Grant Agreement N° 215483

Title:	*Knowledge extraction from service usage*
Authors:	*CNR, TUW, SZTAKI*
Editor:	*Fabrizio Silvestri (CNR)*
Reviewers:	*Osama Sammodi (UniDuE)*
	Elisabetta di Nitto (PoliMi)
Identifier:	*Deliverable #PO-JRA-2.3.7*
Type:	*Deliverable*
Version:	*1.0*
Date:	*29 June 2011*
Status:	*Final*
Class:	*External*

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° 215483 (S-Cube).

Management Summary

This deliverable is aimed at summarizing the joint research in WP-JRA-2.3. related to knowledge extraction from service usage. The work is focused on methodologies to extract knowledge from service/Web logs and possible applications of them in order to enhance service/Web applications. Results are presented in four published papers and one technical report that constitute the core contribution of this deliverable. The work is positioned within the Integrated Research Framework (IRF, WP-IA-3.1), internal WP-JRA-2.3 research architecture and overall WP-JRA-2.3 goals and visions.

Members of the S-Cube consortium

University of Duisburg-Essen (Coordinator)	Germany
Tilburg University	Netherlands
City University London	U.K.
Consiglio Nazionale delle Ricerche	Italy
Center for Scientific and Technological Research	Italy
The French National Institute for Research in Computer Science and Control	France
Lero - The Irish Software Engineering Research Centre	Ireland
Politecnico di Milano	Italy
MTA SZTAKI Computer and Automation Research Institute	Hungary
Vienna University of Technology	Austria
Universit Claude Bernard Lyon	France
University of Crete	Greece
Universidad Politcnica de Madrid	Spain
University of Stuttgart	Germany
University of Hamburg	Germany
Vrije Universiteit Amsterdam	Netherlands

Published S-Cube documents

All public S-Cube deliverables are available from the S-Cube Web Portal at the following URL: `http://www.s-cube-network.eu/results/deliverables/`

The S-Cube Deliverable Series

Vision and Objectives of S-Cube

The Software Services and Systems Network (S-Cube) will establish a unified, multidisciplinary, vibrant research community which will enable Europe to lead the software-services revolution, helping shape the software-service based Internet which is the backbone of our future interactive society.

By integrating diverse research communities, S-Cube intends to achieve world-wide scientific excellence in a field that is critical for European competitiveness. S-Cube will accomplish its aims by meeting the following objectives:

- Re-aligning, re-shaping and integrating research agendas of key European players from diverse research areas and by synthesizing and integrating diversified knowledge, thereby establishing a long-lasting foundation for steering research and for achieving innovation at the highest level.
- Inaugurating a Europe-wide common program of education and training for researchers and industry thereby creating a common culture that will have a profound impact on the future of the field.

- Establishing a pro-active mobility plan to enable cross-fertilisation and thereby fostering the integration of research communities and the establishment of a common software services research culture.
- Establishing trust relationships with industry via European Technology Platforms (specifically NESSI) to achieve a catalytic effect in shaping European research, strengthening industrial competitiveness and addressing main societal challenges.
- Defining a broader research vision and perspective that will shape the software-service based Internet of the future and will accelerate economic growth and improve the living conditions of European citizens.

S-Cube will produce an integrated research community of international reputation and acclaim that will help define the future shape of the field of software services which is of critical for European competitiveness. S-Cube will provide service engineering methodologies which facilitate the development, deployment and adjustment of sophisticated hybrid service-based systems that cannot be addressed with todays limited software engineering approaches. S-Cube will further introduce an advanced training program for researchers and practitioners. Finally, S-Cube intends to bring strategic added value to European industry by using industry best-practice models and by implementing research results into pilot business cases and prototype systems.

S-Cube materials are available from URL: `http://www.s-cube-network.eu/`

Contents

List of Figures

Chapter 1

Foreword

According to the Description of Work (Grant Agreement for: Network of Excellence Annex I Description of Work) this deliverable is aimed at documenting the ways a service/web log can be mined in order to derive useful knowledge. In particular, this deliverable presents different ways of using knowledge extracted from Service Oriented Architectures (SOAs) event logs and Web search engines log and different applications of such knowledge. The rationale of focusing our interest on these techniques within the SOA context is motivated by the fact that they have proven to be effective in other domains (e.g. Web search engines).

The joint research work in WP-JRA-2.3 is composed of two tasks: T-JRA-2.3.1: Infrastructure Mechanisms for the Run-Time Adaptation of Services and T-JRA-2.3.2 Service Registration and Search. The current deliverable is related to T-JRA-2.3.2. As stated in the Description of Work, this deliverable "will report on studies about the knowledge extracted from the activities in online services. The deliverable will also report on the techniques studied and used to perform the knowledge extraction task. As one of the case studies we will use logs from search engines activities to extract knowledge regarding users activity". Therefore, it aims at summarizing the investigations related to extracting knowledge from event logs collected by complex software systems.

The joint work is focused on applying several state-of-the-art data mining algorithms to event logs provided by S-Cube partners. In order to fully present the possible outcomes resulting from the application of these techniques in the SOA domain we describe, as our running case study, how log mining has proven to be effective in enhancing the overall performances of complex software systems such as Web search engines. Results of the research work are presented in four published papers and one technical report and constitute the core contribution of this deliverable. The work is positioned within the Integrated Research Framework (IRF, WP-IA-3.1), internal WP-JRA-2.3 research architecture and overall WP-JRA-2.3 goals and visions.

Chapter 2

Introduction

Data is everywhere. Computer systems keep track of activities of users in the form of log files. Ranging from system logs on Web servers to logs collected by large-scale service based applications, this type of data represents a goldmine of knowledge that, once extracted, can help the stakeholders of the whole system to understand better if, and how, the application can be improved. To this aim, data mining consist of a set of techniques aiming at extracting patterns from large data sets by combining methods from *statistics* and *artificial intelligence* with *database management*. With recent tremendous technical advances in processing power, storage capacity, and inter-connectivity of computer technology, data mining is seen as an increasingly important tool by modern business to transform unprecedented quantities of digital data into business intelligence giving an informational advantage.

Service-centric systems are said to be flexible and dynamic. To support this flexibility, event processing mechanisms can be used to record which events occur within the system. This includes both basic "service events" (e.g., service is created) and complex events regarding QoS (e.g., average response time of service X has changed) and invocations (e.g., service X has been invoked), supporting complex event processing. Users can subscribe to various events of interest, and get notified either via email or Web service notifications (e.g., WS-Eventing). Such notifications may trigger adaptive behavior (e.g., rebinding to other services). Service Oriented Architectures (SOAs) are thus complex infrastructures consisting of thousands or millions of service interacting together in order to achieve complex operations (tasks). Service invocation logs are file tracing the interactions between services. As in other contexts, data mining techniques can be thus applied in order to derive useful knowledge. Such knowledge can be spent in order to enhance both effectiveness and efficiency of the overall infrastructure.

The same approach within other fields like, for example, the Web domain is proven to be effective. The knowledge extracted by means of data mining techniques from query logs (files containing the interactions of the users with the search engine) is the first way a search engine improve its performances in terms of effectiveness and efficiency. In this deliverable we thus investigate how useful knowledge can be extracted from service logs and possible ways of applications within the SOA context. In order to do that as one of the case studies we will use logs from search engines activities to extract knowledge regarding users activity.

2.1 Connections to the Integrated Research Framework

The Integrated Research Framework (IRF) [1] defines four views (other aspects of the IRF, like research challenges, questions and results are being developed simultaneously with this deliverable and omitted.)

The *Conceptual Research Framework* organizes the joint research activities by providing a high-level conceptual architecture for the principles and methods for engineering service based applications, as well as for the technologies and mechanisms which are used to realize those applications. The work presented in this deliverable is clearly related to the "Service Composition and Coordination" and "Service Infrastructure" domains in the horizontal classification whereas addresses issues of "Adaptation",

"Monitoring" and "Quality Definition, Negotiation and Assurance" of the cross-cutting vertical categorization.

The *Reference life-cycle* complements the static view of the conceptual research framework. It is composed of two main cycles: one corresponds to the classical application design, deployment and provisioning while the second one corresponds to the run-time perspective, including monitoring and adaptation. The work presented in this deliverable corresponds to both of the two cycles. The classical analysis and design could benefit of the techniques proposed here for adaptation cycle by investigating techniques aiming at enhancing the analysis and design phase. Furthermore, the run-time perspective exploits these techniques for detecting problems and changes. The techniques presented in this deliverable could also be used for identifying possible adaptation/monitoring strategies and enacting them.

The *Logical Run-Time Architecture* unifies all runtime mechanisms into a coherent framework. The work reported in this deliverable is related to nearly all components of the logical architecture: Monitoring Engine, Adaptation Engine, Negotiation Engine, Runtime QA Engine and Resource Broker.

The *Logical Design Environment* is complementary to the run-time architecture and its purpose is to provide a context where to place the envisioned techniques and mechanisms that support the analyst and designer during the whole SBA's lifecycle. Our work is related to modeling and verification. The techniques presented in this work could help SBA designers as they could reveal possible frequent patterns contained in historical data that could be exploited within new activities.

2.2 Connections to the JRA-WP-2.3 research architecture

Research work in WP-JRA-2.3 is driven by the Work Package vision that structures the research work internally. Figure 2.1 illustrates the overall research architecture of WP-JRA-2.3: research on service infrastructures is comprised in three threads, Service Discovery, Service Registries and Service Execution. Orthogonally different approaches are separated in three layers.

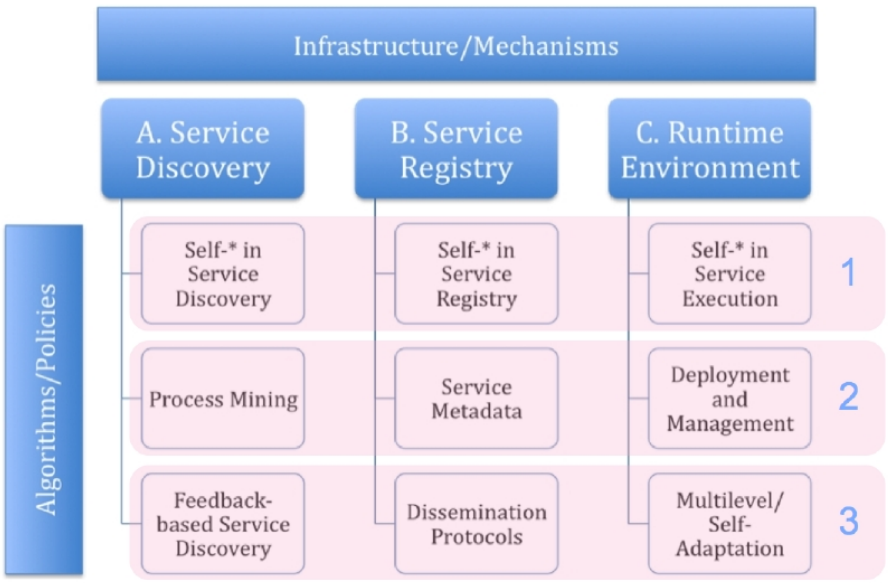


Figure 2.1: WP-JRA-2.3 Research Architecture.

- Service Discovery Thread (A) - Service discovery is a fundamental element of service oriented architectures, services heavily rely on it to enable the execution of service-based applications. Novel discovery mechanisms must be able to deal with millions of services. Additionally, these discovery mechanisms need to consider new constraints, which are not prevalent today, such as

Quality of Experience requirements and expectations of users, geographical constraints, pricing and contractual issues, or invocability.

- Service Registry Research Thread (B) - Service registries are tools for the implementation of loosely-coupled service-based systems. The next generation of registries for Internet-scale service ecosystems are emerging, where fault tolerance and scalability of registries is of eminent importance. Autonomic registries need to be able to form loose federations, which are able to work in spite of heavy load or faults. Additionally, a richer set of metadata is needed in order to capture novel aspects such as self-adaptation, user feedback evaluation, or Internet-scale process discovery. Another research topic is the dissemination of metadata: the distributed and heterogeneous nature of these ecosystems asks for new dissemination methods between physically and logically disjoint registry entities, which work in spite of missing, untrusted, inconsistent and wrong metadata.
- Runtime Environment Research Thread (C) - There is an obvious need for automatic, autonomic approaches at run-time. As opposed to current approaches we envision an infrastructure that is able to adapt autonomously and dynamically to changing conditions. Such adaptation should be supported by past experience, should be able to take into consideration a complex set of conditions and their correlations, act proactively to avoid problems before they can occur and have a long lasting, stabilizing effect.

In alignment with the lifecycle aspect of the Integrated Research Framework, the presented work – due to its nature -, is related mostly to runtime activities hence, to research topics in thread A in Figure 1. While topic A2 is directly covered topic C1 also presented progress. Service registries and runtime environments could be enhanced by the techniques presented in the following. In particular, some aspects of B2 and C2 are covered.

Chapter 3

Description of Data Analyzed

Modern complex software systems save traces of their activities to proper files called "logs". Such files could store "events" regarding the whole infrastructure (like in Service Oriented environments) or real "activities" performed by users interacting with the infrastructure. A possible example of the latter case could be represented by logs of queries coming from Web search engines.

In this deliverable we focus on these two different types of logs: "event logs" coming from Service Oriented Architectures and "query logs" coming from Web search engines. We now describe in more details the main characteristics of an "event log" and a "query log".

3.1 SOA Event Logs

Events and complex event processing [11] (CEP) are frequently used tools to document and track the lifecycle of applications in various domains. For instance, in the business domain the idea of business activity monitoring [9] (BAM) uses events to monitor business process performance. Analogously, technical implementations of business processes on top of SOAs (service compositions) are often monitored using CEP. To this end, many service composition engines can be configured to track their current state in event logs. For instance, the Apache ODE WS-BPEL engine triggers a rich model of execution events[1]. Similarly, service compositions implemented using Windows Workflow Foundation can use the .NET tracking service[2] to persist event logs. However, tracking system state via event logs in SOA is not confined to composition engines. For instance, The Vienna Runtime Environment for Service-Oriented Computing (VRESCo) [12] uses events to track not only service compositions, but all entities and interactions in a SOA (services, users, compositions, metadata and interactions).

In its most general form, an event log $\mathcal{E}$ consists of a sequence of n recorded events, i.e., $\mathcal{E} = \langle e_1, e_2, \ldots, e_n \rangle$. Each event $e_i \in \mathcal{E}$ usually contains at least an unique identifier, an event timestamp, the publisher of the event (e.g., the BPEL engine), the subject of the event (e.g., the composition instance that triggered the event), and the event type. Depending on the concrete event type, more detailed information is available. This type-specific information cannot be described generally, i.e., it is different from event type to event type as well as from system to system. In the following we describe the event types triggered by the VRESCo system as an example of the possibilities provided by event logs. The VRESCo event log will also constitute our running data set all along with this deliverable.

VRESCo is an experimental runtime environment developed at Vienna University of Technology. VRESCo is being developed under an open source license, and can be accessed via the project Web page[3]. The project aims at solving some of the research problems identified in [14], e.g., dynamic selection of services based on Quality-of-Service (QoS), dynamic rebinding and service composition, service metadata and event-based services computing.

[1] `http://ode.apache.org/ode-execution-events.html`
[2] `http://msdn.microsoft.com/en-us/library/ms735887(v=vs.85).aspx`
[3] `http://www.infosys.tuwien.ac.at/prototypes/VRESCo/`

In the following we focus on the latter aspect. The foundations of event-based service-oriented computing have been discussed in [2, 3]. In a nutshell, the goals of this earlier work were to track what is going on in a service-based application by constantly triggering events and using CEP to construct meaningful information from those events.

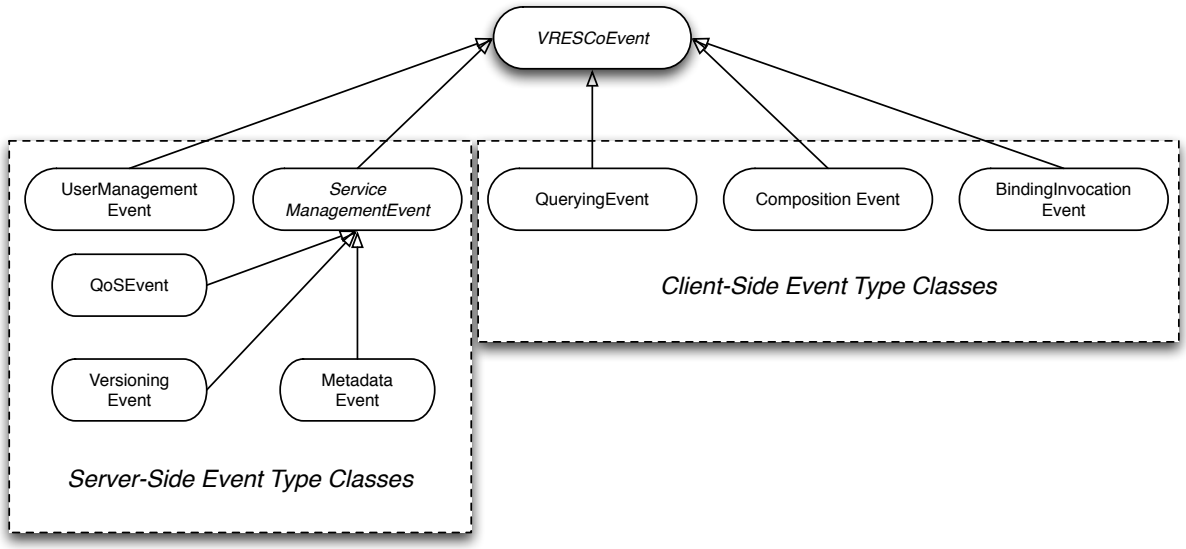


Figure 3.1: VRESCo Event Type Hierarchy

In a VRESCo system, events of various types are triggered. A simplified taxonomy of event type classes is depicted in Figure 3.1. As can be seen, events are triggered when services are queried, bound and invoked. Additionally, events indicate if the data or metadata about services changes (e.g., the QoS is changed, new operations are available). Each of the concrete event type classes (those with non-italic name) in turn contains a number of concrete event types that can be triggered. For full details on all events refer to [2].

Events in VRESCo can be triggered either on client- or server-side. While events concerning metadata are triggered by the VRESCo server, all querying and invocation events are triggered by clients and only processed throughout the VRESCo event engine. These client-triggered events are listed in more detail in Table 3.1. In the table, we provide the condition that triggers each event along with the event type and event type class of the event. All of these events provide the basic information discussed above (sequence number, timestamp, ...). In addition, events generally provide some type-specific additional information, which we also summarize in the table. For reasons of brevity, we have omitted composition events. Composition events in VRESCo are of comparable expressiveness as the events triggered by Apache ODE.

Event Type: *BindingInvocationEvent*		
Event Name	**Event Condition**	**Event Payload**
ServiceInvokedEvent	Specific service is invoked	Message sent to service
ServiceInvocationFailedEvent	Service invocation failed	Message sent to service, fault
ProxyRebindingEvent	Service proxy is (re-)bound to a specific service	Selected service
Event Type: *QueryingEvent*		
Event Name	**Event Condition**	**Event Payload**
RegistryQueriedEvent	Registry is queried	Query string
ServiceFoundEvent	Specific service is found by a query	Query string, query results
NoServiceFoundEvent	No services are found by a query	Query string

Table 3.1: Client-Triggered VRESCo Events.

The VRESCo event engine stores triggered events in an event log. Therein, events are serialized as XML and can be accessed and analyzed via a RESTful service interface. In Listing 3.1 we provide an example event serialized to XML. Evidently, the event reflects a service invocation with a very simple input message (<order>) as payload.

Listing 3.1: Serialized Invocation Event

```
<ServiceInvokedEvent xmlns="http://www.vitalab.tuwien.ac.at/vresco/usertypes">
  <Priority>0</Priority>
  <Publisher>guest</Publisher>
  <PublisherGroup>GuestGroup</PublisherGroup>
  <UserName>a007b09b-8c23-4fac-af30-0142a61f3795</UserName>
  <SeqNum>74006756-64f1-40cb-858e-565d4bc6a94c:24</SeqNum>
  <Timestamp>2010-11-09T09:58:49</Timestamp>
  <CurrentRevisionId>180</CurrentRevisionId>
  <CurrentRevisionWsdl>
    http://localhost:60000/AssemblyAtomicServices/IAssemblingPlanningService?wsdl
  </CurrentRevisionWsdl>
  <FeatureName>GetPartFeature</FeatureName>
  <InvocationInfo>
    <service_input>
      <order><part1>text</part1></order>
    </service_input>
  </InvocationInfo>
</ServiceInvokedEvent>
```

3.2 Web Search Engine Query Logs

Query logs keep track of information regarding interaction between users and the Web search engine. They record the queries issued to a search engine and also a lot of additional information such as the user submitting the query, the pages viewed and clicked in the result set, the ranking of each result, the exact time at which a particular action was done, etc. In general, a query log is comprised by a large number of records $\langle q_i, u_i, t_i, V_i, C_i \rangle$ where for each submitted query q_i, the following information is recorded: i) the anonymized identifier of the user u_i, ii) the timestamp t_i, iii) the set V_i of documents returned by the WSE, and iv) the set C_i of documents clicked by u_i.

From query log information it is possible to derive *Search Sessions*, sets of user actions recorded in a limited period of time. The concept can be further refined into: (i) *Physical Sessions*, (ii) *Logical Sessions*, and (iii) *Supersessions*.

Physical Sessions: a physical session is defined as the sequence of queries issued by the same user before a predefined period of inactivity. A typical timeout threshold used in web log analysis is $t_0 = 30$ minutes. [16, 21].

Logical Sessions: a logical session [5] or *chain* [18] is a topically coherent sequence of queries. A logical session is not strictly related to timeout constraints but collects all the queries that are motivated by the same information need (i.e., planning an holiday in a foreign country, gathering information about a car to buy and so on). A physical session can contain one or more logical session. Jones *et al.* [8] introduced the concepts of *mission* and *goal* to consider coherent information needs at different level of granularity, being a goal a sub-task of a mission (i.e., booking the flight is one of the goal in the more general mission of organizing an holiday).

Supersessions: we refer to the sequence of all queries of a user in the query log, ordered by timestamp, as a supersession. Thus, a supersession is a concatenation of sessions.

Sessions are, thus, sequences of queries submitted by the same user in the same period of time. This data can be used to devise typical query patterns, used to enable advanced query processing techniques. Click-through data (representing a sort of implicit relevance feedback information) is another piece of information that is generally mined by search engines. In particular, every single kind of user action (also, for instance, the action of not clicking on a query result) can be exploited to derive aggregate statistics which are very useful for the optimization of search engine effectiveness. How query logs interact with search engines has been studied in many papers. Good starting point references are [20, 4, 19].

Query Log Name	Public	Period	# Queries	# Sessions	# Users
Excite (1997)	Y	Sep 1997	1,025,908	211,063	∼410,360
Excite Small (1997)	Y	Sep 1997	51,473	–	∼18,113
Altavista	N	Aug 2, 1998 Sep 13, 1998	993,208,159	285,474,117	–
Excite (1999)	Y	Dec 1999	1,025,910	325,711	∼540,000
Excite (2001)	Y	May 2001	1,025,910	262,025	∼446,000
Altavista (public)	Y	Sep 2001	7,175,648	–	–
Tiscali	N	Apr 2002	3,278,211	–	–
TodoBR	Y	Jan 2003 Oct 2003	22,589,568	–	–
TodoCL	N	May 2003 Nov 2003	–	–	–
AOL (big)	N	Dec 26, 2003 Jan 01, 2004	∼100,000,000	–	∼50,000,000
Yahoo!	N	Nov 2005 Nov 2006	–	–	–
AOL (small)	Y	Mar 1, 2006 May 31, 2006	∼20,000,000	–	∼650,000
Microsoft RFP 2006	Y	Spring 2006 (one month)	∼15,000,000	–	–

Table 3.2: Features of the most important query logs that have been studied in the latest years. The dash sign (–) means that the feature in the relative column was non-disclosed.

An important key issue in query log mining is the pre-processing of logs in order to produce a good basis of data to be mined. An important step in usage analysis is thus the *data preparation*. This step includes: data cleaning, session identification, merging logs from several applications and removing requests for robots. This techniques aims to remove irrelevant items, so that the resulting associations and statistics reflects accurately the interactions of users with the search engine.

Very few query logs have been released to the public community in the last years due to their commercial importance and to privacy issues. Starting from 1997 the query logs that have been released to the public are: Excite (1997), AltaVista (1998–1999), AOL (2003–2004), AOL (2006), MSN (2006). Table 3.2 resumes the most important features of the query logs that have been examined in the latest years.

The most famous query log is undoubtedly AOL and it is also the data set we will refer to in the following of this document. The AOL data set contains about 20 million queries issued by about $650,000$ different users, submitted to the AOL search portal over a period of three months from 1st March, 2006 to 31st May, 2006. After the controversial discussion related to users' privacy issues followed to its initial public delivery, AOL has withdrawn the query log from their servers and is not offering it for download anymore.

Figure 3.2 shows a fragment of the AOL query log. Each row of this query log consist of records collecting five fields: i) the ID referring to the user issuing the query, ii) the issued query, iii) the time the

```
507  kbb.com            2006-03-01 16:45:19  1   http://www.kbb.com
507  kbb.com            2006-03-01 16:55:46  1   http://www.kbb.com
507  autotrader         2006-03-02 14:48:05
507  ebay               2006-03-05 10:50:35
507  ebay               2006-03-05 10:50:52
507  ebay               2006-03-05 10:51:24
507  ebay               2006-03-05 10:52:04
507  ebay               2006-03-05 10:52:36  69  http://antiques.ebay.com
507  ebay               2006-03-05 10:58:00
507  ebay               2006-03-05 10:58:21
507  ebay electronics   2006-03-05 10:59:26  5   http://www.internetretailer.com
507  ebay electronics   2006-03-05 11:00:21  20  http://www.amazon.com
507  ebay electronics   2006-03-05 11:00:21  22  http://gizmodo.com
507  ebay electronics   2006-03-05 11:00:21  22  http://gizmodo.com
507  ebay electronics   2006-03-05 11:18:56
507  ebay electronics   2006-03-05 11:20:59
507  ebay electronics   2006-03-05 11:21:53  66  http://portals.ebay.com
507  ebay electronics   2006-03-05 11:25:35
```

Figure 3.2: An example of the AOL query log [15].

query was issued to the search engine, iii) the position of the clicked result in the results page, and iv) the host of the clicked document.

Topics covered by queries contained in search engines are the most disparate. Figure 3.3 highlights the 250 most frequent terms in the AOL query log by means of a tag cloud. The dimension of each term in the tag cloud is directly related to its frequency in the log. The larger the term in the tag cloud, the more frequent it is in the log. As an example,“google” and “free” are the two most frequent terms in the log. Other very frequent words are: “yahoo”, “new”, “county”, “pictures”, “http”.

Figure 3.3: A tag cloud of the 250 most frequent words in the AOL query log [15]. Picture has been generated using wordle.net. From [20].

Furthemore, the queries submitted by users to search engines and thus contained in query logs are sometimes incredible. Just to give an idea, from the AOL query log user #2386968 submitted the query “why is my husband so talkative with my female friends”. Another funny example is the query submitted by the user #427326 looking for “where is my computer?”.

Chapter 4

Different Types of Knowledge Extracted and Possible Applications

The knowledge extracted from service logs can be fruitfully applied for different purposes within SOAs or Web domains. While the use of this type of knowledge within Web domains is useful for enhancing both the *effectiveness* (i.e., quality of results returned to the users) and the *efficiency* (i.e., response time) of the overall search infrastructure, SOAs could exploit this knowledge during different steps of the life-cycle of their compliant applications. In particular, at design-time it is possible to provide service designers with tools aiming at simplifying/suggesting possible (and actually used) interactions between groups of services. These tools could thus simplify the whole design phase, by giving service designers hints on possible associations between services, or possible workflows of activities (tasks) carried on by groups of services. Furthermore, at run-time phase it is also possible to study adaptation strategies that exploit this historical knowledge. In this deliverable we present five applications of different types of knowledge derived from logs coming from SOA or Web domains.

The first contribution (see Appendix A) refers to the use of knowledge extracted from service logs in order to perform a semantic resource allocation within Cloud Computing environments [6].
In this work authors introduce a generic framework for prediction and adaptation, and describe its application in a concrete scenario (Cloud resource scheduling). The basic requirements for such an environment are:

- to provide generic capability of collecting log data about internal events;
- to unify the collected data so that global coherences can be revealed;
- to provide customizable methods for getting predictions based on the collected data;
- to feed back predictions into the realization of adaptation mechanisms.

This framework combines the prediction techniques with the semantic technologies which introduces semantic knowledge to the data evaluated by predictors. Furthermore, it also exploits a multi-agent system to introduce proactiveness and distributed problem solving for increasing the scalability, adaptability and self management of the system. The predictions extracted from the semantic data are taken into account by a group of agents for allocating the different costumer jobs in the most reliable resources for each case.
Authors emphasize on the importance of using historical data by service and cloud providers. They present a generic approach and a re-usable solution for the collection and exploitation of historical log data produced by services and demonstrate its usability and usefulness in a concrete resource scheduling scenario. The heterogeneity of log data arriving from various resources calls for a semantic data representation, which can facilitate the unification of these data and a query mechanism supported by

inferencing. The paper belongs to work package JRA-2.3 as the presented solution is mostly effective on the infrastructure layer, however the methodology and the semantic approach addresses work package JRA-1.3 as well. Regarding WP-JRA-2.3 research challenges the paper is related to "Supporting adaptation of service-based applications" and "Runtime SLA Violation Prevention". The presented solution provides a generic technique to predict the quality attributes of services and resources, which serves as an enabler to various self-adaptation techniques. The paper is more focused on the resource scheduling task, and provides a semantic model for the representation of QoS measurements and requirements used in the scheduling process.
As the decision making is distributed in the suggested generic framework, there is a possibility to extend standard decision making with additional behaviors representing strategies and policies on different levels; on the level of resource providers and on the level of service providers.

The second contribution (App. B) consists of an application of process mining techniques on a real-world service log coming from the VRESCo runtime environment [13].
The vast majority of nowadays software-based systems, ranging from the simplest, i.e., small-scale, to the most complex, i.e., large-scale, record massive amounts of data in the form of *logs*. Such logs could either refer to the functioning of the system as well as keep trace of any possible software or human interaction with the system itself. For this reason, logs represent a valuable source of hidden knowledge that can be exploited in order to enhance the overall performances of any software-based system.
Well-known examples of systems that have started trying to improve their performances by analyzing event logs are surely Web Search Engines (SEs). Roughly, SEs are increasingly exploiting past user behaviors recorded in *query logs* in order to better understand people search intents, thus, for providing users with better search experiences. Indeed, by accurately recognizing and predicting actual user information needs, SEs are now able to offer more sophisticated functionalities (e.g., *query suggestion*) as well as better relevant result sets in response to a specific query (e.g., *query diversification*).
Moreover, there are plenty of modern enterprise software systems that need to operate in highly dynamic and distributed environments in a standardized way. Such systems implement their business logic according to the *Service-oriented Architecture* (SOA) principles, thus, assembling their business processes as the composition and orchestration of autonomous, protocol-independent, and distributed logic units, i.e., *software services*.
Service-based systems and applications (SBAs) require proper *run-time* environments where their composing services can be searched, bound, invoked, monitored and managed. Therefore, SBA's run-time support might keep track of what is going on during the whole application lifecycle by roughly recording all such events to log files, i.e., *service event logs*.
Analysis of such service event logs could reveal interesting *patterns*, which in turn might be exploited for improving the overall performances of SOA's run-time frameworks as well as supporting SBA designers during the whole application lifecycle.
The main contribution of this work concerns the application of data mining techniques to a real-life service event log collected by the VRESCo SOA run-time framework. Our aim is to analyze the historical events stored on VRESCo in order to discover software services that are frequently invoked and composed together, i.e., *process mining*.
Although traditional process mining refers to a set of techniques and methodologies whose aim is to distill a structured process description from a set of actual traces of executions recorded in event logs, here we treat it as an instance of the *sequential pattern mining problem*.
Finally, authors apply two sequential pattern mining algorithms to a real event log provided by the Vienna Runtime Environment for Service-oriented Computing, i.e., VRESCo. The obtained results show that they are able to find services that are frequently invoked together within the same sequence. Such knowledge could be useful at *design-time*, when service-based application developers could be provided with *service recommendation tools* that are able to predict and thus to suggest next services that should be included in the current service composition.

