use Java API which facilitates the logging of events and activities.

On the *Operations level*, the information to be logged depends highly on the Web service's characteristics as well as on the goals and needs of the developer. Therefore, neither the mining approach nor the data to be logged can be standardized fully. Extension mechanisms need to be provided.

On the *Interaction level*, our API will supply methods to log interactions between Web services. The API should be used appropriately before and after every programmatic interaction between WS. By doing so, all interactions, even those with third party owned WS, are recorded and can later be mined.

On the *Workflow level* we need to find a solution to the problem of how to exchange and retrieve workflow information between WS. Since any Web service may be part of multiple workflows, we must find a means for a WS to know what business process it is currently part of when it creates its log entries. The only way to do that in a service-oriented system is to include both a process-ID and an instance-ID into the SOAP messages that are being passed between Web services. SOAP messages are designed in such a way that they allow including any information into the message header in a key-value style. A data-item in string-format, e.g., "makeFilm" is written into the message header and bound to a certain key, e.g.,

"processID". The value can then be obtained by the receiver of the message using that key. In this way the process-ID as well as the instance-ID can be retrieved by any WS in our system to be included in log entries. To avoid interference with any other data in the message header, a unique key should be used to put the information in the header; e.g., we use "process4wsim" and "instance4wsim" in our prototype implementation.

5 Related work

Valuable research results have been achieved in data mining, process mining, and web mining. However, the idea of *Web service Interaction mining* as proposed in this paper, is yet a new hot research topic. In this Section, we discuss process mining works that are the most relevant to our area. Process mining is the major issue in WSIM on the Workflow level. [2] provides an overview of the ideas behind process mining, or workflow mining. They describe process mining as "a method of distilling a structured process description from a set of real executions". Also, the major challenges in process mining are discussed in detail, which gives the reader a very good idea of the problems one might be faced with. These challenges are e.g., mining hidden tasks, mining non-free-choice constructs or loops, dealing with noise and incompleteness or gathering data from heterogeneous sources. Furthermore, an overview over different mining algorithms is given as well as a brief description of the other papers which are part of this special issue on process mining. In [3] a detailed description of what the input data should look like in order to allow for the mining of exact workflow models is presented. Some of these specifications are used in Section 3 of this paper where we present our suggestions of log specifications. Furthermore, [3] elaborates in detail on a step-by-step description of the workflow mining process itself. This process includes the pre-processing of workflow logs and the building of sub-models. The theoretical description of the process is followed by an example, which improves the reader's understanding of process mining significantly. Also, an implementation of the algorithm is presented in the form of an application named *Process Miner*. In order to monitor business process quality, [5] proposes a solution, based on data warehousing and mining techniques for analyzing, predicting, and preventing the occurrence of *exceptions*. Other works in process mining focus on discovering workflow transactional behaviour among workflow instance through execution log [4].

6 Conclusion and Perspectives

In this paper we have outlined our novel idea of Web Service Interaction Mining (WSIM). We have identified three levels of abstraction with respect to WSIM: the operation level, the interaction level and the workflow level. The term mining implies that available log data should be analyzed to acquire additional knowledge about a system.

We believe that developing Web services with consideration for WSIM can significantly improve the manageability of a WS or of an entire service-oriented system. We especially discussed WSIM on the operations and interactions level. The information regarding all interaction partners can be vital during e.g., an impact analysis of changes made to a Web service. In the near future we will therefore direct our attention to developing an easy-to-use framework that allows for the implementation of WS which are ready for WSIM.

However, we also want to take into consideration Web services that have already been deployed. In our future work, we will examine standard logging and maybe mining tools and test them for their usability in WSIM. Especially on the Web services interactions level we see some opportunities of mining for Web service interactions, which were discussed in this paper. WSIM on the workflow level seems to pose the greatest difficulties. As we have shown, WSIM on the workflow level *does* require additional development effort. A workflow-ID and an instance-ID are needed and can only be available if provided by the WS itself.

7 References

1. G. Alonso, F. Casati, H. Kuno, and V. Machiraju, Web Services Concepts, Architectures and Applications, Springer-Verlag, 2004.
2. W. M. P. van der Aalst, and A. J. M. M. Weijters, Process mining: a research agenda, in: Computers in Industry 53, Elsevier B.V., 2003.
3. G. Schimm, Mining exact models of concurrent workflows, in: Computers in Industry 53, Elsevier B.V., 2003.
4. D. Grigori, F. Casati, U. Dayal, Ming-Chien Shan: Improving Business Process Quality through Exception Understanding, Prediction, and Prevention. VLDB'2001, Roma, Italy
5. W. Gaaloul, S. Bhiri, and C. Godart, Discovering Workflow Transactional Behavior From Event-Based Log, CoopIS'2004, Agia Napa, Cyprus, 25-29 Oct. 2004.
6. XProlog, http://www.iro.umontreal.ca/~vaucher/xprolog/
7. The Apache Software Foundation, http://ws.apache.org/soap
8. Mindreef SOAPscope, http://www.mindreef.com/products/overview.html
9. The Apache Software Foundation, http://jakarta.apache.org/tomcat