The third work (App. C) addresses the problem of interface-based test coverage for Web services [7].

In recent years, the Service-Oriented Architecture (SOA) has become a widely adopted paradigm to create loosely coupled distributed systems, and Web services are the most commonly used technology to build SOA. One of the defining characteristics of SOA is that the API (Application Programming Interface) of the available services is published to service consumers using standard, machine-readable description languages. In the case of Web services, this is achieved using the Web Services Description Language (WSDL), paired with XML Schema Definitions (XSD) of the service operations' input and output messages.
Now that the WS-* stack builds a solid technological foundation, traditional software engineering disciplines are being applied to Web services. Among these disciplines is software testing in terms of Verification and Validation (V & V). An important field in software testing is concerned with test coverage, i.e., the extent to which an implemented system has been tested or used. Classical code-based coverage metrics such as function, statement or branch coverage require access to the source code, which is not always possible for Web services. Therefore, testing methods for Web services can usually only rely on the service's API definition, and hence focus on black-box testing of single services or testing the composition of services.
API coverage is commonly expressed as the ratio of previously performed, distinct invocations to the number of (theoretically) possible invocations as defined by the API. In the simplest case, single parameter values are varied and tested, e.g., with extreme values, to verify functionality and detect failures. Since an isolated analysis of parameters is often not sufficient, input combinations need also be considered. Achieving an API coverage of (near) 100% in this case is generally subject to the problem of combinatorial explosion, because all input parameter combinations need to be considered. Hence, it is desirable to restrict the value domain of each parameter to its smallest possible size, in order to minimize the total number of possible combinations. The authors of this work see two possibilities to achieve that. Firstly, Web service developers need to provide a more precise specification of the valid operation parameters, e.g., in terms of string patterns or allowed numeric ranges. Secondly, service testers may analyze these parameters to identify similarities or combinations that seem less important to be tested. XSD already provides a solution for the first point in the form of facets, but expressing such schema-based restrictions is surprisingly hard to achieve in major Web service frameworks such as JAX-WS (Java API for XML Web Services) or Microsoft's .NET platform. Essentially, developers cannot rely on the automated XSD generation, but have to manually define the XML Schema that uses facet restrictions. Concerning the second point, there is still an evident lack for API coverage frameworks with customizable coverage metrics that can be easily plugged into (existing) service-based systems.
In a previous work, authors developed coverage metrics for data-centric dynamic service compositions. Under the same umbrella project named TeCoS (Test Coverage for Service-based systems) they now investigate coverage of service APIs. In this paper, authors apply software testing concepts to Web services and present a solution for measuring API coverage based on historical invocations. Overall, their contribution is threefold: 1) they define API coverage metrics and their instantiation for Web services, 2) they suggest the implementation of XSD facets in the JAX-WS framework, and 3) they present and evaluate our prototype in an experimental evaluation.
The proposed approach is based on domain partitioning, a technique used to narrow down the domain space of invocation parameters, and hence to reduce the number of possible invocations to obtain more meaningful coverage metrics. The prototype implementation of TeCoS intercepts Web service invocation messages and converts the XML tree representation to a relational schema for storage in a relational database management system (DBMS). The domain partitions are defined by the user in a tailor-made expression language. To compute a certain coverage metric of some Web service operation o, the user defines which partitions should be applied to which part of the elements in the schema of the input message of o. The actual computation of the coverage value is performed on the underlying DBMS. In an ongoing work, authors plan also to extend the scope of API coverage to invocation sequences and semantic input description. Moreover, they will further enhance TeCoS with support for distributed storage and coverage computation.

The fourth contribution (App. D) aims at devising effective techniques for identifying task-based sessions, i.e. sets of possibly non contiguous queries issued by the user of a Web Search Engine for carrying out a given task [10].
The *World Wide Web* (Web) was born as a *platform* to connect academic and research people, which exploit the Internet as the communication medium infrastructure. Rapidly, an increasing number of users, which were not directly involved in academia or research activities, have started to have access to the Web as well.
Nevertheless, in the first Web era there was still a clear separation of roles between *few* content providers, i.e., typically skilled workers and professionals, and *many* content consumers, i.e., common end users.
During the last years, a new trend has gained momentum: new applications that allow easy authoring and content creation have lead to an increased *democratization* and *collaborative involvement* in the Web. Somehow, this process caused the end of the first Web era, by bringing down the wall between content providers and consumers, which now can play both roles interchangeably from time to time. Therefore, information made available on the Web have started raising at a tremendous speed rate, reaching nowadays a huge and still growing number of contents, which spread over several media types (e.g., text, images, audio/video, etc.).
This great repository of data makes the Web the place of choice where people look at whenever they come up with *any* sort of information need. Indeed, there is a common belief that the Web is increasingly used not only for consulting documents but also for trying to simplify the accomplishment of various everyday activities, i.e., *tasks*.
Moreover, most of the interactions between users and the Web are often mediated by *Web search engines*, which are amongst the most important and used Web-based tools. This trend is confirmed by a rising "addiction to Web search": no matter what an information need is, user will ask it to a Web search engine that will hopefully give her the answer she expects.
Although the huge number of features which now the most popular Web search engines come with, in essence they still belongs to the category of Web documents retrieval tools. The results they provide in response to a user query are given according to the traditional "*ten blue links*" paradigm, i.e., links to Web pages that are considered *relevant* to the given user query. If results are not satisfactory, decision taken by looking at some of them, users may decide to "re-phrase" the query to try to refine the retrieved results. However, when the need behind a certain query is a task to be accomplished, this "*query-look-refine*" paradigm could not be effective. In other words, for certain (quite popular) tasks, Web search engines as we know can be considered obsolete tools.
Therefore, authors believe next-generation Web search engines should turn from mere Web documents retrieval tools to multifaceted systems, which fully support users while they are interacting with the Web. Of course, this opens up novel and exciting research challenges, in particular the ability to recognize implicit user tasks from the issued queries. Thus, authors focus on identifying *task-oriented sessions* from past issued queries, i.e., sets of possibly non-contiguous queries phrased by users for carrying out various tasks.
First, they built, by means of a manual labeling process, a *ground-truth* where the queries of a real query log have been grouped in tasks. The analysis conducted by authors on this ground-truth shows that users tend to perform more than one task at the same time, since about 75% of the submitted queries involve a multi-tasking activity.
Moreover, authors define the *Task-oriented Session Discovery Problem* (TSDP) as the problem of best approximating the above ground-truth. The TSDP deals with two aspects: *(i)* a robust measure of the *task relatedness* between any two queries, i.e., *task-based query similarity*, and *(ii)* an effective method for actually discovering task-oriented sessions by using the above measure of task relatedness. Concerning *(i)*, we propose and compare both *unsupervised* and *supervised* approaches for devising several task-based query similarity functions.
These functions also exploit the collaborative knowledge collected by *Wiktionary* and *Wikipedia* for detecting query pairs that are not similar from a lexical content point of view, but actually *semantically*

related. Therefore, authors tackle *(ii)* by introducing a set of *query clustering* methods that exploit the above similarity functions for detecting user tasks.
All the proposed solutions have been evaluated on the ground-truth, and two of them have been shown to perform better than state-of-the-art approaches.

The fifth contribution (App. E) outlines the requirements for a reliable management in crowd-computing environments.
Open Web-based and social platforms dramatically influence models of work. Today, there is an increasing interest in outsourcing tasks to crowdsourcing environments that guarantee professional processing [17].
Crowdsourcing follows the "open world" assumption allowing humans to provide their capabilities through the platform by registering themselves as members. While conventional enterprise systems rely on well-defined in-house knowledge of their processes and services, i.e., on established policies, crowdsourcing has a more loosely-coupled, dynamic, and flexible structure and depends especially on the preferences and behavior of the individual crowd members. Hence, it is necessary to monitor, log, and extract knowledge on the behavior of the crowd members to derive an adaptive, optimized usage.
Crowd behavior can lead to an incomplete and unsatisfactory task state at deadline. As a consequence, meeting promised service contracts is challenging and demands for sophisticated management techniques of a crowd platform.
The challenge is to gain the crowd customer's confidence by organizing the crowd's mixture of capabilities and structure to become reliable. The prerequisite is a monitoring infrastructure that updates a crowd-based resource model. Authors focus on an agreement model derived from the collected knowledge and combined with an adaptation approach for reliable service usage and task execution.
This model integrates crowdsourcing with the well-known concept of SLAs. For SLAs, independent from the agreed parameters of quality, the two common categories are hard- and soft-constraints. In this work, for crowdsourcing authors will distinguish between criteria that must be met, e.g., expertise area of crowd members and their principal participation interest and soft constraints that are used for ranking potential crowd members, including their capacity, reputation, and costs. These categories combined with on-line feedback data from behavior monitoring will allow to adapt and optimize assignments according to the agreements and the current crowd status. Eventually, authors show an integration of observed crowdsourcing in a SOA environment and the related enforcement of agreements by the WSLA standard.

Chapter 5

Conclusions

In this deliverable we presented five different contributions regarding the exploitation of usage data coming from Service Oriented Architectures or real world Web search engines. Such knowledge is mostly derived by using data mining techniques. Five possible applications of this derived knowledge is presented here.

First, we introduced a resource allocation mechanism that uses semantic annotated historical data to enable adaptation mechanisms of the run-time environment. The contribution shows the importance of using historical data for service and cloud providers. Furthermore, the approach demonstrates the coupling of semantic data processing with data mining as a promising novel combination.

Secondly, two well-known data mining techniques have been applied to real event logs coming from a event-based service-oriented architecture. The contribution focuses on process mining as a specific instance of the more general sequential pattern mining problem. The aim of this contribution is to detect frequent sequential patterns that might be present in actual traces of service executions recorded in event logs. To this end, two sequential pattern mining algorithms have been applied to a real event log provided by the Vienna Runtime Environment for Service-oriented Computing, (i.e., VRESCo). The results obtained show that it is possible to find services that are frequently invoked together within the same sequence. Such knowledge could be useful at design-time, when service-based application developers could be provided with service recommendation tools that are able to predict and thus to suggest next services that should be included in the current service composition.

In the third contribution we presented an efficient, novel solution to measure API coverage of service-based systems implemented with Web services. User-specified domain partitioning allows for the definition of customizable and reusable API coverage metrics. The proposed end-to-end framework TeCoS can be easily plugged into existing service execution engines to log service invocations, calculate coverage data and render the results in a Web UI. This user-friendly extension remarkably reduces the required development effort and is a step towards meaningful coverage data and schema-based validation of invocation parameters. The performance and scalability of the proposed approach are successfully evaluated within the experimentation.

The fourth contribution discussed a technique for splitting into meaningful user sessions a very large, long-term log of queries submitted to a Web Search Engine (WSE). The contribution formally introduced the Task-based Session Discovery Problem as the problem of extracting from a stream of users queries several subsequences of queries which are all related to the same search goal, i.e., a Web-mediated task. The contribution also proposed a clustering-based solution, leveraging distance measures based on query content and semantics, while query timestamps were used for a first pre-processing breaking phase. In particular, the proposed technique exploited both Wikipedia and Wiktionary to infer the semantics of a query. The novel graph-based heuristic, namely QC-htc, which is a simplification of the weighted connected components QC-wcc, significantly outperforms other heuristics in terms of F-measure, Rand and Jaccard index.

In the last contribution a framework for successfully managing task assignment in a crowd-computing

environment is presented. The framework solves the problem of managing the task assignment with an adaptive and multi-objective task scheduling. The environment is based on a large-scale SOA infrastructure. The extension to one of the existing standards for agreements (WSLA and WS-Agreement) which includes assignment identifying information and relation to different objectives, fits the requirements of our crowd-computing scenario. When deploying the assignment as independent and dependent tasks to capable members, these objectives can than be used as soft- or hard-constraints for a weighted scheduling strategy. The results highlight the advantages of an objective-aware metric ordered strategy in contrast to plain random scheduling while task loads remain in between the boundaries. Nevertheless, the results show that the effort for ordering the assignment lists induces a higher effort in scheduling.

Bibliography

[1] Integration framework baseline, s-cube deliverable cd-ia-3.1.1, 2009.

[2] Anton Michlmayr and Florian Rosenberg and Philipp Leitner and Schahram Dustdar. Advanced Event Processing and Notifications in Service Runtime Environments. In *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08)*, 2008.

[3] Anton Michlmayr and Philipp Leitner and Florian Rosenberg and Schahram Dustdar. Publish/-Subscribe in the VRESCo SOA Runtime. In *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08)*, 2008. invited demo paper.

[4] R. Baeza-Yates. Applications of web query mining. *Advances in Information Retrieval*, pages 7–22, 2005.

[5] R. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *Proc. KDD'07*. ACM, 2007.

[6] J. Ejarque, A. Micsik, R. Sirvent, P. Pallinger, L. Kovacs, and R. M. Badia. Semantic resource allocation with historical data based predictions. In *The First International Conference on Cloud Computing, GRIDs, and Virtualization, CLOUD COMPUTING 2010*, 2010.

[7] W. Hummer, O. Raz, and S. Dustdar. Towards efficient measuring of web services api coverage. In *3rd International Workshop on Principles of Engineering Service-Oriented Systems (PESOS), co-located with ICSE 2011*, 2011.

[8] R. Jones and K. L. Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In *CIKM '08*, pages 699–708. ACM, 2008.

[9] H. Kochar. Business activity monitoring and business intelligence, 2005.

[10] C. Lucchese, S. Orlando, R. Perego, F. Silvestri, and G. Tolomei. Identifying task-based sessions in search engine query logs. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 277–286, New York City, NY, USA, 2011. ACM Press.

[11] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[12] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo. *IEEE Transactions on Services Computing*, 3:193–205, July 2010.

[13] F. M. Nardini, G. Tolomei, P. Leitner, F. Silvestri, and S. Dustdar. Mining lifecycle event logs for enhancing service-based applications. Technical Report N. /cnr.isti/2010-TR-017, CNR ISTI Pisa Italy, 2011.

[14] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11):38–45, November 2007.

[15] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *Proceedings of the 1st international conference on Scalable information systems*, InfoScale '06, New York, NY, USA, 2006. ACM.

[16] B. Piwowarski and H. Zaragoza. Predictive user click models based on click-through history. In *Proc. CIKM'07*. ACM, 2007.

[17] H. Psaier, F. Skopik, D. Schall, and S. Dustdar. Resource and agreement management in dynamic crowdcomputing environments. In *2011 15th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2011)*. IEEE Computer Society, 2011.

[18] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In *Proc. KDD'05*. ACM Press, 2005.

[19] A. Scime. *Web Mining: Applications and Techniques*. IGI Publishing Hershey, PA, USA, 2004.

[20] F. Silvestri. Mining query logs: Turning search usage data into knowledge. *Foundations and Trends in Information Retrieval*, 1(1-2):1–174, 2010.

[21] J. Teevan, E. Adar, R. Jones, and M. A. S. Potts. Information re-retrieval: repeat queries in yahoo's logs. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 151–158, New York, NY, USA, 2007. ACM.

Appendix A

Semantic Resource Allocation with Historical Data Based Predictions

Semantic Resource Allocation with Historical Data Based Predictions

Jorge Ejarque*, Andras Micsik‡, Raül Sirvent*, Peter Pallinger‡, Laszlo Kovacs‡ and Rosa M. Badia*†
*Grid Computing and Clusters Group - Barcelona Supercomputing Center (BSC), Barcelona, Spain
†Artificial Intelligence Research Institute - Spanish National Research Council (IIIA-CSIC), Barcelona, Spain
‡Distributed Systems Department - Computer and Automation Research Institute
Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
{jorge.ejarque, raul.sirvent, rosa.m.badia}@bsc.es, {micsik, pallinger, kovacs}@sztaki.hu

***Abstract*—One of the most important issues for Service Providers in Cloud Computing is delivering a good quality of service. This is achieved by means of the adaptation to a changing environment where different failures can occur during the execution of different services and tasks. Some of these failures can be predicted taking into account the information obtained from previous executions. The results of these predictions will help the schedulers to improve the allocation of resources to the different tasks. In this paper, we present a framework which uses semantically enhanced historical data for predicting the behavior of tasks and resources in the system, and allocating the resources according to these predictions.**

***Keywords*-multi-agent, semantics, scheduling, resource allocation, historical data, predictions, grid computing, cloud computing, distributed systems.**

I. INTRODUCTION

The Service-Oriented Computing (SOC) paradigm [1]. relies on the composition of services to build distributed applications by using basic services offered by third parties. Those services are offered by service providers that create their description and their implementation. From the business point of view, the service provider agrees with its customers the Quality of Service (QoS) and level of service through a Service Level Agreement (SLA). The fulfillment or violation of the SLAs indicate the grade of satisfaction of customers with the Service Provider (SP), affecting directly or indirectly to the benefit of these provider. One of the most common SLA violation happens when unexpected events such as failures appear and the system is not able to adapt to this change.

Service adaptation is a current research topic, addressing the automatic reactions to unanticipated events. Adaptation mechanisms usually get active when something already went wrong. However, adaptation can be used also when we anticipate a problem to occur. Prediction mechanisms can help to warn about statistically probable unwanted situations, or we can just calculate the likelihood of certain service parameters. The simple aim is to learn from the past, in order to project events happened in the past to the future. There are several software toolkits supporting similar calculations in the area of data mining, etc.

In this paper, we introduce a generic framework for prediction and adaptation, and describe its application in a specific scenario (Cloud resource scheduling). The basic requirements for such an environment are:

- to provide generic capability of collecting log data about internal events,
- to unify the collected data so that global coherence can be revealed,
- to provide customizable methods for getting predictions based on the collected data,
- to feedback predictions into the realization of the scheduling and adaptation mechanisms.

This framework combines prediction techniques with semantic technologies, which introduces semantic knowledge to the data evaluated by predictors, and multi-agent systems, which introduce the pro-activity and distributed problem solving for increasing scalability, adaptability and self-management of the system. Predictions extracted from the semantic historical data are taken into account by a group of agents for allocating different customer's jobs in the most reliable resources.

The paper is organized as follows: Section II gives an overview of the architecture of our solution; Section III is focused on the implementation of job predictions and their usage in the resource allocation process; Section IV evaluates the framework; Section V compares our proposal with the related work; and Section VI concludes the paper.

II. ARCHITECTURE

Figure 1 shows the architecture of our proposed framework. It depicts a resource allocator distributed across multiple agents based on the Multi-Agent Resource Allocation (MARA) approach [2] whose decision are based on predictions based on historical data. There are two differentiated parts in the architecture: the part which is related to the management of jobs and the part which is in charge of resource provisioning. The job management part allows the customers to make all the actions related to their jobs (submit, cancel, etc), while the resource provisioning part allows the system administrator to add and remove resources.

Both parts are built on top of a JADE agent platform [3]. The platform can be distributed across multiple locations deploying containers on each of them. Moreover, it

 ISBN: 978-1-61208-001-7

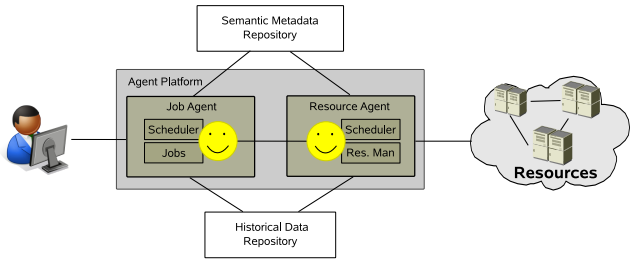


Figure 1. Semantic Resource Allocation with predictions

implements a messaging system, which allows the communication between agents located on different containers. The distributed configuration of the agent platform can improve the scalability of the system because the different parts can be processed in parallel on multiple hosts.

Customers jobs and resources are represented in the system by software agents. Job Agents are in charge of managing the customers jobs and Resource Agents are in charge of managing the providers resources. Moreover, the scheduling of the jobs in the different resources is made by an agreement reached from a negotiation between a Job Agent and different Resource Agents.

Apart from the job management and resource provisioning parts, the system architecture contains a Semantic Metadata Repository (SMR), which contains the semantic resource description registered in the system, and the Historical Data Repository (HDR), which contains semantically annotated logs from system events such as job executions, failures and other monitoring data, which is important to make predictions for current jobs. All the data stored in those semantic repositories is described according to a shared ontology providing a common framework for semantic data. The following paragraphs are focused on the most important part of the architecture.

A. Scenario Ontology

One of the most important issues about semantic technologies is the ontology, which models a common understanding of the concepts used on a scenario of study. In this case, we require an ontology for modeling the system entities such as customers, service providers, jobs and resources and other important data such as resource allocation data for job scheduling and historical data for making predictions.

The Grid Resource Ontology (GRO) [4] provides a model where the most important concepts and entities are described. This ontology provides a definition of the customer jobs and resources but it lacks concepts for describing resource allocation and historical data. Therefore, the GRO has been extended in the scope of the BREIN project [5] in order to cover the mentioned gaps.

The gap of resource allocation data in ontologies was studied in our previous work [6]. Resource requirements have been introduced in the GRO as a set of required abstract *GRO Resources* in the *GRO Task* definition, which models an abstract job. This definition has been also extended with time constraints such as expected duration, deadline, earliest possible start, etc. The scheduling result (assigned resources and time slot) has been introduced on the *GRO Process*, which incarnates the *GRO Task*. Required resources are linked to the task definition as resource sets. A resource set is typically a host, a container of more detailed resource descriptions (disk, CPU, etc.) with resource properties (e.g. size of disk). Furthermore, tasks are also linked to their representing agents and to related business information (such as customer data, service provider, agreed SLA) via the BREIN Business Ontology. These extensions provide the possibility for multi-scope description of job executions merging data about technical capabilities, schedules and business aspects into a single model. It is future work to fully exploit the new capabilities provided by this model. In this paper we provide the first steps in this direction.

GRO Process and *Task* descriptions are also useful for historical data. When a job has ended, the *GRO Process* description contains the result of this job (start time, end time, final status, ...) and the *GRO Task* (abstract job) contains the requested resources, the expected duration and the scheduled start and end times described. The GRO extension for describing historical data also contains classes for describing resources used by each task and resource downtimes. The comparison and evaluation of this data can be used for making predictions about how different types of jobs will run on different resources.

B. System Agents

Our system contains two types of system agents for managing jobs and resources. These agents have been implemented using a Belief-Desire-Intention (BDI) model [7]. For each type of agent, a set of data (Beliefs), goals (Desires) and plans (Intentions) are defined to model the behavior of the agent. Depending on the values of the data, events and the active goals on each moment, the BDI engine decides which plans has to execute the agent for reaching its goals. Our BDI agents have been developed with the Jadex framework [8] used on top of the JADE platform. The following paragraphs give more details about the job and resource agent functionalities and how the BDI model is applied for implementing them.

1) Job Agent (JA): The JA has the main goal of executing jobs in the resources of the system. This execution has different states, some of them indicate that the execution is running correctly and other ones indicate problems in the execution. Depending on the state of the job execution, the JA has to act in a different way. For this reason, the main goal of the JAs has been separated in several subgoals, which will be activated depending on the job status. The activation of subgoals triggers the execution of the plans

 ISBN: 978-1-61208-001-7

to achieve them. For instance, when a new job execution is requested, the JA activates the goal for negotiating a resource allocation. In the *running* state, the JA activates the goal for monitoring the job execution evaluating if the required performance is fulfilled. Finally, if the job is *finished* the JA execute plans for deallocating the resource.

The JA has also to recover itself from status, which can alter the normal execution of the job (*stopped, suspended, non scheduled*). In those situations, the JA will execute plans to resubmit the job or to look for new resources.

2) Resource Agent (RA): The main goal of a RA is the management of resource capabilites for executing the jobs requested by the customer according to resource capabilities and the status and number of jobs assigned to the resources controlled by the agent (jobs *scheduled* and *running*). To provide this main feature, a set of subgoals and plans have been defined to negotiate resource allocations with JAs.

Apart from the negotiation subgoal, RAs contain two subgoals for monitoring the scheduled jobs and running jobs assigned to their resources. These subgoals are in charge of initiating the job execution depending on the planned start time or canceling a job if the deadline has been reached and new jobs are waiting for execution.

Additionally, the RA is also prepared to react to resource failures. The plan for recovering a failure sends a stopped notification message to JAs whose jobs were scheduled at the failed resources. Similarly, it also sends a suspended notification message to JAs whose jobs were already running. JAs treat these notifications according to the plans explained in the JA part.

C. The Historical Data Repository (HDR)

The HDR is a flexible, generic component implemented by SZTAKI to provide facilities for log data collection and on-demand predictions. The HDR is implemented in Java and can be embedded into Java code or can be run as a separate Web Service. In both cases, other software components can send their log data to the HDR, either by our customizable client APIs or via direct calls. Incoming information must be in the format of RDF [9], based on the core ontologies available to all components. The conversion to RDF can be implemented in client APIs if necessary. Thus, in our scenario, the service collects and stores information about past events (i.e., job executions and resource usage), and provides mining and searching for these mentioned past events.

The collected status information is stored as RDF inside the HDR. The repository can then be used for extracting statistics- and knowledge-based information or predictions. In its simple form, the repository can answer SPARQL queries to provide historic information. Clients can use the extended SPARQL provided by Jena ARQ interface [10], and ask for various aggregations of data, such as average, maximum, minimum. Additionally, Pellet [11] or the Jena built-in reasoner can be applied on the data for simple inferencing. For example, the use of subclasses can help to flexibly select a range of resource types for querying.

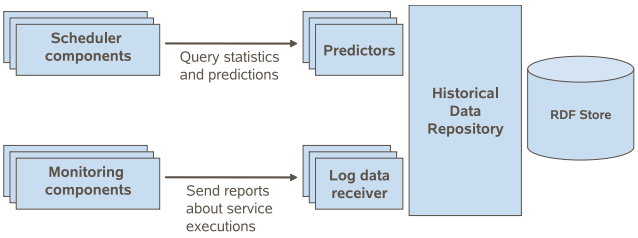


Figure 2. Historical Data Repository overview

For application-specific querying purposes a general plug-in mechanism has been developed. Predictors can be installed as plug-ins for HDR to answer specific questions. Within HDR, the RDF storage is coupled with the Weka data mining software [12]. In this way, a predictor can use both semantic querying and statistical methods. For example, a predictor can build a classification model on top of the results of a semantic query. The classifier can then be used to provide predictions based on the past.

In our specific scenario, the HDR is loaded with the description of the executed jobs, their resource usage and the resource downtimes. This data is used to train the classifier in order to provide estimations on the probabilities of delays in the schedule and problems with the job completion and estimations on the reliability of the used resources. As the statistical model is periodically updated, the provided values dynamically reflect the experiences of the recently finished executions.

D. Semantic Resource Allocation Process

The resource allocation for a particular job is decided between the JA and a set of RAs using the Contract Net Protocol (CNP) [13]. Figure 3 shows the message exchange between these agents for coordinating the job scheduling. When a JA has activated a goal for allocating resources to a job, it sends a query to the SMR to get the RAs whose resources match with the job requirements (1). Afterwards, the JA initiates a negotiation sending to the selected RAs a call for scheduling proposals (2). Each RA makes its own proposal and returns it to the JA (3). The JA evaluates all proposals, accepts the best for its interest and rejects the rest (4).

The RA proposals and the JA evaluations are done using an agent scheduler module. It is based on a rule engine evaluating a set of scheduling rules over semantic metadata bound to the GRO ontology (Section II-A). Despite of both agents using the same module, they behaves in a different way because they are loaded with different information and rules. The RA makes the scheduling proposals evaluating

 ISBN: 978-1-61208-001-7

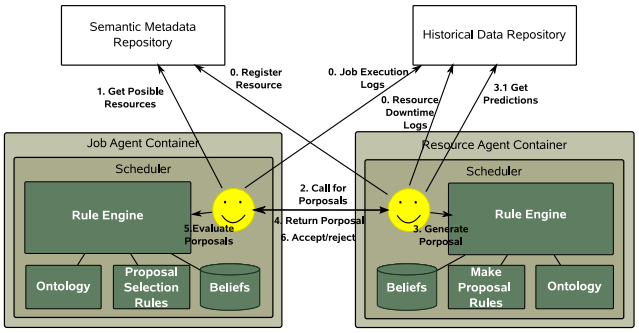


Figure 3. Distributed Semantic Scheduling

scheduling rules over the resource and assigned job descriptions, while the JA evaluates the scheduling proposal according to the customer rules and job description.

Once the RA has obtained a scheduling for a job in a resource, the RA consults the HDR to obtain predictions for this resource allocation (3.1). Resource allocation predictions are attached to the scheduling solution inferred by the scheduler to create a scheduling proposal, which will be evaluated by the JA. The following section gives more details about this process.

III. Job Prediction in Resource Allocation Process

In order to learn from past experiences, agents use the HDR as a service. In our HPC scenario, a separate HDR web service is applied for each Service Provider. This ensures that private internal data (such as log of execution details) remain within the boundaries of the provider, but also creates a merged knowledge base across various HPC clusters of the provider. A new plug-in was developed for the HDR, which is responsible for providing statistics and predictions for the scheduling agents. This predictor plug-in works on the basis of the common GRO explained in Section II-A containing the descriptions of hosts, software, host capabilities and QoS metrics.

When a new job is submitted to the system, it is described semantically indicating time constraints and the collection of resource requirements. During resource allocation process (described in Section II-D), the job is scheduled in the available resource and the results of this process are attached to the job description. During job execution, the JA monitors the execution introducing the relevant data in the job description (completion status, start time, end time and resource usage). Once the job has finished, the JA sends the job execution results to the HDR component using the HDR API client (Fig. 3, step 7). An example of this job execution report is shown in the following lines.

```
<rdf:Description rdf:about="gro:job-0">
  <gro:hasActionState rdf:resource="tech:done"/>
  <gro:hasResourceSet rdf:resource="gro:requirement_job-0"/>
  <rdf:type rdf:resource="gro:Execute_Task"/>
  <tech:hasDeadline>2010-06-14T19:41:07</tech:hasDeadline>
  <tech:expectedDuration>20</tech:expectedDuration>
  <gro:isExecutedAt rdf:resource="tech:host_1"/>
  <gro:incarnatedToProcess rdf:resource="gro:Incarnated_job-0"/>
</rdf:Description>

<rdf:Description rdf:about="gro:Incarnated_job-0">
  <rdf:type rdf:resource="gro:ProcessInstance"/>
  <tech:hasPlannedStart>2010-06-14T19:21:07</tech:hasPlannedStart>
  <tech:hasPlannedEnd>2010-06-14T19:41:07</tech:hasPlannedEnd>
  <gro:usesResource rdf:resource="tech:host_1"/>
  <gro:incarnatedFromTask rdf:resource="gro:job-0"/>
  <tech:hasActualStart>2010-06-14T19:21:07</tech:hasActualStart>
  <tech:hasActualEnd>2010-06-14T19:39:07</tech:hasActualEnd>
</rdf:Description>

<rdf:Description rdf:about="tech:usage_host_1_job-0">
  <tech:hasMeanMemoryUsage>25696.0</tech:hasMeanMemoryUsage>
  <tech:hasMeanCPUUsage>41720.0</tech:hasMeanCPUUsage>
  <tech:hasEnd>2010-06-14T19:39:07</tech:hasEnd>
  <tech:hasStart>2010-06-14T19:21:07</tech:hasStart>
  <tech:hasResource rdf:resource="tech:host_1"/>
  <tech:hasJob rdf:resource="gro:job-0"/>
  <rdf:type rdf:resource="tech:Usage"/>
</rdf:Description>
```

On the other hand, RAs report to the HDR about resource downtimes (Fig. 3, step 8). Job execution and resource downtime reports build up an extended log of actions inside the SP, which will be used as a knowledge base for generating the job predictions.

```
<rdf:Description rdf:about="tech:Downtime_host_1_1">
  <tech:hasEnd>2010-06-13T19:52:05</tech:hasEnd>
  <tech:isCausedBy rdf:resource="tech:powerOut"/>
  <tech:hasStart>2010-06-13T19:44:05</tech:hasStart>
  <tech:hasResource rdf:resource="tech:host_1"/>
  <rdf:type rdf:resource="tech:DownTime"/>
</rdf:Description>
```

Based on these data, a statistical classifier is trained, which can provide estimations on the probability of delays in the schedule, probability of problems with the job completion and on the reliability of the used resources. The relevant data for creating the statistical model is obtained from the HDR RDF store by means of SPARQL queries. For instance, we get the number of resource failures for the different types of resources and jobs from making the predictions of resource reliability and the job failure probability. The planned start and end times can be compared with the real start and end time for calculating delays on the scheduling and job duration. The statistical model is periodically updated with the historical data of recent execution. In this way, predictions dynamically reflect the experiences of past executions in the defined time window. The mechanism demonstrates the coupling of semantic data processing with data mining, as a promising novel combination.

Predictions are queried by the RA during the resource allocation process in order to make its scheduling proposal. The HDR provides the probability of delays in the job schedule (assigned host and time slot) inferred by the RA, the probability of problems during execution based on past executions with similar descriptions on similar hosts, and the reliability of the assigned host. The RA introduces these job predictions in the scheduling proposals and sends them to the

 ISBN: 978-1-61208-001-7

JA in order to be analyzed according to customer rules. In this case, the JA calculates reliability cost for the proposed scheduling based on a pondered mean of the predictions provided by the HDR. Once the job reliability cost for each proposal is analyzed, the JA will select the most reliable host, which fullfills the job constraints.

IV. Evaluation

In our experiments, we used the data from HPC resources of Barcelona Supercomputing Center (BSC). The test environment contained 8 hosts running 2 types of software. A HDR plug-in was implemented for the specific prediction tasks of the experiments. The plug-in at start-up extracts the necessary data using a SPARQL query:

```
SELECT ?task ?plannedStart ?plannedEnd ?start ?end
?resource ?status ?mem ?cpu WHERE{
  ?proc rdf:type gro:ProcessInstance .
  ?proc gro:usesResource ?resource .
  ?proc gro:incarnatedFromTask ?task .
  ?task gro:hasActionState ?status .
  ?proc tech:hasActualStart ?start .
  ?proc tech:hasActualEnd ?end .
  ?proc tech:hasPlannedStart ?plannedStart .
  ?proc tech:hasPlannedEnd ?plannedEnd .
  ?usage tech:hasJob ?task;
  ?usage tech:hasMeanMemoryUsage ?mem;
  ?usage tech:hasMeanCPUUsage ?cpu .
  FILTER (?start > "2010-01-01T00:00:00")
}
```

The data selected for this case includes the scheduled start and end for the job, the actual start and end times, the host where the job was run, the status of job completion and the mean memory and CPU usage of the job. Then, the necessary Weka data structure is built, and the classifiers are configured and run. Further data arriving during the predictor is in use can be added incrementally to the plug-in.

The expected duration to complete the job was calculated using various classifiers built into Weka. The best results were achieved by the M5P regression tree and the KStar instance-based classifier. The mean absolute error for the predicted value was 43 and 74 seconds respectively, where the actual value range was between 240 and 2400 seconds (i.e. the mean error is 3% of the mean predicted value).

Similarly, the status field can be used to predict the probability of failure. In this respect, Bayes-style, decision tree and decision rules algorithms were equally good at classifying the job completion status correctly for 85% of the job executions.

Furthermore, agents could assess the reliability of hosts by asking their availability metric from the plug-in. The availability is calculated based on the list of downtimes for the host. This function is supported directly by SPARQL queries, without using prediction mechanisms.

The time to load the predictor in the experiments can be broken up to the time needed for fetching the necessary data from the semantic log store (SPARQL query), and the time for building the prediction model with Weka. The first action required 1-2 seconds of time, while the second action could be accomplished in about 90 ms for 500 rows of data up to 141 ms for 5000 rows of data. As it was expected, the prediction time was independent of the number of data rows, yielding the result in approximately 15 ms.

We performed further testing of the prediction algorithms with log data available from two public archives: the Parallel Workloads Archive [14], and the Grid Workloads Archive [15]. The mean absolute error of the predictions in this case could be forced below 7% of the mean of the predicted value by careful preprocessing of the data. The comparison with our own data revealed that more details about job execution can yield better predictions in general. In contrast to regression used in cited related work (Section V), we preferred the instance-based learning algorithms, which relate similar job executions with each other for prediction purposes, and thus provided better predictions on job properties.

V. Related Work

Foster et al raised the benefit of integrating the results from agents and grids research areas in [16]. Agents could improve the autonomy, flexibility and scalabity of current grid systems. Regarding the area of resource allocation and job scheduling, the multi-agent system researchers have been already focusing on this problem. Several solutions have been proposed, such as the ones based on market-control where each agent tries to maximize its benefit function and the market controls them, the social wellfare where the multi-agent system tries to maximize a collective benefit and other proposals like game and decision theory. In the market-based solutions, we would like to highlight proposals like Challenger [17], Tycoon [18], other studies more focused on grids such as TRACE [19] or ARAM [20] and also projects such as CatNets [21] or SORMA [22]. The work on wellfare engineering and game theory for multiagents resource allocation has been compiled in [23] and [24] respectively.

All of these solutions implement specific allocation policies for solving a problem with their advantatges and drawbacks, but they are static and cannot be changed easily. In our paper we do not try to offer a new solution for the allocation algorithms, but we have introduced the multi-agent solution in the Semantic Resource Allocation process leaving users the availability of extending or changing the policies. In our system, customers and providers can describe the scheduling rules that are the most convenient for their interests. Those policies will be combined during the negotiation, trying to get a solution which satisfies all the policies.

In this case we have also introduced the capability of taking into account predictions based on semantic historical data about how similar jobs have been executed on the different resources. It allows the user taking into account in their policies the information provided by themselves as

 ISBN: 978-1-61208-001-7

well as information about how the job execution is predicted by the system.

Predictions have been used also in other systems. The work in [25] describes an approach for predicting SLA values during the execution of WS-BPEL processes. At given points in the process, the collected QoS data are fed into a prediction engine, which provides predictions for numerical SLO values using neural networks. Predictions are presented on a graphical user interface and open facilities for manual interaction in the executing process. In [26], the authors present an architecture for event-based collection of historical data, and performing prediction on QoS data in a SOA environment. This approach does not use semantics and its focus on QoS is different than current paper.

VI. Conclusion

The paper emphasizes on the importance of using historical data by service and cloud providers. We present a generic approach and a re-usable solution for the collection and exploitation of historical log data produced by services. The heterogeneity of log data arriving from various resources calls for a semantic data representation, which can facilitate the unification of these data and a query mechanism supported by inference. Our approach demonstrates the coupling of semantic data processing with data mining as a promising novel combination.

Acknowledgment

This work is supported by the Ministry of Science and Technology of Spain and the European Union under contract TIN2007-60625 (FEDER funds), Generalitat de Catalunya under contract 2009-SGR-980 and the European Commission with FP6-IST project 34556 (BREIN) and FP7-ICT project 215483 (S-CUBE).

References

[1] M. Papazoglou and D. Georgakopolous, "Service-oriented computing," *Communications of the ACM*, vol. 46, no. 10, p. 25, 2003.

[2] Y.Chevaleyre, et al, "Issues in Multiagent Resource Allocation," *Informatica*, vol. 30, pp. 3–31, 2006.

[3] "Java Agent Development Framework," http://jade.tilab.com[1].

[4] J. Brooke, D. Fellows, K. Garwood, and C. Goble, "Semantic matching of grid resource descriptions," in *Grid Computing*. Springer, 2004, p. 240.

[5] BREIN Consortium, "Final Report on the BREIN Core Ontologies," BREIN Project, Public Deliverable D3.2.5, 2008.

[6] J. Ejarque, et al, "Exploiting semantics and virtualization for SLA-driven resource allocation in SP," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 5, p. 541, 2010.

[7] A. Rao and M. Georgeff, "BDI agents: From theory to practice," in *The 1st Int. Conf. on Multi-agent Systems*, 1995, p. 312.

[8] "Jadex system," http://jadex.informatik.uni-hamburg.de[1].

[9] "Resource Description Framework," http://www.w3.org/RDF[1].

[10] "ARQ - A SPARQL Processor for Jena," http://jena.sourceforge.net/ARQ[1].

[11] "Pellet OWL reasoner," http://clarkparsia.com/pellet/[1].

[12] I. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann Pub, 2005.

[13] R. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. on Computers*, vol. 100, no. 29, pp. 1104–1113, 1980.

[14] "Parallel workload archive," http://www.cs.huji.ac.il/labs/parallel/workload[1].

[15] "Grid workload archive," http://gwa.ewi.tudelft.nl[1].

[16] I. Foster, N. Jennings, and C. Kesselman, "Brain meets brawn: Why grid and agents need each other," in *The 3rd Int. Conf. on Autonomous Agents and Multiagent Systems*, 2004, p. 15.

[17] A. Chavez, A. Moukas, and P. Maes, "Challenger: A multiagent system for distributed resource allocation," in *The 1st Int. Conf. on Autonomous Agents*, 1997, p. 331.

[18] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. Huberman, "Tycoon: An implementation of a distributed, market-based resource allocation system," *Multiagent and Grid Systems*, vol. 1, no. 3, pp. 169–182, 2005.

[19] S. Fatima and M. Wooldridge, "Adaptive task resources allocation in multi-agent systems," in *The 5th Int. Conf. on Autonomous Agents*, 2001, p. 544.

[20] S. Manvi, M. Birje, and B. Prasad, "An agent-based resource allocation model for computational grids," *Multiagent and Grid Systems*, vol. 1, no. 1, p. 17, 2005.

[21] "CatNets Project," http://www.catnets.uni-bayreuth.de[1].

[22] "Sorma Project," http://www.sorma-project.eu[1].

[23] Y. Chevaleyre, U. Endriss, S. Estivie, and N. Maudet, "Welfare engineering in practice: On the variety of multiagent resource allocation problems," *Engineering Societies in the Agents World V*, p. 335, 2005.

[24] S. Parsons and M. Wooldridge, "Game theory and decision theory in multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 5, no. 3, p. 243, 2002.

[25] P. Leitner, et al, "Runtime Prediction of SLA Violations for Composite Services," in *3rd Workshop on Non-Functional Properties and SLA Management in SOC*, 2009.

[26] L. Zeng, C. Lingenfelder, H. Lei, and H. Chang, "Event-driven QoS prediction," in *6th Int.Conf on SOC*, 2008, p. 147.

[1]Last access to links in the references is August 18th, 2010

 ISBN: 978-1-61208-001-7

Appendix B

Mining Lifecycle Event Logs for Enhancing Service-based Applications

Mining Lifecycle Event Logs for Enhancing Service-based Applications

Franco Maria Nardini[1], Gabriele Tolomei[1,3], Philipp Leitner[2], Fabrizio Silvestri[1], Schahram Dustdar[2]

[1]*ISTI-CNR*
Pisa, Italy
{*firstname.lastname*}*@isti.cnr.it*

[2]*Distributed Systems Group*
Vienna University of Technology, Vienna, Austria
{*lastname*}*@infosys.tuwien.ac.at*

[3]*Ca' Foscari University*
Venice, Italy
{*lastname*}*@dsi.unive.it*

***Abstract*—*Service-Oriented Architectures* (SOAs), and traditional enterprise systems in general, record a variety of events (e.g., messages being sent and received between service components) to proper log files, i.e., *event logs*. These files constitute a huge and valuable source of knowledge that may be extracted through data mining techniques. To this end, *process mining* is increasingly gaining interest across the SOA community. The goal of process mining is to build models without *a priori* knowledge, i.e., to discover structured process models derived from specific *patterns* that are present in actual traces of service executions recorded in event logs. However, in this work we focus on detecting *frequent sequential patterns*, thus considering process mining as a specific instance of the more general *sequential pattern mining* problem. Furthermore, we apply two sequential pattern mining algorithms to a real event log provided by the Vienna Runtime Environment for Service-oriented Computing, i.e., VRESCo. The obtained results show that we are able to find services that are frequently invoked together within the same sequence. Such knowledge could be useful at *design-time*, when service-based application developers could be provided with *service recommendation tools* that are able to predict and thus to suggest next services that should be included in the current service composition.**

I. Introduction

The vast majority of nowadays software-based systems, ranging from the simplest, i.e., small-scale, to the most complex, i.e., large-scale, record massive amounts of data in the form of *logs*. Such logs could either refer to the functioning of the system as well as keep trace of any possible software or human interaction with the system itself. For this reason, logs represent a valuable source of hidden knowledge that can be exploited in order to enhance the overall performances of any software-based system.

Well-known examples of systems that have started trying to improve their performances by analyzing event logs are surely Web Search Engines (SEs). Roughly, SEs are increasingly exploiting past user behaviors recorded in *query logs* in order to better understand people search intents, thus, for providing users with better search experiences. Indeed, by accurately recognizing and predicting actual user information needs, SEs are now able to offer more sophisticated functionalities (e.g., *query suggestion*) as well as better relevant result sets in response to a specific query (e.g., *query diversification*).

Moreover, there are plenty of modern enterprise software systems that need to operate in highly dynamic and distributed environments in a standardized way. Such systems implement their business logic according to the *Service-oriented Architecture* (SOA) principles, thus, assembling their business processes as the composition and orchestration of autonomous, protocol-independent, and distributed logic units, i.e., *software services*.

Service-based systems and applications (SBAs) require proper *run-time* environments where their composing services can be searched, bound, invoked, monitored and managed. Therefore, SBA's run-time support might keep track of what is going on during the whole application lifecycle by roughly recording all such events to log files, i.e., *service event logs*.

Analysis of such service event logs could reveal interesting *patterns*, which in turn might be exploited for improving the overall performances of SOA's run-time frameworks as well as supporting SBA designers during the whole application lifecycle.

The main contribution of this work concerns the application of data mining techniques to a real-life service event log collected by the VRESCo SOA run-time framework. Our aim is to analyze the historical events stored on VRESCo in order to discover software services that are frequently invoked and composed together, i.e., *process mining*.

Although traditional process mining refers to a set of techniques and methodologies whose aim is to distill a structured process description from a set of actual traces of executions recorded in event logs, here we treat it as an instance of the *sequential pattern mining problem*.

The remaining of the paper is structured as follows. Section II describes the information collected by SOA lifecycle event logs, in particular focusing on the VRESCo run-time framework. In Section III, we propose how VRESCo event log may be analyzed for approaching our research challenge. Therefore, Section IV shows the experiments we conduct on a real VRESCo log data set. In Section V, we describe past work that somehow concerns with service event log analysis. Finally, Section VI summarizes the contributions we provide in this work together with any further idea that could be better investigated as future work.

II. SOA Lifecycle Event Logs

In the following, we will discuss SOA lifecycle event logs as they are used in this paper. We then present the lifecycle events emitted by the VRESCo SOA runtime environment as a concrete example used in Section III and Section IV of this paper.

A. Lifecycle Events

Events and complex event processing [11] (CEP) are frequently used tools to document and track the lifecycle of applications in various domains. For instance, in the business domain the idea of business activity monitoring [9] (BAM) uses events to monitor business process performance. Analogously, technical implementations of business processes on top of SOAs (service compositions) are often monitored using CEP. To this end, many service composition engines can be configured to track their current state in event logs. For instance, the Apache ODE WS-BPEL engine triggers a rich model of execution events[1]. Similarly, service compositions implemented using Windows Workflow Foundation can use the .NET tracking service[2] to persist event logs. However, tracking system state via event logs in SOA is not confined to composition engines. For instance, The Vienna Runtime Environment for Service-Oriented Computing (VRESCo) [4] uses events to track not only service compositions, but all entities and interactions in a SOA (services, users, compositions, metadata and interactions).

In its most general form, an event log $\mathcal{E}$ consists of a sequence of n recorded events, i.e., $\mathcal{E} = \langle e_1, e_2, \ldots, e_n \rangle$. Each event $e_i \in \mathcal{E}$ usually contains at least an unique identifier, an event timestamp, the publisher of the event (e.g., the BPEL engine), the subject of the event (e.g., the composition instance that triggered the event), and the event type. Depending on the concrete event type, more detailed information is available. This type-specific information cannot be describe generally, i.e., it is different from event type to event type as well as from system to system. In the following we describe the event types triggered by the VRESCo system as an example of the possibilities provided by event logs.

B. SOA Event Log: VRESCo

VRESCo is an experimental runtime environment developed at Vienna University of Technology. VRESCo is being developed under an open source license, and can be accessed via the project Web page[3]. The project aims at solving some of the research problems identified in [15], e.g., dynamic selection of services based on Quality-of-Service (QoS), dynamic rebinding and service composition, service metadata and event-based services computing.

[1] http://ode.apache.org/ode-execution-events.html
[2] http://msdn.microsoft.com/en-us/library/ms735887(v=vs.85).aspx
[3] http://www.infosys.tuwien.ac.at/prototypes/VRESCo/

In the following we focus on the latter aspect. The foundations of event-based service-oriented computing have been discussed in [3], [5]. In a nutshell, the goals of this earlier work were to track what is going on in a service-based application by constantly triggering events and using CEP to construct meaningful information from those events.

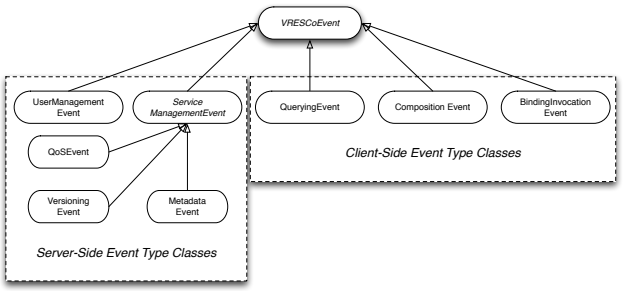


Figure 1. VRESCo Event Type Classes

In a VRESCo system, events of various types are triggered. A simplified taxonomy of event type classes is depicted in Figure 1. As can be seen, events are triggered when services are queried, bound and invoked. Additionally, events indicate if the data or metadata about services changes (e.g., the QoS is changed, new operations are available). Each of the concrete event type classes (those with non-italic name) in turn contains a number of concrete event types that can be triggered. For full details on all events refer to [3].

Events in VRESCo can be triggered either on client- or server-side. While events concerning metadata are triggered by the VRESCo server, all querying and invocation events are triggered by clients and only processed throughout the VRESCo event engine. These client-triggered events are listed in more detail in Table I. In the table, we provide the condition that triggers each event along with the event type and event type class of the event. All of these events provide the basic information discussed above (sequence number, timestamp, ...). In addition, events generally provide some type-specific additional information, which we also summarize in the table. For reasons of brevity, we have omitted composition events. Composition events in VRESCo are of comparable expressiveness as the events triggered by Apache ODE.

The VRESCo event engine stores triggered events in an event log. Therein, events are serialized as XML and can be accessed and analyzed via a RESTful service interface. In Listing 1 we provide an example event serialized to XML. Evidently, the event reflects a service invocation with a very simple input message (`<order>`) as payload.

III. Lifecycle Event Log Mining

In this work, we are interested in exploring how event log collected by the VRESCo framework, i.e., *service event*

Table I
CLIENT-TRIGGERED VRESCO EVENTS

Event Type Class	Event Type	Event Condition	Additional Event Information
BindingInvocationEvent	ServiceInvokedEvent	Specific service is invoked	Message sent to service Invoking user
	ServiceInvocationFailedEvent	Service invocation failed	Message sent to service Triggered fault
	ProxyRebindingEvent	Service proxy is (re-)bound to a new service	Old service New service
QueryingEvent	RegistryQueriedEvent	Registry is queried	Query string
	ServiceFoundEvent	Specific service is found by a query	Query string Query results
	NoServiceFoundEvent	No services are found by a query	Query string

```
<ServiceInvokedEvent xmlns="http://www.vitalab.tuwien.
    ac.at/vresco/usertypes">
  <Priority>0</Priority>
  <Publisher>guest</Publisher>
  <PublisherGroup>GuestGroup</PublisherGroup>
  <UserName>a007b09b-8c23-4fac-af30-0142a61f3795</
      UserName>
  <SeqNum>74006756-64f1-40cb-858e-565d4bc6a94c:24</
      SeqNum>
  <Timestamp>2010-11-09T09:58:49</Timestamp>
  <CurrentRevisionId>180</CurrentRevisionId>
  <CurrentRevisionWsdl>
    http://localhost:60000/AssemblyAtomicServices/
        IAssemblingPlanningService?wsdl
  </CurrentRevisionWsdl>
  <FeatureName>GetPartFeature</FeatureName>
  <InvocationInfo>
    <service_input>
      <order><part1>text</part1></order>
    </service_input>
  </InvocationInfo>
</ServiceInvokedEvent>
```

Listing 1. Serialized Invocation Event

log, could be harnessed for better supporting service-based applications during their whole lifecycle.

Roughly, analysis of VRESCO event log data is finalized to the discovery of *sequences* of services that are frequently invoked together, thus, to detect *processes*, or part of those, as result of the compositions of highly co-invoked services.

This knowledge could be useful for improving the overall performances of the VRESCO run-time framework as well as supporting SBA designers during the whole application lifecycle. As an example, service-based application developers could be provided with *service recommendation tools* that are able to predict and, thus, to suggest next services that should be included in the current service composition at *design-time*.

Due to the huge amount of data collected in the VRESCO event log, data mining techniques represent a suitable approach for addressing our research challenge. In the following, we describe how processes may be mined from the VRESCO event log, namely how sequences of co-invoked services that frequently appear in the event log may be discovered and exploited.

A. Process Mining

According to van der Aalst *et al.* [19], the term *process mining*, also referred to as *workflow mining*, describes a set of techniques and methodologies whose aim is to distill a structured process description from a set of actual traces of executions recorded in event logs.

In our vision, a service log might be viewed as a database consisting of sequences of events that change with time, i.e., a *time-series database* [8]. Such kind of database records the valid time of each data set. For example, in a time-series database that records service invocation transactions, each transaction includes the unique identifier of the invoked service as well as an extra time-stamp attribute indicating when the event happened [24].

Several kinds of patterns can be extracted from various types of time-series data. In this work, we are interested in finding *sequences* of services that are frequently invoked together in a specific order, i.e., *sequential patterns* [2].

Each process instance recorded on event logs might be expressed as an *unrolled* trace of invoked services. Thus, let $S = \{s_1, s_2, \ldots, s_m\}$ be a set of services and let $S^j \subseteq S$ be an itemset of services invoked at the same time (or within a small time window), i.e., $S^j = \{s_{j_1}, s_{j_2}, \ldots, s_{j_k}\}$. Therefore, a process $p_j = \langle S^j_1, S^j_2, \ldots, S^j_{|p_j|}\rangle$ has a unique identifier and represents a sequence of service itemsets, chronologically-ordered according to their time-stamps. Globally, a process database is a sequence database $P = \{p_1, p_2, \ldots, p_n\}$ of executed and recorded processes.

Roughly, a sequential pattern is a sequence of service itemsets that occur *frequently* in P according to a specific order, i.e., appear as subsequence of a large percentage of sequences of P. More formally, a sequence $p' = \langle S'_1, S'_2, \ldots, S'_u\rangle$ is a *subsequence* of $p'' = \langle S''_1, S''_2, \ldots, S''_v\rangle$, i.e., $p' \preceq p''$, if there exists integers $1 \leq i_1 < \ldots < i_u \leq v$ such that $\forall_{1\leq j\leq u} S'_x \subseteq S''_{i_x}$.

Then we may define the support $supp(p')$ of a sequence of service itemsets p' as the proportion of processes in the database P that contains p' as its subsequence, that is:

$$supp(p') = \frac{|\{p_j \mid p' \preceq p_j\}|}{|P|}$$

Therefore, *sequential pattern mining* is the process of extracting certain sequential patterns whose support exceed a predefined minimal support threshold min_supp.

To this end, as a first approach we apply one of the more efficient sequence pattern mining algorithm to the VRESCo event log, namely *PrefixSpan* [16].

Furthermore, we extend our first approach in order to exploit the temporal information associated with each service invocation in a different way. Indeed, in traditional sequential pattern mining, event time-stamps are only used for establishing the chronological order between service invocations, i.e., for simply stating that service s_i is invoked *before* service s_j. However, observing that s_i and s_j are invoked really closed to each other, e.g., within 5 seconds, rather than noting that s_i and s_j are farther away from each other, e.g. 5 minutes, could lead to different conclusions. Thus, we apply another sequential pattern mining algorithm, which is able to deal with this issue, i.e., *MiSTA* [7].

Finally, in Section IV, we describe the different results we obtain on the VRESCo event log when using the two approaches described above.

IV. Experimentation

In order to test our claim about finding frequent sequential patterns inside service event logs, we use a real-life log of events collected by the VRESCo runtime framework.

This event log consists of 89 transactions. Each transaction is in turn composed of several events according to the ones described in Table I (e.g., ServiceInvokedEvent, ServiceInvocationFailedEvent, etc.).

For the sake of our purposes, namely for discovering sequences of services that frequently are invoked together, we only consider the list of *ServiceInvokedEvent* for each transaction.

Firstly, we run the *PrefixSpan* algorithm [16] on the VRESCo data set. We use several thresholds on the *minimum support* of the sequential patterns to be extracted, ranging from 20% to 100%. However, the maximum support for frequent sequences found in our data set is 66%.

The following figures show the distribution of the lengths of extracted sequence patterns, i.e., the number of invoked services that are inside a frequent sequence. In particular, Figure 2, Figure 3, and Figure 4 show the results obtained by using a minimum support threshold of 25%, 50%, and 66%, respectively.

As one would expect, independently from the minimum support chosen, 2-length sequences are the most popular since for each k-length frequent sequences ($k > 2$) any i-length subsequence, i.e., $2 \leq i < k$, is also frequent by definition. Moreover, as the minimum support threshold increases, the maximum length of frequent sequences decreases as well from 17 to 11, and, also, most popular frequent sequences result to be globally shorter.

Finally, on average frequent sequences are composed of 5.86, 4.60, and 4.07 services for minimum support values of 25%, 50%, and 66%, respectively.

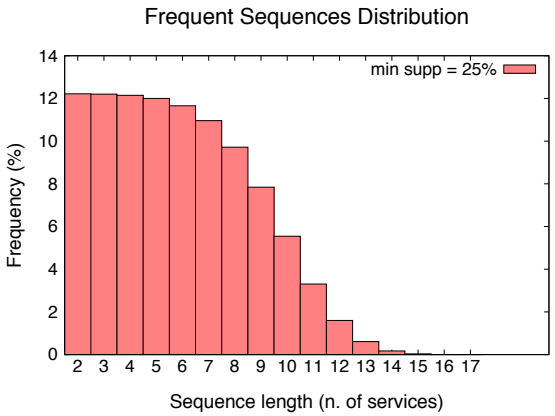


Figure 2. PrefixSpan [*min_supp* = 25%]: Pattern length distribution.

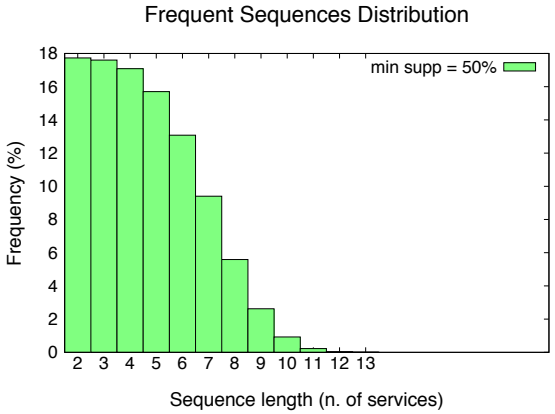


Figure 3. PrefixSpan [*min_supp* = 50%]: Pattern length distribution.

As a second step of our experimental phase, we also run the *MiSTA* algorithm [7] on the VRESCo event log. This algorithm differs from classical sequential pattern mining algorithms like *PrefixSpan* because it also takes care of the time gaps between consecutive items in a sequence. In other words, *MiSTA* extracts frequent sequences by considering not only the *minimum support* but also another parameter, i.e., τ, which basically represents a threshold on the time-gap between pairs of consecutive items.

In our experiments, we use several combinations of such two parameters, namely min_supp and τ. In the following, we describe the distribution of frequent sequence size obtained by using two values for the minimum support, i.e.,

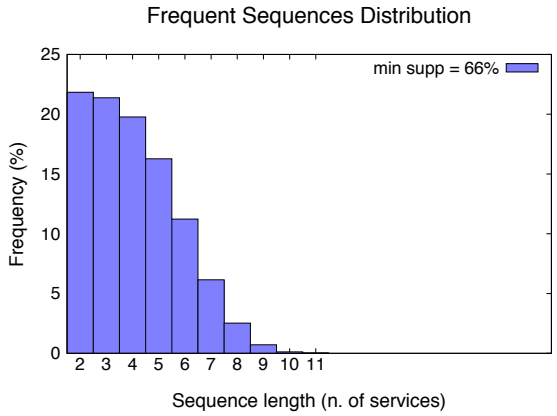


Figure 4. PrefixSpan [*min_supp* = 66%]: Pattern length distribution.

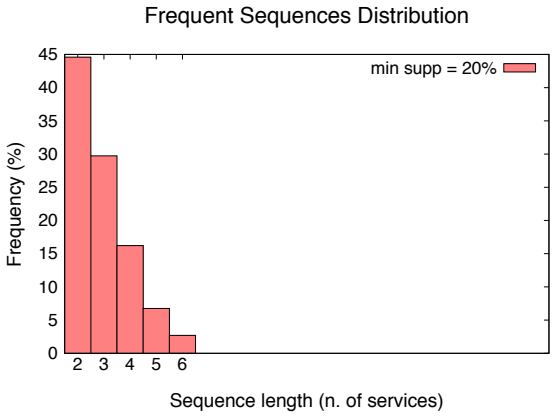


Figure 5. MiSTA [*min_supp* = 20%, τ = 5 sec.]: Pattern length distribution.

20% and 32%, and three values for the time threshold τ, i.e., 5, 60, and 300 seconds, which are shown in Figure 5, 6, 7, 8, 9, 10, respectively.

The total number of frequent sequences found when $min\_supp = 20\%$ is 33, both for $\tau = 5$ and $\tau = 60$, whereas it reaches the value of 35 when $\tau = 300$. Moreover, when $min\_supp = 32\%$ the total amount of frequent sequences is 20, both for $\tau = 5$ and $\tau = 60$, whereas it reaches the value of 27 when $\tau = 300$.

Finally, the maximum size of frequent sequences is 7, whereas the average ranges from a minimum of 2.80 to a maximum of 3.07. These results show that, by taking into account the time gap between service invocations using *MiSTA* instead of *PrefixSpan*, we are able to detect less and shorter frequent sequences on average.

We argue that *MiSTA* could provide better results if we previously analyze how time gaps are distributed across consecutive service invocations, and we leave this as a possible future work.

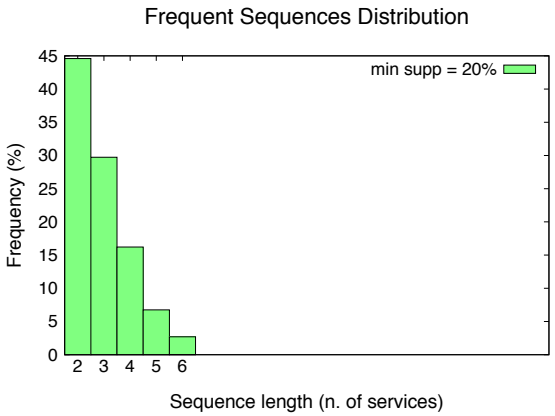


Figure 6. MiSTA [*min_supp* = 20%, τ = 60 sec.]: Pattern length distribution.

V. Related Work

In this paper, we present a use case for event log mining in service-based systems. This idea bears some resemblance to the established idea of business activity management [9] (BAM). BAM considers the event-driven governance of business processes, and is, hence, mostly a term from the business domain. Technically, BAM is enabled by monitoring runtime monitoring of services and their interactions within company SOAs. To this end, event-based monitoring approaches [22], [21], [6] produce a steady stream of low-level lifecycle events, similarly to the lifecycle events discussed in Section II-B and to the event logs produced by VRESCo. These low-level events need to be aggregated so that real business information can be gained from them. Existing techniques to do this include SLA aggregation [18] or event-based SLA monitoring [17], [13]. Related to the ideas of BAM is research work by [14], which considers event-based monitoring of business compliance. Our research, specifically mining for invocation sequences that lead to failed service invocations, is complementary to BAM. While BAM is mostly concerned with discovering failures, our research can be used to identify or predict them in advance.

In literature, events emitted by service-based applications have found various other uses. For instance, [12] uses eventing information (along with various other data sources) to generate visualizations of the past behavior and quality of Web services, mostly to ease management and selection of services for other uses. This is related to our work, which identifies services which have in the past been used together, mostly to suggest suitable services for new uses. [23],

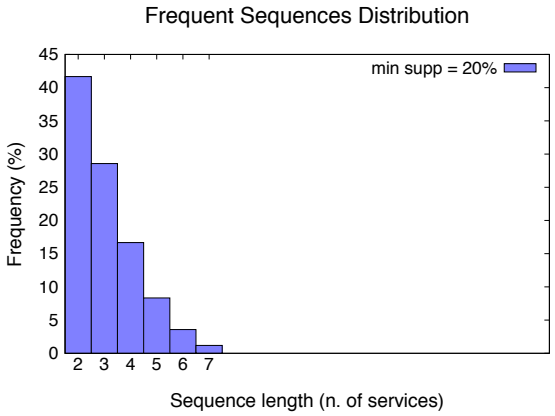


Figure 7. MiSTA [*min_supp* = 20%, τ = 300 sec.]: Pattern length distribution.

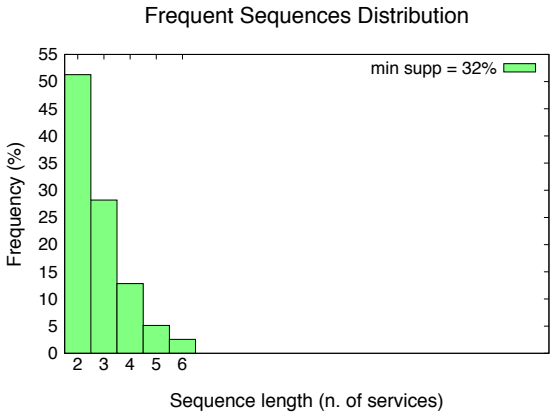


Figure 9. MiSTA [*min_supp* = 32%, τ = 60 sec.]: Pattern length distribution.

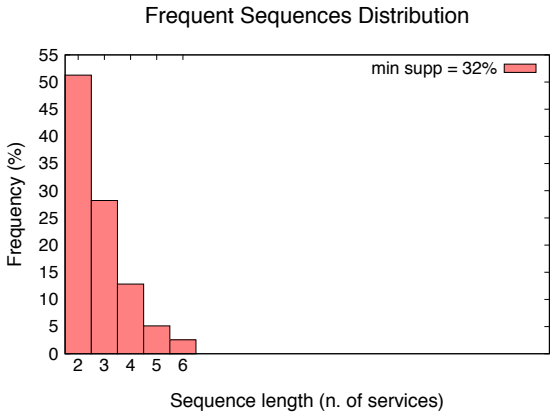


Figure 8. MiSTA [*min_supp* = 32%, τ = 5 sec.]: Pattern length distribution.

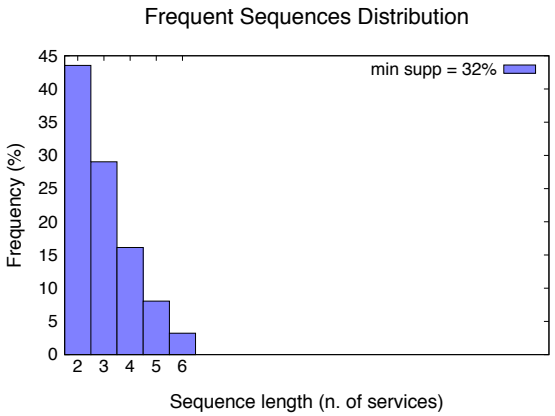


Figure 10. MiSTA [*min_supp* = 32%, τ = 300 sec.]: Pattern length distribution.

[10] have used SOA events to generate predictions of SLA violations. This is done by training machine learning models from the collected event data, and using runtime event information as input to those models.

Finally, our research is related to the idea of process mining [20], [1], [19]. Generally speaking, process mining considers discovering structures (mostly business processes) from traces of earlier executions of information systems. This also includes making implicit processes (that people are subconsciously following) explicit, so that they can be optimized using the techniques of business process reengineering. Our work is different to process mining in the sense that we do not suggest new processes from event logs. Instead, we rather give recommendations which combinations of services have in the past been used together (indicating that it might make sense to use them in combination).

VI. Conclusions and Future Work

Service-Oriented Architectures (SOAs), and traditional enterprise systems in general, record a variety of events (e.g., messages being sent and received between service components) to proper log files, i.e., *event logs*. These files constitute a huge and valuable source of knowledge that may be extracted through data mining techniques.

In this work, we focus on *process mining* as a specific instance of the more general *sequential pattern mining* problem. Basically, our aim is to detect *frequent sequential patterns* that might be present in actual traces of service executions recorded in event logs. To this end, we apply two sequential pattern mining algorithms to a real event log provided by the Vienna Runtime Environment for Service-oriented Computing, i.e., VRESCo.

The obtained results show that we are able to find services that are frequently invoked together within the same sequence. Such knowledge could be useful at *design-time*, when service-based application developers could be provided with *service recommendation tools* that are able to predict and thus to suggest next services that should be included in the current service composition.

Finally, as a future work, we aim at discovering possible sequences of invoked services, which frequently lead to failures or unexpected behaviors. This knowledge could be exploited either for preventing SBA designers to deploy possible faulty service compositions as well as for devising novel run-time adaptation mechanisms in response to undesired events.

ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

REFERENCES

[1] W. M. P. V. D. Aalst, B. F. V. Dongen, C. Günther, A. Rozinat, H. M. W. Verbeek, and A. J. M. M. Weijters, "ProM : The Process Mining Toolkit," in *Industrial Engineering*, vol. 489, Technische Universiteit Eindhoven Eindhoven, The Netherlands. CEUR-WS.org, 2009, pp. 1–4. [Online]. Available: http://prom.win.tue.nl/research/wiki/_media/publications/prom2009bpmdemo.pdf

[2] R. Agrawal and R. Srikant, "Mining sequential patterns," in *ICDE '95*. IEEE, 1995, pp. 3–14.

[3] Anton Michlmayr and Florian Rosenberg and Philipp Leitner and Schahram Dustdar, "Advanced Event Processing and Notifications in Service Runtime Environments," in *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08)*, 2008.

[4] ——, "End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo," *IEEE Transactions on Services Computing*, vol. 3, pp. 193–205, July 2010.

[5] Anton Michlmayr and Philipp Leitner and Florian Rosenberg and Schahram Dustdar, "Publish/Subscribe in the VRESCo SOA Runtime," in *Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08)*, 2008, invited demo paper.

[6] L. Baresi, S. Guinea, M. Pistore, and M. Trainotti, "Dynamo + Astro: An Integrated Approach for BPEL Monitoring," in *Proceedings of the 2009 IEEE International Conference on Web Services (ICWS'09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 230–237.

[7] F. Giannotti, M. Nanni, D. Pedreschi, and F. Pinelli, "Mining sequences with temporal annotations," in *Proceedings of the 2006 ACM Symposium on Applied Computing*, ser. SAC '06. New York, NY, USA: ACM, 2006, pp. 593–597. [Online]. Available: http://doi.acm.org/10.1145/1141277.1141413

[8] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[9] H. Kochar, "Business activity monitoring and business intelligence," 2005. [Online]. Available: http://www.ebizq.net/topics/bam/features/6596.html

[10] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann, "Runtime Prediction of Service Level Agreement Violations for Composite Services," in *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC'09)*, 2009, pp. 176–186.

[11] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[12] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Selective Service Provenance in the VRESCo Runtime," *International Journal of Web Services Research*, vol. 7, no. 2, pp. 65–86, 2008. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-77954090267\&partnerID=40\&md5=10b3d8328dbd86f8a2f623dd1d6cf886

[13] Michlmayr, Anton and Rosenberg, Florian and Leitner, Philipp and Dustdar, Schahram, "Comprehensive QoS Monitoring of Web Services and Event-Based SLA Violation Detection," in *Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing (MWSOC'09)*. New York, NY, USA: ACM, 2009, pp. 1–6.

[14] E. Mulo, U. Zdun, and S. Dustdar, "Monitoring Web Service Event Trails for Business Compliance," in *IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, 2009, pp. 1 –8.

[15] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer*, vol. 40, no. 11, pp. 38–45, November 2007.

[16] J. Pei, J. Han, B. Mortazavi-Asl, and H. Pinto, "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *ICDE '01*. IEEE, 2001.

[17] A. Sahai, V. Machiraju, M. Sayal, A. P. A. van Moorsel, and F. Casati, "Automated SLA Monitoring for Web Services," in *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM)*, 2002.

[18] T. Unger, F. Leymann, F. Leymann, and T. Scheibler, "Aggregation of Service Level Agreements in the Context of Business Processes," in *Proceedings of the Twelfth IEEE Enterprise Distributed Object Conference (EDOC 2008)*, vol. 0, Universit{ä}t Stuttgart, Fakult{ä}t Informatik, Elektrotechnik und Informationstechnik, Germany. IEEE Computer Society, 2008, pp. 43–52. [Online]. Available: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2008-93\&engl=0

[19] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1316839

[20] W. van der Aalst, "Process mining: a research agenda," *Computers in Industry*, vol. 53, no. 3, pp. 231–244, Apr. 2004. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0166361503001945

[21] B. Wetzstein, S. Strauch, and F. Leymann, "Measuring Performance Metrics of WS-BPEL Service Compositions," in *Proceedings of the Fifth International Conference on Networking and Services (ICNS'09)*. IEEE Computer Society, Apr. 2009.

[22] L. Zeng, H. Lei, and H. Chang, "Monitoring the QoS for Web Services," in *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07)*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 132–144.

[23] L. Zeng, C. Lingenfelder, H. Lei, and H. Chang, "Event-Driven Quality of Service Prediction," in *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC'08)*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 147–161.

[24] Q. Zhao and S. S. Bhowmick, "Sequential pattern matching: A survey," 2003.

Appendix C

Towards Efficient Measuring of Web Services API Coverage

Towards Efficient Measuring of Web Services API Coverage

Waldemar Hummer[1], Orna Raz[2], and Schahram Dustdar[1]
[1] Distributed Systems Group
Vienna University of Technology, Austria
{hummer,dustdar}@infosys.tuwien.ac.at
[2] IBM Haifa Research Lab
Haifa University Campus, Israel
ornar@il.ibm.com

ABSTRACT

We address the problem of interface-based test coverage for Web services. We suggest an approach to analyze the Application Programming Interface (API) of Web services, calculate the number of possible input combinations and compare it to the number of actual historical invocations. Such API coverage metrics are an indicator to which extent the service has been used. Measuring API coverage is a key concern for assessing the significance of Verification and Validation (V&V) techniques; on the other hand, API coverage metrics can also yield interesting usage reports for a service-based system in production use. The coverage metrics rely on the exact specification of service interfaces, and we provide a mechanism to specify restrictions for data types in the Java Web services framework (JAX-WS). As full enumeration of all possible inputs is often infeasible, we allow the definition of custom coverage metrics by means of domain partitioning: the user divides domain ranges into subsets, and a coverage of 100% means that the logged invocations contain at least one sample for each subset. Based on a prototype implementation, we evaluate different aspects of our approach.

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Web-based Services

General Terms

Design, Management, Reliability

Keywords

API Coverage Metrics, Web Services, TeCoS Framework

1. INTRODUCTION

In recent years, the Service-Oriented Architecture [15] (SOA) has become a widely adopted paradigm to create loosely coupled distributed systems, and Web services[1] are the most commonly used technology to build SOA. One of the defining characteristics of SOA is that the API (application programming interface) of the available services is published to service consumers using standard, machine-readable description languages. In the case of Web services, this is achieved using the Web Services Description Language (WSDL), paired with XML Schema Definitions (XSD) of the service operations' input and output messages.

Now that the WS-* stack builds a solid technological foundation, traditional software engineering disciplines are being applied to Web services. Among these disciplines is software testing in terms of Verification and Validation (V&V) [6, 16]. An important field in software testing is concerned with test coverage, i.e., the extent to which an implemented system has been tested or used. Classical code-based coverage metrics such as function, statement or branch coverage require access to the source code, which is not always possible for Web services. Therefore, testing methods for Web services can usually only rely on the service's API definition, and hence focus on black-box testing of single services or testing the composition of services [9]. A variety of professional commercial services have evolved around Web services API testing[2] [3] [4], which clearly shows the practical relevance of this field.

API coverage is commonly expressed as the ratio of previously performed, distinct invocations to the number of (theoretically) possible invocations as defined by the API [12]. In the simplest case, single parameter values are varied and tested, e.g., with extreme values, to verify functionality and detect failures. Since an isolated analysis of parameters is often not sufficient, input combinations need also be considered. Achieving an API coverage of (near) 100% in this case is generally subject to the problem of *combinatorial explosion* [16], because all input parameter combinations need to be considered. Hence, it is desirable to restrict the value domain of each parameter to its smallest possible size, in order to minimize the total number of possible combinations. We see two possibilities to achieve that. Firstly, Web service developers need to provide a more precise specification of the valid operation parameters, e.g., in terms of string patterns or allowed numeric ranges. Secondly, service testers may analyze these parameters to identify similarities or combinations that seem less important to be tested. XSD already provides a solution for the first point in the form of *facets*, but expressing such schema-based restrictions is surprisingly hard to achieve in major Web service frameworks such as JAX-WS[5] (Java API for XML Web Services) or Microsoft's .NET platform. Essentially, developers cannot rely on the automated XSD generation, but have to manually define the XML Schema that uses facet restrictions. Concerning the second point, there is still an evident lack for

[1] http://www.w3.org/2002/ws/

[2] http://qualitylogic.com/tuneup/uploads/docfiles/web-api-testing.pdf
[3] http://www.crosschecknet.com/
[4] http://www.stylusstudio.com/ws_tester.html
[5] http://jcp.org/en/jsr/detail?id=224

PESOS '11, May 23-24, 2011, Waikiki, Honolulu, HI, USA

API coverage frameworks with customizable coverage metrics that can be easily plugged into (existing) service-based systems.

In our previous work, we developed coverage metrics for data-centric dynamic service compositions [11]. Under the same umbrella project named *TeCoS* (Test Coverage for Service-based systems) we now investigate coverage of service APIs. In this paper we apply software testing concepts to Web services and present a solution for measuring API coverage based on historical invocations. Our contribution is threefold: 1) we define API coverage metrics and their instantiation for Web services, 2) we suggest the implementation of XSD facets in the JAX-WS framework, and 3) we present and assess our prototype in an experimental evaluation.

The remainder of this paper is structured as follows. In Section 2 we briefly introduce a scenario Web service to which the API coverage metrics will be applied. Section 3 illustrates our approach for exact Web service interface definition using Java annotations. Section 4 discusses how different API coverage metrics can be defined by SOA testers and developers, and in Section 5 we detail the technique for measuring and calculating these metrics. Section 6 presents our prototype implementation, and the overall approach is evaluated in Section 7. We then discuss related work in the field of API coverage for service-oriented systems in Section 8, and Section 9 concludes the paper with an outlook for future work.

2. SCENARIO

As an illustrative scenario for this paper we assume a *Chart Web service* (CWS) that is capable of generating chart images from numeric input values, similar to the *Google Chart API*[6]. For the sake of brevity we consider only a simplified version of a service with 1 operation (*generateChart*) and 4 parameters (see Table 1). However, our approach is also applicable to more comprehensive APIs.

Service *ChartService*

Operation *generateChart*

Parameters:

Name	XSD Type	Constraints
$type$	`string`	$type \in \{'line', 'bar', 'pie'\}$
$values$	list of `integers`	$values \in \{-100, .., 100\}^n$, $1 \leq n \leq 10$
$name$	`string`	pattern `"[a-z][a-z0-9]{0,4}"`
$config$	list of `Configs`	list has less than 100 `Configs`

Table 1: API of Scenario Service

The *generateChart* operation has a parameter $type$, whose domain is an enumeration of 3 values. The parameter $values$ is a list of `integers`, with a length between 1 and 10, and an integer range of $\{-100, .., 100\}$. A short alphanumerical identifier (maximum length 5, starting with a letter) is provided using the parameter $name$. Finally, various configuration settings (at most 99) in the form of key-value pairs are passed to the operation using the parameter $config$. The XSD complex type `Config` contains a sequence with a `key` element and a list of one ore more `value` elements. More details about the XML schema are given in Section 3.

3. EXACT DEFINITION OF WEB SERVICE API WITH JAX-WS AND JAXB

To provide meaningful data for API coverage metrics, it is desirable to limit the domain range of the parameters of service operations. For example, the numeric values in our scenario are of type `integer`. Considering the service is implemented in Java, each value has a range of 32 bits (4.294.967.296 possibilities), while the API requires the values to be between -100 and +100 (201 possibilities). Note that constraining the domain range does not eliminate the combinatorial explosion problem per se, but can lead to a significantly smaller size of the parameters' combined domain range.

```
1  @XmlRootElement
2  public class GenerateChart {
3    public static enum ChartType { line, bar, pie }
4    public static class Config {
5      public String key;
6      public List<String> value;
7    }
8    @XmlElement(required=true)
9    public ChartType type;
10   @MinOccurs(1) @MaxOccurs(10)
11   @Facets(minInclusive=-100, maxInclusive=100)
12   public List<Integer> value;
13   @Facets(pattern="[a-z][a-z0-9]{0,4}")
14   public String name;
15   @MaxOccurs(99)
16   public List<Config> config;
17 }
18 @WebService
19 public class ChartService {
20   public String generateChart(GenerateChart request){
21     // generate chart, return in Base64 format
22   }
23 }
```

Listing 1: Implementation of Chart Web Service

We argue that an exact definition of interfaces should be an integral part of engineering service-oriented systems to provide for reasonable coverage analysis. The Web services framework is based on XML messaging and allows to limit domain ranges of simple types using XSD *facets*, and the range of elements with multiple occurrences using the XSD attributes `minOccurs` and `maxOccurs`. An exact specification of parameters is also useful to enforce the integrity of invocations at runtime. Instead of validating parameters manually in the implementation (e.g., checking for `null` values to avoid a Java `NullPointerException`), the service execution engine can automatically filter invalid parameters by matching incoming messages against the XSD using XML Schema validation.

```
<complexType name="generateChart">
 <sequence>
  <element name="type">
   <simpleType>
    <restriction base="xs:string">
     <enumeration value="line" />
     <enumeration value="bar" />
     <enumeration value="pie" />
    </restriction>
   </simpleType>
  </element>
  <element name="value" minOccurs="1" maxOccurs="10">
   <simpleType>
    <restriction base="int">
     <minInclusive value="-100" />
     <maxInclusive value="100" />
    </restriction>
   </simpleType>
  </element>
  <element name="name">
   <simpleType>
    <restriction base="string">
     <pattern value="[a-z][a-z0-9]{0,4}" />
    </restriction>
   </simpleType>
  </element>
  <element name="config" type="Config" maxOccurs="99"/>
 </sequence>
</complexType>
<complexType name="Config">...</complexType>
```

Listing 2: Generated XML Schema for *GenerateChart*

[6]http://code.google.com/apis/chart/

In the world of Java Web services (JAX-WS), JAXB (Java Architecture for XML Binding) is used to create a mapping between XML elements and Java objects. Because JAXB currently supports neither XSD facets nor occurrence ranges, we suggest to extend JAXB with 3 new annotations: `@Facets`, `@MinOccurs` and `@MaxOccurs`. The Java implementation of the Chart Web service using the extended JAXB annotations is illustrated in Listing 1.

When deploying the Web service, the WSDL file is automatically generated and contains the according XSD type for the class *GenerateChart*, which is printed in Listing 2. Details concerning the implementation are presented in Section 6. Based on this exact description of the service API, we are now able to define metrics to measure the coverage of the Chart Web service in Section 4.

4. CONFIGURABLE API COVERAGE METRICS FOR WEB SERVICES

API coverage can be measured in various ways, and the ability to define coverage metrics is vital for SOA testers and developers. For instance, we could request that the Chart Web service needs to be invoked at least with the extreme values for the parameter $values$ (i.e., -100 and +100) and with all possible chart $type$s. Furthermore, it could be of interest to see a service invocation fail which uses values beyond the range (e.g., -200). Apart from testing the functionality of services, coverage metrics can also be used to generate usage reports of the running system, e.g., how often service consumers have requested different chart types, or which configuration parameters were frequently used.

Determining the aforementioned coverage metrics is supported in our approach by means of customizable *domain partitioning*. The basic idea is that the user specifies rules which divide the range of a parameter domain d into (usually disjoint) subsets $P_d = \{d_1, .., d_n\}$, $d_1 \cup .. \cup d_n = d$. At runtime, service invocation messages are logged and matched against the specified rules to find out which subset the message belongs to. A 100% coverage would then indicate that at least one message has been logged for each subset.

We distinguish two methods to define domain partitioning rules:

1. *Membership Test* (MT): For each subset s in the partion P_d of domain d, an indicator function $m_s : d \rightarrow boolean$ determines for every element of d whether it is a member of s. That is, the user has to define n different indicator functions, and all n expressions need to be evaluated to determine which subset(s) an element is member of.

2. *Membership Identifier* (MI): The user specifies 1) the total number of subsets in the partition, $|P_d|$, and 2) a single membership function $i : d \rightarrow ID$, which returns for each element in the domain d a unique numeric identifier ($ID \subset \mathbb{R}$, $|ID| = |P_d|$) of the subset it is member of. Evidently, with this approach the subsets $d_1, .., d_n$ are automatically disjoint.

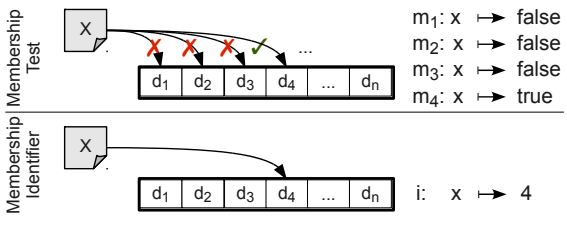


Figure 1: Domain Partitioning Methods

The two methods are illustrated in Figure 1. The MT method requires more function evaluations, but it is more expressive and supports non-disjoint subsets (i.e., each element may be member in more than one subset). For both methods, the expressions are defined using the syntax of the *Groovy*[7] scripting language for Java. Seven exemplary domain partitions are illustrated in Table 2. The predefined variable `x` references the target value for which the function expression should be evaluated (e.g., the parameter $values$ introduced in Section 2), and `MIN`/`MAX` are predefined variables for the minimum/maximum value of the domain of `x`. It is important to mention that domain partitions may apply to different targets: 1) the numerical occurrence count (e.g., `x`$\in\{1, .., 10\}$ for $values$), and 2) the value (text content) of XML elements (e.g., `x`$\in\{-100, .., 100\}$ for $values$). Hence, the variable `x` refers to either of the two, depending on the target for which the partition has been specified. The examples in Table 2 contain numerical partitions for extreme values (t), negative/zero/positive values (n, z), blocks of values (b) or values that are out of the valid range (o), as well as a partition to select the type `pie` (p) and an "ignore" partition (returns `true` on all membership tests). In addition, the predefined indicator function $m_0(x) = (\bigwedge_{i\in\{1,..,|P_d|\}} \neg m_i(x))$ embraces elements that belong to no other subset. Furthermore, we provide the predefined *default* partition (d) which defines no custom subsets but reflects the value domains as specified in the XSD.

ID	Name	Type	$\lvert\mathbf{P_d}\rvert$	$\mathbf{m_1(x)}$	$\mathbf{m_2(x)}$	$\mathbf{m_3(x)}$	$\mathbf{i(x)}$
i	*ignore*	MT	1	`true`	-	-	-
n	*negZeroPos*	MT	3	`x<0`	`x==0`	`x>0`	-
z	*zero*	MT	1	`x==0`	-	-	-
t	*extreme*	MT	2	`x==MIN`	`x==MAX`	-	-
b	*blocksOf10*	MI	`int(abs(MAX -MIN)/10)+1`	-	-	-	`(int) x/10`
o	*outOfRange*	MT	1	`x<MIN|| x>MAX`	-	-	-
p	*pieChart*	MT	1	`x=='pie'`	-	-	-
d	*default*	predefined, based on XSD of the Web service					

Table 2: User-Defined Domain Partitions

5. COVERAGE COMPUTATION

In the following we discuss how Web service API coverage is computed taking into consideration the configurable coverage metrics described in Section 4. A prerequisite is that all invocations performed in the service-based system are accessible. Details on how our implementation intercepts and logs the invocations messages are given later in Section 6. For now, we assume that the XML messages are stored in tables of a relational database management system (DBMS). To enable detailed queries on their structure and content, the XML messages are parsed and stored in a structured format as depicted in Figure 2. This figure uses UML notation and the model maps directly to the persistence (database) level.

The base class is `XMLNode`, which models nodes in an XML structure. The type (simple/complex element, attribute, text, ...) is distinguished by the attribute `type`, and the `value` (text content) may be empty for complex type elements. Additionally, the complete XML source of the root elements is stored in `xml` for performance optimizations (see Section 7). The actual XSD type of elements is stored separately, and is left out in the figure. The combination of `name` and `value` is unique, i.e., every encountered XML is only stored once and there are no redundancies. In turn this means that the parent-child relationship has many-to-many cardinality: an element may have multiple children and each element may be the child of more than one parent. In addition to the parent-child association, we store the list of all descendants of

[7] http://groovy.codehaus.org/

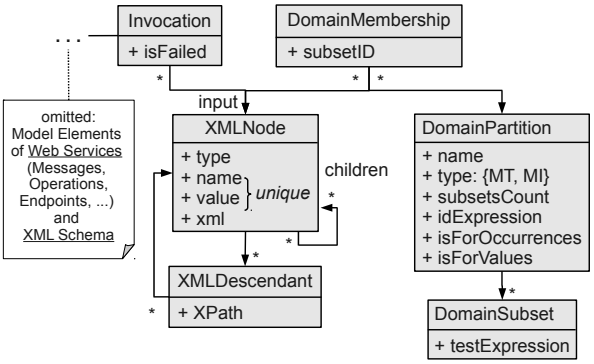


Figure 2: Domain Model for API Coverage with Custom Domain Partitioning

an invocation input message m using the class `XMLDescendant`. The position within the element tree rooted in m is specified as an XPath [18] expression. The class `DomainPartition` describes a domain partition, including the name, type and number of subsets (`subsetsCount`). Depending on the type (MT or MI), the subsets are either expressed using several membership test expressions (`testExpression`), or using a single membership ID expression (`idExpression`). The mapping between a node and the partition subset it belongs to is stored in `DomainMembership`.

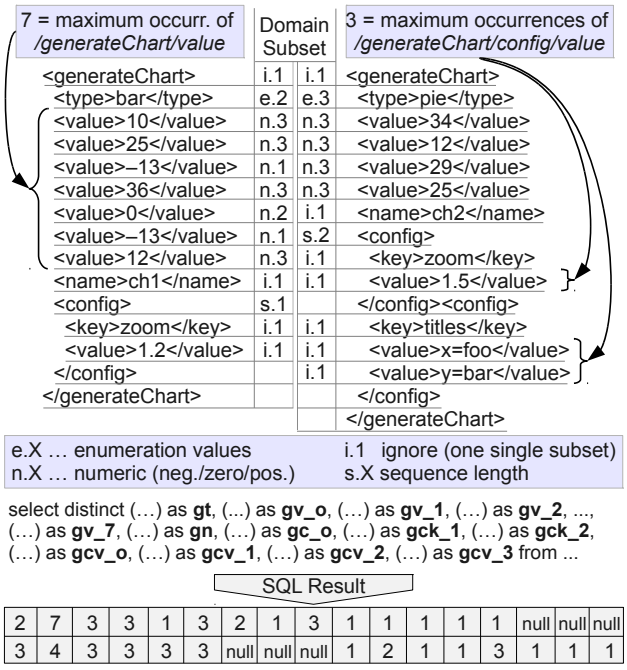


Figure 3: SQL-Based Querying of Distinct Invocations

Essentially, API coverage expresses the ratio of previously performed, distinct invocations (DI) to the number of possible invocations (PI). While PI can be calculated statically based on the XSD schema of the service's input messages, determining DI requires to query the database for logged invocations. Since the log database may become very large, our goal is to outsource the computation of DI to the DBMS. We hence formulate a single SQL query, which can be optimized and efficiently executed by the DBMS.

Figure 3 illustrates the SQL-based querying of distinct invocations, based on two invocations of the *generateChart* operation. We specify that $type$ should have one subset for each enumeration value (abbreviated `e.1`, `e.2`, `e.3`), the numeric $values$ are divided into negative/zero/positive (`n.1`, `n.2`, `n.3`), the number of element occurrences for $values$ and $config$ are measured using the MI partitioning method (`s.X`, where `X` is the length of the sequence), and the actual keys and $value$s all fall into one single partition (`i.1`) and are hence "ignored", i.e., have no effect on the computation of DI. The depicted SQL *select* clause joins the stored invocations with their calculated domain subset membership and computes all distinct combinations. The core idea of our method is to flatten the XML tree and to use a sub-query for each possible value that can occur in this tree. We abbreviate an element's XPath as the concatenation of the first characters of its ancestors' names (e.g., `gcv` for `/generateChart/config/value`). The total multiplicity of elements is abbreviated with the suffix "*_o*", e.g., *gcv_o* is 1 in the first example invocation and 3 in the second invocation. The single values of these elements with multiple occurrences are referenced by their index, e.g., *gcv_1*, *gcv_2* etc. We hence first determine for each descendant the maximum number of occurrences recorded so far. For each relevant value we then construct an SQL sub-query and generate the total query string. This means that the SQL result (distinct invocations) contains `null` values for those elements with less than the maximum occurrences.

For now, we assume that the ordering plays a role and work mainly on XSD *sequences*, because they are arguably more relevant in the Java Web services world than the order indicators *all* and *choice*. However, extending the approach is straightforward.

	PI_v	PI_o	PI_a	DP_v	DP_o	PI_v	DI_v	PI_o	DI_o	PI_a	DI_a	Cvg%
generateChart	-	1	∞	**i**	-	-	-	1	1	265716	2	0.0008
├ type	3	1	3	**d**	-	3	2	1	1	3	2	66.667
├ value	201	10	10^{23}	**n**	**d**	3	3	10	2	88572	2	0.0023
├ name	45M	1	45M	**i**	-	1	1	1	1	1	1	100
└ config	-	100	∞	**i**	**i**	-	-	1	1	1	1	100
├ key	∞	1	∞	**i**	-	1	1	1	1	1	1	100
└ value	∞	∞	∞	**i**	**i**	1	1	1	1	1	1	100
	based on Schema					based on Partitioning						

Figure 4: Calculation of Possible Invocations

Figure 4 illustrates the calculation of possible invocations and the percental API coverage (*Cvg%*), based on our scenario and the two logged invocations (see Figure 3). Figure 4 contains numbers based on the partitioning, and, as a comparison, the number of possibilities determined from the Schema, which constitutes the upper bound. The depicted table lists the following values for every XSD (sub-)element el of the invocation message `generateChart`:

1. the user-defined domain partition to be used for evaluation of el concerning its values (DP_v) and occurrences (DP_o),
2. all possible invocations (PI_a) with regards to the value of el (PI_v) and the number of occurrences of el (PI_o),
3. the actually measured distinct invocations (DI_a) of el, when evaluating partition DP_v on its values (DI_v) and partition DP_o on its number of occurrences (DI_o),
4. the overall coverage (*Cvg%*) of el as the percental ratio of distinct invocations (DI_a) to possible invocations (PI_a). The coverage of the root element `generateChart` can be seen as the API coverage of the operation with the same name.

Concerning point 1, DP_v and DP_o can be arbitrarily chosen for each XML descendant of the input message schema, which allows for detailed coverage analysis on a per-element basis. Note that

DP_o is not available for elements that have a fixed occurrence count within their parent element (e.g., elements `type` and `name`).

For point 2, PI_v can be inferred from the domain partition DP_v and is generally equal to the number of subsets in this partition. PI_o expresses the number of possible occurrences of an element, and is subject to partitioning with DP_o. Besides custom partitions, our approach also computes the number of possible invocations directly from the XSD definitions (see Figure 4). Usually this number is much higher, e.g., the regular expression of `name` specifies 44917730 (≈45M) possibilities (see Section 6 for details about how this is determined). The total number of possibilities PI_a is a combination of PI_v and PI_o. Consider the `value` element and its assigned partition $n.X$ (negative-zero-positive) which has 3 subsets (PI_v=3). As specified in the XSD, this element may occur between 1 and 10 times, which means that the total number of possible instantiations is $3^1+3^2+\ldots+3^{10}$. We observe that this term can be calculated by means of a geometric series, as printed in Equation 1.

$$\sum_{k=m}^{n} r^k = \frac{r^{n+1} - r^m}{r-1} \quad (1)$$

In our example, $m = 1$, $n = 10$, $r = 3$, the value of PI_a hence amounts to 88572 possible combinations. The calculation of PI_a for complex XSD types (e.g., `generateChart`, `config`) is different, because all combinations of the contained sub-elements must be taken into account. PI_a of a complex type is therefore defined as the product of the PI_a values of all its direct child elements. For instance, `generateChart` has a total of 265716 possibilities (3∗88572∗1∗1) when we apply our example partitioning. Note that it is also possible that a complex type contains a text content which is not embraced by another element (e.g., `<a><b/>text</a>`), for instance resulting from an XSD *any* element. Our approach supports such constructs, since in terms of API coverage we can treat the text node like a simple type element.

Point 3 of the above list concerns the number of distinct invocations, DI_a, the calculation of which has been described earlier in this section. Additionally, we calculate DI_v and DI_o with two modified versions of the SQL query in Figure 3. In our example for the element `value`, DI_v is 3 because values from all 3 domain subsets (negative, zero, positive) have been logged, and DI_o is 2 because we logged invocations with 2 different occurrence counts of the `value` element (7 and 4).

Having computed all required values, the API coverage is calculated as DI_a/PI_a. The example coverage of 0.0008% is very low, which is hardly surprising considering the fact that we only logged 2 invocations for illustrative purposes. The overall coverage increases considerably when either more invocations are stored in the DB or other (more restricting) domain partitions are chosen.

6. IMPLEMENTATION

In this section we discuss the implementation of the presented concepts. The overview of the TeCoS framework architecture in Figure 5 shows a SOA under test consisting of Web services, which are invoked by a business process and end users. The services are deployed in a container, whose responsibility is to transparently log incoming invocations to the *Tracing Service* (TS). To that end, we provide an implementation of `SOAPHandler` which can be easily plugged into the JAX-WS handler chain. The TS makes use of a *Service Registry* and logs all messages to the *Invocation Database* (InvDB), which also incorporates a simplified *XML Schema Database*. MySQL[8] (version 5.1) is used as the DBMS. The *Coverage Calculator* (CC) operates on the InvDB and contains the core business logic for computing API coverage metrics. Furthermore, the CC manages the user-defined domain partitions. The TeCoS framework also defines coverage metrics for WS-BPEL service compositions, which we have presented in earlier work [11]. We further plan to extend our notion of API coverage from parameter combinations of a single operation to invocation sequences.

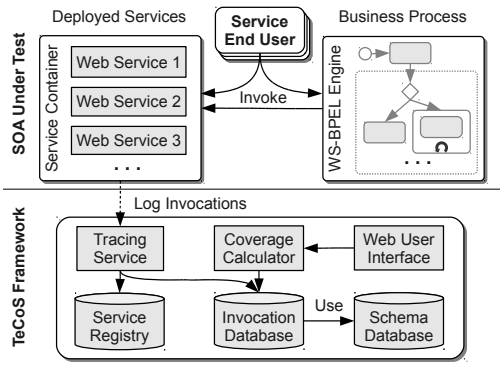


Figure 5: TeCoS Framework Architecture

To allow for a graphical feedback and experimental exploration of coverage metrics, we have implemented a graphical Web user interface (UI) using Java Server Faces (JSF) technology. The UI allows the configuration of user-defined partitions, visualizes the information stored in the Service Registry, and depicts invocations and API coverage data. A screenshot of the Web UI, showing the coverage results of our test scenario, is depicted in Figure 6. The leftmost table column lists the element structure together with the applicable facets. By clicking on the name of an element, the user receives additional information such as the actual occurrences and values that were logged for this element. We are currently also working on extended coverage reports with graphical charts.

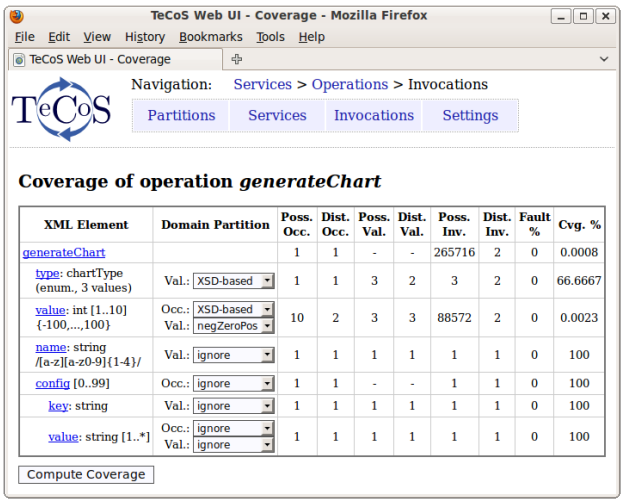


Figure 6: Screenshot of Web User Interface

As part of its extension mechanism, the JAX-WS reference implementation[9] (RI) employs a means to implement special-purpose interceptors for custom WSDL generation (e.g., WS-Addressing headers). However, JAX-WS RI does not allow custom schema

[8]http://www.mysql.com/

[9]http://jax-ws.java.net/

generation based on JAXB. In 2007, a JIRA entry[10] with priority *"Major"* was created for this issue, but as of January 2011 the status of this feature request is still *"Open"*. We therefore investigated the source code of JAXB RI to hook into the schema generation process. Our modification requires 3 new annotation interfaces (used in Listing 1), 1 new class (`XsdFacets`) and 3 additional lines of code in the class `com.sun.xml.bind.v2.XmlSchemaGenerator` (lines with comments on right margin in Listing 3). We have initiated an open discussion on the JAXB *dev* mailing list about whether our solution will be merged into JAXB RI.

```
private Tree handleElementProp(
                final ElementPropertyInfo<T,C> ep) {
 ...
 if(!XsdFacets.hasFacets(t, e))                          //1
     writeTypeRef(e,t, "type");
 ...
 if(!XsdFacets.writeOccurs(t,e,isOptional,repeated)) //2
     writeOccurs(e,isOptional,repeated);
 XsdFacetsGenerator.addFacets(t, e);                     //3
 ...
}
```

Listing 3: Modifications of *XmlSchemaGenerator* in JAXB RI

One of the key prerequisites for determining the number of possible invocations is the analysis of regular expressions (regex). Usually, regex engines allow only to match a given string against a regex string, but have no support for generating all strings that match the regex. Our current implementation is hence based on a small (yet powerful) Python script[11] by Paul McGuire, which can determine the number of all possible matching strings for a regex.

7. DISCUSSION AND EVALUATION

To evaluate different aspects of our approach, we have implemented the test service presented in Section 2 and generated 10000 invocations with randomized parameters (concerning both the length of sequences and the values of simple types). Figure 7 depicts the measured results when applying the domain partitions of Figure 4: number of distinct invocations (DI), time required to calculate this number (CT), and XML Elements (XE) stored in the database. As described in Section 5, only unique XML elements are stored in the database; e.g., if two subsequent invocations i_1 and i_2 use a `value` parameter with value 5, a new database entry is inserted for i_1, but the stored invocation for i_2 then points to the existing row. This means that more queries are required when we store an invocation message, with the advantage that the DB is free of duplicates. We can observe the memory effect in the figure: the minimum number of XML elements per invocation is 3 (1 `type`, 1 `value`, 1 `name`), but for 10000 invocations only 10209 distinct elements were saved.

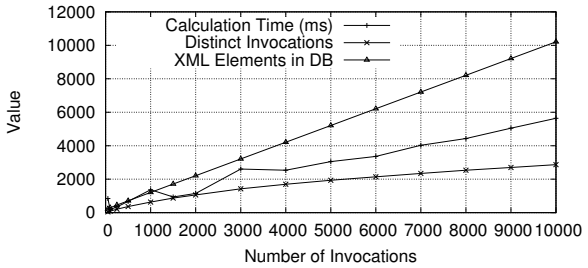


Figure 7: Performance Evaluation Results

[10]http://java.net/jira/browse/JAXB-392

[11]http://pyparsing.wikispaces.com/file/view/invRegex.py

Another interesting aspect is the trend of DI with increasing number of total random invocations. We calculate DI with the partition settings and the query depicted in Figure 3. Whereas DI rises quite sharply in the initial phase (635 out of 1000, 1074 out of 2000), the curve flattens out when more invocations are recorded (ΔDI between 9000 and 10000 was 160). The reason for this is clearly that an increasing number of stored messages decreases the possibility that a new distinct invocation arrives. Finally, the figure indicates the time required to calculate the coverage metrics with the aid of the DBMS. The time for 10000 stored invocations is still feasible (< 6 seconds) and shows that the approach generally scales well. However, for very large systems with millions of records the current solution poses limitations for real-time usage. In our future work, we are therefore investigating techniques for caching and storing/calculating coverage data in a distributed manner.

For simple processing of XML data, MySQL provides the function `ExtractValue` to access node values in XML markup via XPath expressions. To evaluate the use of this function, we have stored the complete XML source of invocation messages in the column `xml` of table `XMLNode` (cf. Figure 2). This allows to execute the coverage sub-queries (cf. Figure 3) directly on the XML source, e.g., `select ExtractValue(xml,'//type') from XMLNode ...` Such XPath-based queries show very good performance, even with several thousand rows in the table, and in some cases an improvement could be achieved over the alternative approach of joining `XMLDescendant` with the `XMLNode` table. The queries using `ExtractValue` are especially profitable when the stored invocations differ in content, but the tradeoff is that always the complete XML is stored, which results in redundant and duplicate data.

#	Example Metrics	v(gt)	o(gv)	v(gv)	v(gn)	o(gc)	v(gck)	o(gcv)	v(gcv)
		User-Specified Partitions							
1	Usage Counter	d	d	d	d	d	d	d	d
2	Extreme Values	i	t	t	i	i	i	i	i
3	Chart Types	d	i	i	i	i	i	i	i
4	Blocks of 10	i	i	b	i	i	i	i	i
5	Pie Charts	p	d	d	d	d	d	d	d
6	Out of Range	i	i	o	i	i	i	i	i
7	Config. Settings	i	i	i	i	d	i	i	i
8	Config. Keys	i	i	i	i	i	d	i	i
9	Values of Default Pie Charts	p	i	d	i	z	i	i	i

Table 3: Partition Settings for Example Coverage Metrics

The introductory paragraph of section 4 mentions that TeCoS allows various coverage metrics and usage reports. To provide a complete taxonomy of supported metrics is out of the scope of this paper, but we evaluate this claim by giving a number of concrete examples in Table 3. The column titles refer to the partitions used for the values (*v(x)*) and occurrences (*o(x)*) of some node *x* (*x* again represents an XPath abbreviation as used in Section 5). The partition IDs in the table refer back to Table 2. In the default setting (#1) all historical (distinct) invocations are computed, which provides a general usage counter of the chart service. To analyze specific parameters, the domain partitions of all remaining elements are set to i (ignore). For instance, example #2 calculates the coverage of the extreme values (concerning both the occurrence and the numeric value) of the `value` parameter, and metric #3 determines which chart types are covered. Conversely, if a subset of parameters should be fixed, all remaining parameter partitions can be set to d: e.g., #5 represents the partition settings to determine all `pie` charts requests. Metric #9 is a more advanced example which shows the potential of combining different parameter partitions. It calculates the chart values (gv_v=d) of all pie chart requests (gt_v=p) with default settings, i.e., no additional user configurations (gc_o=z).

8. RELATED WORK

In the following we discuss related work in the area of API test coverage and its application to service-oriented systems.

Xu et al. [19] present an approach for testing Web services based on the operations' XML schema. The paper defines a formal model for XSD, and defines operators for schema "perturbation", i.e., for creating XML messages that are *invalid* with respect to the schema. These operators, which include insertion, deletion and changing of nodes, are the basis for test coverage criteria. Their approach is complementary to ours, as it focuses on generating invalid test cases, whereas we perform logging and analysis of performed invocations, which serves both as a reporting tool for service providers and as a means to enforce coverage goals for service testers.

Apart from interface-based Web service testing, different methods have been proposed for, e.g., group testing [17], collaborative contract-based testing [2], and testing of data-centric service compositions [14, 11]. Further coverage criteria have been defined for operation/message combinations [5], and execution paths in processes defined with the Web Services Business Process Execution Language (WS-BPEL) [10, 13]. Baresi et al. [3] presented various techniques for Web services testing and verification, including monitoring, modeling, and reliability analysis.

A certain similarity can be seen with the approach of Bertoloni et al. [7] who also utilize XML Schema based partitioning. A major difference is that only the partitions defined by the schema are considered, whereas we support customizable partitioning of domain ranges. Furthermore, their approach aims at test data generation, which is not the primary concern in this paper. Under the term "whitening" of SOA testing [4], Bertoloni et al. have suggested that *testable* services should expose additional metadata in the form of coverage data. Similarly, we argue that providing an exact interface definition is a key requirement for meaningful API coverage testing and analysis. Conversely to Bertoloni et al., we do not require the service under test to measure the coverage itself, but merely to log its invocations to the TeCoS tracing service.

Also Bai et al. have studied automatic test case generation based on WSDL documents and the XML schema exposed therein [1]. Their approach mostly concentrates on random generation based on the interface and dependencies (message flows) of operations, but does not define in detail how coverage metrics are calculated.

Domain partitioning for API testing has been proposed previously, e.g., in the form of the Category-Partition test design pattern by Binder [8]. Jorgensen and Whittaker [12] do not differentiate between methods for defining partitions, whereas we provide two alternative ways (MT and MI) and additionally support default partitioning that is automatically derived from WSDL documents.

9. CONCLUSION

We presented an efficient, novel solution to measure API coverage of service-based systems implemented with Web services. User-specified domain partitioning allows for the definition of customizable and reusable API coverage metrics. Our end-to-end framework TeCoS can be easily plugged into existing service execution engines to log service invocations, calculate coverage data and render the results in a Web UI. Furthermore, we suggest to enhance JAX-WS and JAXB with support for XSD facets, and provide a solution that integrates seamlessly with the JAXB RI. This user-friendly extension greatly reduces the required development effort and is a step towards meaningful coverage data and schema-based validation of invocation parameters. The performance and scalability was successfully evaluated in our experimentation. In our ongoing work, we plan to extend the scope of API coverage to invocation sequences and semantic input description, as well as to enhance TeCoS with distributed storage and coverage computation.

10. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7) under grant agreements 215483 (S-Cube) and 257483 (Indenica).

11. REFERENCES

[1] X. Bai, W. Dong, W.-T. Tsai, and Y. Chen. WSDL-based automatic test case generation for Web services testing. In *Int. Workshop Service-Oriented Syst. Eng.*, pages 215–220, 2005.

[2] X. Bai, Y. Wang, G. Dai, W.-T. Tsai, and Y. Chen. A framework for contract-based collaborative verification and validation of web services. In *CBSE '10*, pages 258–273, 2007.

[3] L. Baresi and E. D. Nitto. *Test and Analysis of Web Services*. Springer-Verlag New York, Inc., 2007.

[4] C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti. Whitening SOA Testing. In *ESEC/SIGSOFT FSE '09*, 2009.

[5] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini. WS-TAXI: A WSDL-based Testing Tool for Web Services. In *ICST 2009*, pages 326–335, 2009.

[6] B. Beizer. *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., New York, USA, 1990.

[7] A. Bertolino, J. Gao, E. Marchetti, and A. Polini. Automatic Test Data Generation for XML Schema-based Partition Testing. In *Int. Workshop Automation of Software Test*, 2007.

[8] R. V. Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Longman, 1999.

[9] G. Canfora and M. Di Penta. Testing Services and Service-Centric Systems: Challenges and Opportunities. *IT Professional*, 8(2):10–17, 2006.

[10] J. Garcïa-fanjul, J. Tuya, and C. D. L. Riva. Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN. In *WS-MaTe 2006*, pages 83–94, 2006.

[11] W. Hummer, O. Raz, O. Shehory, P. Leitner, and S. Dustdar. Test coverage of data-centric dynamic compositions in service-based systems. In *4th IEEE International Conference on Software Testing, Verification and Validation*, 2011.

[12] A. Jorgensen and J. Whittaker. An API Testing Method. In *STAREAST Conf. on Softw. Testing Analysis & Review*, 2000.

[13] D. Lübke, L. Singer, and A. Salnikow. Calculating BPEL Test Coverage Through Instrumentation. In *Int. Workshop on Automation of Software Test*, pages 115–122, 2009.

[14] L. Mei, W. Chan, and T. Tse. Data flow testing of service-oriented workflow applications. In *ICSE*, 2008.

[15] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45, 2007.

[16] J. O. Paul Ammann. *Introduction to Software Testing*. Cambridge University Press, 2008.

[17] W. T. Tsai, Y. Chen, R. Paul, N. Liao, and H. Huang. Cooperative and Group Testing in Verification of Dynamic Composite Web Services. In *COMPSAC*, pages 170–173, 2004.

[18] World Wide Web Consortium (W3C). XML Path Language (XPath). http://www.w3.org/TR/xpath/, 1999.

[19] W. Xu, J. Offutt, and J. Luo. Testing Web Services by XML Perturbation. In *16th IEEE Int. Symposium on Software Reliability Engineering*, pages 257–266, 2005.

Appendix D

Identifying Task-based Sessions in Search Engine Query Logs

Identifying Task-based Sessions in Search Engine Query Logs

Claudio Lucchese‡, Salvatore Orlando†,
Raffaele Perego‡, Fabrizio Silvestri‡, Gabriele Tolomei‡†
‡ISTI-CNR, Pisa, Italy – e-mail: {firstname.lastname}@isti.cnr.it
†Dip.to di Informatica, Università Ca' Foscari Venezia, Italy – e-mail: {lastname}@dsi.unive.it

ABSTRACT

The research challenge addressed in this paper is to devise effective techniques for identifying *task-based sessions*, i.e. sets of possibly non contiguous queries issued by the user of a Web Search Engine for carrying out a given *task*. In order to evaluate and compare different approaches, we built, by means of a manual labeling process, a *ground-truth* where the queries of a given query log have been grouped in tasks. Our analysis of this ground-truth shows that users tend to perform more than one task at the same time, since about 75% of the submitted queries involve a multi-tasking activity. We formally define the *Task-based Session Discovery Problem* (TSDP) as the problem of best approximating the manually annotated tasks, and we propose several variants of well known clustering algorithms, as well as a novel efficient heuristic algorithm, specifically tuned for solving the TSDP. These algorithms also exploit the collaborative knowledge collected by *Wiktionary* and *Wikipedia* for detecting query pairs that are not similar from a lexical content point of view, but actually *semantically* related. The proposed algorithms have been evaluated on the above ground-truth, and are shown to perform better than state-of-the-art approaches, because they effectively take into account the multi-tasking behavior of users.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering, Query formulation, Search process*

General Terms

Algorithms, Design, Experimentation

Keywords

Query log analysis, Query log session detection, Task-based session, Query clustering, User search intent

WSDM'11, February 9–12, 2011, Hong Kong, China.

1. INTRODUCTION

There is a common belief that the Web is increasingly used to simplify the accomplishment of various everyday activities. Since nowadays Web Search Engines (WSEs) are the most important and used Web portals, such users' behaviors can be revealed by analyzing and mining WSE query logs [2, 26, 14, 8, 19, 30]. A very important piece of information we can extract from a query log is represented by "*query sessions*", i.e. specific sets/sequences submitted by a user while interacting with a WSE. Sessions represent the basic unit of information for tasks like query suggestion [1], learning to rank [23], enhancing interactions with the Web Search Engine [34], etc.

In the literature, there are many definitions of query sessions. In this paper, we are interested in identifying sessions composed of queries issued by users having in mind a particular task/goal [31]. Unfortunately, the well-known time-based detection methods fail in revealing such *task-based sessions*, i.e., *Web-mediated tasks*, due to the *multi-tasking* users' behavior. Multi-tasking refers to the way users interact with a WSE, by intertwining different tasks within the same time period. Therefore, the extraction of such task-based sessions requires to detect whether pairs of users' queries are *similar* and, thus, related to the same task/goal.

The main contributions of our work are the following.
(i) We start by showing that users perform multi-tasking search activities in the query streams issued to a WSE. This makes it unsuitable to identify task-based sessions by only exploiting techniques that simply split the stream of queries. Then, we investigate three well-known *clustering-based* approaches and we propose a new heuristic for detecting Web-mediated tasks. The obtained results show that the new algorithm performs similarly to the best clustering-based approach (i.e., weighted connected components) but it is computationally lighter on average.
(ii) We use a *query distance function*, exploited by those algorithms, which combines classical lexical content distance measures, with the collaborative knowledge provided by *Wiktionary*[1] and *Wikipedia*[2]. This knowledge is used to enrich the meaning of each issued query and, thus, to make more accurate decisions during clustering.
(iii) Finally, we compare and evaluate the quality of all those methods by exploiting a manually generated *ground-truth*, i.e. a set of task-based sessions manually detected over the queries submitted by several users.

[1] http://www.wiktionary.org
[2] http://www.wikipedia.org

2. RELATED WORK

Analysis of query logs collected by most Web Search Engines (WSEs) has increasingly gained interest across Web mining research community. Roughly, query logs record information about the *search activities* of users and so they are a suitable source of information for understanding how people search the Web or, in other words, the real intent behind issued queries [30].

Previous work on session identification can be classified into: 1) *time-based*, 2) *content-based*, and 3) *mixed-heuristics*, which usually combine both 1) and 2).

1) Time-based. Usually, time-based techniques have been adopted for their simplicity in previous research work.
Silverstein *et al.* [29] firstly defined a concept of "*session*" as follows: two consecutive queries are part of the same session if they are issued at most within a 5-minutes time window. According to this definition, they found that the average number of queries per session in the data they analyzed was 2.02. He and Göker [6] used different timeouts to split user sessions of Excite query log, ranging from 1 to 50 minutes.
Radlinski and Joachims [23] observed that users often perform a sequence of queries with a similar information need, and they referred to those sequences of reformulated queries as *query chains*. Their paper presented a method for automatically detecting query chains in query and click-through logs using 30 minutes threshold for determining if two consecutive queries belong to the same search session.
Another definition of session, i.e. *search episode*, was given by Jansen and Spink [8]. They described a session as the period of time occurring from the first to the last recorded time-stamp on the WSE server from a particular user in a single day, so that session length might vary from less than a minute to a few hours. Moreover, using the same concept of search episode, Spink *et al.* [31] investigated also *multi-tasking* behaviors while users interacting with a WSE.
In this paper, we show the presence of multi-tasking also within shorter user activities.

2) Content-based. Some work suggested to exploit the lexical content of the query themselves for determining a possible topic shift in the stream of issued queries and, thus, a session boundary [12, 7, 20]. To this extent, several *search patterns* have been proposed by means of lexical comparison, using different string similarity scores (e.g., Levenstein, Jaccard, etc.). However, approaches relying only on content features suffer of the so-called *vocabulary-mismatch problem*, namely the existence of topically-related queries without any shared terms. In order to overcome this issue, Shen *et al.* [28] compared "expanded representation" of queries, instead of the actual queries themselves. Each individual expanded query was obtained by concatenating the titles and the Web-snippets for the top 50 results provided by a WSE for the specific query. Thus, the relatedness between query pairs was computed using cosine similarity between the corresponding expanded queries.

3) Mixed heuristics. Jansen *et al.* [9] assumed that a new search pattern always identifies the start of a new session. Moreover, He *et al.* [7] showed that statistical information collected from query logs could be used for finding out the probability that a search pattern actually implies a session boundary. In particular, they extended their previous work [6] to consider both temporal and lexical information.
Boldi *et al.* [1] introduced the *query-flow graph* as a model for representing data collected in WSE query logs. They exploited this model for segmenting the query stream into sets of related information-seeking queries, leveraging on an instance of the Asymmetric Traveling Salesman Problem.
Finally, Jones and Klinkner [11] addressed a problem that appears to be similar to ours. In particular, they argue that within a user's query stream it is possible to recognize particular hierarchical units, i.e., *search missions*, which are in turn subdivided into disjoint *search goals*. A search goal is defined as an atomic information need, resulting in one or more queries, while a search mission is a set of topically-related information needs, resulting in one or more goals. Given a manually generated ground-truth, Jones and Klinkner [11] investigated how to *learn* a suitable binary classifier, which is aimed to precisely detect whether two queries belong to the same task or not. Among various results, they realized that timeouts, whatever their lengths, are of limited utility in predicting whether two queries belong to the same goal, and thus to identify session boundaries. Indeed, authors did not to explore how such binary classifier could be exploited for actually segmenting users' query streams into goals and missions.

3. THEORETICAL MODEL

A WSE query log stores queries submitted by users, along with other information, such as userIDs, time-stamps, etc. We denote with $\mathcal{QL}$ a WSE log of the queries submitted by a set of users $\mathcal{U} = \{u_1, u_2, \ldots, u_N\}$ during a given observation period. Moreover, let $q_i \in \mathcal{QL}$ be a generic query issued by user u_i, and $q_{i,j} \in \mathcal{QL}$ be the j-th query issued by user u_i.

The methods that extract meaningful user sessions from $\mathcal{QL}$ have to analyze all the queries issued by each user u_i. Let $\mathcal{S}_i$ be the sequence of *all* the queries $q_i \in \mathcal{QL}$ issued by user $u_i \in \mathcal{U}$, chronologically ordered during the period of observation recorded in the query log: $\mathcal{S}_i = \langle q_{i,1}, q_{i,2}, \ldots, q_{i,K} \rangle$. Therefore,

$$\mathcal{QL} = \bigcup_{i=1}^{N} \mathcal{S}_i$$

Since users tend to issue *bursts* of queries for relatively short periods of time, which are usually followed by longer periods of inactivity, the *time gap* between queries plays a significative role in detecting session boundaries. According to [29], we detect the session boundaries by considering the user's inactivity periods, i.e. the time gaps between consecutive queries in each $\mathcal{S}_i$.

DEFINITION 3.1 (TIME-GAP SESSION $\phi_{i,k}$). *Let $\tau(q_{i,j})$ bet the time at which the query $q_{i,j}$ is issued, and t_ϕ be the maximum* time gap *threshold. The ordered set of consecutive queries $\phi_{i,k} = \langle q_{i,s_k}, \ldots, q_{i,e_k} \rangle \subseteq \mathcal{S}_i$, with $s_k < e_k$, is said to be a* time-gap session *if it holds that:* (i) $\tau(q_{i,j+1}) - \tau(q_{i,j}) \leq t_\phi$ *for every j, $s_k \leq j < e_k$, and* (ii) *there is no time-gap session being a superset of $\phi_{i,k}$.* □

It is worth noticing that this splitting technique makes no restrictions on the total elapsed time between the first and the last query of the sequence $\phi_{i,k}$. Moreover, even if the inactivity threshold is usually fixed arbitrarily, in our tests we set $t_\phi = 26$ minutes by analyzing the distribution of the time gaps in the query log $\mathcal{QL}$ used for the experiments (see Section 4.1).

In this paper, we are interested in studying to which extent in such time-gap sessions we can further recognize *task-based sessions*, i.e. sets of queries aimed at performing some Web-mediated tasks. Queries within the same task-based session do not necessarily occur consecutively in the time-gap session $\phi_{i,k}$. Indeed, we will show that a generic user u_i usually interleaves many different information needs and related queries in each $\phi_{i,k}$.

DEFINITION 3.2 (TASK-BASED SESSION $\theta^j_{i,k}$). *Let $\phi_{i,k}$ be a time-gap session included in $\mathcal{S}_i$, and let $\theta^j_{i,k} \subseteq \phi_{i,k}$ be a* task-based session*, i.e., a set of (not necessarily consecutive) queries issued by user u_i for performing a given Web-mediated task. Such tasks form a disjoint partitioning of a time-gap session.* □

We denote with $\Theta_{i,k} = \cup_j \theta^j_{i,k}$ all the task-based sessions in a given time-gap session $\phi_{i,k}$, and with $\Theta = \cup_{i,k}\Theta_{i,k}$ the set of all the task-based sessions in the query log $\mathcal{QL}$, i.e. the set-union of $\Theta_{i,k}$ for all users i and associated time-gap sessions k.

The problem of finding Θ in a given query log can thus be formulated as the *Task-based Session Discovery Problem* (TSDP), whose goal is to find the best *query partitioning strategy* π that, when used to segment each time-gap session $\phi_{i,k}$ in $\mathcal{QL}$, approximates the *actual* user task-based sessions $\Theta_{i,k}$.

DEFINITION 3.3 (TSDP). *Given a query log $\mathcal{QL}$, let $\mathcal{C}_{i,k} = \{c^1_{i,k}, c^2_{i,k}, \ldots\}$ be the task-based sessions determined by the* query partitioning strategy *π, when applied onto $\phi_{i,k}$, i.e. $\pi(\phi_{i,k}) = \mathcal{C}_{i,k}$. Let $\Theta = \cup_{i,k}\Theta_{i,k}$ and $\mathcal{C}_\pi = \cup_{i,k}\mathcal{C}_{i,k}$. The TSDP requires to find the best partitioning $\bar{\pi}$ such that*

$$\bar{\pi} = \arg\max_{\pi} \xi(\Theta, \mathcal{C}_\pi)$$

where ξ is a given function that measures the quality of partitioning $\mathcal{C}_\pi$ with respect to Θ. □

Several quality measures can be used to evaluate the accuracy of a task-based session extraction, and consequently, several ξ functions can be devised. In Section 6 we instantiate ξ in terms of *F-measure*, *Rand index* and *Jaccard index*.

4. DATA ANALYSIS

We used the 2006 *AOL* query log as our testing data set. This query log is a very large and long-term collection consisting of about 20 million of Web queries issued by more than 657000 users over 3 months (from 03/01/2006 to 05/31/2006)[3].

First of all, we removed query log records containing both empty and "non-sense" query strings (e.g., query strings composed of only punctuation symbols). Also, we removed all the *stop-words* from each query string. Then, we run the *Porter stemming algorithm* [21] for removing the most common morphological and inflexional English endings from the terms of each query string. Finally, the data cleaning phase involved removing the long-term user sessions containing too much queries, which were probably generated by *robots*, instead of human users. Then, we considered as a sample the 1,000 user sessions with the highest number of queries (*top*-1000).

[3] http://sifaka.cs.uiuc.edu/xshen/aol_querylog.html

Hence, according to Def. 3.1, we split each user session into several time-gap sessions. To this end, we had to devise a suitable time threshold t_ϕ, which can be obtained by analyzing the distribution of time gaps between all the consecutive query pairs in our data set. We divided all the time gaps into several *buckets*, 60-seconds each. Therefore, we analyzed the query inter-arrival times distribution, which is revealed to be a power-law (Fig. 1(a)).

This model tightly fits user behaviors during Web search activities, when consecutive queries issued within a short period of time are often not independent because they are also task-related.

More formally, given the following general form of a power-law distribution $p(x)$,

$$p(x) = \frac{\alpha - 1}{x_{min}} \left(\frac{x}{x_{min}} \right)^{-\alpha}$$

where $\alpha > 1$ and x_{min} is the minimum value of x from which the law holds, we were interested in finding the value $\bar{x}$, such that two consecutive queries whose time gap is smaller than $\bar{x}$ are considered to belong to the same time-gap session. When the underlying distribution is unknown, it makes sense to assume a Gaussian distribution and use a threshold $\bar{x} = \mu + \sigma$ being equal to mean μ plus standard deviation σ, which results in "accepting" $\lambda = 84.1\%$ of the samples. This is equivalent to consider the cumulative distribution $P(\bar{x}) = Pr(X \leq \bar{x})$ and to determine $\bar{x}$, such that $P(\bar{x}) = \lambda$. Since we know the underlying distribution, we map the threshold λ into our context as follows:

$$P(\bar{x}) = C \int_{-\infty}^{\bar{x}} p(X)\, dX = \frac{\alpha - 1}{x_{min}^{-\alpha+1}} \int_{-\infty}^{\bar{x}} X^{-\alpha}\, dX = \left(\frac{\bar{x}}{x_{min}} \right)^{-\alpha+1}$$

Hence, for our purpose we had to solve the following equation w.r.t. $\bar{x}$:

$$P(\bar{x}) = \left(\frac{\bar{x}}{x_{min}} \right)^{-\alpha+1} = \lambda = 0.841 \qquad (1)$$

The value x_{min} represents the minimum query pair time gap and corresponds to the first interval, i.e., 60 seconds. Therefore, we estimated $\alpha = 1.58$ and finally we can solve Eq. 1 finding $\bar{x} \simeq 26$ minutes. This means to assume 84.1% of consecutive query pairs are issued within 26 minutes. We used this value, $\bar{x}$, as the threshold t_ϕ for splitting each long-term user session of the the query log.

4.1 Ground-truth construction

In order to approach the Task-based Session Discovery Problem, according to our Def. 3.3, we need to find the query partitioning strategy that best approximates the *actual* task-based segmentation. Such optimal task-based partitioning can be manually built from real WSE query log data. To this end, we developed a Web application that helps human assessors to manually identify the optimal task-based query sessions from the previously prepared AOL query log, thus producing a *ground-truth* that can be used for evaluating automatic task-based session discovery methods.

Human annotators grouped together queries that they claimed to be task-related within each time-gap session. Also, they had chance to discard meaningless queries from those sessions. For each manually identified task (i.e., set of task-related queries), evaluators had to add a *tag* and, optionally,

a longer description. Such data source could possibly represent a semantic knowledge base of users search goals (i.e., taxonomy of tasks).

Since long-term sessions in AOL query log were too long, we only consider the first week of activities for each *top*-1000 user session. Finally, human evaluators were people selected from our laboratory, but not directly involved in this work.

4.2 Ground-truth statistics

Manual annotation procedure concerned a total of 2,004 queries, from which 446 time-gap sessions were extracted automatically. A total of 139 time-gap session were discarded as meaningless by the annotators, and therefore they were removed from the ground-truth. Eventually, 1,424 queries were actually clustered from 307 time-gap sessions.

Fig. 1(b) shows the distribution of time-gap session length, using a discretization factor of 60 seconds. While there are many session being less than 1 minute long, probably short sessions with one or two queries, the average length of a time-gap session is about 15 minutes. It is not infrequent to have sessions lasting for 40 minutes. Also in this case, the length of these sessions suggests that the interaction of the user with the Web Search Engine is non trivial, and it is likely to involve multi-tasking. Finally, the longest time-gap session lasts 9207 seconds, i.e. about 2 hours and a half.

In Fig. 1(c) we report the time-gap session size distribution. On average, each time-gap session contains 4.49 queries, the sessions with at most 5 queries cover slightly more than half of the query log. The other half of the query log contains longer sessions with high probability of having multiple tasks being carried on in the same session.

The total number of human annotated task-based sessions is 554, with an average of 2.57 queries per task. The distribution of the task-based sessions size is illustrated in Fig. 1(d). The number of tasks accomplished in a time-gap session is 1.80, see Fig. 1(e). In particular, only 162 out of 307 time-gap sessions contain one task only. We found that this 50% split between single-tasking and multi-tasking sessions is consistent across the various users. Interestingly enough, this shows that a good algorithm should be able to handle efficiently both single- and multi-tasking sessions. If we consider the queries included in each task, then 1,046 out of 1,424 queries are included in multi-tasking sessions, meaning that about 74% of the user activity is multi-tasking.

Finally, we also evaluated the degree of multi-tasking by taking into account the number of overlapping task-based sessions. We say that a *jump* occurs whenever two queries in a manually labelled task-based session are not consecutive. For instance, let $\phi = \langle q_1, q_2, \ldots, q_9 \rangle$ be a time-gap session and let $\pi(\phi) = \{\theta_1, \theta_2, \theta_3\}$ be the result of the manual annotation procedure for ϕ, where $\theta_1 = \{q_1, q_2, q_3, q_4\}$, $\theta_2 = \{q_5, q_7\}$, and $\theta_3 = \{q_6, q_8, q_9\}$. In this case, the number of jumps observed in ϕ is 2, because there are two query pairs $(q_5, q_7) \in \theta_2$ and $(q_6, q_8) \in \theta_3$, which are not consecutive. The number of jumps gives a measure of the *simultaneous* multi-tasking activity. We denote with $j(\phi)$ the simultaneous multi-tasking degree of ϕ as the ratio of task-based sessions in ϕ having at least one jump. In the previous example $j(\phi) \simeq 0.67$, since 2 out of 3 tasks contain at least one jump. In Fig. 1(f), we show the distribution of the multi-tasking degree over all the time-gap sessions. Note that the result for $j(\phi) = 0$ is omitted, because we already know that 50% of the sessions are single-tasking.

5. SESSION DISCOVERY METHODS

In this section, we address the Task-based Session Discovery Problem (TSDP) introduced in Section 3 by proposing and comparing several approaches and techniques. We group the session discovery mechanisms into two broad families: *(i) TimeSplitting-t* and *(ii) QueryClustering-m*. Basically, *TimeSplitting-t* consists of splitting each session when the time between two query submissions is greater than a threshold t. Besides, *QueryClustering-m* aims to detect task-based sessions using a given clustering method m.

5.1 TimeSplitting-t (TS-t)

Intuitively, the simplest techniques for identifying sets of task-related queries from a WSE's log take *only* into account query submission time. Time splitting techniques simply break the stream of queries as long as the time gap between two adjacent queries is greater than a certain threshold t. This is based on the assumption that if two consecutive queries are far away enough than they are also likely to be unrelated. Note that time splitting techniques differ one from each other only for the actual value of t.

According to Def. 3.1, time splitting techniques are used for detecting time-gap sessions. In particular, we use a time threshold $t = 26$ minutes for identifying time-gap sessions of our query log (i.e., TS-26). This threshold have been figured out from the testing data set using the methodology described in Section 4.

Moreover, according to Def. 3.3, TSDP requires to find the best partitioning strategy over all the time-gap sessions available in the query log. A trivial partitioning strategy is the one that simply consider each time-gap session as a task-based session. In our case, this is equivalent to use only TS-26 for addressing the TSDP. However, other partitioning strategies might be figured out simply by applying different time splitting techniques to each identified time-gap session. In this regard, there are several time thresholds that have been extensively proposed in literature [29, 6]. In this work, we used TS-5 and TS-15, i.e., 5 and 15 minutes thresholds, respectively.

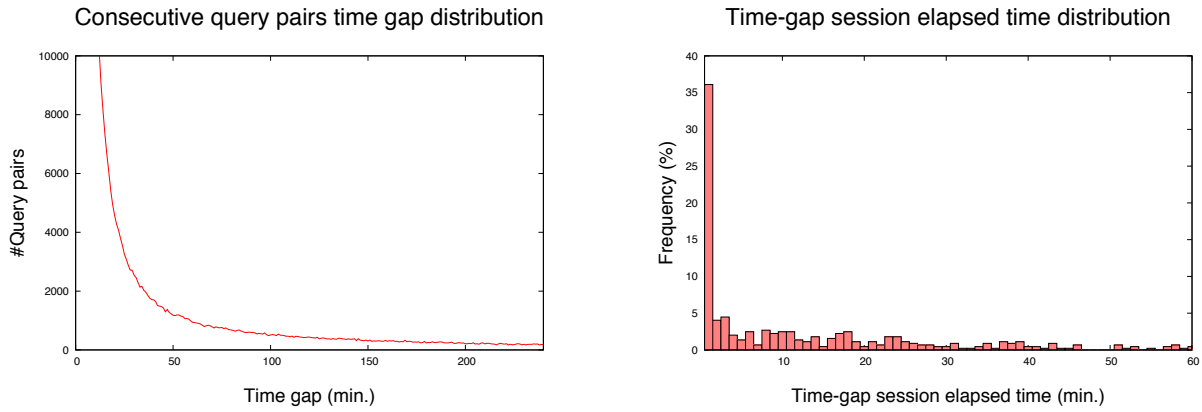
The main drawback of time splitting methods is that they are unable to properly deal with multi-tasking sessions, since identified sets of task-related queries are actually composed of temporarily ordered consecutive queries. Moreover, according to the analysis we provided in Section 4.2, multi-tasking sessions represent a significative sample of the total available sessions, at least for our testing data set.

In Section 6, we compare the results provided by TS-5, TS-15, and TS-26. Also, we show that they alone are not suitable for identifying task-based sessions.

5.2 QueryClustering-m (QC-m)

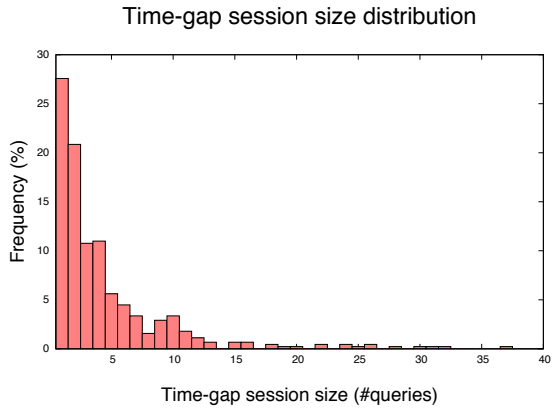
We study three algorithms derived from well-known clustering methods: QC-MEANS, QC-SCAN, and QC-WCC. Moreover, we propose a novel algorithm as a variation of QC-WCC, named QC-HTC. All clustering algorithms have been applied to time-gap sessions, which in turn have been preliminary identified using TS-26 time splitting technique.

As for any other clustering problem, most important choices involve both the features selected for computing the *distance function* used by the algorithms and how such features might be composed, as we show in the following.

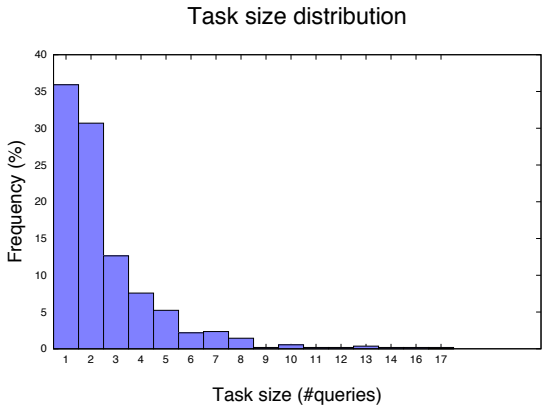


(a)

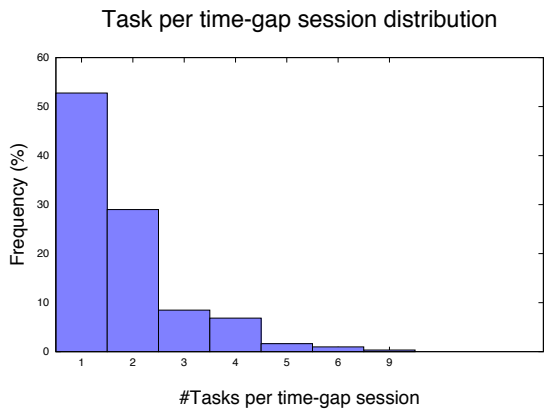
(b)



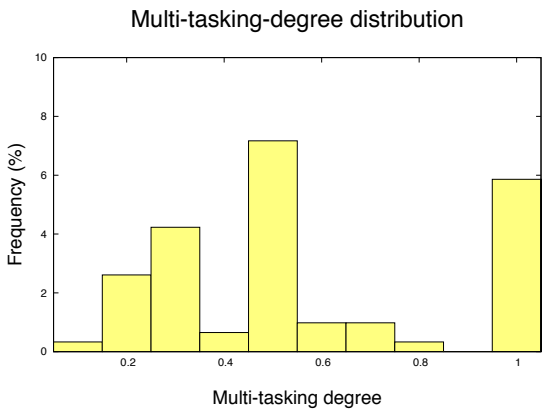
(c)



(d)



(e)



(f)

Figure 1: Statistics about the ground-truth data.

5.2.1 Feature Selection

Evaluating the similarity between two queries is a very complex issue. Most of the previous approaches are based on distance between query lexical content [27]. The precision of those approaches results to be quite low due to the short length of queries [29] and the lack of the contextual information in which queries are issued [33]. Thus, some approaches try to expand those short queries by exploiting resulting URLs returned by WSEs [5], or the returned Web-snippets [16], or the documents themselves [24]. Two queries might be considered similar if they return similar results, or similar documents. Unfortunately, it might be the case for unrelated queries to share some results.

In our work, we propose two features and two similarity measures for assessing the relatedness of two queries both in terms of their lexicographical content and their semantics.

Content-based ($\mu_{content}$). Two queries that share some common terms are likely related. Sometime, such terms may be very similar, but not identical, due to mispelling, or different prefixes/suffixes. To capture content distance between queries, we adopt a Jaccard index on tri-grams [10]. Let $T(q)$ be the tri-grams resulting from the terms of query q, we define the distance $\mu_{jaccard}$ as follows:

$$\mu_{jaccard}(q_1, q_2) = 1 - \frac{|T(q_1) \cap T(q_2)|}{|T(q_1) \cup T(q_2)|}.$$

In addition, we exploit a normalized Levenstein distance $\mu_{levenstein}$, which Jones and Klinkner [11] claimed to be the best edit-based feature for identifying goal boundaries. Finally, the overall content-based distance is computed as follows:

$$\mu_{content}(q_1, q_2) = \frac{(\mu_{jaccard} + \mu_{levenstein})}{2}.$$

Semantic-based ($\mu_{semantic}$). We are interested in finding a measure of the *semantic relatedness* between query pairs. Typically, humans can easily judge the semantic relatedness between two terms. This human ability is backed by their experience and knowledge, which makes it a hard task for machines. If a machine should solve this task, it also needs some source of knowledge. Usually, this knowledge comes from: *(i)* large text collections (i.e., *corpora*) or from *(ii)* semantic resources. Thus, we figured out that we could expand each query with its "*wikification*". Basically, we exploit both Wiktionary and Wikipedia data sources for increasing the meaningfulness of each query, trying to overcome its lack of semantic information.

Several semantic relatedness metrics dealing with semantic resources have been proposed in the past. They can be classified into: *(i) path-based*, in which knowledge is modeled as a graph of concepts and the metrics rely on paths over that graph [22, 13], *(ii) information content-based* that takes into account the information content of a concept [25], *(iii) gloss-based*, which is based on term overlaps between definitions of concepts [15], and *(iv) vector-based* that models each concept as a vector of anchor links [18] or terms [4].

Following the last approach, we assume that a Wiktionary or a Wikipedia article describes a certain concept and that the presence of a term in a given article is an evidence of the correlation between that term and that concept. Thus, we describe the *wikification* $\overrightarrow{C}(t)$ of a term t as its representation in a high dimensional concept space $\overrightarrow{C}(t) = (c_1, c_2, \ldots, c_W)$, where W is the number of articles in our collections and c_i scores the relevance of the term t for the i-th article. We measure this relevance by using the well known tf-idf score [27].

In order to "*wikify*" the whole string associated with a query q, we sum up the contribution from its terms, i.e.:

$$\overrightarrow{C}(q) = \sum_{t \in q} \overrightarrow{C}(t).$$

Then, we compute the relatedness $rel(q_1, q_2)$ between two queries as the cosine of their corresponding concept vectors:

$$rel(q_1, q_2) = \frac{\overrightarrow{C}(q_1) \cdot \overrightarrow{C}(q_2)}{|\overrightarrow{C}(q_1)||\overrightarrow{C}(q_1)|}.$$

Thus, the distance score $\mu_{wikification}$ can be written as follows:

$$\mu_{wikification}(q_1, q_2) = 1 - rel(q_1, q_2).$$

Of course, we use the same approach both for computing $\mu_{wiktionary}$ and $\mu_{wikipedia}$ distances, taking into account Wiktionary and Wikipedia corpora, respectively. Finally, the overall semantic-based distance is obtained as follows:

$$\mu_{semantic}(q_1, q_2) = min(\mu_{wiktionary}, \mu_{wikipedia}).$$

5.2.2 Distance Function

An immediate way to put together *(i)* the lexical content ($\mu_{content}$) and *(ii)* the semantic expansion ($\mu_{semantic}$) distance is via a *convex combination*:

$$\mu_1 = \alpha \cdot \mu_{content} + (1 - \alpha) \cdot \mu_{semantic} \quad (2)$$

In addition, we propose a novel *conditional* distance function μ_2 based on the following heuristic: if the content-based distance between two queries does not exceed a certain threshold then we can be confident that queries are also task-related; otherwise, we look at the semantic expansion of the queries and we compute the final distance score as the minimum between content-based and semantic-based distance values.

$$\mu_2 = \begin{cases} \mu_{content} & \text{if } \mu_{content} < \mathbf{t} \\ \min(\mu_{content}, \mathbf{b} \cdot \mu_{semantic}) & \text{otherwise.} \end{cases} \quad (3)$$

Both μ_1 and μ_2 relies on the estimation of some parameters, i.e., α for μ_1 and $\mathbf{t}$, and $\mathbf{b}$ for μ_2. We exploited the ground-truth for learning such parameters but here we do not show the whole procedure we followed because it is not part of the contributions we wanted to highlight in this work.

The rationale for introducing the *conditional* distance function μ_2 is the following: we conjecture that if two queries are close in term of lexical content, the semantic expansion could be unhelpful. Vice-versa, nothing can be said when queries do not share any common content feature (e.g., consider the two queries "kobe bryant" and "nba").

5.2.3 Algorithms

In the following we describe four clustering algorithms that exploit the above functions. The first three are insipired to well-known clustering algorithms, while the last is a novel algorithm aiming at reducing the computational cost of the clustering.

QC-Means. QC-MEANS is a *centroid-based* algorithm and represents a variation of the well-known K-MEANS [17]. We replaced the usual K parameter, i.e. the number of clusters to be extracted, with a ρ threshold that defines the maximum radius of a centroid-based cluster. This allows us to better deal with the variance in length of the various user sessions as well as to avoid specifying the number of final clusters *apriori*.

QC-Scan. QC-SCAN is the *density-based* DB-SCAN algorithm [3], specifically tailored for extracting task-based sessions from WSE query logs. The rationale for evaluating also a variation of DB-SCAN is that centroid-based approach may suffer the presence of noise in query logs.

QC-wcc. QC-WCC, query clustering based on *weighted connected components*, is a *graph-based* algorithm. Given a time-gap based session ϕ, it builds a complete graph $G_\phi = (V, E, w)$, whose nodes V are the queries in ϕ, and whose E edges are weighted by the similarity of the corresponding nodes. The weighting function w is a similarity function $w : E \longmapsto \mathbb{R} \in [0, 1]$ that can be easily instantiated in terms of the distance functions μ_1 or μ_2, described in the previous section (i.e., $w = 1 - \mu_1$ or $w = 1 - \mu_2$). The graph G_ϕ describes the similarity between any pair of queries in the given tim-gap based session.
The rationale of QC-WCC is to drop *weak edges*, i.e. with low similarity, since the corresponding queries are not related, and to build clusters on the basis of the *strong edges*, i.e. with high similarity, which identify the related query pairs. The algorithm preforms two steps. In the first step, given the graph G_ϕ all the edges $e \in E$ whose weight is smaller than a given threshold, that is $w(e) < \eta$, are removed, thus obtaining a pruned graph G'_ϕ. In the second step, the connected components of G'_ϕ are extracted. Such connected components identify the clusters of related queries which are returned by the algorithm.
Note that this step could have been accomplished by adopting the classification-based method proposed by Jones and Klinkner [11]. However, our aim was to detect task-based sessions using heuristics that extract suitable features directly from the data we had, thus avoiding any kind of preliminary step, which involves the training of a classifier.
Indeed, assuming a robust similarity function, the QC-WCC algorithm is able to handle the multi-tasking nature of users sessions. Groups of related queries are isolated by the pruning of weak edges, and links with large similarity identify the generalization/specialization steps of the users, as well as restarts from a previous query when the current query-chain is found to be unsuccessful.
The cost of QC-WCC is dominated by the construction of the graph G_ϕ. Indeed, the similarity between any pair of edges must be computed, resulting in a number of similarity computations quadratic in the number of nodes, i.e. queries, in the session.

QC-htc. In this section, we propose QC-HTC, query clustering based on *head-tail* components, a variation of the connected components based algorithm, which does not need to compute the full similarity graph. Since queries are submitted one after the other by the user, the QC-HTC algorithms takes advantage of this sequentiality to reduce the number of similarity computations needed by QC-WCC. The algorithm works in two phases as follows.
The first step aims at creating an approximate fine-grained clustering of the given time-gap session $\phi = \langle q_1, q_2, \ldots, q_m \rangle$. Every single Web-mediated task generates a sequence of queries. Due to the multi-tasking behavior of users, multiple Web-mediated tasks are carried on at the same time, and the query log records such overlapping tasks, and the corresponding query sequences. As a consequence, each Web-mediated task is observed as a set of *fragments*, i.e. smaller sets of consecutive queries, and fragments of different tasks are interleaved in the query log because of multi-tasking.
The algorithm exploits the sequentiality of user queries, and tries to detect the above fragments, by partitioning the given time-gap session into *sequential clusters*, where a sequential cluster, denoted with $\tilde{c}^i$, must contain only queries that occur in a row within the query log, and such that each query is sufficiently similar to the chronologically following one. Since we are only focussing on the similarity between one query and the next, the detection of such *sequential clusters* can be done in linear time.
The second step of the algorithm merges together those fragments when they are related, trying to overcome the interleaving of different tasks. Here we introduce another assumption that reduces the computational cost of the algorithm. We assume that a cluster of queries can be described well by just the chronologically first and last of its queries, respectively denoted with $head(\tilde{c}^i)$ and $tail(\tilde{c}^i)$. Therefore, the similarity s between two clusters $\tilde{c}^i$,$\tilde{c}^j$ is computed as:

$$s(\tilde{c}^i, \tilde{c}^j) = \min_{\substack{q \in \{head(\tilde{c}^i), tail(\tilde{c}^i)\} \\ p \in \{head(\tilde{c}^j), tail(\tilde{c}^j)\}}} w(e(q, p))$$

where w weights the edge $e(q, p)$ linking the queries p and q on the basis of their similarity, analogously to QC-WCC.
The final clustering is produced as follows. The first cluster c^1 is initialized with the oldest sequential cluster $\tilde{c}^1$, which is removed from the set of sequential clusters. Then, c^1 is compared with any other sequential cluster $\tilde{c}^i$ (ordered chronologically) by computing the similarity as above. Given a threshold η, if $s(c^1, \tilde{c}^i) < \eta$, then $\tilde{c}^i$ is merged into c^1, the *head* and *tail* queries of c^1 are updated consequently, and $\tilde{c}^i$ is removed from the set of sequential clusters. The algorithm continues comparing the new cluster c^1 with the remaining sequential clusters. When all the sequential clusters have been considered, the oldest sequential cluster available is used to build a new cluster c^2.
The algorithm iterates this procedure until no more sequential clusters are left.
In the worst case, the complexity of QC-HTC is still quadratic in the number of queries. However, there are frequent cases in which the complexity is much smaller. We have seen that 52.8% of the time-gap bases sessions contain one task only. In this case, it is very likely that this task is found after the first step of the algorithm, if each query is sufficiently similar to the next one, with a cost that is only linear in the number of nodes. For multi-tasking sessions, the complexity in the second step is quadratic in the number of sequential clusters extracted. Also in this case, the cost reduction may be significant. Suppose that the number of sequential clusters is $\rho|\phi|$, with $0 \leq \rho \leq 1$, then the complexity of the algorithm is $O(\rho^2|\phi|^2)$. Suppose that the number of sequential clusters is half the number of queries, than the algorithm is four times cheaper than QC-WCC. Still, this is an upper bound of the cost, since the QC-HTC algorithm does not compute the pair-wise similarities among sequential clusters in advance.

6. EXPERIMENTS

In this section we analyze and compare the results obtained with all the methods we described in Section 5 for approaching the TSDP. Moreover, we compare our results with the ones provided by two other methods: *(i)* the simple time splitting technique TS-26, which is considered as the *baseline* solution and *(ii)* the session extraction method based on the *query-flow graph* model proposed by Boldi *et al.* [1], which represents the state of the art.

6.1 Measures of validity

In order to evaluate and compare all the methods we mentioned, we need to measure the degree of correspondence between manually-extracted tasks of the ground-truth and tasks produced by our algorithms. To this end, we use both *classification-* and *similarity-oriented* measures [32]. In the following, we thus refer to "*predicted classes*" as the task-based sessions detected by a specific algorithm, whereas the "*actual classes*" just correspond to the task-based sessions of the ground-truth.

Classification-oriented approaches measure the degree to which predicted classes correspond to actual classes and *F-measure* is one of the most popular measure in this category. It combines both *precision* and *recall*: precision measures the fraction of a task that consists of objects of a specified class, while recall measures the extent to which a task contains all objects of a specified class. Thus, globally F-measure evaluates the extent to which a task contains *only* objects of a particular class and *all* objects of that class. Given $p(i,j)$, $r(i,j)$ the precision and recall of task i with respect to class j respectively, the F-measure is computed as: $F(i,j) = \frac{2 \times p(i,j) \times r(i,j)}{p(i,j)+r(i,j)}$.

We have to compute F-measure for each task-based session detected by an algorithm, where discarded queries are considered as singleton tasks (i.e., tasks containing only one query). Finally, an overall F-measure value is computed by a weighted average over all the tasks, by considering the size of each task as weight.

Besides, similarity-oriented measures consider pairs of objects instead of single objects. Hence, given f_{00} = number of pairs of objects having a different class and a different task, f_{01} = number of pairs of objects having a different class and the same task, f_{10} = number of pairs of objects having the same class and a different task, and f_{11} = number of pairs of objects having the same class and the same task, two measures are defined: *(i) Rand index* $R = \frac{f_{00}+f_{11}}{f_{00}+f_{01}+f_{10}+f_{11}}$ and *(ii) Jaccard index* $J = \frac{f_{11}}{f_{01}+f_{10}+f_{11}}$.

When computing both Rand and Jaccard index we did not consider time-gap sessions containing only one singleton task. Anyway, we still took into account time-gap sessions that are composed of a single task with more than one query.

6.2 Evaluation

TimeSplitting-t. In this work, we compare three different time splitting techniques: TS-5, TS-15, and TS-26, which use 5, 15, and 26 minutes thresholds, respectively. Tab. 1 shows the results we obtained using those techniques on the ground-truth. The best result in terms of F-measure was found considering the whole time-gap sessions identified with TS-26, without additionally splitting them into shorter time-gap sessions.

Hence, we consider TS-26 as the *baseline* approach for addressing the TSDP. Roughly, according to Def. 3.1 and Def. 3.2, this is equivalent to identify task-based sessions with time-gap sessions.

Table 1: TS-5**,** TS-15**, and** TS-26**.**

	F-measure	Rand	Jaccard
TS-5	0.28	**0.75**	0.03
TS-15	0.28	0.71	0.08
TS-26	**0.65**	0.34	**0.34**

Query Flow Graph. In order to better evaluate our proposed approaches we decided to compare them to the *query-flow graph* (QFG) presented by Boldi *et al.* [1].

QFG has been constructed over a training segment of the AOL *top*-1000 user sessions. This method uses *chaining probabilities* measured by means of a machine learning method. The initial step was to extract those features from the training log, and storing them into a compressed graph representation. In particular, we extracted 25 different features (i.e., time-related, session and textual features) for each pair of queries (q, q') that are consecutive in at least one session of the query log.

The validity of QFG has been tested on the ground-truth and the results we obtained are showed in Tab. 2. We found the best values using a threshold $\eta = 0.7$. In fact, results do not improve using a greater threshold value.

QFG significantly improved the baseline TS-26. In particular, F-measure is improved with a gain of $\approx 16\%$. Furthermore, QFG gained $\approx 52\%$ in terms of Rand and $\approx 15\%$ in terms of Jaccard.

Table 2: QFG**: varying the threshold** η**.**

	η	F-measure	Rand	Jaccard
QFG	0.1	0.68	0.47	0.36
	0.2	0.68	0.49	0.36
	0.3	0.69	0.51	0.37
	0.4	0.70	0.55	0.38
	0.5	0.71	0.59	0.38
	0.6	0.74	0.65	0.39
	0.7	**0.77**	**0.71**	**0.40**
	0.8	0.77	0.71	0.40
	0.9	0.77	0.71	0.40

QueryClustering-m. We now compare all the clustering methods described in Section 5.2.3.

We start evaluating QC-Means algorithm using both the distance functions μ_1 and μ_2. We empirically set the radius of this centroid-based algorithm to 0.4 for both distance functions and the results are showed in Tab. 3.

Concerning μ_1, the best results were obtained by using only the content-based distance, i.e., with $\alpha = 1$. However, the very best results for QC-Means were found when using μ_2. Here, we significantly improved the baseline TS-26 in terms of F-measure ($\approx 10\%$) and Rand ($\approx 54\%$), while we lost $\approx 21\%$ in terms of Jaccard. Moreover, if we compare the best QC-Means with the best QFG we can notice that QC-Means lost $\approx 6\%$ for F-measure, $\approx 33\%$ for Jaccard but it gained $\approx 4\%$ in terms of Rand.

Then, we analyzed QC-Scan algorithm, again using both the distance functions μ_1 and μ_2. We used several combinations of the two density-based parameters, i.e., $minPts$ and eps, and we found the best results with $minPts = 2$ and $eps = 0.4$.

Tab. 4 highlights that QC-Scan provided globally better results than QC-Means for both μ_1 and μ_2. Still, for μ_1

Table 3: QC-MEANS: μ_1 **vs.** μ_2.

QC-MEANS μ_1				
		F-measure	Rand	Jaccard
α	$(1-\alpha)$			
1	0	0.71	0.73	0.26
0.5	0.5	0.68	0.70	0.14
0	1	0.68	0.70	0.13

QC-MEANS μ_2				
		F-measure	Rand	Jaccard
t	b			
0.5	4	**0.72**	**0.74**	**0.27**

the best results were obtained by using only content-based distance, i.e., with $\alpha = 1$. However, our proposed conditional function μ_2 revealed a significative improvement with respect to all measures.
Finally, it is worth noticing that QC-SCAN behaved exactly as QFG, except for the Jaccard where QC-SCAN lost $\approx 53\%$.

Table 4: QC-SCAN: μ_1 **vs.** μ_2.

QC-SCAN μ_1				
		F-measure	Rand	Jaccard
α	$(1-\alpha)$			
1	0	0.77	0.71	0.17
0.5	0.5	0.74	0.68	0.06
0	1	0.75	0.68	0.07

QC-SCAN μ_2				
		F-measure	Rand	Jaccard
t	b			
0.5	4	**0.77**	**0.71**	**0.19**

The third algorithm we considered is QC-WCC. Tab. 5 shows the results we found using this algorithm either with distance function μ_1 and μ_2 and by varying the pruning threshold η. In particular, concerning μ_1 we only considered the best convex combination when $\alpha = 0.5$.
The best results with μ_1 were obtained when $\eta = 0.2$, while even better results were found with μ_2 when $\eta = 0.3$. In this last case, the overall evaluation was significantly higher than the baseline TS-26 but also than the state-of-the-art approach QFG. Concerning TS-26, the best QC-WCC gained $\approx 20\%$, $\approx 56\%$, and $\approx 23\%$ in terms of F-measure, Rand, and Jaccard, respectively. Moreover, QC-WCC improved also the results of QFG, gaining $\approx 5\%$ for F-measure, $\approx 9\%$ for Rand, and $\approx 10\%$ for Jaccard.
QC-HTC is the last algorithm we introduced and represents one of the novel contribution of our work. The results we got using this approach with both distance functions μ_1 and μ_2 and by varying the pruning threshold η are showed in Tab. 6. As for QC-WCC, regarding μ_1 we only considered the best convex combination when $\alpha = 0.5$. Again, the best results with μ_1 were obtained when $\eta = 0.2$, while the global best results were found with μ_2 when $\eta = 0.3$. As the table shows, the overall results are very close to the ones obtained with QC-WCC. In particular, QC-HTC improved TS-26 by gaining $\approx 19\%$, $\approx 56\%$, and $\approx 21\%$ in terms of F-measure, Rand, and Jaccard, respectively. Therefore, QC-HTC provided better results than QFG and gained $\approx 4\%$ for F-measure, $\approx 9\%$ for Rand, and $\approx 8\%$ for Jaccard.
Globally, Tab. 7 gives an overview and compares the best results found with each examined approach.
Finally, Tab. 8 clearly points out the benefit of exploiting collaborative knowledge like Wikipedia. QC-HTC was able

Table 5: QC-WCC: μ_1 **vs.** μ_2 **varying the threshold** η.

QC-WCC μ_1 ($\alpha = 0.5$)			
η	F-measure	Rand	Jaccard
0.1	0.78	0.71	0.42
0.2	**0.81**	**0.78**	**0.43**
0.3	0.79	0.77	0.37
0.4	0.75	0.73	0.27
0.5	0.72	0.71	0.20
0.6	0.75	0.70	0.14
0.7	0.74	0.69	0.11
0.8	0.74	0.68	0.07
0.9	0.72	0.67	0.04

QC-WCC μ_2 ($t = 0.5, b = 4$)			
η	F-measure	Rand	Jaccard
0.1	0.67	0.45	0.33
0.2	0.78	0.71	0.42
0.3	**0.81**	**0.78**	**0.44**
0.4	0.81	0.78	0.41
0.5	0.80	0.77	0.37
0.6	0.78	0.75	0.32
0.7	0.75	0.73	0.23
0.8	0.71	0.70	0.15
0.9	0.69	0.68	0.08

Table 6: QC-HTC: μ_1 **vs.** μ_2 **varying the threshold** η.

QC-HTC μ_1 ($\alpha = 0.5$)			
η	F-measure	Rand	Jaccard
0.1	0.78	0.72	0.41
0.2	**0.80**	**0.78**	**0.41**
0.3	0.78	0.76	0.35
0.4	0.75	0.73	0.25
0.5	0.73	0.70	0.18
0.6	0.75	0.70	0.13
0.7	0.74	0.69	0.10
0.8	0.74	0.68	0.06
0.9	0.72	0.67	0.03

QC-HTC μ_2 ($t = 0.5, b = 4$)			
η	F-measure	Rand	Jaccard
0.1	0.68	0.56	0.32
0.2	0.78	0.73	0.41
0.3	**0.80**	**0.78**	**0.43**
0.4	0.80	0.77	0.38
0.5	0.78	0.76	0.34
0.6	0.77	0.74	0.30
0.7	0.74	0.72	0.21
0.8	0.71	0.70	0.14
0.9	0.68	0.67	0.07

Table 7: Best results obtained with each method.

	F-measure	Rand	Jaccard
TS-26 (*baseline*)	0.65	0.34	0.34
QFG $_{best}$ (*state of the art*)	0.77	0.71	0.40
QC-MEANS $_{best}$	0.72	0.74	0.27
QC-SCAN $_{best}$	0.77	0.71	0.19
QC-WCC $_{best}$	**0.81**	**0.78**	**0.44**
QC-HTC $_{best}$	0.80	0.78	0.43

to capture and group together two queries that are completely different from a content-based perspective, but that are strictly semantically-related, using the function μ_2. Indeed, "Cancun" is one of the region where the "Hurricane Wilma" impacted during the 2005 season (see the cross reference inside the corresponding Wikipedia article[4]). Moreover, "Los Cabos" and "Cancun" are both in Mexico despite they are far away from each other. It might be the case,

[4] http://en.wikipedia.org/wiki/Cancun

of course with no absolute certainty, the user was looking for the relative position of Los Cabos from Cancun just to understand if Los Cabos was struck by the hurricane as well.

Table 8: The impact of Wikipedia: μ_1 vs. μ_2

QC-HTC μ_1 ($\alpha = 1$)		QC-HTC μ_2 (0.5, 4)	
Query ID	Query String	Query ID	Query String
		63	los cabos
		64	cancun
65	hurricane wilma	65	hurricane wilma
68	hurricane wilma	68	hurricane wilma

7. CONCLUSION AND FUTURE WORK

We have discussed a technique for splitting into meaningful user sessions a very large, long-term log of queries submitted to a Web Search Engine (WSE). We have formally introduced the *Task-based Session Discovery Problem* as the problem of extracting from a stream of user's queries several subsequences of queries which are all related to the same search goal, i.e., a *Web-mediated task*. We have also proposed a clustering-based solution, leveraging distance measures based on query content and semantics, while query timestamps were used for a first pre-processing breaking phase. In particular, we exploited both Wikipedia and Wiktionary to infer the semantics of a query. Our novel graph-based heuristic, QC-HTC, which is a simplification of the weighted connected components QC-WCC, significantly outperforms other heuristics in terms of F-measure, Rand and Jaccard index. As future work, we plan to learn a model that describes how users compose together several tasks for enacting more complex *Web-mediated processes*. Finally, we aim at investigating how such processes or parts of them can be recommended, devising a novel recommender system that goes beyond the simple query suggestion of modern WSEs.

8. ACKNOWLEDGMENTS

We acknowledge the partial support of S-CUBE (EU-FP7-215483), ASSETS (CIP-ICT-PSP-250527), and VISITO Tuscany (POR-FESR-63748) projects. Also, we acknowledge the authors of [1] and the Yahoo! Research Lab of Barcelona for providing us the possibility of using their Query Flow Graph implementation for evaluation purposes. Authors also thank Franco Maria Nardini for the help provided in adapting the original Query Flow Graph implementation.

9. REFERENCES

[1] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: model and applications. In *CIKM '08*, pages 609–618. ACM, 2008.

[2] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, 2002.

[3] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD '96*, pages 226–231. ACM, 1996.

[4] E. Gabrilovich and S. Markovitch. Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis. In *IJCAI*, pages 6–12, 2007.

[5] N. S. Glance. Community search assistant. In *IUI '01*, pages 91–96. ACM, 2001.

[6] D. He and A. Göker. Detecting session boundaries from web user logs. In *BCS-IRSG*, pages 57–66, 2000.

[7] D. He and D. J. Harper. Combining evidence for automatic web session identification. *IPM*, 38(5):727–742, 2002.

[8] B. J. Jansen and A. Spink. How are we searching the world wide web?: a comparison of nine search engine transaction logs. *IPM*, 42(1):248–263, 2006.

[9] B. J. Jansen, A. Spink, C. Blakely, and S. Koshman. Defining a session on web search engines: Research articles. *JASIST*, 58(6):862–871, 2007.

[10] A. Järvelin, A. Järvelin, and K. Järvelin. s-grams: Defining generalized n-grams for information retrieval. *IPM*, 43(4):1005 – 1019, 2007.

[11] R. Jones and K. L. Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In *CIKM '08*, pages 699–708. ACM, 2008.

[12] T. Lau and E. Horvitz. Patterns of search: Analyzing and modeling web query refinement. In *UM '99*, pages 119–128. Springer Wien, 1999.

[13] C. Leacock and M. Chodorow. *Combining Local Context and WordNet Similarity for Word Sense Identification*, chapter 11, pages 265–283. The MIT Press, May 1998.

[14] U. Lee, Z. Liu, and J. Cho. Automatic identification of user goals in web search. In *WWW '05*, pages 391–400. ACM, 2005.

[15] M. Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *SIGDOC '86*, pages 24–26. ACM, 1986.

[16] K. W.-T. Leung, W. Ng, and D. L. Lee. Personalized concept-based clustering of search engine queries. *IEEE TKDE*, 20(11):1505–1518, 2008.

[17] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. UC Press, 1967.

[18] D. Milne and I. H. Witten. An effective, low-cost measure of semantic relatedness obtained from wikipedia links. In *AAAI '08*, pages 25–30, 2008.

[19] S. Orlando and F. Silvestri. Mining query logs. In *ECIR*, volume 5478 of *LNCS*, pages 814–817. Springer, 2009.

[20] H. C. Ozmutlu and F. Çavdur. Application of automatic topic identification on excite web search engine data logs. *IPM*, 41(5):1243–1262, 2005.

[21] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[22] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE TSMC*, 19(1):17–30, 1989.

[23] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. In *KDD '05*, pages 239–248. ACM, 2005

[24] V. V. Raghavan and H. Sever. On the reuse of past optimal queries. In *SIGIR '95*, pages 344–350. ACM, 1995

[25] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, pages 448–453, 1995.

[26] D. E. Rose and D. Levinson. Understanding user goals in web search. In *WWW '04*, pages 13–19. ACM, 2004.

[27] G. Salton and M. J. Mcgill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1986.

[28] X. Shen, B. Tan, and C. Zhai. Implicit user modeling for personalized search. In *CIKM '05*, pages 824–831. ACM, 2005.

[29] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

[30] F. Silvestri. Mining query logs: Turning search usage data into knowledge. *Foundations and Trends in Information Retrieval*, 1(1-2):1–174, 2010.

[31] A. Spink, M. Park, B. J. Jansen, and J. Pedersen. Multitasking during web search sessions. *IPM*, 42(1):264–275, 2006.

[32] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, May 2005.

[33] J. R. Wen, J. Y. Nie, and H. Zhang. Query clustering using user logs. *ACM TOIS*, 20(1):59–81, 2002.

[34] R. W. White, M. Bilenko, and S. Cucerzan. Leveraging popular destinations to enhance web search interaction. *ACM TWEB*, 2(3):1–30, 2008.

Appendix E

Resource and Agreement Management in Dynamic Crowdcomputing Environments

Resource and Agreement Management in Dynamic Crowdcomputing Environments

Harald Psaier, Florian Skopik, Daniel Schall, Schahram Dustdar
Distributed Systems Group
Vienna University of Technology
Argentinierstraße 8/184-1, A-1040 Vienna, Austria
{lastname}@infosys.tuwien.ac.at

***Abstract*—Open Web-based and social platforms dramatically influence models of work. Today, there is an increasing interest in outsourcing tasks to crowdsourcing environments that guarantee professional processing. The challenge is to gain the customer's confidence by organizing the crowd's mixture of capabilities and structure to become reliable. This work outlines the requirements for a reliable management in crowdcomputing environments. For that purpose, distinguished crowd members act as responsible points of reference. These members mediate the crowd's workforce, settle agreements, organize activities, schedule tasks, and monitor behavior. At the center of this work we provide a hard/soft constraints scheduling algorithm that integrates existing agreement models for service-oriented systems with crowdcomputing environments. We outline an architecture that monitors the capabilities of crowd members, triggers agreement violations, and deploys counteractions to compensate service quality degradation.**

***Keywords*-crowdsourcing, service agreements, scheduling, behavior-based adaptation**

I. Introduction

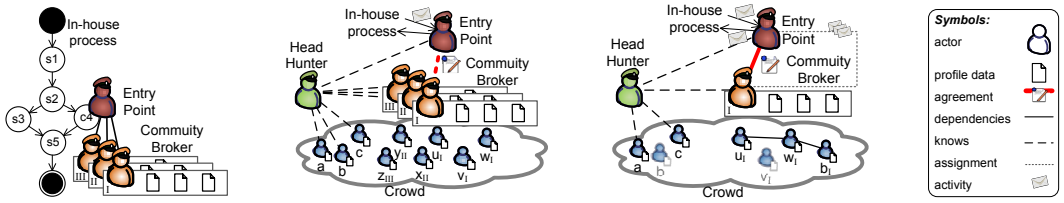
Crowdsourcing applications [1] are typically open Internet based platforms where problem-solving tasks are distributed among a group of humans. Crowdsourcing follows the 'open world' assumption, allowing humans to provide their capabilities through the platform by registering themselves as members. A popular crowdsourcing platform is for example Amazon Mechanical Turk [2]. Currently, two types of crowdsourcing modes have been identified [3]. In marketplace oriented crowdsourcing crowds are organized by providers or brokers that bid for and distribute requests. In competition based crowdsourcing the request is an open call to the crowd and the winning submission is picked.

While conventional enterprise systems rely on well established policies, crowdsourcing has a more loosely-coupled, dynamic, and flexible structure and depends especially on the preferences and behavior of the individual crowd members. Even if an advantage of a crowd platform is the possibility to choose from a larger number of skilled members, the selection must consider, e.g., the distinguished working hours of the members possibly contradicting with the current requirements. The members availability will mostly depend on their context. Their working hours, for example, also depend on their location and the related timezone. Context does not only influence task assignment strategies but certain changes can also cause unpredictable interruptions of services. This leads to an incomplete and unsatisfactory task state at deadline. As a consequence, meeting promised service contracts is challenging and demands for sophisticated management techniques for a crowd platform. One of the key issues of the management investigated in this work is to find an appropriate scheduling for task assignments that matches tasks to skills and to the availability of the members, and above all, also meets the agreed contract. This adaptive scheduling strategy must constantly update on changes, and ensure, that shifts in interest, skills, and behavior of members including task-related misbehavior, such as degrading worker performance, refusal of tasks, or missing feedback that affects successful task completion is detected, avoided, and balanced with alternative workforce.

In this paper we describe a framework which integrates agreements into service-oriented crowdsourcing platforms. The prerequisite is a monitoring infrastructure that updates a crowd-based resource model. In our previous work [4], [5], we studied a monitoring and behavior adaptation architecture serving as the basis for the presented agreement management. Here, we focus on an agreement model combined with an adaptation approach for reliable task execution. An accurate resource model supports a sophisticated scheduling of crowd activities. A self-adaptive mechanism must anticipate misbehavior and update the crowd's task scheduling to enforce behavior rules also dictated by the agreement.

Here we address the following challenges:

- *Human Behavior Characteristics.* The crowd is a transient network where humans can join and leave the platform at any time. Furthermore, in contrast to software services, human behavior is subject to numerous contextual constraints.
- *Feedback Loop.* Since the crowd's resources, i.e., available members and their capabilities, are in a constant flux and change, models for monitoring the environment and accounting for given constraints need to be applied.
- *Quality Guarantees.* It is challenging to provide any guarantees regarding execution time and reliability of actors in highly dynamic environments. Our flexible agreement management models tackle that problem by allowing management of negotiated agreements.



(a) Test cycle. (b) Agreement setup. (c) Assignment. (d) Symbols.

Figure 1. Crowdsourcing software tests.

This paper is structured as follows: in Section II related work is discussed followed by a section describing a crowdcomputing scenario. Section IV details a framework that addresses identified challenges regarding agreement management. Section V discusses the layout of agreements and use of quality metrics for adapting crowd scenarios in order to meet agreements. One promising adaptation approach is dynamic re-scheduling of tasks in crowds which is highlighted in Section VI. Section VII shows evaluation results and Section VIII concludes the paper.

II. Related Work

Crowdsourcing applications [1] are online, distributed problem-solving and production models that have emerged in recent years ([2], [6], [7]). A vast number of registered individuals offer solutions to the problem. Apart from the benefit of multiple, redundant workforce and collective intelligence, crowdsourcing poses some difficult challenges related to its distributed and open nature. The main challenge remains how to organize and manage the crowd and identify potential leaks of resources. Two operating modes, marketplace and competition based crowd platforms, have been identified so far [3]. Our assumptions base on a marketplace oriented crowdsourcing environment where mediators manage the crowd. Moreover, we base our ideas around existing mediator services that post tasks on behalf of their clients [8].

There has been substantial research on translations of **service level agreements** to a Web-service applicable standard. Two of the main contributions are the specification from the Grid Resource Allocation Agreement Protocol (GRAAP) Working Group [9] and from IBM [10]. Both present similar XML-based model for an SLA, however, differ in the details. When creating their specification the GRAAP Working Group in particular focuses on the setup, negotiation, and renegotiation phases of the agreement, thus, presents a rather flexible structure [11]. IBM's WSLA focus was on defining agreement objectives, their constraints and combination. For this purpose parameters can be linked to SLOs together with thresholds. In our work we reuse the parameter scheme to define our quality attributes.

Scheduling is a well-known subject in computer science. The novel contribution in this work is to consider multidimensional assignment and allocation of tasks. A thorough analysis and investigation in the area of multidimensional optimal auctions and the design of optimal scoring rules has been done by [12]. In [13] iterative multi-attribute procurement auctions are introduced while focusing on mechanism design issues and on solving the multi-attribute allocation problem. Focusing on task-based adaptation, [14] near-optimal resource allocations and reallocations of human tasks were presented. Workforce scheduling research investigates the impact of weights on the multiple, at times contradicting, objectives in real work workforce scheduling [15]. Staff scheduling related to closed systems was discussed in [16], [17]. However, unlike in closed enterprise systems, crucial scheduling information, i.e., the current user load or precise working hours are usually not directly provided by the crowd. Instead, the scheduling relevant information must be gathered by monitoring. The work in [18] details the challenges for collaborative workforce in crowdsourcing where activities are coordinated, workforce contributions are not wasted and final results are guaranteed.

III. Crowdcomputing Environment

This section motivates our work and outlines a crowdcomputing environment. With reference to existing testing marketplaces, c.f. [19], we present a software testing scenario and discuss the different roles of crowd members.

A. Roles in Crowdcomputing

The crowd *Entry Point(EP)* is a mediator that connects customers from outside the crowd with the required crowd members (see Figure 1). Generally, crowd customers look for a certain knowledge or capability which their company environments lack. Thus, the EP maintains regular contacts to required crowd members. It acts as representative of a company's outsourced assignments to the crowd and must assure all implications to the dependencies with the company's internals. The *Community Broker (CB)* is a proxy for a certain crowd segment or platform. It maintains and represents a group of registered members with similar capabilities and offers the joint knowledge to interested parties, e.g., EPs. In the example of Figure 1, the representative of platform I is the CB of the crowd member group u, v and w with a I subscript each. A CB is in charge of a fair distribution of the incoming assignments and settles agreements. The *Head Hunter (HH)* acts as a kind of registry

for a crowd environment. It monitors the tendencies of the required capabilities in the crowd environment, and discovers new knowledge sources (single members or groups). Additionally, it offers an interface that allows new members to register, and further, to be discovered. EPs and CBs can find required partners and members using the HH's lookup service. In the example in Figure 1, the HH provides, e.g., for the new crowd members a, b, and c a first point of reference to enter the crowd business. We argue that, between entities of these roles, relations based on agreements also need to be established. Agreements state rules that organize and regulate the assignments of tasks in the crowd. They help to assure dependencies and regulate the rather unpredictable behavior of an unorganized crowd.

B. Use Case Scenario

In Figure 1 we outline the various phases of a typical crowdsourcing software test scenario. Beginning with Figure 1(a) the in-house quality assurance (QA) process is split into five repetitive and automated steps (*s1*, *s2*, *s3*, the crowdsourcing step *c4*, and *s5*). In *s1* all modules that need to be tested are collected for the next upcoming QA cycle. In *s2* a sorting process differs between automated tests that run in the company's own test environment and those that need to be crowdsourced by, e.g., monitoring a flag on the test units provided by the QA team. Step three (*s3*) represents the test period run in-house. In parallel, in step *c4* a company's representative, EP, is in charge of a smooth flow of the crowdsourced test activities. In order to get into this position, s/he needs to have some previously established relations to available crowd platforms, e.g., *I*, *II*, and *III*, and their representative CBs. Having numerous alternatives for outsourcing guarantees a reliable management of this test period. The final step *s5* collects the testing results for an evaluation and merge.

Next, in Figure 1(b), the EP has to decide which crowd community can handle the currently pending test activities. The assignment depends on the requirements of the tasks and resources of current platforms, in particular, the members' capacities and capabilities. Next, the EP must balance the effort, expected quality, and costs of the available CBs' offers. If none of the known CBs fits the requirements, the EP can invoke the HH lookup service to find a new CB. Thus a HH mediates CBs to an EP on request. In the example, however, this is not the case and EP contacts the chosen platform I's representative. An agreement for the following assignment is negotiated.

Figure 1(c) illustrates the scenario at runtime. Testing activity segments, i.e. tasks, are delegated to the appropriate crowd members. As mentioned in the introduction, the crowd's structure is transient and its members' processing attitude is context dependent and individual, thus, at times unpredictable. In the given example, nodes u, v, and w process dependent tasks. However, v is not able to process the tasks as scheduled. It is now the challenge of the crowd management to find a solution. A first solution would be to reschedule the task. Unfortunately, in the presented scenario no member can replace v. The CB must call the assistance of the HH and request a fitting, though previously unknown, crowd member. As depicted, member b is mediated to CB of platform *I* and takes over the duties of node v. Finally, once all tasks have been completed, the results are collected and merged by the EP. Thereafter, the final result is provided to the evaluation step (*s5*).

C. Agreement Management

A CB is usually either a business person who established a dedicated platform and invited crowd members to join, or, has emerged from a formation of members with the same interests to represent them. Because crowd environments are open systems with no guarantees, the main role of the CB is to fill this gap. S/He provides the otherwise missing, however, necessary guarantees to the EP. In particular, guarantees in this scenario include a satisfying, proper conduct of the tests. Just like in a cooperation between two companies, EP and CB, set-up an agreement for the outsourced test activity. The agreement identifies the activity, settles the test scheduling, and states metrics to measure the demanded quality. The quality preferences include attributes, such as, maximum tolerated running time for the assigned tasks, fees, demands on the result, etc..

Our proposed agreement management approach employs the following fundamental concepts when distributing tasks in the crowd:

Hard- and Soft-Constraints. We distinguish between criteria that must be met, e.g., expertise area of crowd members and their principal participation interest and so-called soft constraints that are used for ranking potential crowd members, including their capacity, reputation, and costs.

Environment Observation. Periodic run-time monitoring and evaluation of the crowd members' behavior in terms of reliability and task execution progress enables timely detection of misbehavior and quality degradations.

Adaptation and Optimization. Using feedback data obtained from behavior monitoring enables numerous adaptation mechanisms to optimize the assignment and execution of tasks in the crowdcomputing environment; for instance the reassignment and/or rescheduling of tasks in case of deadline misses.

IV. Architecture

The new extensions to our previous work on the $VieCure\ Framework$ [5] are detailed in the following section. The additions include monitoring of agreements and scheduling of assignments based on policies.

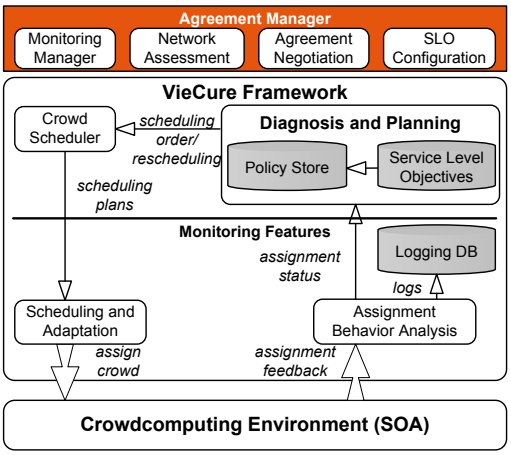


Figure 2. Agreement management framework.

A. Architectural Overview

Figure 2 outlines the three-layer infrastructure of the framework extension. The top layer comprises the *Agreement Manager*. This is a tool-set to monitor, track, and analyze the crowd structure. Additionally, by hiding the particularities of the scheduling technique, it allows to extend and change the framework's *SLOs* and *Policy Store* entries, thus, adapt the environment to new agreements. The layer in the middle represents the framework itself. It is organized according to an **automated adaptation loop** and its main purpose is to adapt the *Crowd Scheduler*'s assignment strategy. The strategy depends on the current crowd's acceptance behavior and capacities, as well as, on policies representing system and agreement constraints. Therefore, interfacing with the environment, the *Assignment Behavior Analysis* collects feedback to a log database and forwards the current status to the *Diagnosis and Planning* module. Considering the valid policies and the fresh assignment status from analysis this module adapts the scheduling order, and/or, on an assignment reject, issues a rescheduling directive with new ordering rules to the *Crowd Scheduler*. Depending on the current situation the *Crowd Scheduler* uses its algorithm to assign a batch of tasks, or on request of *Diagnosis and Planning* module, reschedules a unsuccessful assignment. Finally, the scheduling result is transmitted to the *Scheduling and Adaptation* module. This deploys the assignments and scheduling changes to the crowd.

B. Further Building Blocks

The bottom layer of the architecture in Figure 2 bases on a *service-oriented architecture (SOA)*. In our previous work, we studied flexible interactions [20], metrics in crowds [21], and monitoring of service-oriented collaboration environments [5]. Here we give a quick overview of the main principles and our findings.

SOA-based Interactions in Crowds. Dynamic discovery of services, flexible interactions and compositions at runtime are only some properties that make SOAs an intuitive and convenient technical grounding for large-scale crowdsourcing environments. However, not only service interactions, but also human interactions may be performed using SOAP (see Human-Provided Services [20] for collaborative environments and BPEL4People [22] for human interactions in business processes), which is the state-of-the-art technology to exchange XML-based messages in service-oriented environments, and well supported by a wide variety of software frameworks. There exist well-known and accepted standards for modeling and monitoring service level agreements that act as an integral part of our approach (c.f., Section II).

Failure Compensation through Dynamic Adaptation. Performance degradation and failures may arise due to various reasons. Especially in crowdsourcing environments, human (mis-)behavior has a fundamental influence on the overall success rate regarding task execution and throughput. We studied an approach to rate and categorize human behavior earlier [5] to be able to compensate malicious behavior in collaborative networks. For that purpose, typically adaptation rules are pre-defined (e.g., seize tasks from unreliable workers) and applied according to adaptation policies. These mechanisms rely on monitoring data that is captured from the service-oriented infrastructure.

The next section gives a detailed description of the sequence of operations between the framework's modules and outlines their interaction with the system in the various phases of an agreement's life-cycle.

C. Agreement Life-Cycle

The life-cycle of the agreement includes three distinct phases. In the first phase, offers are invited and an agreement for the assignment is negotiated. With the agreement as a base for the business relation, the additional environment policies need to be applied in the line with the agreement. Next, tasks are scheduled and assigned to the crowd members according to the policies order. In parallel, the assignments status is diagnosed. A rejected assignment must be rescheduled.

The sequence in Figure 3 presents an agreement's life-cycle. It details the interactions between the involved parties, EP and CB presented in Section III, and the automated $VieCure$ crowd assignment management.

Negotiation. While an EP browses for interesting bids, the CB uses the $VieCure\ Framework$ to create offers within the limits of the current crowd's capacities. On a request an individual offer can be created and provided. The negotiable items of a later assignment and their boundaries depend on the available resources and their current scheduling. Both can be gathered by checking availability at the *Crowd Scheduler* and the current crowd member status at the *Diagnosis and Planning* module. The provided offer is revised by the

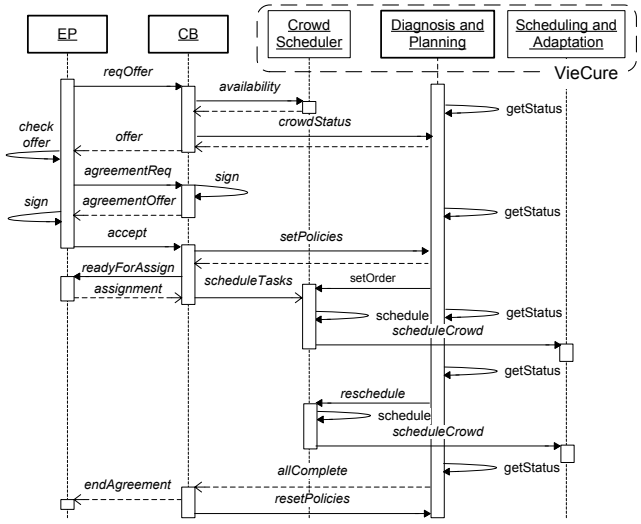


Figure 3. Agreement management life-cycle.

EP comparing it against the in-house requirements. If not pleased, negotiation starts over again or a different CB is considered as service provider. Finally, a satisfying offer including the agreement details in the objectives is signed by both parties and enacted with the according policies for assignment management.

Scheduling. Once the agreement's objectives are translated into new scheduling policy rules, the CB is ready to take over the assignment and schedule the tasks in sequence. Meanwhile, tasks are arranged by the *Crowd Scheduler*'s strategy according to a valid order and their priorities. The scheduling plan is propagated to the *Scheduling and Adaptation* module. Then, the in sequence distribution of the tasks to the members concludes the scheduling phase.

Rescheduling. Situation and behaviors change. Some of the members will reject their scheduling plans. The *Diagnosis and Planning* module receives the current assignment status of all involved members and reacts to rejects with rescheduling orders for the *Crowd Scheduler*. As the members' status has changed with previous assignments, the *Diagnosis and Planning* module must also adapt the scheduling strategy for any reassignment. Generally, it keeps a record of the rejects and accepts and adapts the present scheduling strategy to the current crowd's acceptance behavior.

In the line with the requirements emerging from the agreement's life-cycle in the next section we detail a structure for the agreements and discuss examples of fundamental quality attributes applicable for crowdcomputing.

V. Agreements and Quality

The growing interest in outsourcing tasks to platforms hosting crowds entails the demand for reliable business contacts, clear rules, and applicable agreements. A prerequisite of our approach is a reliable behavior monitoring. This ensures up-to-date data for metrics and quality attributes.

A. Agreement Structure

The used agreement model is inspired by the work of IBM and the GRAAP Working Group presented in Section II. The overall structure includes header, agreement items, and terms. The header comprises the agreement's parties details and contact information. In the contractual items the agreement's subjects are listed. These include the service content (i.e., in Web-service environments WSDL location, endpoint, and operation) along with scheduling information, metrics and their measuring method. Finally, the terms provide the objectives, SLOs respectively, and their validity period. Threshold values expresses the desired relation between objectives and metrics defined in the items. In the next section, we provide a number of quality metrics that can be applied and measured in SOA based crowdcomputing environments, and thus, aligned to the described structure.

B. Quality Parameters

Analyzing the requirements of the scenario in Section III, Table I represents a plausible list of quality attributes for negotiation in crowdcomputing. The crowd broker manages the attributes for registered crowd members and constantly updates their value.

Table I
Negotiated quality attributes.

Quality Attributes	Description
reliability	predicted confidence in the assignment acceptance of a member.
load	estimated task queue size of a member.
overlap	match between a member's capabilities and the task's requirements.
cost	fee demanded by a member for processing a task.

The first listed quality attribute, *reliability*, is related to the assignment acceptance behavior of a particular member. It reflects the difference between total assigned and rejected tasks. The monitored *load* represents the current task load at a member. The *overlap* factor indicates how suitable a member is for a certain task assignment by calculating the overlap of its capabilities and the task's requirements. Lastly, the *cost* attribute states the maximum fee that can be charged by a member for a processed task.

VI. Task Scheduling in the Crowd

Next, we provide an example of an agreement in extended WSLA (Web Service Level Agreements)[1] notation. The format is XML-based, thus, processable for the phases of negotiation, and extraction of agreement items and objectives. Further, this helps to fit the extracted hard- and soft-constraints into a self-adaptive scheduling algorithm that is formalized in Algorithm 1.

[1] http://www.research.ibm.com/wsla/WSLA093.xsd

A. Agreement Setup

The XML examples in Listing 1 and Listing 2 detail a sample WSLA agreement applicable to crowdcomputing environments. Only the important parts are fully listed. Additionally, the structure has been extended to fit the crowdcomputing particulars.

```
1  <wsla:SLA
2  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xmlns:wsla="http://www.ibm.com/wsla"
4  xmlns:hps="http://www.infosys.tuwien.ac.at/hps/"
5  name="SwTestCrowdSLA5312">
6   <wsla:Parties>
7    <wsla:ServiceProvider name="CommunityBroker">
8     <!-- ... -->
9    </wsla:ServiceProvider>
10   <wsla:ServiceConsumer name="EntryPoint">
11    <!-- ... -->
12   </wsla:ServiceConsumer>
13  </wsla:Parties>
14  <wsla:ServiceDefinition name="TestService">
15   <wsla:Operation
16    xsi:type="wsla:WSDLSOAPOperationDescriptionType"
17    name="AddActivity">
18    <!-- schedule period -->
19    <wsla:SLAParameter name="FeedbackExpected"
20     type="int" unit="Days">
21     <wsla:Metric>CountDays</wsla:Metric>
22    </wsla:SLAParameter>
23    <!-- config details: name, wsdl-location, binding -->
24   </wsla:Operation>
25   <hps:SLAActivity name="Testing">
26    <hps:SLAInvolvedProfiles>
27     <hps:SLAProfileTyp>UI Test</hps:SLAProfileTyp>
28     <!-- ... functional, performance, security -->
29    </hps:SLAInvolvedProfiles>
30    <hps:SLAReportURI>http://.../reports</hps:SLAReportURI>
31    <wsla:SLAParameter name="TasksCost"
32     type="float" unit="Euro">
33     <wsla:Metric>MaxCost</wsla:Metric>
34    </wsla:SLAParameter>
35       <!-- reliability, load, overlap -->
36   </hps:SLAActivity>
37  </wsla:ServiceDefinition>
38 <!-- wsla Obligations -->
39 </wsla:SLA>
```

Listing 1. Involved parties and body.

After the contract parties' details, *ServiceProvider* and *ServiceConsumer* listed in lines 6 to 13, Listing 1 states the items from line 14 to 37. These are a collection of *ServiceObjectType* items including scheduling, operation description, and configuration, and also, an *SLAParameter* (`FeedbackExpected`) arranging periodic feedback. Important and a new contribution to this part is the *SLAActivity* (lines 25 to 36). It extends WSLA's *ServiceObjectType* and states what kind of member capabilities must be involved in the testing activity (line 27), at which URI the final testing reports are expected (line 30), and at the end, the *SLAParameter*s for the assigned activity.

These include for example, all the quality attributes as presented previously in Table I. The parameter is identified by a name, a type, a unit, and related to a metric for estimation.

Listing 2 shows the terms as *Obligations* of the contract including all SLOs. An SLO consists of an obliged party, a validity period, and *Expression*s that can be combined

```
1  <wsla:Obligations>
2  <wsla:ServiceLevelObjective name="sloSrv"
3       serviceObject="AddActivity">
4   <wsla:Obliged>CommunityBroker</wsla:Obliged>
5   <!-- Validity -->
6    <wsla:Expression>
7     <wsla:Predicate xsi:type="wsla:Equal">
8      <wsla:SLAParameter>FeedbackExpected</wsla:SLAParameter>
9      <wsla:Value>7</wsla:Value>
10    </wsla:Predicate>
11   </wsla:Expression>
12  </wsla:ServiceLevelObjective>
13  <wsla:ServiceLevelObjective name="sloAct"
14       serviceObject="Testing">
15   <wsla:Obliged>CommunityBroker</wsla:Obliged>
16   <!-- Validity -->
17   <wsla:And>
18    <wsla:Expression>
19     <wsla:Predicate xsi:type="wsla:LessEqual">
20      <wsla:SLAParameter>TasksCost</wsla:SLAParameter>
21      <wsla:Value>50.0</wsla:Value>
22     </wsla:Predicate>
23    </wsla:Expression>
24    <!-- expressions for reliability, load, overlap -->
25   </wsla:And>
26   <wsla:EvaluationEvent>TaskAssignment</wsla:EvaluationEvent>
27  </wsla:ServiceLevelObjective>
28  <wsla:QualifiedAction>
29   <wsla:Party>CommunityBroker</wsla:Party>
30   <wsla:Action actionName="violation" xsi:type="Notification">
31    <wsla:NotificationType>Violation</wsla:NotificationType>
32    <wsla:CausingGuarantee>sloAct</wsla:CausingGuarantee>
33    <wsla:SLAParameter>MaxCost</wsla:SLAParameter>
34    <!-- expressions for reliability, load, overlap -->
35   </wsla:Action>
36  </wsla:QualifiedAction>
37  </wsla:Obligations>
```

Listing 2. Obligations and SLOs.

with a logic expression (e.g., *And*). The content of the expressions connects the pool of *SLAParameters* of the items to a predicate (e.g, `Equal`) and threshold value (*Value*). The final tag *QualifiedAction* defines the consequence of an SLO violation. In the example case, if a threshold of SLO `solAct` is violated an action *Notification* is called.

B. Scheduling Algorithm

As mentioned in the introduction there is numerous factors that influence the human behavior. Thus, one crucial factor when scheduling tasks in crowdcomputing environments is that one cannot rely on the constant availability of resources (i.e., humans). These dynamics and the system size inhibit a fully automated scheduling approach. Instead in this work we base our assumptions on a semi-automated task assignment algorithm with a human, e.g. CB, in-the-loop. Such an approach remains highly adaptable and suitable for crowds.

Tasks that are outsourced to the crowd have three important categories of properties. First, keywords describe the task's type and required capabilities. These are matched to the members' profiles. Second, a task has temporal constraints for the scheduling process. A task has a *latest begin time*, and a *length* as the estimated time spent to complete the task. Combined, these properties define the *deadline*, and also, the latest possible *reassign time*. A task assignment fails if members do not acknowledge processing until the task's latest begin time. In many scenarios including soft-

ware testing tasks depend on one another. Thus, a property of a task can either represent a parent task with dependent subtasks, a subtask or an independent task. The impact of a failed processing of a parent task results also in a failure of the dependent subtasks. This needs to be considered when scheduling complex tasks in crowdcomputing environments.

In the following we detail the steps of the crowd scheduling algorithm. Let us define the set of crowd members $U = \{u_1, u_2, \ldots\}$ and the set of tasks $T = \{t_1, t_2, \ldots\}$ to be processed by the crowd. The goal of the algorithm is to assign each task to one individual crowd member. Notice, we do not make any assumption about the role of the broker. In fact, the broker may be implemented as a software service, thus the procedure is fully automated, or the procedure may be performed under human supervision.

Algorithm 1 Task scheduling in the crowd.

Require: $T \neq \emptyset \wedge U \neq \emptyset$
1 $AT \leftarrow \emptyset$ /* Set of assigned tasks */
2 $FT \leftarrow \emptyset$ /* Set of failed (to assign) tasks */
3 **foreach** Task $t \in T$ **do**
4 /* Retrieve matching members with $U' \subseteq U$ */
5 $U' \leftarrow$ `getAllMembersMatchingTask(`t`)`
6 **foreach** Member $u \in U'$ **do**
7 /* Check additional constraints */
8 **if** `meetsDeadline(`u`,` t`)` **== false** $\vee$ `meetsResponseTime(`u`,` t`)` **== false then**
9 **continue** /* Do not consider as candidate */
10 **end if**
11 $score_{(u,t)} \leftarrow$ `calculateScore(`u`,`t`)`
12 $U'' \leftarrow$ `insertByScore(`$score_{(u,t)}$`,`U''`)`
13 **end foreach**
14 $CM[t] \leftarrow U''$ /* Save CM */
15 **foreach** Member $u \in CM[t]$ **do**
16 **if** `assignTask(`u`,`t`)` **== true then**
17 $AT \leftarrow AT \cup t$
18 **break**
19 **else**
20 $CM[t] \leftarrow CM[t] \backslash u$ /* Remove u from CM */
21 **end if**
22 **end foreach**
23 **if** $t \notin AT$ **then**
24 $FT \leftarrow FT \cup t$ /* Add t to FT */
25 **end if**
26 **end foreach**

Given the loop in Algorithm 1 (lines 3 to 26), three essential steps are performed:

Matching. A set of members whose profiles satisfy a task's required capabilities (see line 5) are selected. The detailed profile matching procedure is, however, not detailed in this work. Also, the demanded degree of match depends on the nature of a task (parent or subtask). In our system, parent tasks demand for a broader area of expertise thereby requiring a full match of a task's keywords and a members capabilities.

Ranking. Next (see lines 6 to 13) additional filtering and ranking is performed. The steps are a mixture of hard- and soft constraints. First, `meetsDeadline` is a filter to evaluate a task's deadline against the user u's expertise profile and estimated load. An expert who has already proven experience collected by processing a given type of tasks may finish a task faster compared to a relatively unexperienced user. Also, the estimated current load of a user from a particular broker's point of view is taken into account; i.e., whether or not u will be able start processing a task without violating time constraints such as deadline. Second, the filter `meetsResponseTime` is based on the user's context. Crowd members may be scattered around the globe. Therefore, different timezones as well as preferred working hours may prevent a member from processing a task in a given time frame. These two filters rely on hard constraints and cannot be influenced. The soft constraints are covered by a third type procedure. It performs a ranking based on a set of metrics that are specified in the context of an agreement. The details regarding this step (line 11) are given in the following.

Assignment. Finally, based on the ranked candidate list $CM[t]$ (see line 14), the broker attempts to assign the task t (see line 16). Indeed, an assignment may fail due to the aforementioned challenges in crowdcomputing such as limited knowledge of the user's actual load. If the assignment succeeds, the task t is added to the set AT and our algorithm continues to process the next task. Otherwise, the member u is removed from the list of candidate members $CM[t]$ (see line 20). Notice, the list $CM[t]$ is kept for reference in case a given task t needs to be seized from the assigned member due to lack of processing progress. In this case, the next (top-ranked) member in $CM[t]$ is picked. The set of failed tasks FT may require renegotiation of agreement metrics. Renegotiation procedures are currently not covered by our approach. The scoring function used in our algorithm (line 11) is defined as

$$score_{(u,t)} = \left[\sum_{m \in M'} |w_m| \times score(u,m)^p \right]^{1/p} \quad (1)$$

The detailed parameter description can be found in Table II. This approach is based on a model for *simultaneity* and *replaceability* of preference parameters known as Logic-Scoring of Preferences (LSP), e.g. [23].

The parameter p can be assigned manually based on the desired scoring behavior [23] or calculated automatically. Here we use a simple pattern to calculate p based on the homogeneity (or diversity) of the preference weights $w_m \in W$ where W is the set holding preference weights for each metric m: if $\texttt{max}(W) - \texttt{min}(W) > \texttt{avg}(W)$ use $p = 1.5$, if $\texttt{max}(W) - \texttt{min}(W) = \texttt{avg}(W)$ use $p = 1$, otherwise use $p = 0.5$. This means that replaceability should be preferred over simultaneity if the weight values (preferences) vary highly expressed by the relationship between `max` and `min` values compared to the average (`avg`) weight.

Table II
DESCRIPTION OF RANKING PROCEDURE.

Symbol	Description
m	Metric $m \in M'$ with $M' \subseteq M$ as defined in the SLA. Examples of a set of metrics and corresponding values that are obtained through monitoring and mining include *reliability* and *load*.
w_m	The weight assigned to a given metric m such that $\sum_{m \in M'} \lvert w_m \rvert = 1$. Weights are thereby not assigned independently or arbitrarily but rather with respect to preferences for individual metrics.
$score(u, m)$	Scoring function for a given user u. The sign of a weight w_m is used to determine whether higher or lower values denote better scores. For example, higher reliability results in higher scores (i.e., $+w_{rel}$) whereas lower values in costs are more desirable (i.e., $-w_{cost}$).
p	Parameter to configure *simultaneity* or *replaceability* of a metric. Simultaneity is a desired property if *each* metric m is important. As an example, a member should have good scores for *both* overlap and reliability as opposed to having only good overlap. Replaceability means that higher overlap may compensate for low reliability or vice versa.

VII. SIMULATION AND EVALUATION

The main idea of the evaluations is to identify the boundaries of the scheduling algorithm presented in Section VI in crowdcomputing environments to support reasonable negotiations of quality attributes in agreements. Monitored with the metrics defined by common crowdsourcing quality parameters presented in Section V we setup a simulated crowdcomputing environment with properties related to the environment outlined in the scenario. We consider only a subgroup of a larger and more complex crowdcomputing network because in contrast to existing environments with our broker approach resources can be organized for specific skills. In particular, we study the challenges and effort of scheduling, and also of rescheduling, tasks for differently behaving crowd members.

A. Environment Setup

The **simulated crowd environment** comprises a framework implemented in Java language with a CB singleton instance, a crowd of 128 members, a task model, and various helper instances for the score calculation as detailed in the previous section. The single CB holds a reference to all crowd members in a registry. In a loop it tries to schedule batches of tasks for the members according to our scheduling algorithm and to reschedule tasks if rejected.

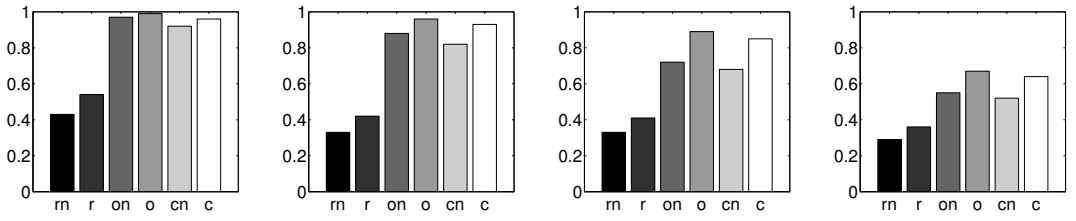
Each **crowd member** has its own capability profile. Additionally, the member exposes a predefined behavior in task assignment and task processing. The acceptance behavior in task assignment depends on the current task queue size. Whilst on an empty queue the member is eager to get task assignments, the enthusiasm decreases linearly to full reject at a number of 6 tasks in queue. The processing behavior is assigned at bootstrapping. Three different types of behavior patterns are known to the system. The first one processes tasks with a probability of 20%, the second one 50%, and the last one 80%. The behavior patterns are equally distributed among the members. Finally, members charge a fee for their service. In a further extension, a quarter of the members are randomly assigned with a fee rate which we consider to exceed the fixed rate negotiated in the agreement.

Tasks have temporal properties as discussed previously. Tasks have a latest begin time and length in time slices. In our experiments reassign time was set to 3 slices prior to latest begin. If an assigned member does not acknowledge processing until the task's latest begin time, the task fails. The impact of a failed processing of a complex (parent) task results also in a failure of the dependent subtasks.

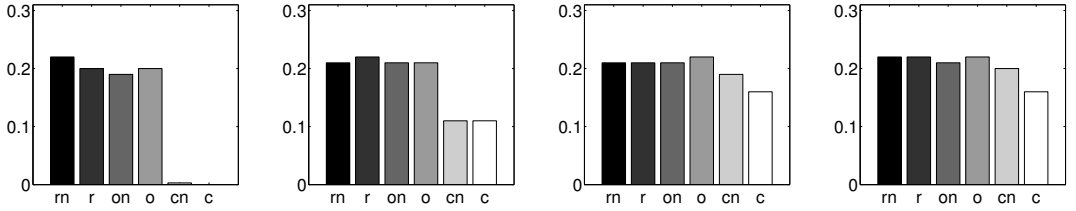
At bootstrap the framework instantiates broker and members and fills the registry. The runtime is split in two phases including **scheduling and rescheduling** activities. Both phases apply Algorithm 1 to assign a batch of tasks. Members have different behavior in accepting and processing tasks. Thus, the goal of the two phases is to minimize the task failure rate, and in the later experiments, to minimize the costs of processing by choosing the currently less expensive crowd member. Two different scheduling **strategies** have been implemented and evaluated. The first one, a random strategy, picks from the set of available and capable crowd members randomly the next candidate for the assignment. The second one, the metrics assisted strategy, uses the previously presented metrics in Table I to select the next candidate from the set. During an independent task assignment the metrics weight factor is equally distributed. Nevertheless, this strategy is also aware of task dependencies. Once a parent task is assigned, an extra weight (60%) is set to the *overlap* metric to move the most overlapping members to the begin of the selection queue.

B. Experiment Results

The main goal of the experiments is to illustrate the effectiveness of the metrics enhanced scheduling algorithm outlined in Algorithm 1; also compared to random scheduling. In the first set of experiments Figures 4(a), 4(b), 4(c), and 4(d), the percentage of *completed tasks* with respect to the total number of assigned tasks is an indicator for the effectiveness. In the next set of figures (4(e), 4(f), 4(g), and 4(h)), the *exceeding costs* are considered. This second type of experiments illustrates the percentage of completed tasks that exceed the SLA cost objective. As aforementioned, this is caused by members that exceed the cost with their fee. The results of the two sets are presented by the rows in Figure 4. Furthermore, to explore lower, and in particular, upper bounds of the algorithm different batch sizes of tasks are scheduled. A batch size defines the number of tasks that are received from the EP and need to be scheduled in the next period. In our simulations, once all tasks of a batch assignment are past their deadline, a new similar size batch



(a) Processed - 256 batch. (b) Processed - 512 batch. (c) Processed - 768 batch. (d) Processed - 1024 batch.



(e) Over cost - 256 batch. (f) Over cost - 512 batch. (g) Over cost - 768 batch. (h) Over cost - 1024 batch.

Figure 4. Results of the experiments for various scheduling strategies with different task batch size. Strategies include *random* scheduling with *no* rescheduling phase *(rn)* and with rescheduling *(r)*, according to metrics *ordered* scheduling with *no* rescheduling (on) and with rescheduling (o), and finally, metric-ordered scheduling including *costs* with *no* rescheduling *(cn)* and with rescheduling *(c)*.

is scheduled until a number of 10000 tasks total are assigned. Note, the fixed size crowd has a maximum capacity of 768 tasks per period. The columns of the results in Figure 4 represent the different batch sizes.

As one immediately realizes form the first row of results, the amount of successfully processed tasks decreases with increasing batch size. From left to right, the bar at the very left of the experiment figures shows the results for a random scheduling strategy with no rescheduling phase **(rn)**. Together with the next bar, representing random scheduling with rescheduling **(r)**, they perform the worst with task processing peeks of only 40% for **(rn)**, and a few more than half (54%) for **(r)**, respectively, at a batch size of 256. Even with the rescheduling enabled, this strategy cannot be considered satisfactory for any of the batch sizes. An interesting fact is however, that their success rate remains the same for the 512 and 768 batches. This indicates that for this strategy a medium to high task queue load results in a similar success rate. The next two bars represent metric-ordered scheduling. In contrast to the 4th bar (**(o)**) the 3rd bar shows the result without rescheduling phase **(on)**. Starting with 97% and 99%, respectively, for **(on)** and **(o)** at size 256, they decrease to 55% and 67% at size 1024 when task queues are too small to schedule all tasks. Here the improvement of rescheduling phase is apparent when testing higher batch sizes. At size 1024, if a batch is to large for the task queues, the success rate decreases notably for both settings. The last two configurations, cost aware ordering with no rescheduling **(cn)** and with rescheduling **(c)**, also consider costs when ordering the candidate set for a task. Interestingly, the impact to the success rate in comparison to cost-unaware ordering is only marginal. This is the result of simultaneity between the metrics.

The second row of experiments reflects the percentage of successfully processed tasks that, however, exceeded the expected costs. The results show, that with strategies that do not consider costs, the amount of tasks exceeding costs is around 20% for all batch sizes (first four rows). Only if the cost metric is included, costs can be saved for the minor and medium batch sizes. With size 256 almost no costs accumulate with **(cn)** and **(c)**. At size 512 half the costs can be saved as opposed to the other strategies. Starting with batches of size 768, the results of the two cost considering strategies diverge notably and perform similarly unacceptable with respect to cost-unaware strategies. At size 1024, for example, **(cn)** results in 20% over cost and **(c)** in 22%. As a synthesis, it can be observed that metric-assisted scheduling, generally, preforms twice as well as random scheduling. This remains true as long as task queues can schedule all tasks. Rescheduling helps to increase the success of processing in each case. Whilst the success difference remains on average around 9% for random scheduling, in metric-assisted scheduling the difference depends on the batch sizes with extremes at size 256 with only 2% difference and 16% at size 768. If costs are also taken into account, both random and metric-assisted scheduling perform comparably. The success of an additional cost factor for metric-assisted scheduling depends on the batch size and is negligible for large batch sizes.

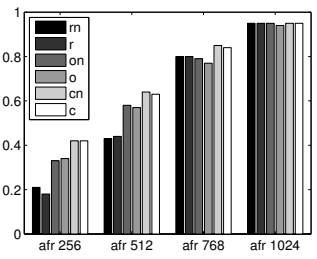


Figure 5. Assignment failure rate (afr) for different batch sizes.

The scheduling effort is another cost source of task assignments. Referring to the experiments in Figure 5 we list the assignment failure rate (*afr*) for the different batch sizes and varying strategies. The results show the percentage of scheduling requests that are rejected on the first request. The results highlight how the increasing batch size reduces the impact of the *afr* for the metric-assisted strategies (**(on)**,**(o)**, **(cn)**, and **(c)**). These strategies force similar member selections for comparable tasks. Thus, for small batches and cost on focus, the low fee members reject on full queues. For size 256 and 512 the effort difference between **(rn)**, **(r)** and **(cn)**, **(c)** is around 20% and decreases rapidly for larger sizes.

VIII. Conclusion and Future Work

The main objectives of this work were to present a framework which successfully manages task assignment in a crowdcomputing environment. In this work we solve the problem with an adaptive and multi-objective task scheduling. Objectives derive from an agreement between a company and a crowd broker. As we base our environment on an SOA infrastructure with its convenient technical grounding for large-scale environments, we are also able to take advantage of the already existing models for agreements (WSLA and WS-Agreement). Our extension to one of the existing standards which includes assignment identifying information and relation to different objectives, fits the requirements of our crowdcomputing scenario. When deploying the assignment as independent and dependent tasks to capable members, these objectives can than be used as soft- or hard-constraints for a weighted scheduling strategy. The results of our experiments highlight the advantages of an objective-aware metric ordered strategy in contrast to plain random scheduling while task loads remain in between the boundaries. Nevertheless, the results show, the effort for ordering the assignment lists induces a higher effort in scheduling.

In future work we focus our research on two additional challenges. There remains the need for a fully automated translation of SLOs into the scheduling objectives. Vice-versa, this would also assist a semi-automatic approach for the negotiation phase where current policies are translated into SLOs that the crowd broker wishes to negotiate. Finally, a major challenge of crowdsourcing is the fluctuation and unpredictable behavior of the resources. As already mentioned in the scenario, however out of scope for this work, we see great potential in the role of the Head Hunter broker. Its main function is to lend the missing but required members to crowd brokers on resource shortage. For that purpose we plan to study methods and algorithms to derive tendencies of expertise evolution in crowdsourcing.

Acknowledgments

This work is supported by the EU through the FP7 projects S-Cube (215483) and COIN (216256).

References

[1] D. Brabham, "Crowdsourcing as a model for problem solving: An introduction and cases," *Convergence*, vol. 14, no. 1, p. 75, 2008.
[2] Amazon Mechnical Turk, http://www.mturk.com, May 2011.
[3] M. Vukovic, "Crowdsourcing for Enterprises," in *Proceedings of the 2009 Congress on Services*. IEEE, 2009, pp. 686–692.
[4] H. Psaier, L. Juszczyk, F. Skopik, D. Schall, and S. Dustdar, "Runtime Behavior Monitoring and Self-Adaptation in Service-Oriented Systems," in *SASO*. IEEE, 2010, pp. 164–174.
[5] H. Psaier, F. Skopik, D. Schall, and S. Dustdar, "Behavior Monitoring in Self-healing Service-oriented Systems," in *COMPSAC*. IEEE, 2010, pp. 357–366.
[6] LiveOps, https://www.livework.com/, May 2011.
[7] Yahoo! Answers, http://answers.yahoo.com/, May 2011.
[8] P. G. Ipeirotis, "Analyzing the Amazon Mechanical Turk Marketplace," *SSRN eLibrary*, vol. 17, no. 2, pp. 16–21, 2010.
[9] A. Andrieux *et al.*, "Web Services Agreement Specification." Open Grid Forum, 2007.
[10] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck, "Web Service Level Agreement (WSLA) Language Specification." IBM Corporation, 2003.
[11] H. Ludwig, T. Nakata, O. Wäldrich, P. Wieder, and W. Ziegler, "Reliable orchestration of resources using WS-Agreement," *HPCC*, pp. 753–762, 2006.
[12] Y. Che, "Design competition through multidimensional auctions," *The RAND Journal of Economics*, vol. 24, no. 4, pp. 668–680, 1993.
[13] D. Parkes and J. Kalagnanam, "Models for iterative multiattribute procurement auctions," *Management Science*, vol. 51, no. 3, pp. 435–451, 2005.
[14] J. Sousa, V. Poladian, D. Garlan, B. Schmerl, and M. Shaw, "Task-based adaptation for ubiquitous computing," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 36, no. 3, pp. 328 –340, May 2006.
[15] P. Cowling, N. Colledge, K. Dahal, and S. Remde, "The Trade Off Between Diversity and Quality for Multi-objective Workforce Scheduling," in *Evolutionary Computation in Combinatorial Optimization*, vol. 3906, 2006, pp. 13–24.
[16] A. Caprara, M. Monaci, and P. Toth, "Models and algorithms for a staff scheduling problem," *Math. Program.*, vol. 98, no. 1-3, pp. 445–476, 2003.
[17] A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier, "Staff scheduling and rostering: A review of applications, methods and models," *European Journal of Operational Research*, vol. 153, no. 1, pp. 3 – 27, 2004, timetabling and Rostering.
[18] G. La Vecchia and A. Cisternino, "Collaborative Workforce, Business Process Crowdsourcing as an Alternative of BPO," in *Current Trends in Web Eng.*, vol. 6385, 2010, pp. 425–430.
[19] uTest, http://www.utest.com/, May 2011.
[20] D. Schall, H.-L. Truong, and S. Dustdar, "Unifying Human and Software Services in Web-Scale Collaborations," *IEEE Internet Computing*, vol. 12, no. 3, pp. 62–68, 2008.
[21] F. Skopik, D. Schall, and S. Dustdar, "Modeling and Mining of Dynamic Trust in Complex Service-oriented Systems," *Information Systems*, vol. 35, pp. 735–757, 2010.
[22] A. Agrawal *et al.*, "WS-BPEL Extension for People (BPEL4People), Version 1.0." 2007.
[23] J. J. Dujmović and H. L. Larsen, "Generalized conjunction/disjunction," *Int. J. Approx. Reasoning*, vol. 46, pp. 423–446, December 2007.